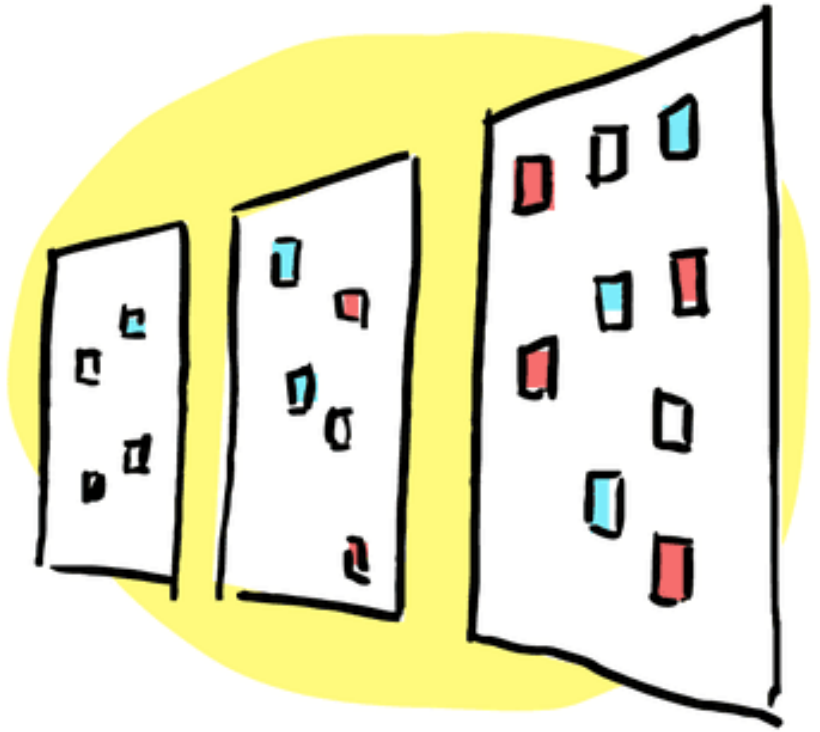
Yazılım Geliştirmede Çevik Yöntemler

6. Hafta



25.03.2025

Mühendislik Fakültesi

Bilgisayar Mühendisliği

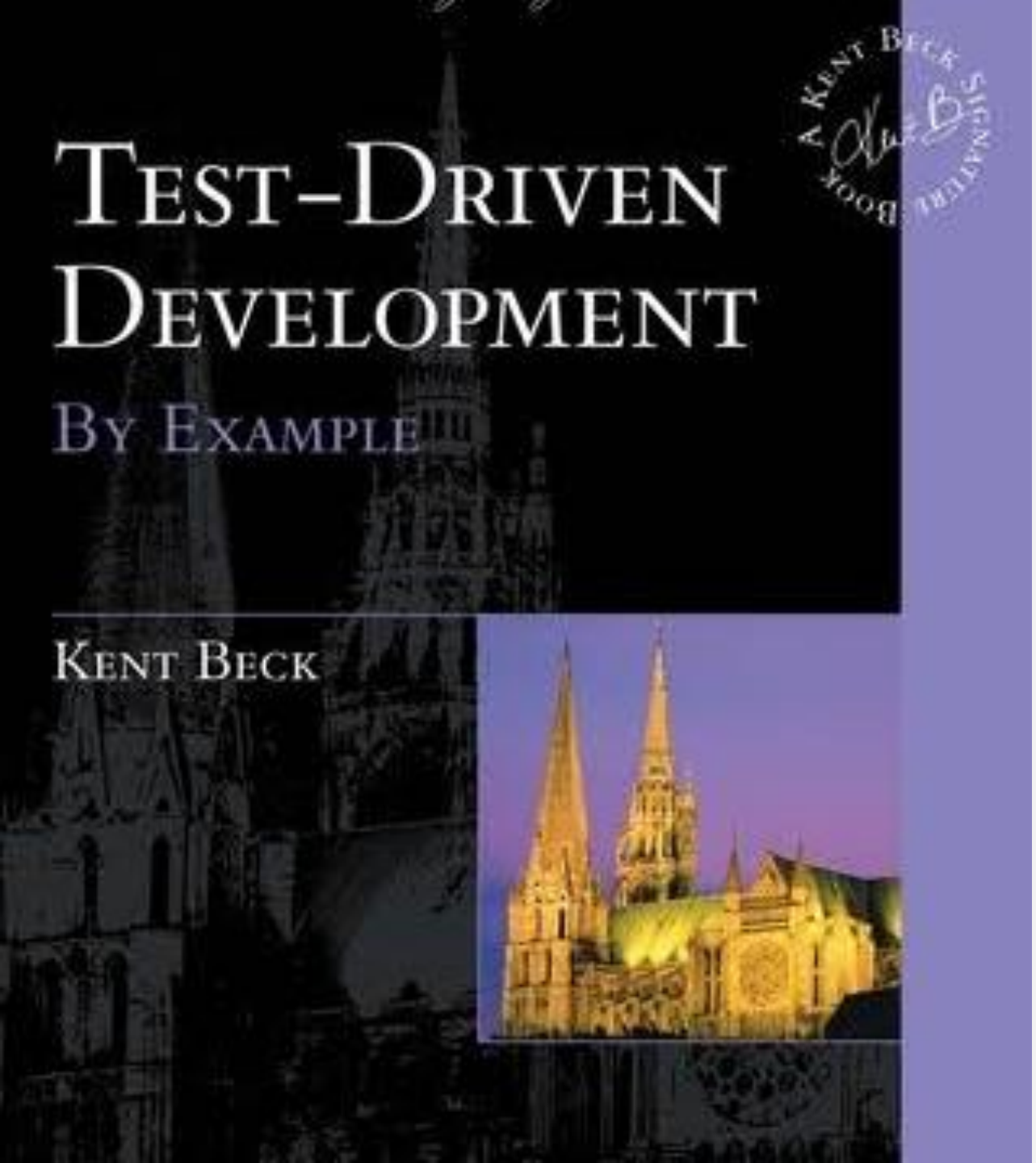
Hazırlayan: Dr. Ercan Ezin

GİRİŞ

Test Driven Development (TDD)

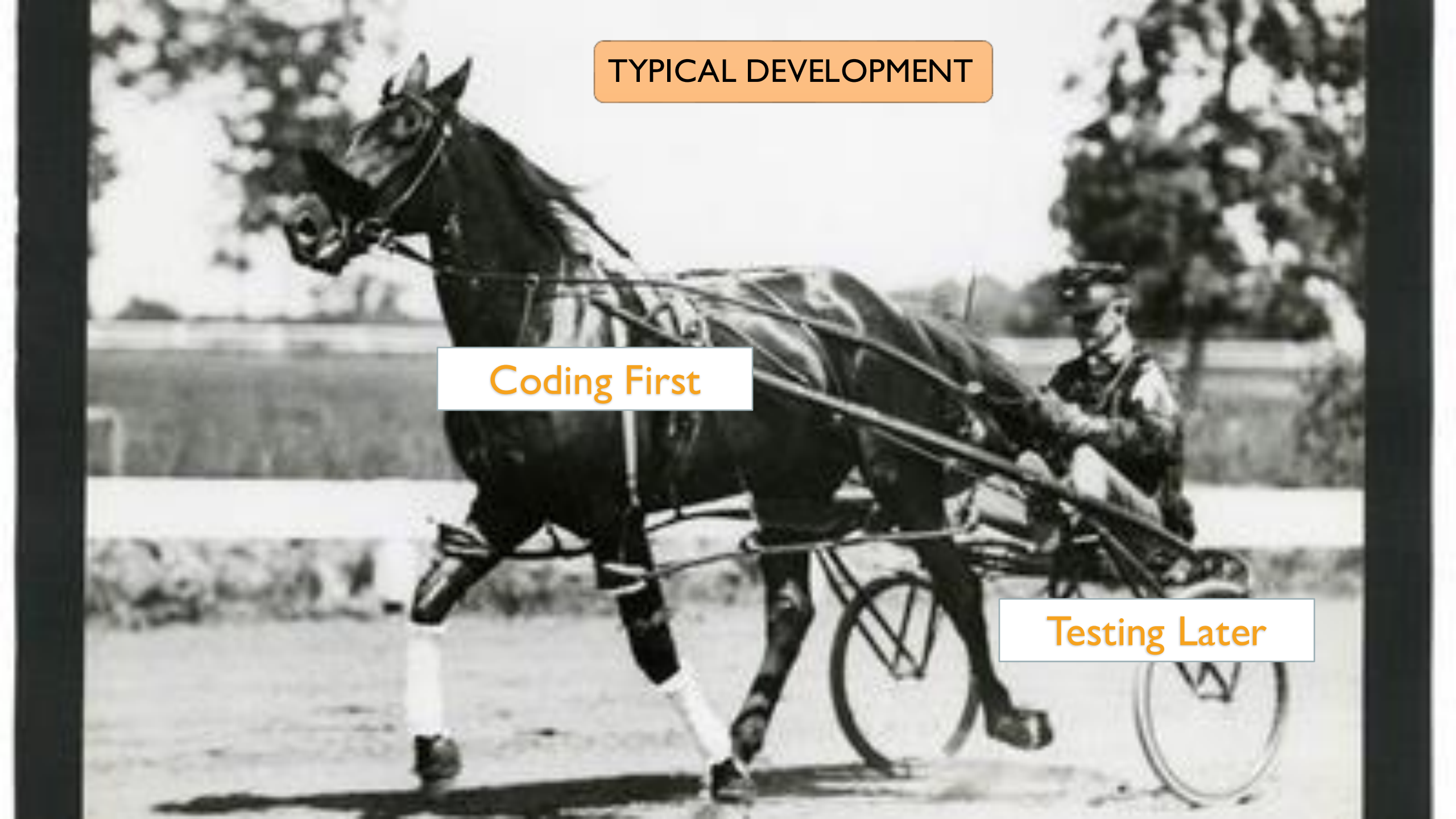
TEST ODAKLI GELİŞTIRME: AGILE PROJELERDE KALITE OLUŞTURMA

- TDD, testlerin test edilecek kodlardan *önce* yazıldığı bir yazılım geliştirme yaklaşımıdır.
- Agile metodolojilerde temel bir uygulama olup, yinelemeli geliştirme ve sürekli iyileştirmeyi vurgular.
- TDD kavramını popülerleştiren kişi, Agile hareketinin önemli bir figürü olan Kent Beck'tir. Kod tasarımını iyileştirme ve hata azaltmada rolünü vurgulamıştır.
- Bunu bir ev inşa etmek gibi düşünün – temeli (kod) atmadan önce planları (testler) tasarlarsınız.



KENT BECK TDD BOOK

- Kent Beck 1994 yılında ilk Test kütüphanesini yazdı.
- 1998 yılında XP'de "testleri önce yazmalıyız" dedi.
- 2002 yılında yanda kapağı gösterilen kitabı yazdı.


TYPICAL DEVELOPMENT
Coding First
Testing Later

TDD

MODERN UYGULAMALARDA TEST YAZMA

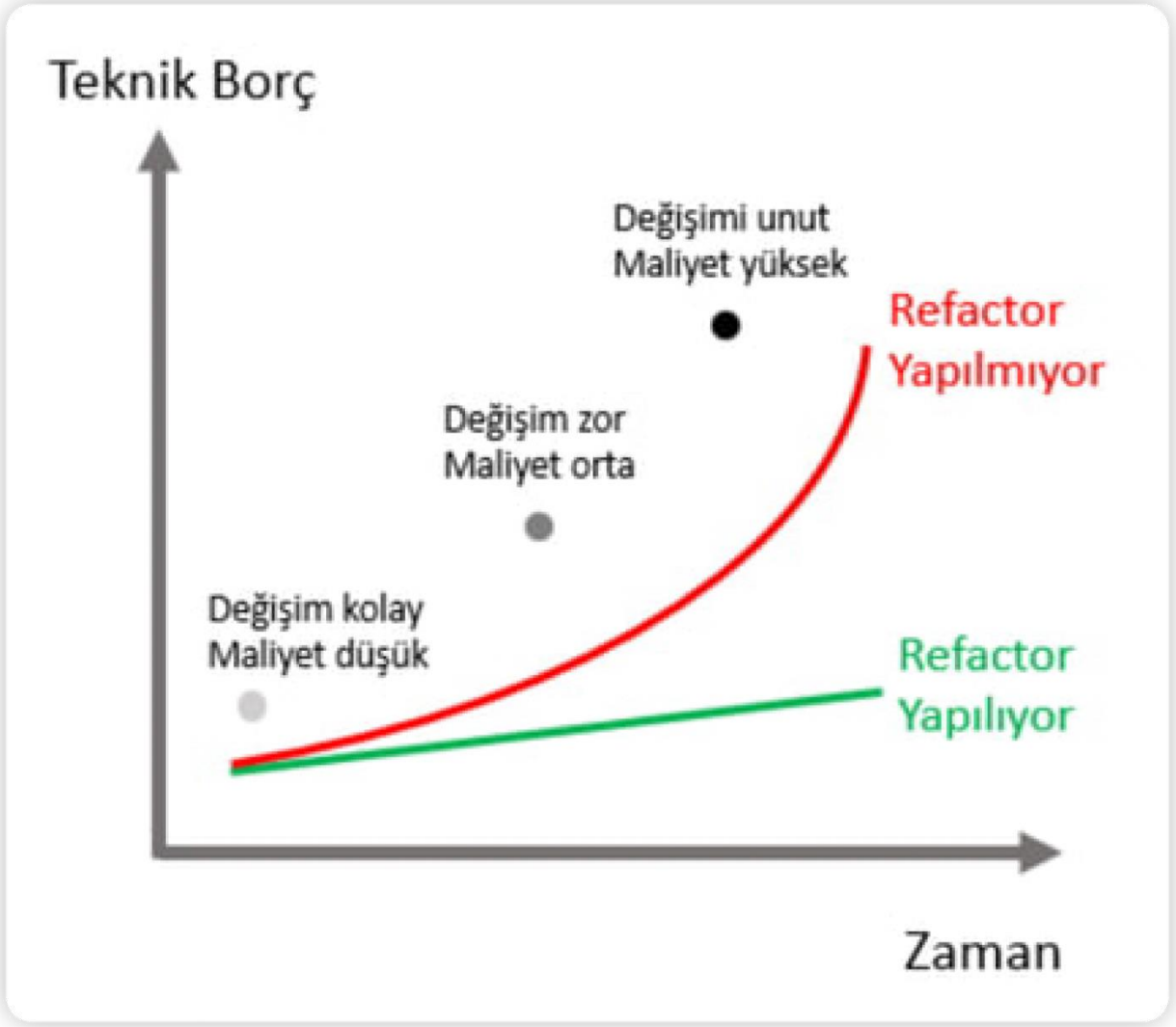
- Modern uygulamalar karmaşıktır, çeşitli katmanları (ön uç, arka uç, veritabanları) içerir. Testlerin tüm bu yönleri kapsaması gerekir.
- Test sürecini otomatikleştirmek için çeşitli test çerçeveleri (örneğin, Jest, Mocha, pytest, Selenium) kullanırız.
- Testler genellikle sürekli geri bildirim ve otomatik dağıtım için CI/CD pipeline hattına entegre edilir.
- Farklı türde testler, farklı yönleri kapsamak için mevcuttur (birim, entegrasyon, uçtan uca).

TDD – ARTILARI VE EKSILERI

- \+ Kod tasarımı ve okunabilirliği iyileştirir.
- \+ Hataları azaltır ve kaliteyi artırır.
- \+ Refaktoring ve bakımı kolaylaştırır.
- \+ Gereksinimleri ve spesifikasyonları netleştirir.
- \+ Kod değişikliklerinde güveni artırır.

- − Başlangıçta geliştirme süresini artırır.
- − Disiplin ve önceden planlama gerektirir.
- − Karmaşık sistemler için başlangıçta zor olabilir.
- − Başlangıçta dik bir öğrenme eğrisi vardır.
- − Aşırı test yapma potansiyeli.

TECHNICAL DEBT

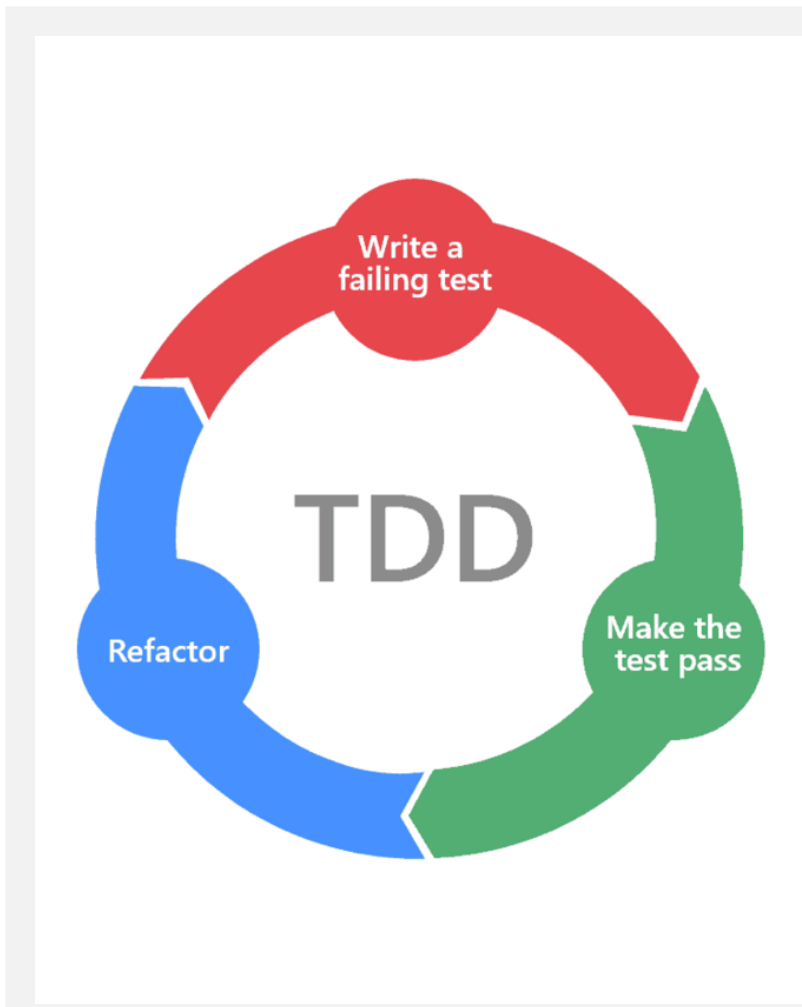


TEST ODAKLI GELİŞTİRME DÖNGÜSÜ

Kırmızı: Belirli bir işlevselliği tanımlayan başarısız bir test yazın. (Test başarısız olur çünkü kod henüz mevcut değildir.)

Yeşil: Testin geçmesi için mümkün olan en basit kodu yazın. İşlevselliğe odaklanın, mükemmelliğe değil.

Refaktör: Testlerin geçmeye devam ettiğinden emin olarak kodun tasarımını ve yapısını iyileştirin. Bu, gereksiz tekrarlamaları ortadan kaldırır ve okunabilirliği artırır. Bu adım, zaman içinde kod kalitesini korumak için çok önemlidir.



ÖNEMLİ!

TDD’da 3 Kural

Geçmeyen bir unit testi yazmadan, kod yazmamalısın.

Aynı anda bir fazla geçmeyen unit testi yazmamalısın.

Geçmeyen unit testi geçirecek kadar kod yazmalısın. Daha fazlası olmamalı.

```python
# Adım 1: Önce testleri yazarız.
# Burada, 'topla' ve 'cikar' fonksiyonları için istenen davranışları tanımlıyoruz.
class TestHesaplamaFonksiyonlari(unittest.TestCase):
    def test_topla(self):
        # Beklenen: 2 + 3 = 5
        self.assertEqual(topla(2, 3), 5)

    def test_cikar(self):
        # Beklenen: 5 - 3 = 2
        self.assertEqual(cikar(5, 3), 2)
```

1- WRITE A FAILING TEST

Testleri çalıştırdığımızda, henüz 'topla' ve 'cikar' fonksiyonları tanımlı olmadığı için testler hata verecektir.
Bu, ilk başarısız test döngüsünü simgeler.

```python
# Adım 2: Minimum kodu yazarak testleri geçmeye çalışırız.
def topla(a, b):
    # Bu fonksiyon, verilen iki sayının toplamını döndürür.
    return a + b

def cikar(a, b):
    # Bu fonksiyon, verilen iki sayının farkını döndürür.
    return a - b
```

2- MAKE THE TEST PASS

```python
# Adım 3: Testleri tekrar çalıştırarak tüm testlerin geçtiğini doğrularız.
if __name__ == '__main__':
    # Hata alınan testleri tespit edip bu döngüde hataları çözüyoruz.
    unittest.main()
```

3-REFACTOR

HANGI DURUMLARDA TEST YAZILMAZ

- Getter & Setter Metodları
- User Interface Kodları
- Kısa(2-3 satır uzunluğunda), çıktısı başka testlerden geçtiğine emin olunan kütüphane metodları.

```
def show_current_datetime():
    now = datetime.datetime.now()
    formatted = now.strftime("%Y-%m-%d %H:%M:%S")
    print(f"Current date and time: {formatted}")

if __name__ == "__main__":
    show_current_datetime()
```

FARKLI TEST YÖNTEMLERI

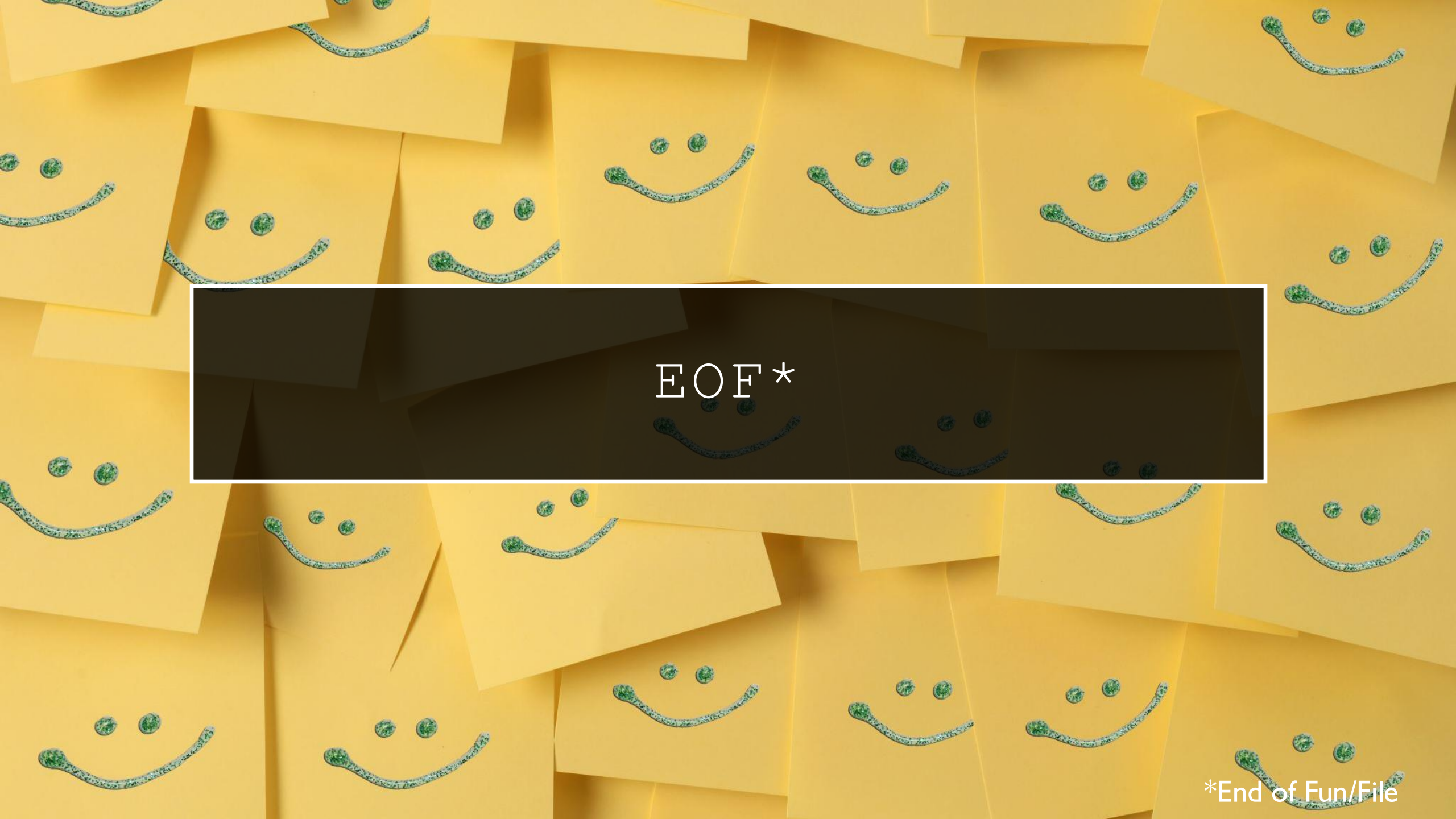
- **Birim Testi:** Kodun bireysel bileşenlerini veya birimlerini izole bir şekilde test etme.
- **Entegrasyon Testi:** Farklı birimler veya modüller arasındaki etkileşimi test etme.
- **Fonksiyonel Test (Kara Kutu Testi):** İç kod bilgisi olmadan sistemin işlevselliğini kullanıcı perspektifinden test etme. Girişlere ve çıkışlara odaklanır.
- **Uçtan Uca Test:** Baştan sona tüm sistem akışını test etme, gerçek dünya kullanıcı senaryolarını simüle etme.
- **Regresyon Testi:** Değişiklikler yapıldıktan sonra mevcut işlevselliğin bozulmadığından emin olmak için sistemi yeniden test etme.

TDD ODAKLI YAZILIM

TDD, Agile çerçevesi içinde yüksek kaliteli yazılım oluşturmak için güçlü bir tekniktir.

Başlangıçta zaman ve çaba gerektirse de, uzun vadede hataların azalması, bakım kolaylığı ve geliştirici güveninin artması açısından önemli faydalar sağlar.

Pratik mükemmelleştirir! Küçük, yönetilebilir görevlerle başlayın ve TDD'yi iş akışınıza kademeli olarak dahil edin.


EOF*
*End of Fun/File

REFS

- https://www.slideshare.net/slideshow/test-gdml-gelitirme-ve-birim-testler/187166228
- https://www.slideshare.net/ilkinulas/tdd-peak-games