UNIVERSITÉ DE SHERBROOKE

Faculté de génie
Département de génie électrique et génie informatique

PROJET S5
ASSURANCE QUALITÉ

Présenté à
Jean-Baptiste Michaud

Présenté par
l'équipe P4

Sherbrooke – 23 février 2017

TABLE DES MATIÈRES

1 ASSURANCE QUALITÉ

1.1 PLAN DE L'AQ (STRATÉGIE, OBJECTIFS)

1.1.1 BUTS DE L'ASSURANCE QUALITÉ DANS LE PROJET

- S'assurer, de manière efficace, que les spécifications techniques sont respectées.
- Faciliter la gestion (et résolutions) d'erreurs des modules techniques.

1.1.2 FONCTIONNEMENT DE L'ASSURANCE QUALITÉ POUR NOTRE ÉQUIPE

On utilise la plateforme Lean Testing (app.leantesting.com)

Préalablement à la conception de chaque module, il faut établir un plan de test unitaire pour chaque fonctionnalité dudit module.

Pour les tests, on a :

- L'équipe qui conçoit les tests doivent être différents de l'équipe qui les exécute.
- Chaque test doit contenir les trois sections suivantes :
    - Étapes préalables (*Preconditions*).
    - Étapes à exécuter (*Steps to execute*).
    - Résultat attend (*Expected result*).
- Il y a deux types de tests : régression et de module. Afin de spécifier un test de type régression, on met la priorité à un niveau critique (*Priority -> Critical)* et **on ajoute un R dans le nom du test**.
    - Environ 20 % des tests d'un module doivent avoir la mention de régression.
        - Ces derniers sont les tests clés qui s'assurent que le module fonctionne.
- Il faut que chaque test soit très clair ; l'équipe d'exécution ne doit rien déduire lors de la procédure.
- Nommer le test comme d'un nom significatif au test. Ne pas oublier si le test est de type régression.

Pour l'exécution de test, on a :

- Trois passes d'exécution possibles (par module) :
    - Première passe :
        - Première exécution du plan de test pour un module X.
        - Contient tous les tests pour le module.
        - Si un test échoue, il faut continuer l'exécution, corriger toutes les erreurs trouvées et refaire la passe.
    - Deuxième passe :
        - Il faut que la première passe soit réussite à 100 % pour un module X.
        - Le module X est dépendant au module Y. Ainsi, lorsque le module Y est implémenté, il faut exécuter la deuxième passe du module X.
        - Contient seulement les **tests de régression** pour le module X.
        - Si un test échoue, il faut continuer l'exécution, corriger toutes les erreurs trouvées et refaire la **première** passe.
    - Troisième passe :
        - Il faut que, pour chacun des modules, la première et deuxième passe soient réussites à 100 %.
        - Contient seulement les **tests de régression** de tout le projet.
- Lorsqu'un module subit des changements importants, il est essentiel de refaire la première passe.
- Il faut créer une *Test Run*.
    - On la nomme comme suit :
        - <#_de_la_Passe> - <Nom_du_module>
    - Mettre une version si besoin est.
    - Sélectionner le module à tester (ne pas oublier que si passe 2, seulement sélectionner les tests de régression).
- Laisser un commentaire significatif à la fin du test.

Pour les résultats des tests, on a :

- Trois résultats possibles :
    - Réussite (*Pass*).
        - Les résultats attendus sont obtenus lors de l'exécution.
    - Échec (*Fail*).
        - Les résultats attendus ne sont pas obtenus lors de l'exécution.
        - Assigner un ticket d'erreur.
    - Bloqué (*Could not test*).
        - Un test préalable à ce test a échoué.

    - À tester de nouveau (*Not applicable*).
        - Impossibilité de tester en ce moment (manque de matériel, etc.).

Pour le ticket d'erreur, on a :

Création :

- Lorsqu'il y a un échec, il faut créer un ticket d'erreur (*bug report*).
    - Un même ticket d'erreur peut être la cause d'échec de plusieurs tests.
- La structure d'un ticket d'erreur est la suivante (juste remplir ce qui est listé ici, le reste est facultatif) :
    - Titre du bogue (*Bug title*).
        - <Nom_du_module> - <#_du_test> - <Court_titre_descriptif>
    - Priorité (*Priority*).
        - Si critique, cela veut dire que l'erreur doit être résolue le plus vite possible (blocker)
    - Type.
        - Type de l'erreur (fonction, texte, visuel, etc.).
    - Sévérité (*Severity*)
        - Si critique, cela veut dire que l'erreur est très grave.
    - Reproductivité (*Reproducibility*)
        - Combien de fois l'erreur arrive (à quelle fréquence).
    - Description
        - Décrire l'erreur de manière précise.
        - Finir avec la date de test
        - Ajouter le numéro du test case.

Résolution d'un ticket (s'assurer que c'est une équipe différente de ceux qui ont testé) :

- Changer le statut du ticket d'erreur à 'Acknowledged'.
- Tester de nouveau pour confirmer l'erreur.
    - S'il n'y a pas d'erreur, discuter avec le reporteur du bogue et déterminer la suite à l'interne.
    - Sinon, changer le statut à 'Confirmed'.
- Changer le statut à 'In progress' lorsqu'on travaille sur un correctif.
- Lorsque l'erreur est résolue, assigner le ticket à l'équipe qui a trouvé l'erreur et changer le statut à 'Resolved' (on peut ajouter la version dans laquelle le correctif est présent, si souhaité).

- L'équipe qui a maintenant le ticket doit tester de nouveau pour voir si le correctif est valide.
    - Si oui, on met le statut à 'Closed' et on refait la passe.
    - Sinon, on ajoute des commentaires expliquant pourquoi l'erreur n'est pas résolue, on réassigne le ticket à l'équipe qui a trouvé le correctif et on met le statut à 'Feedback'.
- Si le statut est à 'Feedback', on recommence le processus comme si le ticket était à 'Confirmed'.
- **Prendre note qu'il ne faut jamais supprimer un ticket d'erreur à des fins de traçabilité.**

1.2 TRAÇABILITÉ (LOGICIELS)

1.2.1 TRELLO

Afin de faire notre gestion de projet, nous utilisons le logiciel de gestion *Trello*. En effet, on utilise ce logiciel pour séparer les tâches entre les membres de l'équipe. Ainsi, on peut facilement relier chacun des mandats du projet aux différentes sous-équipes. De plus, comme chacun des membres a un compte personnel, on peut aussi faire la gestion des heures de travail pour chacun des membres, par mandat, ce qui facilite énormément les documents de gestion (courbe en S, Gantt avec un estimé d'heures, etc.).

1.2.2 GITHUB

Toutes les productions sont ensuite sauvegardées dans le logiciel *GitHub*. Ce qui est intéressant avec ce dernier est que lorsqu'on écrase un fichier, on est peut facilement revenir à la version d'avant au cas où on a fait des erreurs. De plus, chaque membre a un compte personnel, donc chaque version de chaque fichier est associée à un membre de l'équipe, ce qui nous permet de retracer facilement chaque modification a été effectuée par qui.

1.2.3 LEAN TESTING

Finalement, pour la gestion de l'assurance qualité, on utilise la plateforme web *Lean Testing*. Ce logiciel est parfait pour la rétractabilité de toutes nos erreurs, car la plateforme nous permet d'enregistrer des tickets d'erreur suite aux tests qui ont échoué. Par ailleurs, elle contient aussi tous nos tests, classés par module, et les résultats correspondants. On peut aussi voir les commentaires de chacun des membres de l'équipe dû au fait qu'on a tous un compte indépendant.

1.3 PLAN DE TEST POUR CHAQUE ACTIVITÉ DES DIAGRAMMES UML DE NIVEAU 1 ET 2

1.3.1 NIVEAU 1

L'usager doit être capable d'alterner entre les modes 'enregistrer' et 'rejouer partition'. En outre, il pourra aussi décider si l'enregistrement est en mode 'accord' ou bien 'note singulière'. Finalement, le mode 'partition', uniquement actif dans le mode 'note singulière', doit permettre de choisir le tempo désiré. Tout cela découle du monde 'enregistrer'. Ces choix seront tous effectués à l'aide du clavier.

Ainsi, l'AQ doit s'assurer que les dépendances de chacun des niveaux soient respectées.

1.3.2 NIVEAU 2

Le niveau 2 contient deux modules, soit la télécommande et le DSK. Pour commencer, la télécommande doit comprendre un menu qui permet à l'usager de lire, sur un écran LCD, des informations sur l'état du programme. De plus, le module télécommande contient le clavier qui permet à l'usager de changer les modes décrit dans le niveau 1.

Pour le DSK, on doit s'assurer qu'il reçoit les informations envoyées par l'usager à l'aide de la télécommande. Donc, le mode 'enregistrer' active le microphone qui écoute le son du piano joué par l'utilisateur. Par la suite, ce son est amplifié et converti en signal numérique à l'aide d'un ADC qui est enregistré dans la SDRAM. Puis, le signal est traité et traduit en notes/accord à l'aide d'une logique informatique pour être finalement remis dans la SDRAM à des fins d'affichage ou exportation de partition. Le mode 'rejouer partition' prend ce qui est sauvegardé dans la SDRAM et entre le signal numérique et le transforme en analogique à l'aide du CODEC pour qu'il soit par la suite entendu par l'usager.

Ainsi, pour le niveau 2, l'AQ doit s'assurer que l'usager peut choisir les modes à l'aide de la télécommande et que ce dernier voit les bonnes informations affichées sur l'écran LCD. L'AQ doit aussi regarder chacune des étapes intermédiaires à l'interne, c'est donc dire qu'il faudra voir si le signal venant du microphone est entendu, s'il est bien amplifié, si l'ADC transforme le signal analogique en numérique correctement. Aussi, on doit évaluer si le signal est bien enregistré dans la SDRAM, s'il est bien traité et traduit par le code. Pour l'autre mode, l'AQ doit s'assurer que le son enregistré au préalable est identique à lui qui est rejoué.

Donc, l'AQ devra faire des tests qui valident pour chacune de ces interactions entre ces activités, ce qui devrait tester la fonctionnalité du projet de manière efficace.

1.4 PLAN DE TEST POUR L'AUTOCORRÉLATION

Les tests unitaires sur la fonction d'autocorrélation devraient être sensiblement identiques entre la fonction de test sur matlab et celle qui sera codé sur CCS. En effet, notre application ferait une autocorrélation à toutes les 200ms sur les signaux qu'ils recevraient. Ainsi, dans matlab ainsi que dans CCS, il faudrait imposer un signal périodique de plus de 200ms et s'attendre à ce que l'autocorrélation nous renvoie un résultat de 1.

Une autocorrélation fructueuse d'un signal périodique devrait en effet renvoyer un coefficient de 1.

Aussi, il serait intéressant d'entré un signal périodique bruité de plus de 200ms. Il est en effet pertinent de trouver le niveau de bruit acceptable pour qu'un signal soit considéré périodique malgré le fait qu'il y aille du bruit dans une pièce. Ce test permettrait de trouver un seuil acceptable de bruit dans un signal pour qu'une détection soit faite.

De plus, tester un signal non périodique de plus de 200ms serait intéressant puisqu'il permettrait de trouver le coefficient que renvoi l'autocorrélation à une telle opération. Ce test complèterait le précédent en permettant de trouvé un seuil acceptable de détection.

Dans les tests d'intégration, il serait important de tester des signaux avec des durées inférieures au seuil de 200ms établie. Aussi, tester un son tel qu'un claquement de doigt pour vérifier la réponse du système serait un test crucial. Ces tests d'intégration pourraient avoir un impact sur les seuils choisis.