Test Otomasyonu

Otomatik test ifadesinden ne anlıyorsunuz?

JUnit örneği

```java
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TriangleTest {

    @Test
    public void shouldClassifyEquilateral() {
        Triangle.Type result = Triangle.classify(10, 10, 10);
        assertEquals(Triangle.Type.EQUILATERAL, result);
    }

    // ...
```

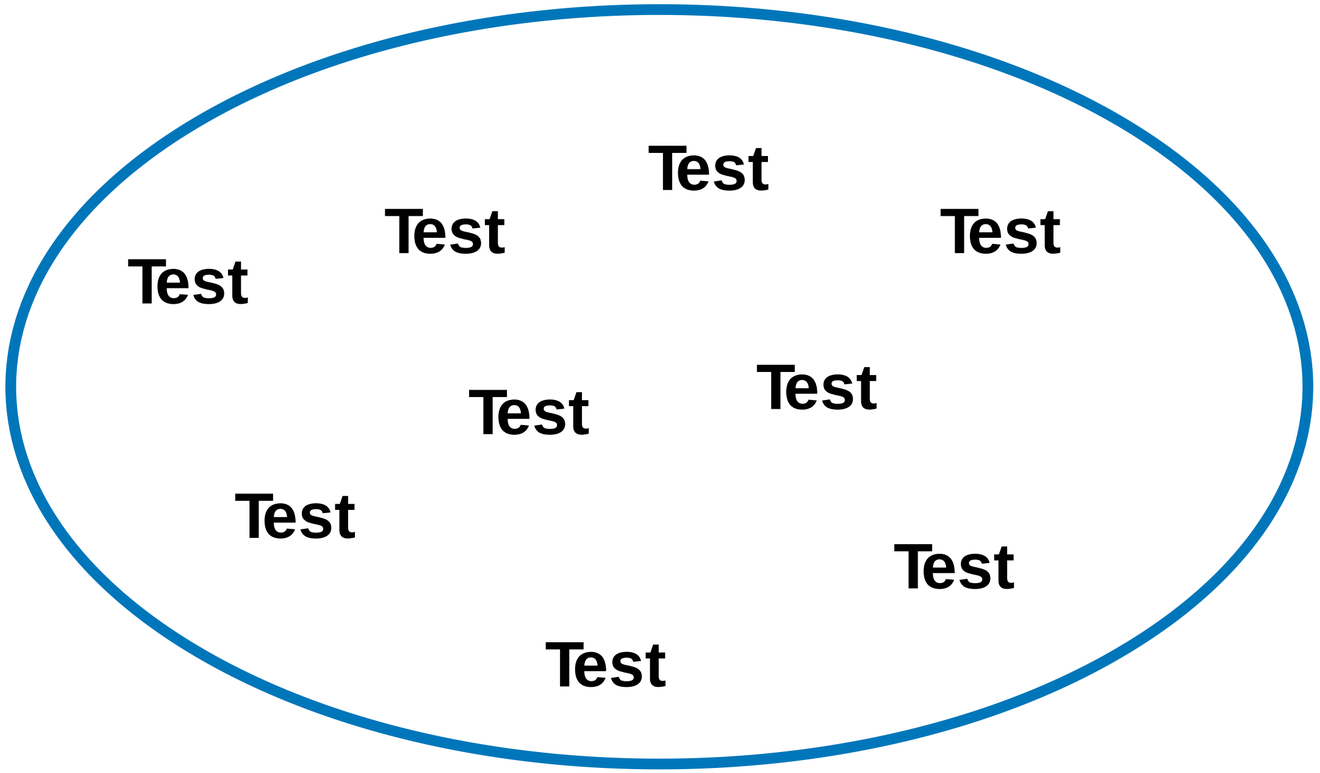
RSpec örneği

```
require_relative "../spec_helper"

describe "the add page" do
  it "is accessible from the search page" do
    visit "/search"
    click_link "Add a new player to the database"
    expect(page).to have_content "Add Player"
  end

  it "will not add a player with no details" do
    visit "/add"
    click_button "Submit"
    expect(page).to have_content "Please correct the errors below"
  end

  it "adds a player when all details are entered" do
    add_test_player
    expect(page).to have_content "George Test"
    clear_database
  end
end
```

Otomatik Test Senaryosunun Bileşenleri

1. Yazılımı test için doğru duruma getirmek için gereken girdiler
2. Mevcut test çalışması girişleri
3. Testin beklenen sonuçları
4. Sistem durumunun sıfırlanması

Bir Test Paketi – Bir Test Kümesi



İdeal olarak, testler herhangi bir sırayla yürütülebilir

Test Otomasyonunun Doğuşu

Test her zaman programlamanın bir parçası olmuştur

… İlk programınızı yazdığınızda, neredeyse kesinlikle bazı örnek verilerle denediniz

Uzun bir süre boyunca, endüstriyel uygulamada en son teknoloji buydu!

2000'li yılların başında yazılım geliştirme uygulamaları değişmeye başladı

Yazılım sistemleri, manuel testlerin, çalıştıklarından ve çalışmaya devam ettiklerinden emin olmanın etkili ve **verimli bir yolu olarak kalamayacak** kadar büyük ve karmaşık hale geldi.

Modern Yazılım Geliştirme Hızında Test Etme

Yazılım sistemleri giderek büyüyor ve her zamankinden daha karmaşık hale geliyor.

Örneğin, Google'daki tipik bir uygulama veya hizmet, binlerce veya milyonlarca kod satırından oluşur.

İnsanların bir **sistemdeki her davranışı manuel olarak doğrulama yeteneği**, çoğu yazılımdaki özellik ve platform patlamasına **ayak uyduramadı.**

Modern Yazılım Geliştirme Hızında Test Etme

Kod her değiştirildiğinde, Google aramanın işlevselliğini manuel olarak test etmenin ne kadar süreceğini bir düşünün.

… Sadece web araması değil, resimler, uçuşlar, film saatleri vb.

Ardından, desteklenmesi gereken her dil, ülke ve cihaz için bunu çarpın.

Ardından erişilebilirlik ve güvenlik gibi faktörleri ekleyin.

El ile test ölçeklendirilemez. Otomasyona ihtiyacımız var.

Geliştirici Odaklı Otomatik Test

Üretkenliği ve hızı artırmanın bir yolu olarak otomatik testleri kodlama fikri (örneğin JUnit'te) biraz zıt görünebilir.

Sonuçta, test yazma eylemi, ilk etapta bir özelliğin uygulanması kadar (hatta daha uzun değilse!) uzun sürebilir… değil mi?

Aksine!

Endüstride, yazılım testlerine yatırım yapmak, geliştirici üretkenliğine birkaç önemli fayda sağlar.

Daha Az Hata Ayıklama

Test edilen kod gönderildiğinde daha az kusura sahiptir.

En önemlisi, kod kullanım ömrü boyunca güncellenme eğiliminde olduğundan, varlığı boyunca daha az kusura sahiptir.

... diğer ekipler ve hatta otomatik kod bakım sistemleri tarafından değiştirilecek.

Kodda veya bağımlılıklarında yapılan değişiklikler, otomatik bir testle hızlı bir şekilde tespit edilebilir ve sorun üretime ulaşmadan önce geri alınabilir.

Değişikliklere Artan Güven

İyi testlere sahip projeler, projelerinin tüm önemli davranışları sürekli olarak doğrulandığından güvenle değiştirilebilir.

Bu projeler yeniden düzenlemeyi (refactor) teşvik eder..

Bir değişiklikten sonra, mevcut işlevlerden hiçbirini bozmadığımızdan emin olmak için otomatik testleri yeniden çalıştırabiliriz.

Geliştirilmiş Dokümantasyon

Yazılım dokümantasyonu herkesin bildiği gibi güvenilmezdir!

Her seferinde bir davranışı uygulayan açık, odaklanmış testler, yürütülebilir dokümantasyon işlevi görür.

Düşünceli Tasarım

Yeni kod için testler yazmak, kodun API tasarımını uygulamanın pratik bir yoludur.

Yeni kodun test edilmesi zorsa, bunun nedeni genellikle test edilen kodun çok fazla sorumluluğa sahip olması veya yönetilmesi zor bağımlılıklara sahip olmasıdır.

İyi tasarlanmış kod modüler olmalı, sıkı bağlantılardan (tight coupling) kaçınılmalı ve belirli sorumluluklara odaklanılmalıdır.

Tasarım sorunlarını erkenden düzeltmek daha sonra daha az yeniden çalışma anlamına gelir.

Hızlı, Yüksek Kaliteli Sürümler

Sağlıklı bir otomatik test paketi ile ekipler, uygulamalarının yeni sürümlerini güvenle yayınlayabilir.

Yüzlerce mühendisin ve her gün gönderilen binlerce kod değişikliğinin yer aldığı birçok büyük proje, genellikle her gün çok kısa yayın döngüleri içerir.

Bu, otomatik test olmadan mümkün olmazdı.

Otomatik Test Paketinin Faydaları

1. Daha Az Hata Ayıklama
2. Değişikliklere Artan Güven
3. Geliştirilmiş Dokümantasyon
4. Düşünceli Tasarım
5. Hızlı, yüksek kaliteli yazılım sürümleri