# Estratègia de testing i qualitat

Dins la metodologia àgil no podem aplicar un pla de testing molt extens a causa de les limitacions de temps de cada sprint, tot i això cal assegurar la qualitat de cada feina que és realitzada i formarà part de la nostra aplicació.

És per això que en aquest document es redacta un recull de la metodologia i les bones pràctiques que farem servir a l’hora de testejar la nostra aplicació, quan es pot donar una funcionalitat per finalitzada estarà en la metodologia de treball.

## Tests Petits (70%)

Són els tests unitaris que s'executen sense cap dependència. Aquests simulen qualsevol mena de component necessari per a la seva realització i han de tenir la capacitat d'executar-se ràpid entorn de desenvolupament.

Tota història d’usuari haurà de tenir un test unitari per a cadascun dels seus criteris d'acceptació, en la mesura que sigui possible. Això assegurarà que les expectatives de l’usuari vinculades a aquesta funcionalitat es compleixen correctament.

A més, per assegurar que la comunicació entre el nostre codi i els serveis externs funciona de manera correcta, en el moment en què s’introdueix una nova API haurem d’introduir tests de serveis externs dins el nostre codi.

### Eines

#### Mockito

Es farà servir per a tots els tests unitaris que requereixen stubs, ja que permet assegurar-te que la teva aplicació completi una determinada acció, sense realitzar realment aquesta acció on no tindria sentit.

#### Robolectric

Es farà servir per els tests unitaris que requereixin interactuar especialment amb el framework d’Android, ja que permet falsificar el codi del framework natiu en el nostre JVM local.

#### Emuladors o dispositius físics

Es faran servir per testejar la compatibilitat del codi amb el maquinari del dispositiu. Especialment ens serà útil per a identificar els problemes que pugui donar el maquinari del dispositiu en funcionalitats que el requereixin com per exemple la geolocalització.

## Tests Mitjans (20%)

Són els tests d'integració, que s'ocupen d'integrar els diferents components de proves unitàries i construir una estructura de programa dictada pel disseny amb l'objectiu de provar si funciona el programa com a conjunt.

### Eines

#### Emuladors en el Núvol

Podem fer servir eines com el Firebase Test Lab, que permet testejar el mateix programari en diferents combinacions de grandària de pantalles i configuracions de maquinari.

#### Dispositius físics

Seran útils per a testejar la coordinació i el temps d'execució de l'aplicació en provar diferents tests unitaris.

## Tests Grans (10%)

Són tests tant d'integració com de la interfície d'usuari, que s'executen per a assegurar el correcte funcionament tant del look & feel de l'aplicació com el workflow de la interfície de l'usuari.

Així mateix es prova el programari de manera que funcioni bé per a diferents sistemes operatius, conté: proves de seguretat, proves de recuperació, proves d’estrès on es donen condicions desfavorables al sistema per comprovar com respon, i proves de rendiment per provar el rendiment en temps d’execució del sistema, la velocitat, eficacia i fluïdesa.

### Eina

#### Espresso

Es farà servir per tal de fer proves a la interfície d'usuari per a simular interaccions d’usuari. D’aquesta manera evitarem que l’usuari es trobi amb resultats inesperats o bé tingui una mala experiència al moment del seu ús.

## Detecció de Bugs

Per assegurar la màxima qualitat del nostre software farem servir l’eina SonarQube per tal de registrar tots els bugs que apareixin durant el desenvolupament del projecte, descubrir vulnerabilitats i identificar altres possibles errors.

## Bibliografía:

[https://developer.android.com/training/testing/](https://developer.android.com/training/testing/#UIAutomator)

<https://www.testingexcellence.com/agile-test-strategy-example-template/>

<https://raygun.com/blog/android-testing-with-junit-mockito-and-robolectric/>

# Estrategia de testing

Referència: <https://www.testingexcellence.com/agile-test-strategy-example-template/>

Dins la metodologia àgil no podem aplicar un pla de testing molt extens degut a les limitacions de temps de cada sprint, tot i així cal assegurar la qualitat de cada feina que es realitzada i formarà part de la nostra aplicació. És per això que en aquest document es redacta un recull de les bones pràctiques a l’hora de testejar la nostra aplicació i quan es pot donar una funcionalitat per finalitzada.

## Histories d’usuari

Aquest és el primer nivell de la fase de testing. Una correcta definició de la historia d’usuari i els seus criteris d’acceptació ens permetrà tenir una idea clara de quins són els objectius que vol assolir el client a través d’aquesta història i ens evitarà possibles errors derivats de la falta de precisió de la historia.

Les històries d’usuari s’han de realitzar en conseqüència amb el concepte d’”INVEST”. Un bona historia d’usuari ha de ser Independent de totes les altres, Negociable, Valuosa, Estimable, petita per tal de poder ser compresa en una iteració (Small), i Testejable.

La podriem no considerar un tècnica de testing en totes les seves facetes però ens facilitarà la feina a l’hora de generar els tests en un futur, és per això que la considerem el primer esglaó dins la nostra estrategia.

## Tests unitaris

Aquesta fase del testing es centra en la menor unitat de disseny, es proven unitats individualment o en grup d'unitats relacionades entre si. S’acostuma a fer amb exemples d'entrada i observant els resultats corresponents.

La seva finalitats principal es asegurar que el codi es desenvolupa correctament. La feina recau ens els programadors i s’ha d’aplicar sobre cada nova peça de codi o sobre codi refactoritzat tant bon punt aquest s’escriu.

EXPLICAR L’EINA QUE UTILITZAREM

## Test d’acceptació

Aquest tipus de proves es realitzen sobre funcionalitats que ja estan apunt per funcionar i que han passar els tests unitaris. La finalitat principal dels test d’acceptació es assegurar que les expectatives de l’usuari vinculades a aquesta funcionalitat es compleixen correctament.

## Test de serveis externs

Aquest tipus de casos de testeig serveixen per assegurar que la comunicació entre el nostre codi i els serveis externs funciona de manera correcta. També està a càrrec dels programadors i s’aplica en el moment en que s’introdueix una nova API s’introdueix dins el nostre codi.

## Test d’integració

L'objectiu és prendre components de proves unitàries i construir una estructura de programa dictada pel disseny. La prova d'integració és una prova en què es combinen un grup de components per produir resultats.

## Test de regressió

Cada vegada que s'afegeix un mòdul nou es produeixen canvis en el programa. Aquest tipus de test prova que tot el component funcioni correctament fins i tot després d'afegir components al programa complet.

## Test del sistema

Es prova el programari de manera que funciona bé per a diferents sistemes operatius. Està cobert per la tècnica de prova de caixa negra, només ens centrem en l'entrada i sortida sense centrar-nos en el treball intern.

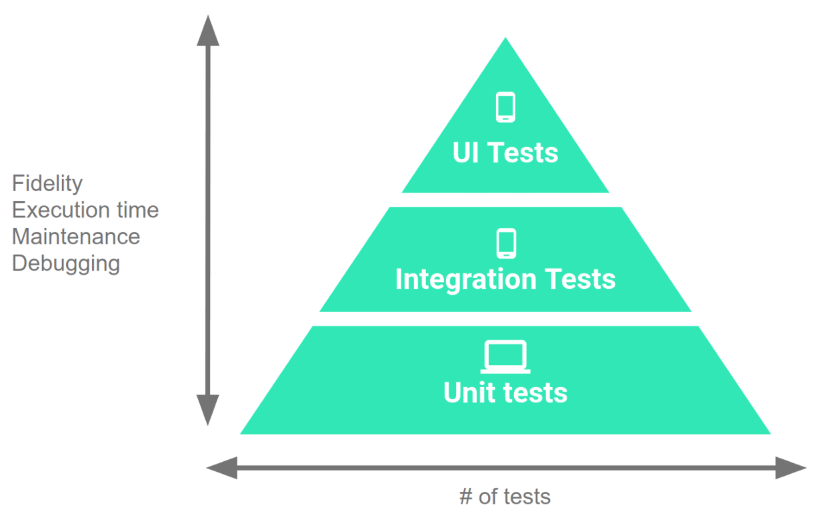
Conté:

* Proves de seguretat
* Proves de recuperació
* Proves d'estrès: donem condicions desfavorables al sistema i comprovem com respon.
* Proves de rendiment: utilitzat per provar el rendiment en temps d'execució del sistema. Prova la velocitat i l'eficàcia del programa.

Documentación de Android para Testing

Referencia: <https://developer.android.com/training/testing/#UIAutomator>

# Estrategia de Testing



## Tests Pequeños (70%)

Son los tests unitarios que se ejecutan sin ninguna dependencia. Estos simulan cualquier tipo de componente necesario para su realización y deben tener la capacidad de ejecutarse rápido en el entorno de desarrollo.

### Herramientas

#### Robolectric

#### Mockito

#### En emuladores o dispositivos físicos

Para testear la compatibilidad del código con el hardware del dispositivo.

## Tests Medianos (20%)

Son los tests de integración, debido a que se ocupan de integrar los diferentes componentes con tal de probar si funciona el programa como conjunto. Pueden ejecutarse en emuladores o en dispositivos reales.

### Herramientas

#### Emuladores en la Nube

Un ejemplo sería el Firebase Test Lab, que permite testear el mismo software en distintas combinaciones de tamaño de pantallas y configuraciones de hardware.

#### Dispositivos físicos

Para testear la coordinación de la aplicación al probar diferentes tests unitarios.

## Tests Grandes (10%)

Son tests tanto de integración como de la interfaz de usuario, que se ejecutan para asegurar el correcto funcionamiento tanto del look & feel de la aplicación como el workflow de la interfaz del usuario.

### Herramientas

#### Espresso

asd

#### UI Automator

asd

Diferencias entre JUnit, Mockito y Roboelectric: <https://raygun.com/blog/android-testing-with-junit-mockito-and-robolectric/>