BS450 Yazılım Test Mühendisliği

Daha Büyük Testler

Daha Büyük Testler

Mock ile bile birim testi tüm ihtiyaçlarımızı karşılayamaz.

Kodumuzun davranışını başka kodların davranışlarıyla veya kodumuzun birlikte çalışması gereken harici hizmetlerle kontrol etmek için **entegrasyon testleri** gereklidir.

Ancak yazılımı bir bütün olarak test etmek için **sistem testlerine** ihtiyacımız var.

Daha büyük testler yazmak istememizin bir dizi başka nedenleri daha bulunmaktadır.

Neden Daha Büyük Testler

1. Aslına Uygunluk
2. Güvensiz dublörler
3. Yapılandırma problemleri
4. Yük altında ortaya çıkan durumlar
5. Beklenmeyen davranışlar, girdiler ve yan etkiler
6. Acil davranışlar

Aslına Uygunluk – Çevre Gerçekçiliği

Aslına uygunluk, bir testin, test edilen sistemin gerçek davranışını yansıtmasını sağlayan özelliktir.

Aslına uygunluğu anlamanın bir yolu **çevresel** açıdan değerlendirmektir:

- **Birim testleri, bir testi ve kodun küçük bir bölümünü çalıştırılabilir bir birim olarak bir araya getirir; bu, kodun test edilmesini sağlar, ancak üretim kodunun çalışma şeklinden çok farklıdır.**
- Üretim, testlerde doğruluğun en yüksek olduğu ortamdır ancak burada test yapılmamalıdır! En iyi alternatif, sistemin **aşamalı bir versiyonudur**; yani, gerçek verileri kullanmadan (veya kaybetmeden) testin yapılabileceği gerçek ortamındaki sistem.

Aslına Uygunluk – Çevre Gerçekçiliği

Testler, test içeriğinin gerçeğe ne kadar sadık olduğu açısından da ölçülebilir.

Test verilerinin kendisi gerçekçi görünmüyorsa, birçok el yapımı test mühendisler tarafından reddedilir.

Üretimden kopyalanan test verileri, bu şekilde yakalandığı için gerçeğe daha sadıktır.

Yeni kodu yayınlamadan önce gerçekçi test trafiğinin nasıl oluşturulacağı büyük bir zorluktur!

Güvensiz Dublörler

Dublörler, dublörü oldukları şeyi tam olarak kopyalamazlar.

Bu, birim testlerinde boşluklara yol açabilir.

Ayrıca, çiftler ve bunların orijinalleri, özellikle de orijinal sınıfın yazarı aynı zamanda dublörden sorumlu değilse, senkronizasyon bozulabilir.

Yapılandırma Problemleri

Birim testleri, sistemin tamamıyla ilgili yapılandırma ile ilgili problemleri test edemez.

Google'da, şirketin yaşadığı birçok büyük kesintinin bir numaralı nedeni yapılandırma değişiklikleridir.

2013 yılında, hiç test edilmemiş kötü bir ağ yapılandırmasının devreye sokulması nedeniyle küresel bir Google kesintisi yaşandı.

https://www.wired.co.uk/article/googledip

Yalnızca Yük Altında Ortaya Çıkan Sorunlar

Birim testleri sistem performansını ve özellikle **aşırı yükler** altındaki performansını **kontrol edemez** (**stres testi** ile yapılabilir).

Oldukça büyük verilerle gerçekleşir! Genellikle saniyede binlerce veya milyonlarca sorgu ile.

Beklenmeyen davranışlar, girdiler ve yan etkiler

Birim testleri onları yazan mühendisin hayal gücüyle sınırlıdır!

Kullanıcıların bir üründe bulduğu sorunlar çoğunlukla beklenmedik sorunlardır (aksi takdirde test edilmiş olurlardı!)

Otomatik test tekniklerinin (ve yapay zekanın) işe yaradığı yer burasıdır (örneğin **bulanıklaştırma (fuzzing)**).

Acil Davranışlar

Birim testleri kapsadıkları kapsam ile sınırlıdır, dolayısıyla bu kapsamın dışındaki davranış değişiklikleri tespit edilemez.

… ve birim testleri hızlı ve güvenilir olacak şekilde tasarlandığından, gerçek bağımlılıklar, ağ ve verilerden kaynaklanan kaosu bilinçli olarak ortadan kaldırır.

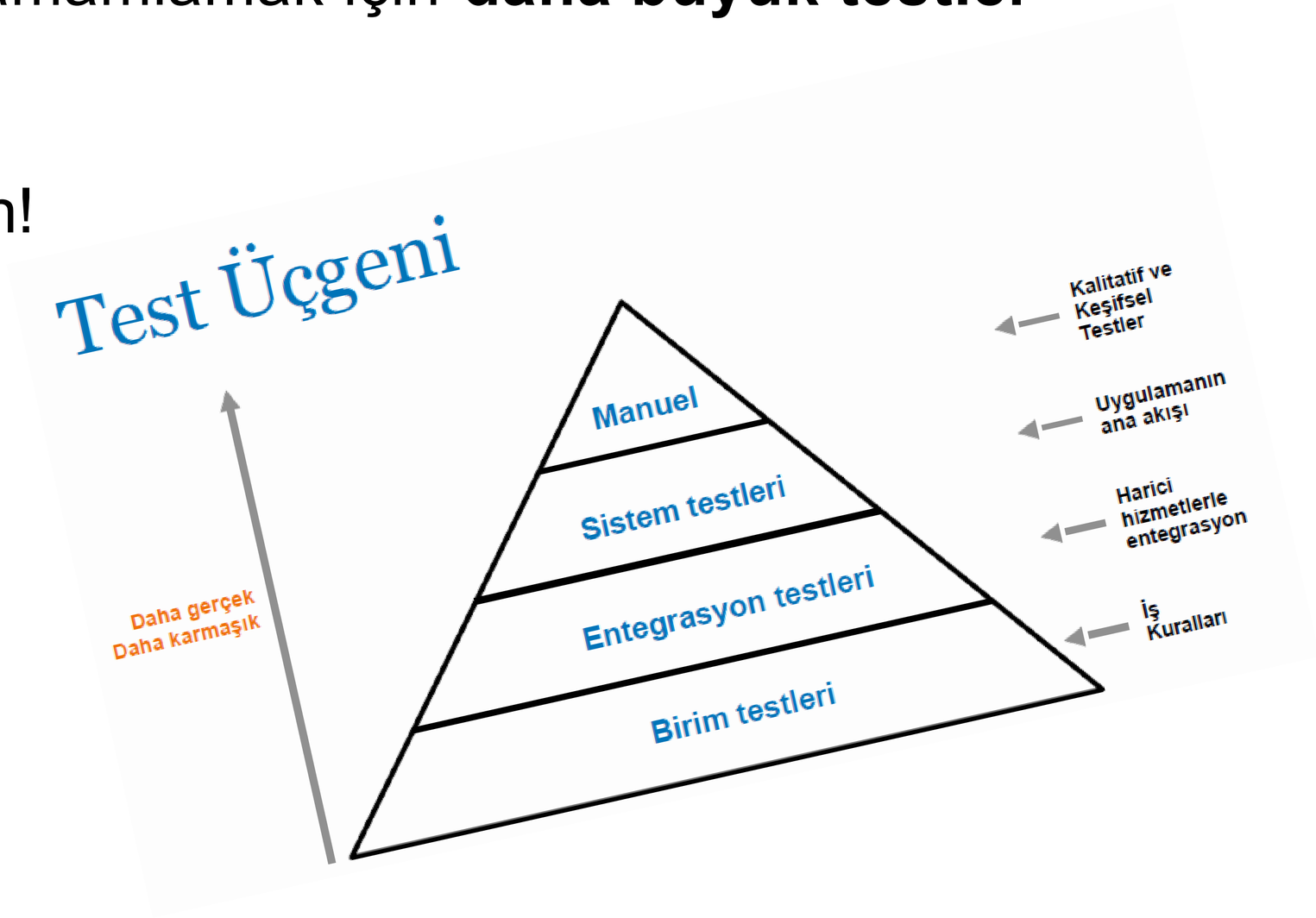
Birim testleri teorik fizikteki problemler gibidir... boşluğa yerleştirilmiştir, gerçek dünyanın karmaşasından özenle gizlenmiştir.

Bu, hız ve güvenilirlik açısından harikadır ancak **bazı kusur türlerini gözden kaçırır.**

Bu nedenler "birim testi yapmayın" demek değildir

Ancak birim testleri tamamlamak için **daha büyük testler** kullanılmalıdır.

Test üçgenini hatırlayın!



Bazı Sistem Testi Türleri

Tarayıcı ve Cihaz Testi

Tarayıcılar ve cihazlar farklılık gösterir ve genellikle kullanıcıların uygulamayla nasıl etkileşimde bulunacağını belirler.

Örneğin, ekran boyutu göz önüne alındığında ne/ne kadar içeriğin oluşturulabileceği.

Cihazların tüm bu sürümlerine sahip olmak sorunlu olabilir ve bu da genel cihaz kütüphanelerinin kullanılmasına yol açabilir.

Performans, Yük ve Stres Testi

Bu testler, sürümler arasında performansta herhangi bir bozulma olmadığından ve sistemin trafikte beklenen ani artışları karşılayabildiğinden emin olmak açısından kritik öneme sahiptir.

Deployment Yapılandırma Testi

Çoğu zaman kusurların kaynağı kod değil, yapılandırmadır:

- Veri dosyaları
- Veritabanları
- Seçenek tanımları

Daha büyük testler bu şeyleri test edebilir çünkü yapılandırma dosyaları ürünün binary dosyasının başlatılması sırasında okunur.

Sıkça, bu testler sistemin duman testleridir ve genellikle ek test verilerine veya iddialara çok fazla ihtiyaç duymazlar: **Sistem başarılı bir şekilde başlarsa, test geçerlidir!**

Araştırma (Exploratory) Testi

Keşif testi, yeni ve kurulmuş kullanıcı senaryolarını deneyerek şüpheli davranışları aramaya odaklanan bir tür manuel testtir.

Eğitimli kullanıcılar/testerlar, beklenen veya sezgisel davranıştan sapma gösteren yeni sistem yollarını veya güvenlik açıklarını aramak için bir ürünle etkileşime girerler.

Kullanıcı Kabul Testleri

Birim testleri geliştiriciler tarafından yazılır, bu nedenle sistem için amaçlanan davranış hakkında yanlış anlamaların, sadece kodda değil, aynı zamanda birim testlerinde de yansıma olasılığı oldukça yüksektir.

Kullanıcı Kabul Testleri, belirli kullanıcı yolculukları için genel davranışın amaçlandığı gibi olduğunu sağlamak için sistem üzerinde uygulama yapar.

Bu tür testlerin sistem "uçtan uca" çalıştığı söylenir ve bazen uçtan uca (E2E) testleri olarak adlandırılır.

Cucumber ve RSpec gibi çeşitli frameworkler, testleri yazılabilir ve kullanıcı dostu bir dilde okunabilir hale getirmek için mevcuttur.

Felaket Kurtarma

Bu testler, veri merkezi yangınları, kötü niyetli saldırılar, depremler gibi beklenmedik değişiklikler veya arızalar karşısında sistemın ne kadar iyi tepki verdiğini ortaya çıkarır.

Sistemler, pratikte ne olduğunun etkilerini gözlemlemek için devre dışı bırakılır veya kapatılır.

Bu genellikle organizasyonun kırılganlığını da test eder - çünkü önemli karar vericiler aniden iletişim dışı kalabilirler.

Kaos Mühendisliği

Netflix tarafından popüler hale getirilen kaos mühendisliği, pratikte çalkantılı koşulları simüle eden programlar yazarak sistemlerin dayanıklılığını test eder.

Bu programlar, sunucu kapanmaları, gecikme enjeksiyonları ve kaynak tükenmeleri gibi hataları/sorunları dahil eder.

Bu hatalar, derste daha sonra ele alacağımız mutasyon analiziyle ortaya çıkan hatalardan daha yüksek düzeydedir ve daha hedefe yöneliktir.

Daha Büyük Testlerle İlgili Sorunlar

Daha Büyük Testlerde Kırılganlık

Daha büyük testler, kırılganlık sorunundan kaçınamaz.

Kırılganlığın nedeni, kodun içsel değişikliklerine (yeniden yapılandırma) bağlı olarak dışsal davranışını etkilemeyen testlerin başarısız olmasıdır.

Örnek:

Uçtan uca sistem testlerinde, testler GUI öğelerinin adlarına veya düzenlerine çok sıkı bağlı olabilir - bu da arayüz değişirse testin başarısız olmasına neden olabilir.

Flaky Testler

Daha büyük testler, flakiness sorunuyla karşılaşma açısından daha yatkındır.

Flakiness, testlerde veya kodda belirsizlikten kaynaklanan ve aslında çalışan testlerin başarısız olmasına neden olan bir durumdur (ve daha az yaygın olarak, başarısız olan testin başarılı olması).

Örnek:

> Bir sunucudan yanıt gerektiren bazı kodları çalıştıran bir sistem testi. Sunucu çalışır durumdaysa, test başarılı olur, ancak sunucu kapalı durumdaysa, test başarısız olur. Ancak, test başarısız olduğunda, kodda bir hata yoktur.

Birim testleri de flaky olabilir, ancak mantığa odaklanır ve belirsizlik içeren unsurları içermezse, bu durum daha az yaygındır.

Flaky Testlerin Neden Olduğu Sorunlar

Flaky test başarısızlıklarının aralıklı olması, geliştiriciler için çılgına döndürücüdür çünkü flakiness kaynağı genellikle **takip etmesi zordur**:

> Sorunun belirtileri (başarısız bir test), özellikle sistem testleri ile birlikte, belirsiz davranışın kaynağından genellikle uzaktır.

Ancak bir test paketinde birkaç flaky test olduğunda, **geliştiriciler genellikle o teste güvenmeyi durdurur ve çalıştırmaz.**

Flaky testler, sürekli entegrasyon sunucularındaki yapıların başarısız olmasında **sıklıkla karşılaşılan** bir faktördür.

Neden flaky olabilirim?

Bazı kodları hangi gün olduğuna bağlı olarak test ediyorum.

Neden flaky olabilirim?

Birçok iş parçacığının başlatıldığı ve yürütüldüğü bir kodu test ediyorum.

Neden flaky olabilirim?

Belirli bir zaman çerçevesinde bazı görevlerin yürütülmesine bağımlı olan bazı kodları test ediyorum.

Neden flaky olabilirim?

En son haber öğelerini kontrol eden bazı kodları test ediyorum.

Neden flaky olabilirim?

Daha Büyük Testlerin Sorunları

1. **Güvenilirlik** – büyük testler kırılgan ve flaky olabilir
2. **Hız** – daha büyük testler yavaş olabilir bu da geliştiricinin çalışma hızını düşürür
3. **Mülkiyet** – birim testlerinin bir sahibi vardır. Sistem testinden kim sorumlu?
4. **Standardizasyon**. Birim testleri standart frameworklere ve forma sahiptir. Ancak daha büyük testler, farklı altyapılar ve ortamlarla başa çıkmak zorundadır. Bu testlerde birden fazla rekabet eden araç ve framework bulunabilir (bazıları şirket içinde geliştirilir).