**Définition de la stratégie et des méthodes de tests par étape :**

1. Démarrage du projet :

Pour pouvoir démarrer un projet, il faut que l’équipe projet se mette d’accord sur des points qui seront essentiels pour mettre en place la bonne stratégie de test. C’est la phase de cadrage projet.

* Quelle méthode de développement (Agile Srum, Agile Kanban, V, autres…)
* Quelle équipe (PO, Scrum Master, développeurs, testeurs, QA, …)
* Les outils (de développement, de TU, de tests d’acceptance, de spécifications, de gestion des sources, de gestion de tickets, d’intégration continue …)

J’ai l’habitude chez Legrand de travailler en transverse avec différentes équipes projet, des méthodes et outils hétérogènes. C’est pourquoi cette phase est nécessaire et importante dans mon poste actuel.

Dans le cas de Wopata, j’imagine que les pratiques et outils sont beaucoup moins hétérogènes et qu’il y a certainement un consensus dans l’entreprise. J’imagine plutôt des méthodes itératives (Agile, Scrum) et en terme d’outils, je crois savoir que vous travaillez avec la suite Atlassian (Confluence, Jira …).

Un point très important concernant les outils, il est vraiment recommandé d’avoir un outil de spécification des Users Stories et des tests commun avec lequel on puisse faire de la traçabilité. Moi je travaille avec l’outil Polarion qui est un outil payant mais très complet. Je suis bien sure capable de m’adapter à tout autre outil. Ce qui est important pour l’assurance qualité c’est uniquement de pouvoir lier les tests d’acceptance avec les stories.

De même, il est très important d’avoir un outil de gestion des sources et d’intégration continue pour permettre de lancer l’ensemble des TUs à chaque commit et tous les tests d’acceptance chaque nuits. L’idée étant de toujours construire sur des bases solides en vérifiant à tout instant la non régression.

1. Vision

Pour dégager la vision du produit et les principaux cas d’usages, il y a un bon outil que je pratique beaucoup chez Legrand c’est le Story Mapping. Il est important que l’équipe projet au complet soit présente (idéalement le client aussi, sinon c’est le PO qui peut faire office de…).

Le test commence vraiment ici car le/les testeurs présents commencent à rentrer dans le sujet et se basent sur une vision commune avec l’ensemble de l’équipe.

1. Définition de la stratégie de test

Après avoir mis à plat la structure du projet et la vision, on peut rentrer dans le vif du sujet et définir concrètement la stratégie. L’objectif c’est de répondre à la question jusqu’où on va en terme de test et quel est le niveau de risque que l’on accepte de prendre dans le projet.

Il y a à mon sens 3 niveaux de tests qui ne sont pas négociables pour tous les projets donc aussi pour le cas particulier du projet OR Master:

* les TUs : idéalement le développement devrait être fait selon le principe de Test Driven Developpement. C’est à les développeurs codes leur script de Tests unitaires avant de coder la fonction et se servent du test comme notion de Done pour la fonction.
* les Tests d’acceptance : la spécification du test d’acceptance est écrite collégialement par les développeurs et les testeurs avant la cotation de la story correspondante (d’où l’importance d’un outil commun de spec de stories et de tests). Les scripts des tests d’acceptances sont en suites écrits par un/des testeurs indépendants. Idéalement les testeurs ne font que cela dans le projet et ne sont pas les développeurs du projet, par contre ils peuvent très bien être développeur sur un autre projet.
* Le suivi des crashs une fois l’App publiée. Ceci permet d’alimenter le bug tracer en cours de maintenance de l’application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TU | Oui | Interne | Automatique | 100% des modules |
| Test Acceptance | Oui | Interne | Automatique | 100% couverture fonctionnelle |
| Tests d’ergonomie/UX | Oui/Non | Interne/Externe | Manuel | Au moins 10 personnes par pays |
| Tests de design | Oui/Non | Interne/Externe | Manuel | tailles d’écran représentant 80% du marché européen |
| Test de langue/Traduction | Oui/Non | Interne/Externe | Manuel | Au mois 1 personne par langue |
| Tests de performance (montée en charge) | Oui/Non | Interne/Externe | Manuel | Selon volumétrie d’utilisation de l’APP |
| Tests de sécurité (intrusion) | Oui/Non | Interne/Externe | Manuel |  |
| Suivi des Crash | Oui | Interne | Automatique | Crashlytics, … |

Dans le contexte de l’exercice OR Master, et de ce que j’ai pu comprendre du projet sans avoir tous les éléments des chapitres 1 et 2, je préconiserais en plus des TU et Tests d’acceptante :

* Des Tests de design : ce sont des essais que je traite chez Legrand de façon externalisée car je ne souhaite pas investir dans une batterie de devices qui sont dépassés à peine achetés. Je travaille aujourd’hui avec la société B/acceptance qui me réalise des essais manuels sur différents devices avec différentes tailles d’écran (4'', 4.3'', 4.5'', 4.7'', 5'', 5.1'', 5.5'', 7'', 10.1'', 11'') pour vérifier la responsivité du design. La société StarDust est aussi bien placée sur ce genre de tests en France.
* Des Tests de traduction et d’UX manuels avec des utilisateurs réels : je n’ai jamais pratiqué ce genre de test à grande échelle chez Legrand. Ce que nous faisons c’est seulement diffuser à un panel restreint au sein des équipes R&D et marketing quelques semaines avant le lancement (c’est ce que nous appelons des chantiers tests). Mais dans votre cas cela peut être intéressant car il y a une vraie diversité de pays et de langue et l’appli s’adresse à une clientèle pro mais non formée à l’outil donc pouvant être considérée comme un marché grand public du point de vue de l’expérience utilisateur. Il y a pleins de façon d’adresser ce type de test en fonction des moyens que l’on veut y mettre. Cela peut simplement passer par une sollicitation directe des clients à qui on fait essayer le produit ou par des programmes de tests externalisés plus complets où on paye un panel de testeur réel pour réaliser des essais définis sur l’application et la noter : ex usertesting …

Dans votre contexte : données non critiques, dénie de service non critique, volumétrie faible, je ne pense pas que des tests de sécurité et de performances soient indispensable. A discuter. En ce qui me concerne, ce sont des essais que j’ai toujours externalisé (ex : test d’intrusion par société Sysdream, ou test de montée en charge sur un service me mise à jour M2M par la société Smile)

1. Processus de tests interne en mode itératif.

Intervention du test dans le process de développement :

* A l’écriture de la Story, le testeur intervient pour écrire la spécification du test d’acceptance en détaillant bien le cas nominal et les cas alternatifs et/ou aux limites.
* Lors de la planification du Sprint, la charge de test est incluse dans la cotation de la story.
* Lors du découpage en tâche, une tache test est crée pour la réalisation du script du test d’acceptance pour le testeur. Remarque : une tâche TU est également créée pour le développeur.
* Le scriptage sur l’outil de test auto (Ranorex, Robot Framework …) intervient dans le Sprint en cours.
* La notion de Done au niveau du script inclus
  + Couverture TU 100% + rapport PASS
  + Couverture Tests acceptance 100% + rapport PASS
  + Une demo est faite et validée par le client final (forme de test utilisateur)
* La notion de Done au niveau de la release inclus :
  + - Couverture TU 100% + rapport PASS
    - Tests acceptance 100% + rapport PASS
* Rapport externe design
* Rapport externe traduction / UX
* Autres rapports de test

Tout ce process permet d’assurer la qualité et la non régression des softs développés. C’est le filet de sécurité.

Il est bien évident que cette partie très standardisée doit être complétée par du test beaucoup moins formel. C’est ce que j’appelle le déverminage. En effet en parallèle de la mise au point des scripts de test, le testeur met en œuvre la fonction à tester manuellement de manière informelle et non tracée.

De même dans la partie écriture des specs de test, il est bon d’avoir un formalisme assez standardisé (un genre de language de test). En ce qui me concerne, j’utilise le formalisme suivant :

* 1 test case par fonction
* Dans chaque test case, on trouve un cas nominal et des cas alternatifs
* Chaque cas est décrit selon le formalisme suivant :
  + GIVEN, WHEN, THEN

Voir dans l’exemple fourni des spécifications des cas de test pour OR Master.

1. Scripts de test, automatisation et rapports

N’étant pas développeur, je travaille avec un éditeur de script ce qui me permet d’avoir une couche d’abstraction par rapport au langage. Je scripte donc avec l’IDE de Robot Framework : Ride pour les tests des logiciels embarqués et des interfaces logiciel et web. J’utilise l’outil Ranorex pour les tests des applications mobile Android et IOS.

Chez Legrand à ce jour nous n’avons développé que des App Android !! Je n’ai donc pas été confrontée au problème des tests Cross- Platform comme vous pouvez l’avoir sur votre projet OR master.

Cependant, j’ai importé la bibliothèque Appium dans l’outil Robot Framework que j’utilise et effectivement les fonctions propres à IOS et Android sont présentes dans la librairie.

Ranorex qui est un outil payant est lui aussi compatible Android et IOS.

En terme de gestion des scripts pour votre projet, je pense qu’une bonne solution pour gérer l’hétérogénéité serait de créer des scripts de setup séparé pour Android, IOS et service web et de scripter les tests fonctionnels avec un script par type de device si les fonctions utilisées dans le script sont spécifiques au device. Ensuite l’utilisation des tags sur les fichiers de test permettra avec un outil d’automatisation type Jenkins de lancer les campagnes en fonction des devices.

On pourrait imaginer que trois campagnes soient lancées chaque nuit : la campagne iphone, la campagne android et la campagne web. Si une campagne unitairement est trop longue on peut envisager une rotation sur 3 jours ou lancer les 3 campagnes sur 3 serveurs en parallèle.

3 rapports sont donc édités par Jenkins à la fin du process de tests automatisés.

1. Gestion des sources, environnement de dev et environnement de tests

Il faut un outil commun pour gérer les sources et les scripts de test du projet (Git, SVN …)

Les fichiers de test sont nommés par le n° de spec correspondant

Les développeurs doivent travailler sur leur environnement de développement et gérer leurs sources en ‘’branche’’ tant que la fonction est en cours de développement et que le TU n’est pas Pass. Seule cette condition permet de ‘’commiter’’ sur le tronc.Le testeur sur son environnement de test, doit récupérer la fonction commitée sur le tronc avant de pouvoir jouer une campagne de test automatique.

1. Tests sur devices réels ou simulés

Test automatiques sur devices simulés.Tests manuels sur devices réels.