Pour tester AntennaPod, nous pourrions choisir la méthode du « Big Bang » ou la méthode incrémentale. Le « Big Bang » consiste à tester le système en entier une fois qu’il est complet. La méthode incrémentale consiste à réaliser des tests unitaires sur chaque composante. Ensuite, lors de l’intégration, il faudrait faire des tests d’intégration et, finalement, il faudrait faire des tests de sytème. Un avantage du « Big Bang » est que cette méthode est plus rapide à exécuter. En effet, il n’y a qu’une seule itération. La méthode incrémentale, quant à elle, contient plusieurs cycles. Elle a comme avantage de mieux trouver des bugs potentiels et de retrouver les problèmes plus facilement s’il y en a. Nous avons opté pour la méthode incrémentale, car nous avions du temps et nous consacrons beaucoup d’importance aux tests. Les tests unitaires ainsi que les tests d’intégration sont principalement dans le module « core ». Finalement les tests de système sont principalement dans le module « app ».

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Objectif | Algorithme(s) concerné(s) | Test ou diagnostic | Description | Logiciel | Objectif de couverture |
| Complétude | Les algorithmes concernés sont décrits avec les descriptions. | Test unitaire d’égalité | Les descriptions de tous les objectifs sont écrites en dessous | JUnit | 100% |
| Conformité | Test unitaire d’égalité | JUnit | 100% |
| Précision | Test unitaire d’égalité | JUnit | 100% |
| Disponibilité | Audit disponibilité | NA | 90% |
| Tolérance aux pannes | Test unitaire d’égalité et Audit tolérance aux pannes | JUnit | 99,5% |
| Récupérabilité | Vérification manuelle | NA | 80% |
| Modifiabilité | Audit modifiabilité | NA | 90% |
| Testabilité | Audit testabilité | NA | 95% |
| Réutilisabilité | Audit réutilisabilité | NA | 90% |

**Fonctionalité :**

Complétude : Nous pouvons voir dans les méthodes de la classe « MainActivityTest » que les entrées sont remises à celles par défaut lorsque l’utilisateur modifie un champ et appuie ensuite sur « back ». Nous pouvons rajouter un test unitaire très simple cette fonctionnalité. Nous voyons dans le fichier « renameItemDialog » que, si l’entrée de l’utilisateur est vide, nous devons la remettre à celle par défaut. Il faudrait tout simplement simuler une entrée vide et s’attendre à recevoir R.string.rename\_tag\_label.



Conformité : Nous pouvons voir que les méthodes de la classe URLCheckerTest s’assurent toutes que les url entrés sont justes. Ainsi, lorsque un URL est entré, on peut voir qu’il respecte les exigences.

Précision : À l’aide des tests « testComboFilter » et « testMinimalDurationFilter », nous pouvons nous assurer que les champs remplis par l’utilisateur sont acceptés. En effet, avec le premier test, il ne peut pas entrer un mot qui est refusé et il doit entrer au moins un mot accepté. Ensuite, avec le second, on confirme que la durée du filtre est plus grande que la durée minimale.

**Fiabilité :**

Disponibilité : Il y a un temps infini entre les pannes. En effet, AntennaPod n’utilise pas de serveur central. Toutes les informations qu’il utilise sont des données publiques sur internet. Par conséquent, même si les sites des créateurs de podcast ne fonctionnent plus, AntennaPod continuera à fonctionner.

Tolérance aux pannes : Nous pouvons voir le test « test404 » qui s’assure que, lors d’une panne de connexion, les fonctionalités du système qui ne nécessitent pas de connexion fonctionnent encore. Par exemple, les éléments téléchargés fonctionnent encore. De plus, sur le site d’AntennaPod, il est dit que l’application va chercher les informations des podcasts sur les sites sur lesquels elles ont publiées.

Récupérabilité : Une multitude de tests sont réalisés sur la base de données que ce soit pour envoyer ou recevoir des données. Ces tests sont situés dans « DbReaderTest » et « DbWriterTest ». Ainsi, si l’application crash, les données seront tout-de-même récupérables vu qu’elles se retrouvent sur la base de données.

**Maintenabilité :**

Modifiabilité : On peut voir que le système est facilement modifiable. Par exemple, le test « testUploadSubscription » et « testUploadSubscription2 » sont des tests très similaires. Nous n’avons ajouté qu’un élément de plus. Nous pouvons donc extrapoler que nous pouvons faire de même pour un nombre infini d’éléments.

Testabilité : Nous pouvons voir que tous les tests sont exécutés dans des fragments individuels. Cela confirme donc que toutes les composantes de l’applications peuvent être testées individuellement.

Réutilisabilité : Les tests « testShareDiaologDsiplayed » et « testShareDialogCancelButton » sont des tests qu’on réutilise un peu partout dans l’application. En effet ces tests s’assurent que les boites de dialogues fonctionnent bien. Les boites de dialogues sont, pour la majorité, des vues de l’application et on réutilise toujours la même composante.

Nous pouvons retrouver la majeure partie des tests d’AntennaPod dans les dossiers ci-dessous.

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated  Graphical user interface, text

Description automatically generated

Ici, on peut voir que tous les tests unitaires passent :

Graphical user interface, text, application, chat or text message

Description automatically generated

Les tests systèmes et d’intégration fonctionnent aussi, car les tests faits par l’émulateur fonctionnent en plus d’avoir testé à la main que le système fonctionnait.

Text

Description automatically generatedGraphical user interface

Description automatically generated with low confidence

Les tests unitaires passent localement comme on peut voir ici en lançant la commande « gradlew test » 

Cependant, pour le module « App », certains des tests de l’émulateur ne fonctionnent pas localement. On peut voir que c’est à cause de l’émulateur utilisé. En effet, on peut voir que, avec deux émulateurs différents, nous obtenons des résultats de tests largement différents. Nous pouvons cependant voir que les tests d’émulateur passent tous sur la pipeline.

App pixel 2 :Graphical user interface, text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated with medium confidenceGraphical user interface, application

Description automatically generated

App tablette custom :

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Core :

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Annexe :

Partie 2 (Testing) : J’ai eu des problèmes à faire fonctionner les tests de app localement. Alors, j’ai dû lancer les tests sur deux appareils différents pour que chaque test passe au moins une fois.