Yazılım Testinin Prensipleri ve Test Metodolojileri – Derinlemesine Analiz

Bugün yazılım test sürecinin iki temel konusunu detaylı olarak inceledim:

1. **Yazılım Testinin 7 Prensibi**
2. **Test Metodolojileri** (İşlevsel ve İşlevsel Olmayan Testler)

Bu bilgiler, test süreçlerini **standartlaştırmak** ve **evrensel kalite standartlarına** uyum sağlamak açısından kritik öneme sahiptir.

---

1. Yazılım Testinin 7 Prensibi

#	Prensip	Açıklama	Avantaj	Dezavantaj	Örnek
1	**Testing shows presence of defects, not their absence**	Testler hataların varlığını ortaya çıkarır, ama hatasızlığı kanıtlayamaz.	Gerçek durumu görmemizi sağlar, güvenlik sağlar.	Yanlış algı oluşabilir, hatasız sanılabilir.	100 test geçse de farklı bir senaryoda hata çıkabilir.
2	**Exhaustive testing is impossible**	Tüm olasılıkları test etmek mümkün değildir.	Kaynakları verimli kullanma fırsatı verir, önceliklendirme yapılır.	Bazı hatalar gözden kaçabilir.	Bir şifre alanında tüm karakter kombinasyonlarını test etmek imkânsızdır.
3	**Early testing**	Test sürecine gereksinim aşamasında başlanmalıdır.	Zaman ve maliyet tasarrufu sağlar, riskleri azaltır.	Gereksinimler değişirse yeniden test maliyeti çıkar.	Prototipte yapılan testler, canlıda düzeltmeye göre ucuzdur.
4	**Defect clustering**	Hatalar genellikle belirli modüllerde yoğunlaşır (80/20 kuralı).	Kritik modüllere odaklanarak verimlilik sağlar.	Diğer modüllerdeki hatalar gözden kaçabilir.	Ödeme modülünde çoğu hata yoğunlaşabilir.
5	**Pesticide paradox**	Aynı testler tekrar edilirse yeni hatalar bulunamaz.	Test senaryolarını yenilemeye teşvik eder.	Sürekli senaryo güncelleme ihtiyacı oluşur.	Hep aynı veri setiyle test edilen API'de edge-case hataları bulunamaz.

#	Prensip	Açıklama	Avantaj	Dezavantaj	Örnek
6	**Testing is context dependent**	Test yaklaşımı, uygulamanın türüne göre değişir.	Doğru strateji seçimiyle verimlilik artar.	Yanlış strateji seçimi ciddi sorun çıkarır.	Bankacılık uygulamasında güvenlik, oyun uygulamasında performans önceliklidir.
7	**Absence-of-errors fallacy**	Hata bulunmaması, kullanıcı ihtiyaçlarının karşılandığını göstermez.	Ürün analizini derinleştirir.	Yanlış güven duygusu oluşturabilir.	Mükemmel çalışan ama yanlış özelliklere sahip yazılım.

2. Test Metodolojileri

Test metodolojileri **Functional** (İşlevsel) ve **Non-Functional** (İşlevsel Olmayan) olmak üzere iki ana kategoriye ayrılır.

2.1 Functional Testing (İşlevsel Test)

Sistemin **gereksinimlere uygun çalışıp çalışmadığını** kontrol eder. Genellikle **Black Box Testing** altında değerlendirilir.

2.1.1 Unit Test

- **Tanım:** Tek birim veya fonksiyonun test edilmesi.
- **Avantaj:** Hataları erken bulur, geliştirme sürecinde hızlı geri bildirim sağlar.
- **Dezavantaj:** Büyük projelerde manuel olarak yazmak zaman alıcıdır, her hatayı yakalayamaz.
- **Örnek:** `topla()` fonksiyonunun pozitif, negatif, sıfır değerleriyle test edilmesi.

2.1.2 Smoke Test

- **Tanım:** Sistemin temel işlevlerinin çalıştığını kontrol eder.
- **Avantaj:** Sistemin "çalışır durumda" olup olmadığını hızlı gösterir.
- **Dezavantaj:** Derinlemesine test yapmaz, detay hatalar gözden kaçar.
- **Örnek:** Login ekranında doğru bilgilerle girişin ana sayfaya yönlendirmesi.

2.1.3 Sanity Test

- **Tanım:** Belirli değişiklik sonrası ilgili kısmın çalıştığını doğrular.
- **Avantaj:** Hızlı kontrol sağlar.
- **Dezavantaj:** Sadece ilgili modülü test eder, sistem geneli gözden kaçabilir.
- **Örnek:** Login algoritması değiştikten sonra sadece login fonksiyonunun test edilmesi.

2.1.4 Integration Test

- **Tanım:** Modüllerin birlikte çalışmasını test eder.
- **Avantaj:** Modüller arası uyumu sağlar.
- **Dezavantaj:** Hata kaynağını bulmak zaman alabilir.
- **Örnek:** Ödeme sistemi ile sipariş yönetim sisteminin uyum testi.

2.1.5 User Acceptance Test (UAT)

- **Tanım:** Ürünün müşteri tarafından onaylanmasını sağlar.
- **Avantaj:** Ürün yayın öncesi son onaydan geçer.
- **Dezavantaj:** Müşteri gereksinimleri net değilse süreç uzar.
- **Örnek:** Müşterinin demo ortamında ürünü test edip onay vermesi.

2.1.6 Localization Test

- **Tanım:** Bölgesel uygunluğu test eder.
- **Avantaj:** Yerel kullanıcı deneyimi iyileşir.
- **Dezavantaj:** Tüm bölge ve dil varyasyonlarını test etmek zor olabilir.
- **Örnek:** Türkiye'de tarih formatının `gg.aa.yyyy` olması.

2.1.7 Internationalization Test

- **Tanım:** Çoklu dil ve bölge desteğinin test edilmesi.
- **Avantaj:** Küresel pazar için uygunluk sağlar.
- **Dezavantaj:** Tüm dillerde test yapmak maliyetlidir.
- **Örnek:** Para birimi otomatik değişimi.

2.2 Non-Functional Testing (İşlevsel Olmayan Test)

Sistemin **kalite özelliklerini** ölçer.

2.2.1 Security Test

- **Avantaj:** Güvenlik açıklarını önceden tespit eder.
- **Dezavantaj:** Tüm güvenlik açıklarını tespit etmek imkânsıza yakındır.
- **Örnek:** SQL Injection testi.

2.2.2 Usability Test

- **Avantaj:** Kullanıcı deneyimini optimize eder.
- **Dezavantaj:** Subjektif sonuçlar çıkabilir.
- **Örnek:** Kullanıcıların bir ürün arama sayfasında aradıkları ürünü ne kadar kolay bulduğunu test etmek.

2.2.3 Portability Test

- **Avantaj:** Çoklu platform uyumluluğunu garanti eder.
- **Dezavantaj:** Tüm platformlarda test yapmak maliyetlidir.
- **Örnek:** Linux uygulamasının Windows'ta çalışması.

2.2.4 Maintainability Test

- **Avantaj:** Yazılım bakımını kolaylaştırır.
- **Dezavantaj:** Kod karmaşıksa süreç zorlaşır.
- **Örnek:** Kodun okunabilirliği ve dokümantasyon testi.

2.2.5 Performance Test

- **Avantaj:** Sistem kapasitesini ölçer, darboğazları bulur.
- **Dezavantaj:** Gerçek ortamı %100 simüle etmek zordur.
- **Örnek:** 1000 eşzamanlı kullanıcı ile sistem tepkisi.

2.2.6 Stress Test

- **Avantaj:** Sistemin limitlerini gösterir.
- **Dezavantaj:** Donanım/altyapı zarar görebilir.
- **Örnek:** 10.000 kullanıcının aynı anda giriş yapması.

2.2.7 Scalability Test

- **Avantaj:** Büyüme planlamasına yardımcı olur.
- **Dezavantaj:** Uzun süreli test gerektirebilir.
- **Örnek:** Kullanıcı sayısı arttığında sistem performansının ölçülmesi.

---

3. Kapanış – Bugünkü Kazanımlarım

Bugün yazılım testinin yalnızca **Unit Test'ten ibaret olmadığını** ve çok daha kapsamlı bir disiplin olduğunu öğrendim.
Her bir test türünün **avantajlarını ve dezavantajlarını** bilmek, gelecekte daha doğru test stratejileri seçmemi sağlayacak.

Özellikle login ekranı örneğinde, sadece doğru giriş durumunu test etmek yerine;

- Yanlış şifre
- Büyük/küçük harf farklılıkları
- Boş alan kontrolleri
- Maksimum karakter uzunluğu
  gibi durumları da test etmenin gerekliliğini kavradım.

Bu vizyon, yazılım kariyerimde **daha kaliteli ve hatasız ürünler** geliştirmeme katkı sağlayacak.