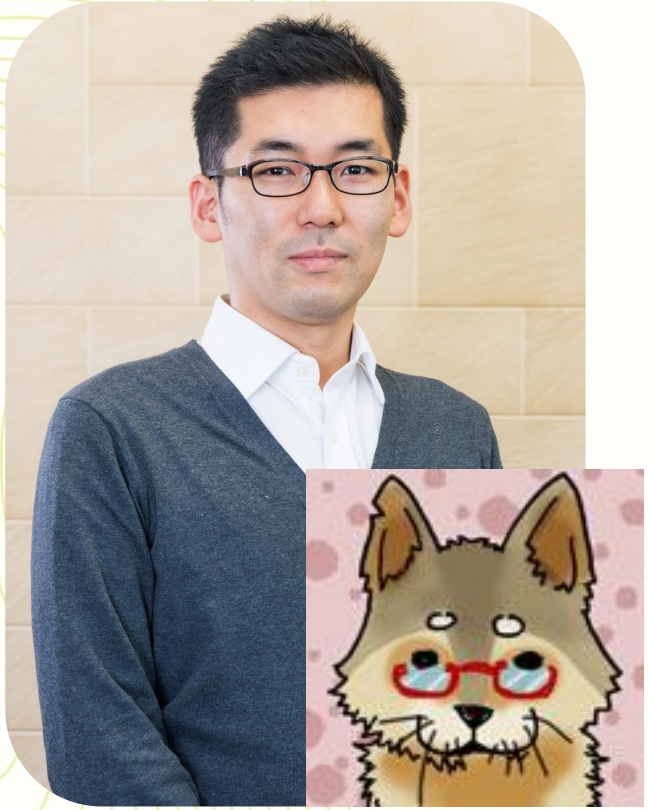
「良いユニットテスト」を書こう

asken 高津 基暢

2024/12/20 Ebisu.mobile #8



自己紹介

株式会社asken
Android Engineer / QA Engineer

Android Engineerとしてasken入社
Android開発は2012年から

元々テストが好きだったこともあって、
2024年からQA Engineer
JSTQB FL保有

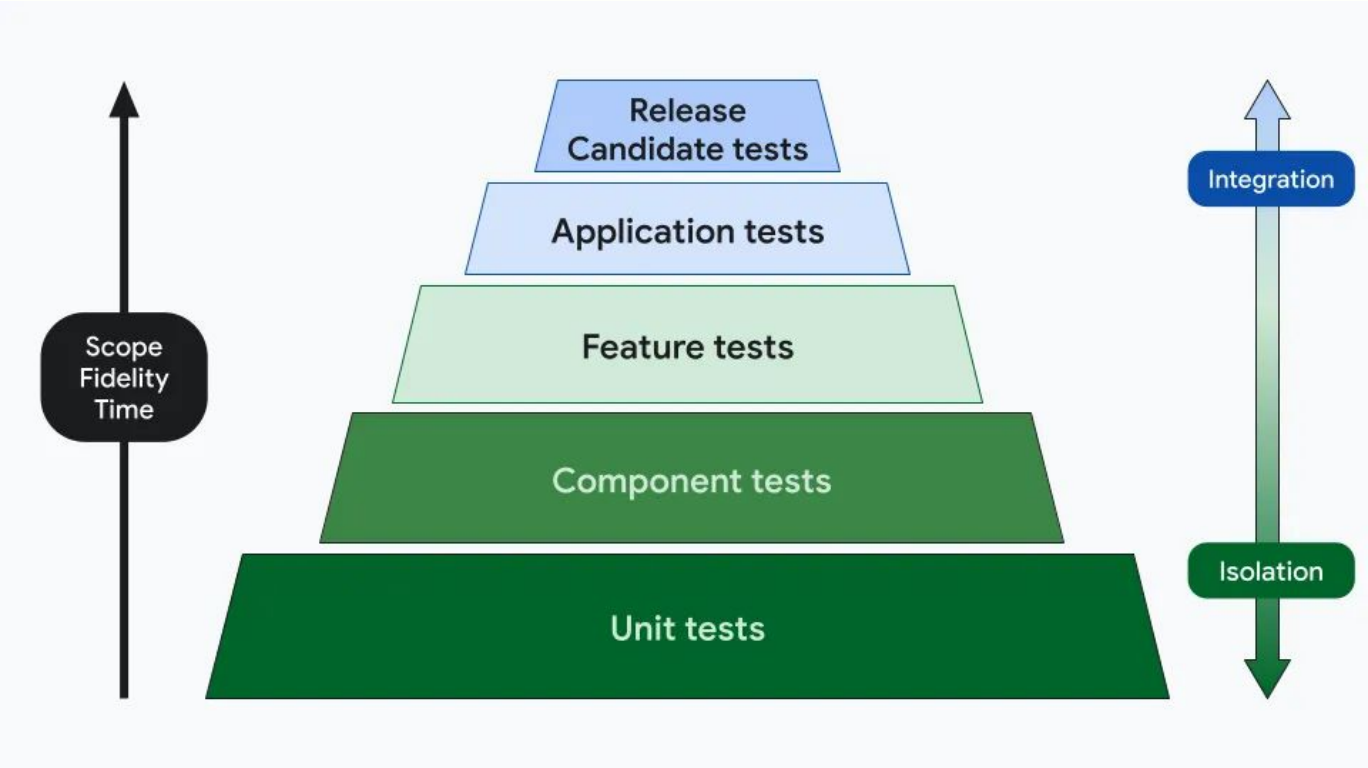
趣味はボードゲームと漫画とラジオ

Agenda

1. なぜユニットテストを書く必要があるのか
2. 何を対象にユニットテストを書いているのか
3. 「良いユニットテスト」とは
4. 実際にどんな効果があったか

1. なぜユニットテストを書く必要があるのか

Googleのテスト戦略



引用: https://developer.android.com/training/testing/fundamentals/strategies

ユニットテストに期待される効果

- バグの早期発見
- リグレッションエラー(デグレ)の検出
- 設計改善

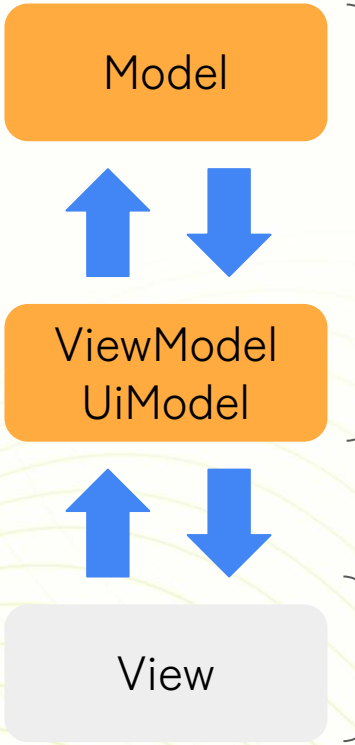
効果を最大化するために

- テストを書く範囲を決めて、カバレッジを上げる

- 質が良いテストコードを書く

2. あすけんAndroidでは何を対象にユニットテストを書いているのか

ユニットテストを書いているところ



(前提)MVVM アーキテクチャを採用

Model, ViewModel(UiModel含む):
ユニットテストを書いている

View:
ユニットテストを書いていない
コンポーネントテストを検討中

3.「良いユニットテスト」とは

信頼性と可読性

テストの結果が信頼できる

信頼できない ＝ 実行するたびに結果が変わる(＝不安定)

テストの結果が不安定になる原因の例

- システム時刻を参照している
- データベースやSharedPreferencesを参照している

テストの結果が信頼できる

偽陽性、偽陰性のどちらも発生しない

	プロダクトコード: 正しい	プロダクトコード: 誤り
テスト結果: 成功	期待どおり	偽陰性 (検知漏れ)
テスト結果: 失敗	偽陽性 (誤検知)	期待どおり

テストの結果が信頼できないと、
障害修正のコストが跳ね上がる!

可読性が高い

可読性を高めるために、以下の点を工夫する

- 1つのテストケースで1つの観点をテストする
- テストケース名でテストの意図がわかるようにする
- AAAパターンまたはGherkin記法で整理する
- ループや条件分岐をなくし、上から下へ素直に読み下せるようにする
- 文字列や数値をベタ書きする

読みにくいテストコード例

asken

```
@Test
fun validate_birthday() {
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    // 15歳の場合 (誕生日当日) は OK
    assertTrue(Birthday.validate(birthday, today))
    // 14歳の場合 (誕生日前日) は NG
    …
    assertFalse(...
}
private fun getDateBefore15Years(today: LocalDateTime): LocalDateTime {
```

NG: 1つのテストケースで複数テストしている

asken

```
@Test
fun validate_birthday() {
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    // 15歳の場合 (誕生日当日) は OK
    assertTrue(Birthday.validate(birthday, today))
    // 14歳の場合 (誕生日前日) は NG
    …
    assertFalse(...
}
private fun getDateBefore15Years(today: LocalDateTime): LocalDateTime {
```

複数の確認項目が1つのテストケースにある

OK: 1つのテストケースで1つの観点をテストする

asken

```
@Test
fun validate_birthday_正常系() {
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    // 15歳の場合 (誕生日当日) は OK
    assertTrue(Birthday.validate(birthday, today))
}

@Test
fun validate_birthday_異常系() {
    ...
```

確認項目ごとにテストケースを分割

NG: 何を確認したいのかわからないテストケース名

asken

```
@Test
fun validate_birthday_正常系() {
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    // 15歳の場合 (誕生日当日) は OK
    assertTrue(Birthday.validate(birthday, today))
}

private fun getDateBeforeFifteenYears(today: LocalDateTime): LocalDateTime {
    ...
```

正常って何? 何がどうなっていれば正しい?

テスト対象の関数名が
テストケース名に含まれている

OK: テストケース名でテストの意図がわかる

asken

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    assertTrue(Birthday.validate(birthday, today))
}

private fun getDateBefore15Years(today: LocalDateTime): LocalDateTime {
    ...
```

NG: テストケースの処理フローの説明がない

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    assertTrue(Birthday.validate(birthday, today))
}

private fun getDateBefore15Years(today: LocalDateTime): LocalDateTime {
    ...
```

AAAパターンのコメントを追加して
テストの流れを整理する

OK: AAAパターンで処理フローを整理する

asken

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    // Arrange
    // 15歳の誕生日を用意する
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    // Act
    val actual = Birthday.validate(birthday, today)
    // Assert
    assertTrue(actual)
}
private fun getDateBefore15Years(today: LocalDateTime): LocalDateTime {
```

Arrange / Act / Assert
に分割して整理する

NG: テストデータ作成を関数化している

asken

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    // Arrange
    // 15歳の誕生日を用意する
    val today = LocalDateTime.now()
    val birthday =  getDateBefore15Years(today)
    // Act
    val actual = Birthday.validate(birthday, today)
    // Assert
    assertTrue(actual)
}
private fun getDateBefore15Years(today: LocalDateTime): LocalDateTime {
```

他のテストケースでも同じ処理を使っているので関数化している

OK: 上から下へ素直に読み下せるようにする

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    // Arrange
    // 15歳の誕生日を用意する
    val today = LocalDateTime.now()
    val birthday =  today.minusYears(15)
    // Act
    val actual = Birthday.validate(birthday, today)
    // Assert
    assertTrue(actual)
}
```

他のテストケースと同じ処理であっても
関数の切り出しをしない

NG: テストデータを計算している

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    // Arrange
    // 15歳の誕生日を用意する
    val today = LocalDateTime.now()
    val birthday =  today.minusYears(15)
    // Act
    val actual = Birthday.validate(birthday, today)
    // Assert
    assertTrue(actual)
}
```

システム時刻から今日が15歳の誕生日となる日付を計算している

OK: テストデータに必要な値をベタ書きする

```
@Test
fun 15歳の誕生日当日は登録可能とする() {
    // Arrange
    // 今日と15歳の誕生日を用意する
    val today = SimpleDateFormat("yyyyMMdd").parse("20241201")
    val birthday = SimpleDateFormat("yyyyMMdd").parse("20091201")
    // Act
    val actual = Birthday.validate(birthday, today)
    // Assert
    assertTrue(actual)
}
```

“今日”の日付と、その日が15歳の誕生日となる日をベタ書きする

4. 実際にどんな効果があったか

askenエンジニアが実感している効果

期待される効果を実際に体験できている

- バグの早期発見
- リグレッションエラー(デグレ)の検出
- 設計改善

予想外の効果

iOSエンジニアにテストコードの
レビューをしてもらえるようになった

これまでの対応によりKotlinに不慣れでも
テストコードの内容は理解できる

レビューしてもらった結果...

iOS - Android間の仕様差異を事前に検出できた!!

We are hiring!

askenで一緒に活躍してくれる方を募集しています!

asken

Thank you!