T.C.
KIRKLARELİ ÜNİVERSİTESİ
TEKNİK BİLİMLER MESLEK YÜKSEKOKULU
BİLGİSAYAR TEKNOLOJİLERİ BÖLÜMÜ
BİLGİSAYAR PROGRAMCILIĞI PROGRAMI

ÖLÇEKLENEBİLİR PROJE MİMARİLERİ

EYLÜL KAY

1247008007

MOBİL PROGRAMLAMA

NADİR SUBAŞI

KIRKLARELİ

12.2025

İÇİNDEKİLER

5. Clean Architecture ve TDD’nin Entegrasyonu

5.1 Domain Katmanında TDD Uygulaması

5.2 Data Katmanında Mocklama Stratejileri

5.3 Presentation Katmanında Test Uygulamaları

5.4 Test Edilebilirlik Açısından Clean Architecture Avantajları

6. Örnek Proje Uygulaması

6.1 Proje Tanımı

6.2 Katman Yapısının Oluşturulması

6.3 Domain Katmanı Kod Örneği

6.4 Data Katmanı Kod Örneği

6.5 Presentation Katmanı Kod Örneği

6.6 TDD Sürecinin Adım Adım Uygulanması

6.6.1 Use Case Testi

6.6.2 Repository Testi

6.6.3 UI Testleri

7. Karşılaşılan Yaygın Sorunlar ve Çözüm Yöntemleri

7.1 Bağımlılık Yönetimi Problemleri

7.2 Test Yazımında Sık Yapılan Hatalar

7.3 Performans ve Kod Okunabilirliği Konuları

8. Sonuç ve Değerlendirme

8.1 Mimari ve TDD’nin Sağladığı Avantajlar

8.2 Projenin Geliştirilebilir Alanları

8.3 Nihai Değerlendirme

9. Kaynakça

KISALTMALAR

TDD : Test Driven Development – Test Odaklı Geliştirme

CI/CD : Continuous Integration / Continuous Deployment – Sürekli Entegrasyon / Sürekli Dağıtım

DI : Dependency Injection – Bağımlılık Enjeksiyonu

UI : User Interface – Kullanıcı Arayüzü

UX : User Experience – Kullanıcı Deneyimi

API : Application Programming Interface – Uygulama Programlama Arayüzü

DTO : Data Transfer Object – Veri Transfer Nesnesi

JSON : JavaScript Object Notation – Veri Gösterim Formatı

HTTP : Hypertext Transfer Protocol – Hiper Metin Aktarım Protokolü

REST : Representational State Transfer – Kaynak Temsili Aktarımı

CRUD : Create, Read, Update, Delete – Oluştur, Oku, Güncelle, Sil

SDK : Software Development Kit – Yazılım Geliştirme Kiti

IDE : Integrated Development Environment – Tümleşik Geliştirme Ortamı

OOP : Object Oriented Programming – Nesne Yönelimli Programlama

SOLID : Nesne yönelimli tasarım prensipleri bütünü

CI : Continuous Integration – Sürekli Entegrasyon

CD : Continuous Delivery / Continuous Deployment – Sürekli Teslim / Sürekli Dağıtım

VM : ViewModel – Görünüm Modeli

BDD : Behaviour Driven Development – Davranış Odaklı Geliştirme

AAA : Arrange–Act–Assert (Test yazım düzeni)

UTC : Coordinated Universal Time – Koordine Evrensel Zaman

CPU : Central Processing Unit – Merkezi İşlem Birimi

RAM : Random Access Memory – Rastgele Erişimli Bell

1. Giriş

1.1 Çalışmanın Amacı

Bu Raporun Amacı, Flutter Projelerinde Clean Architecture (Temiz Mimari) Prensiplerinin Uygulanmasını Ve Test-Driven Development (Tdd) Yaklaşımının Bu Mimari İle Nasıl Entegre Edileceğini Açıklamak, Geliştiricilere Sistematik Ve Sürdürülebilir Bir Proje Geliştirme Yöntemi Sunmaktır.

1.2 Kapsam Ve Sınırlılıklar

Rapor; Flutter Uygulamaları Özelinde Katmanlı Mimari, Bağımlılık Yönetimi, Test Türleri Ve Tdd Süreçlerini Kapsamaktadır. Flutter'ın Tüm Ekosistemi Ele Alınmayıp Yalnızca Mimari Yapı Ve Test Odaklı İçerik İncelenmiştir.

1.3 Temel Kavramlar

- Clean Architecture: Katmanlar Arasında Bağımlılık Kurallarının Kesin Olarak Ayrıldığı Mimari Model.
- Tdd: Önce Testlerin Yazıldığı, Ardından Kodun Geliştirildiği Döngüsel Yazılım Geliştirme Yöntemi.
- Katman: Projeyi Sorumluluklarına Göre Ayıran Yapısal Bölümler.

2. Flutter’da Mimari Yapıların Evrimi

2.1 Geleneksel Flutter Mimari Yaklaşımları

- MVC (Model–View–Controller)
- MVVM (Model–View–ViewModel)
- Provider tabanlı basit mimariler
  Bu yaklaşımlar küçük projelerde hızlı artılar sağlasa da karmaşık yapılarda sürdürülebilirliği azaltır.

2.2 Modern Yaklaşımlar

- BLoC mimarisi
- Redux
- Riverpod
- Clean Architecture + State Management kombinasyonları

2.3 Clean Architecture’ın Flutter Ekosistemindeki Yeri

Modüler, test edilebilir ve genişletilebilir yapılar kurmak isteyen geliştiriciler için Clean Architecture Flutter’da güçlü bir temel oluşturur.

3. Clean Architecture (Temiz Mimari)

3.1 Temel Prensipler

- Bağımlılıklar merkezden dışa doğru değil, dıştan içe doğru yönelir.
- Her katman kendi sorumluluklarını taşır.
- İş kuralları UI’dan bağımsızdır.

3.2 Katmanların Genel Yapısı

1. Domain (Merkez Katman)
2. Data (Orta Katman)
3. Presentation (Dış Katman)

3.3 Domain Katmanı

UI veya veri kaynağı tarafından etkilenmeyen saf iş kurallarıdır.

3.3.1 Entities (Varlıklar)

Temel iş kurallarını temsil eden değişmez yapılardır.

Örnek:

```
class User {
  final int id;
  final String name;
  User({required this.id, required this.name});}
```

3.3.2 Use Cases (Kullanım Senaryoları)

Bir iş kuralının tek sorumluluklu şekilde uygulanmış hâlidir.

Örnek:

```
class GetUser {
  final UserRepository repository;
  GetUser(this.repository);
  Future<User> call(int id) {
  return repository.getUser(id);}}
```

3.4 Data Katmanı

Dış kaynaklardan veri alır (API, veritabanı).

3.4.1 Modeller

API'den gelen veriyi domain entity'lerine dönüştürür.

Örnek:

```
class UserModel extends User {
  UserModel({required super.id, required super.name});
```

```
 factory UserModel.fromJson(Map<String, dynamic> json) {
    return UserModel(id: json["id"], name: json["name"]);}}
```

3.4.2 Repository Implementations

Domain'deki soyut repository'yi uygular.

```
class UserRepositoryImpl implements UserRepository {
  final RemoteDataSource remote;
  UserRepositoryImpl(this.remote);
  @override
  Future<User> getUser(int id) async {
    final data = await remote.fetchUser(id);
    return UserModel.fromJson(data);}}
```

3.5 Presentation Katmanı

3.5.1 State Management Yaklaşımları

- Provider
- Bloc
- Riverpod
- MobX

3.5.2 View ve ViewModel Yapılanması

Örnek Bloc Kullanımı:

```
class UserCubit extends Cubit<UserState> {
  final GetUser getUser;
UserCubit(this.getUser) : super(UserInitial());
  void load(int id) async {
```

```
emit(UserLoading());
final result = await getUser(id);
emit(UserLoaded(result));}}
```

3.6 Katmanlar Arası Bağımlılık Kuralları

- Presentation → Domain
- Data → Domain
- Domain hiçbir katmana bağımlı değildir.

4. Test-Driven Development (TDD)

4.1 TDD Nedir?

Testlerin koddan önce yazıldığı geliştirme yaklaşımıdır.

4.2 Red–Green–Refactor Döngüsü

1. Red: Test yazılır ve başarısız olur.
2. Green: Testi geçecek en basit kod yazılır.
3. Refactor: Kod iyileştirilir, testler geçmeye devam eder.

4.3 Flutter Test Türleri

4.3.1 Unit Test

Tek bir fonksiyonun test edilmesi.

4.3.2 Widget Test

UI bileşenlerinin davranış testi.

4.3.3 Integration Test

Uygulamanın uçtan uca test edilmesi.

4.4 Clean Architecture ile TDD İlişkisi

Use Case tabanlı yapı, TDD'nin uygulanmasını kolaylaştırır. Her katman bağımsız test edilebilir.

5. Clean Architecture ve TDD'nin Entegrasyonu

5.1 Domain Katmanında TDD Uygulaması

Use case için önce test yazılır.

Örnek:

```
test("should return User when id is valid", () async {
  final repository = MockUserRepository();
  final useCase = GetUser(repository);
  when(() => repository.getUser(1))
      .thenAnswer((_) async => User(id: 1, name: "Eylül"));
 final result = await useCase(1);
 expect(result.name, "Eylül");});
```

5.2 Data Katmanında Mocklama

- Repository için mock
- RemoteDataSource için stub veri

5.3 Presentation Katmanında Test

Widget ve Cubit/Bloc testleri yazılır.

5.4 Avantajlar

- Kolay genişletilebilirlik
- Minimum hata
- Yüksek modülerlik

6. Örnek Proje Uygulaması

6.1 Proje Tanımı

Basit bir kullanıcı sorgulama uygulaması: Kullanıcının ID'si girildiğinde API üzerinden kullanıcı bilgisi alınır.

6.2 Katman Yapısı

lib/

└── src/

    ├── domain/

    ├── data/

    └── presentation/

6.3 Domain Katmanı Örneği

- User Entity
- UserRepository (abstract)
- GetUser Use Case

6.4 Data Katmanı Örneği

- UserModel
- UserRepositoryImpl
- RemoteDataSource

6.5 Presentation Katmanı Örneği

Üç durum: Loading, Loaded, Error.

6.6 TDD Süreci

6.6.1 Use Case Testi

Daha önce gösterilen örnek uygulanır.

6.6.2 Repository Testi

Mock RemoteDataSource ile test yapılır.

6.6.3 UI Testleri

testWidgets("shows loading then data", (tester) async { ... });

7. Yaygın Sorunlar ve Çözüm Yöntemleri

7.1 Bağımlılık Yönetimi Problemleri

Çözüm: Dependency Injection (get_it, riverpod).

7.2 Test Yazımında Sık Hatalar

- Mocklama eksikliği
- Katman sınırlarının ihlali

7.3 Performans ve Kod Okunabilirliği

- Gereksiz rebuild'ler önlenmelidir.
- Kod yorum ve dokümantasyonu artırılmalıdır.

8. Sonuç ve Değerlendirme

8.1 Sağlanan Avantajlar

- Test edilebilirlik
- Bağımsız katmanlar
- Uzun vadeli sürdürülebilirlik

8.2 Geliştirilebilir Alanlar

- Katmanlar daha da modüler hâle getirilebilir.
- Otomatik test entegrasyonları artırılabilir.

8.3 Nihai Değerlendirme

Clean Architecture ve TDD birlikte kullanıldığında Flutter projelerinde yüksek güvenilirlik ve güçlü bir yapı elde edilir.

9. Kaynakça

- Robert C. Martin – Clean Architecture
- Flutter Resmî Dokümantasyonu
- Dart.dev Testing Guidelines
- Mockito ve Bloc Test Paketleri Dokümantasyonu
- ChatGPT