1) **Regression test nedir ? Kısaca açıklayınız.**

Regresyon testi canlıda çalışan kodun üzerinde yapılan değişikliklerin kontrolü için kullanılır. Bu değişiklikler yeni bir fonksiyon, hata çözümü ya da performans geliştirmesi olabilir. Regresyon testleri genellikle değişiklikler son aşamaya geldiğinde ve yazılımın yeni sürümü yayınlamadan önce gerçekleştirilir. Regresyon testlerinin öncelikli amacı, uygulamanın kritik alanlarının hala beklendiği gibi çalıştığını kontrol etmektedir.
Regression test adımınları seçimindeki bazı önemli noktalar:

- Kullanıcıların yoğun olarak kullanıldığı alanlar
- Genellikle hata çıkan uygulama alanları
- Ana fonksiyonlar
- Yüksek karmaşık fonksiyonlar
- Son değişikliklerin yapıldığı alanlar
- Önemli entegrasyonlar

2) **A/B test nedir ? Kısaca açıklayınız.**

A/B testi (bölme testi veya kova testi olarak da bilinir) hangisinin daha iyi performans gösterdiğini belirlemek için bir web sayfasının veya uygulamanın iki sürümünü birbiriyle karşılaştırmanın bir yöntemidir. AB testi, esas olarak, bir sayfanın iki veya daha fazla varyantının rasgele kullanıcılara gösterildiği bir deneydir ve hangi varyasyonun belirli bir dönüşüm hedefi için daha iyi performans gösterdiğini belirlemek için istatistiksel analiz kullanılır.

3) **Black box / white box test kavramlarını açıklayınız.**

Black box test: Test edilecek uygulamanın içeriği görülmeden yapılan testlerdir. Sadece input verilip output alınarak yapılan testlerdir.
White box test: Test edilecek uygulamanın içeriği görülerek yapılan testlerdir. Kodları ve uygulamanın yapısı bilinerek yapılır. Hangi basamaklar sonucunda neler elde edilebileceğinin analizi yapılabilir.

4) **Mutation test nedir ? Kısaca açıklayınız**

MT olarak da bilinir hataya dayalı test, program mutasyonu, hataya dayalı test, veya mutasyon analizi. Adından da anlaşılacağı gibi mutasyon testi, değişikliklere veya mutasyonlara dayanan bir yazılım test türüdür. Tanımlanmış test senaryolarının koddaki hataları tespit edip edemediğini kontrol etmek için kaynak kodunda küçük değişiklikler yapılır.İdeal durum, test durumlarından hiçbirinin geçmemesidir. Test başarılı olursa, kodda bir hata var demektir. Mutantın (kodumuzun değiştirilmiş versiyonu) yaşadığını söylüyoruz. Test başarısız olursa, kodda hata yoktur ve mutant öldürülmüştür. Amacımız tüm mutantları öldürmek. Pitest gibi araçlar ile mutant programlar üretmektedir.

5) **Behavior Driven Development (BDD) nedir, neyi amaçlamaktadır?**

Yazılım süreçlerinin daha test odaklı gitmesini sağlayan bir yaklaşımdır. Aynı zamanda müşteri ile aramızda yaşayan bir döküman oluşmasını sağlayabilir. BDD, Test Driven Development (TDD) gibi prensip olarak öncelikle test kodları yazılsın daha sonrasında proje kodu yazılsın anlayışını benimsemektedir.
Cucumber BDD anlayışını benimseyen açık kaynak kodlu bir araçtır. Testler neredeyse düz metin dili olan Gherkin ile yazılmıştır. Bu dilin en güzel tarafı okunduğunda herkes tarafından kolaylıkla anlaşılabilir olmasıdır.

6) **Agile test quadrant nedir? Quadrant'ların kapsamını kısaca açıklayınız.**
Ekiplerin ihtiyaç duyulan testi belirlemesine planlanmasına ve uygulanmasına yardımcı olmak için bir sınıflandırma yapmasıdır.
Q1: Unıt testler ve Compenent testlerin yapıldığı aşamadır. Bu süreç otomasyon olarak ilerler
Q2: Functional testler, story testler, simülasyonlar, örneklerin yapıldığı aşamadır. Manuel ve otomatik testlerle iş odaklı olarak yapılır.
Q3:Senaryolar, User acceptance testler yapılır alpha veya beta aşamaları olarak da geçer. QA mühendisleri tarafından manuel testler ile ilerler.
Q4:Performans,Load test, security testlerin olduğu aşamadır.

The Practical Test Pyramid Özet (Martin Fowler)

Test Otomasyonunun Önemi: Yazılım sektörünün her geçen gün gelişmesiyle kullanıcı sayıları da yazılımlar da artıyor. Büyüyen yazılımlarda test ve deploying işlemleri imkansızlaşmaya başlıyor. Eğer daha hızlı ve kaliteyi bozmadan uygulamalar geliştirmek istiyorsak otomatize edilmiş testler yazmamız çok önemlidir.

Test Piramiti: Otomatize testler yazmak istiyorsak bilmemiz konseptlerden biri test piramitidir. Mike Cohn tarafından Succeeding with Agile kitabında ortaya atmıştır.

Test piramiti 3 aşamadan oluşur ve yukardan aşağı bu katmanları kapsaması gerekir:

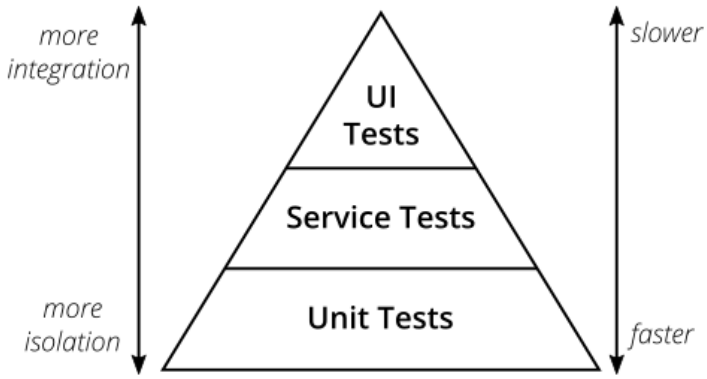

Figure 2: The Test Pyramid

Aşagıya doğru gidildikçe daha hızlı ve daha küçük testler yazmanız gerekirken yukarı da daha kapsamlı ayrıntılı testler yazmamız gerektiğini söyler.

Unit Tests: Unit testler codebase'imizin belirli bir biriminin amaçlandığı gibi çalıştığından emin olur. Unit testler en dar kapsamlı testlerdir.

Peki bir birim nedir? Birçok insana bir birimin anlamını sorarsanız hepsinden farklı cevaplar alırsınız. Fonksiyonel bir dil kullanıyorsanız büyük ihtimal fonksiyonu bir birim olarak kabul edebilirsiniz. OOP'de ise sadece bit metod veya classın tamamı bir birim olabilir.

Sociable unit test en temel kod parçacığını test ederken bağımlı olduğu kod parçacıklarını da unit teste dahil eder.

Solitary unit test en temel kod parçacığını test ederken bağımlı olduğu kod parçacıklarını izole ederek test eder

Test Yapısı: İyi bir test yapısı olarak:

- Test verisini hazırlamak
- Metodları testte çağırmak
- Beklenen ve dönen değeri karşılaştırmak

''Arrange, Act, Assert'' ve ''given, when, them'' yapılarını BDD den hatırlamamız iyi olur.

Integration Tests: Integration testlerinde unit testlerden farklı olarak servislerin database ve kendi aralarındaki iletişimlerini birleşik olarak test etmemizi sağlar. Büyük ve maliyetli testlerdir.

Contract Tests: Consumer-Driven Contracts, kullanılan (provider) ve kullanıcı (consumer) servisleri arasındaki entegrasyonun bir kontrat üzerinden tasarlandığı ve yönetildiği yaklaşım olarak düşünülebilir. Consumer'lara sağlanan API üzerinde herhangi değişiklik yapıldığında, bu servisi kullanan tüm consumer'ların API release'lerini takip ederek kendilerini güncelliyor olması gerekecektir. Veya API tarafının entegrasyonlarda sorun oluşmaması için önceki versiyon(ları) da desteklemeye devam etmesi zorunlu olacaktır. Bu sorunla başa çıkma için, API'lar üzerinde yapılacak değişikliklerin validasyonu için bir kontrat mekanizması kurulabilir. Bu durumda servisi kullanacak olan tüm consumer'lar kendi kontratlarını hazırlayarak provider'ın erişimine açacaktır. Bu yaklaşımdaki önemli fayda, provider'ın hangi fonksiyonlarının nasıl kullanıldığını bilmesi ve bu sayede yapılacak önemli bir değişiklikte hangi kullanıcıların etkileneceğini öngörebilmesi ya da erişim noktalarını sabit tutarak kodun içyapısını değiştirebilmesidir.

UI Tests: Son kullanıcıya yönelik yazılan testlerdir. Arka planı bilmeden sadece arayüzden yapılan testlerdir. Selenium gibi araçlar kullanılır.

Acceptance Tests: ürünün kullanıcılar tarafından kontrol edilerek, ihtiyaçların eksiksiz ve doğru bir şekilde karşılandığından emin olunmasını sağlayan önemli testlerden biridir.