MANUEL TEST iNTERVİEW SORULARI

By AKİF RENCBER
akifrencber@gmail.com

1. SDCL ve STLC'nin tanımı, aralarındaki Farklar ve Benzerlikler ile İki kavramın aşamalarını belirtiniz

SDLC: Yazılım Geliştirme Yaşam Döngüsü (Software Development Life Cycle), yazılım endüstrisi tarafından yüksek kaliteli yazılımlar geliştirmek, bu yazılımları test etmek ve tasarım yapmak için kullanılan bir süreçtir.

STLC: Yazılım Test Yaşam Döngüsü, (Software Testing Life Cycle) yazılımın veya ürünün kalitesini korumak veya garantiye almak için test ekibi tarafından gerçekleştirilen bir dizi faaliyettir.

SDLC ve STLC birbirine çok benzeyen iki süreçtir. Fakat kapsadıkları alan bakımından birbirlerinden ayrılmaktadır. Bu iki kavram birbiri ile iç içe geçmiş iki ayrı süreç olarak ifade edilebilir.

STLC, SDLC süreçlerinin bir parçasıdır ve SDLC bir yazılım ürünü geliştirme sürecinin tümünü kapsar iken STLC sadece test aşamasını kapsamaktadır.

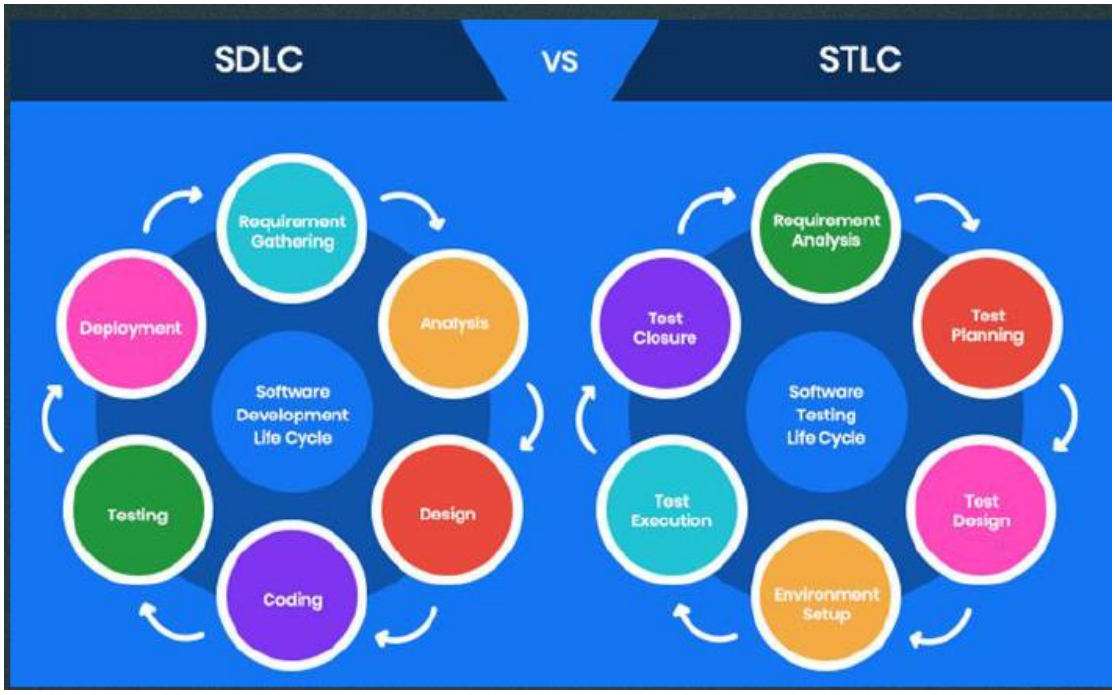
Aşağıdaki görselde görüldüğü gibi ikisi de 6 aşamadan oluşmaktadır.

SDLC'de aşamalar:

1. Gereksinimlerin Toplanması
2. Analiz
3. Dizayn
4. Kodlama
5. Testing
6. Deployment (dağıtım)

STLC'de aşamalar:

1. Gereksiminlerin analizi
2. Test Planlama
3. Test Dizayn
4. Test Ortamının Kurulumu
5. Test Execution
6. Testi Tamamlama



2. Pozitif ve Negatif Test nedir? Örnek ile açıklayın

Pozitif testler, uygulamanın doğru girdi ile beklendiği gibi performans gösterdiğini test etmek için yapılan **happy path** senaryolardır.

Negatif testler ise uygulamanın uygun şekilde yanıt verdiğinden emin olmak için sisteme yanlış veri girilmesi ile yapılan test uygulamasıdır. Bu testler ile yazılımın çalışmadığı noktaları tespit edebiliriz. Bu test ile yazılımın olumsuz durumlarda nasıl çalıştığını, olumsuz sonuçlarda ne gösterdiğini, birimin dayanıklılığını test edebiliriz.

Burada verilebilecek en yaygın ve bilindik örnek bir uygulama login modülüdür. Uygulamanın hayati modullerinden olan ve priority'si high olan login modülü üzerinde hem pozitif hem de negatif testler yapmamız mümkündür ve gereklidir. Bize verilen gereksinimler bağlamında sınır değer analizi yöntemi ile pozitif test senaryoları yazarak uygulamanın geçerli veriler ile doğru bir şekilde çalışıp çalışmadığını test ederiz. Aynı şekilde uygulamanın gereksinimler dışında farklı datalar ile test edilmesi de gerekir. Örneğin geçersiz değerler girerek negatif testler yaparız ve uygulamanın kalitesini test edebiliriz. Bu da negatif test uygulamasıdır.

3. Bug Nedir, Hata Tanımları Nelerdir ve Bug ile Defect'in farkı nedir?

Bug, yazılımın veya uygulamanın gereksinime göre çalışmadığı anlamına gelen kusurların gayri resmi adıdır.

Terimler (Terms)	Tanım (Description)	Sorumlu Kişi
Defect	Uygulamanın gereksinimlere göre çalışmadığı zaman ortaya çıkan durumdur. Henüz canlıya çıkmamış bir projede bulunan hatalara verilen isimdir.	Test Engineer
Bug	Hatanın gayri resmi tanımıdır. Bu tanımı biz canlıya çıkmış bir uygulamada karşımıza çıkan hataları tanımlarken kullanırız. Bu bağlamda defetc ile arasında en temel fark budur.	Test Engineer
Error	Koddaki sorunun ortaya çıkardığı hatadır.	Developer ve Test Engineer
Issue	Uygulama iş gereksinimlerini (business requirement) ortaya çıkan durumdur.	Customer (Müşteri)
Mistake	Dökümandaki problemleri tanımlayan kavramdır.	
Failure	Yazılımın başarısız olmasına neden olan çok sayıda hatayı tanımlar.	

4. Hata Yönetim Süreci ve Hata Raporu İçeriğini Anlatır mısınız?

Yazılım Testinde Hata Raporu, yazılım uygulamasında bulunan hatalar hakkında ayrıntılı bir belgedir. Hata raporu; açıklama, hatanın bulunduğu tarih, hatayı bulan test uzmanının adı, hatayı düzelten geliştiricinin adı vb. gibi hatalarla ilgili her ayrıntıyı içerir. Hata raporu, gelecekte benzer hataların tespit edilmesine yardımcı olur, böylece önlenebilir.

Hatayı geliştiriciye bildirirken, Hata Raporunuz aşağıdaki bilgileri içermelidir;

Defect_ID - Kusur için benzersiz tanımlama numarası.

Hata Açıklaması - Hatanın bulunduğu modül hakkında bilgiler de dahil olmak üzere hatanın ayrıntılı açıklaması.

Sürüm - Hatanın bulunduğu uygulamanın sürümü.

Adımlar - Developerın hataları yeniden üretebileceği ekran görüntüleri ile birlikte ayrıntılı adımlar.

Oluşturulduğu Tarih - Hatanın oluşturulduğu tarih

Referans- Burada, hatanın anlaşılmasına yardımcı olmak için gereksinimler, tasarım, mimari ve hatta hatanın ekran görüntüleri gibi belgelere referans sağlarsınız.

Tespit Eden - Hatayı bildiren test uzmanının adı/ID'si

Durum - Kusurun durumu

Düzelten - Düzelten geliştiricinin adı/kimliği

Kapatıldığı Tarih - Hatanın kapatıldığı tarih

NOT: Hatanın uygulama üzerindeki etkisini tanımlayan önem derecesi, kusur giderme aciliyeti ile ilgili olan öncelik. Önem önceliği, hatanın düzeltilmesi gereken etki aciliyetine bağlı olarak sırasıyla **Yüksek/Orta/Düşük** olabilir.

5. Test ve Hata Ayıklama (debugging) Kavramlarını açıklayıp farklarını belirtiniz.

Bunların her ikisi de bir sistem/yazılım/uygulama/araçtaki hataları bulmak ve ardından bunları uygun şekilde düzeltmek için kullanılır. Ancak test etme ve hata ayıklama arasında çeşitli farklar vardır. Test bug ve error ları bulduğumuz süreçtir. Hata ayıklama ise test sürecinde bulduğumuz hataları düzelttiğimiz süreçtir.

Test Nedir?

Temel olarak, bir uygulama veya yazılımın hatasız olduğunu, tüm teknik gereksinimleri karşıladığını, tüm geliştirme ve tasarım gereksinimlerine uyduğunu ve tüm kullanıcı gereksinimlerini karşıladığını doğruladığımız ve onayladığımız bir süreçtir. Test, amaçlanan yazılımın veya uygulamanın bu gereksinimleri verimli ve etkili bir şekilde karşılamasını ve tüm sınır durumları ve istisnai durumları ele almasını sağlar.

Hata Ayıklama Nedir?

Temel olarak bir yazılımda veya uygulamada bulunan herhangi bir hatayı düzelttiğimiz bir süreçtir. Bu süreçte önce hataları tespit eder, sonra analiz eder ve daha sonra da gideririz. Hata ayıklama, amaçlanan yazılım düzgün bir şekilde yürütülemedikten sonra başlar. Burada, sorunu çözerek ve yazılımı başarıyla test ederek sonuçlandırıyoruz. Bu süreç son derece sıkıcı ve karmaşık olarak kabul edilir çünkü hata ayıklamanın her aşamasında mevcut hataları tanımlamamız ve çözmemiz gerekir.

6. Bug veya defect nasıl meydana gelir?

Yazılım testindeki bug veya defect, yanlış, hatalı veya fazla kodlamadan meydana gelebilir. Yanlış kodlama aslında yanlış uygulama inşa etmektir. Bu bağlamda örnek verecek olursak:

Gmail uygulamasında "Gelen Kutusu" bağlantısına tıkladığımızda "Taslak" sayfasına gittiğini varsayalım, bu developer tarafından yapılan yanlış kodlama nedeniyle ortaya çıkan durumdur ve bu sebeple bir hatadır.

Eksik kodlamada yine gereksinimlerin karşılanmadığı anlamına gelir. Bu doğrultuda: eksik kodlama ile az önce ifade ettiğimiz örnek bağlamında yapılması gereken bir uygulama aslıda yapılmamış olacaktır.

Hem bir developer hem de bir tester için gereksinimler oldukça önemlidir. Bu sebeple **gereksinimlerden bir adım fazla ya da bir adım eksik adım atmamak gerekir.** Fazla kodda bu bağlamda istenilen uygulamanın yanlış seyretmesine neden olabilir.

7. Hata İzleme Araçlarının Öneminden Bahdederek Bu Bağlamda Yaygın Kullanılan Hata İzleme Araçlarından Birkaçını Belirtiniz.

1. Yazılım kalitesini arttırmak şirketler için en temel ihtiyaçlardan biridir. Bu bağlamda bir bug izleme aracı ile yazılımın en az sorun ile piyasaya çıkması sağlanır. Hata izleme araçları sorunları önceden tespit etmeyi, hataların giderilmesini ve düzeltilmesini sağlar. Aynı zamanda bir hatayı veya sorunu düzeltmek için test uzmanları tarafından yapılanları da analiz eder. Bu sayede tahsis edilen bütçe ve planlanan zaman dahilinde verimli bir ürünün zamanında teslim edilmesini sağlar.
2. Bug izleme araçları insan hatasından kaynalı sorunları en aza indirir. Tabii ki bu izleme araçlarına yine manuel olarak girişler olsa da **burası hata hafızası** olduğu için unutma ve diğer sebeplerden gözden kaçabilecek kusurların takibini kolaylaştırır. Tekrar eden hataların görülmesi, test uzmanının işlerini daha kolay ve sistematik bir şekilde takip etmesi ve önemsiz sorunlar ile daha az zaman kaybetme gibi bir çok fayda sağlar. Bu sayede zamandan ve paradan tasarruf edilmiş olur.
3. Bug izleme araçları ekiplerin ortak çalışmasına da uygun olduğu için iletişimi güçlendiren ve bir merkezden sorunları ortak bir şekilde oldukça hızlı çözülmesini sağlayan araçlardır. Sonuçta ortaya çıkan rapor ile bir sonraki projede planlama için gerekli doneler elde edilmiş olur.
4. En büyük avantajlarından biri de hata giderme sürecinin kayıtlarının güvenli bir şekilde merkezi veri sistemine kaydedilmesidir. Bu, hatayı düzelttikten sonra bile, hataların ayrıntılı bir listesine, düzeltme sürecine ve belirli bir sorunu düzeltmenin gerçekte ne kadar sürdüğüne sahip olduğunuz anlamına gelir. Temel olarak, tüm test aşaması merkezi veri sistemine kaydedilir ve bu da hata eğilimlerini takip etmenizi sağlar.

5. Test uzmanlarının sorunları çok daha erken tespit etmesine ve öncelikli olarak harekete geçmesine yardımcı olur. Öte yandan, insan takibi bunu kısa sürede başaramaz. Sonuç olarak, bir hata izleme aracıyla çalıştırdığınızda yazılımın işlevsel aksaklıklarla piyasaya sürülme olasılığı neredeyse sıfırdır.

-- Yaygın Olarak Kullanılan Bug İzleme Araçlarından Birkaçı Aşağıdaki Gibidir:

- **Jira**
- **Bugzilla**
- **BugNet**
- **Redmine**
- **Mantis**
- **Trac**
- **Backlog**
- **Confluence**

8. Bir Yazılım Testindeki Test Seviyeleri Nelerdir Kısaca İfade Ediniz

-- Birim Testi (Unit Testing): Bu test, yazılımın her bir modulünü ayrı ayrı doğrulamayı hedef alır. Dolayısıyla bu testler doğrulanan bu modüllerin kabul kriterlerini karşıladığından ve beklenen işlevselliği sağladığından emin olmak için yapılır. **[Developer tarafından yapılır.]--Kodun iç yapısını bilen test uzmanları da bu testi yapabilirler.**

--Entegrasyon Testi (Integration Testing): Birbiri ile etkileşim halinde olan birimlerin bir arada ve birbiri ile uyumlu halde çalışıp çalışmadığı test etmek için yapılan bir test seviyesidir. Böylece unit test ile gözden kaçan hatalar bu test seviyesinde bulunabilir. **(Birim bazda developerlar, sistem bazında ise test uzmanları tarafından yapılır)**

--Sistem Testi (System Testing): Yazılımın tüm bileşenlerinin bir bütün olarak test edildi test seviyesidir. **(Test uzmanları tarafından yapılır)**

--Kabul Testi (Acceptance Testing): Kabul Testi, bir ürünün onaylandığı veya reddedildiği yazılım test sürecindeki adımdır. Bu tür testlerin amacı, sistemin son kullanıcı gereksinimlerini karşılayıp karşılamadığını ve dağıtıma hazır olup olmadığını doğrulamaktır. (**Kullanıcılar tarafından yapılır**)

9. Test Case ve Test Senaryosu Arasındaki Farklar Nelerdir?

Bir test senaryosu, test edilebilecek her bir özelliği tanımlar. Test koşulu veya Test olasılığı olarak da adlandırılır. Oturum açma özelliğini test etme test senaryosuna örnek olarak verilebilir.

Test Case ise daha özelde yazılımın belirli bir özelliğini test etmeyi amaçlar. Örneğin oturum açmak için yapılan test senaryosunda oturum açmak için geçerli değerler girmek test case'in konusuna girer. Yani Test senaryosu daha genel bir kavram iken test case daha dar bir tanımlamadır. Test senaryosu içinde bir çok test case bulunabilir. Test caseler pozitif ve negatif caselerden oluşabilir. Yani istenilen ve istenmeyen durumların test edilerek uygulamanın doğru tepkiler verdiğini ölçmek için bunlar yapılır.

10. Fonksiyonel Testler ve Fonksiyonel Olmayan Testler Arasındaki Farklar Nelerdir?

Fonksiyonel Testler (Functional Testing): Yazılımın her bir fonksiyonunun spesifikasyona (belirtilen şartlara) uygun olarak çalışmasını sağlayan bir test türüdür. Bu testler, sistemin ne yaptığını test eder.

Fonksiyonel Olmayan testler (Non-functional Testing): Bir yazılım uygulamasının işlevsel olmayan özelliklerini (performans, kullanılabilirlik ve güvenilirlik gibi) değerlendirmek için kullanılan bir test türüdür. Sistemin performansını değerlendirir.

11. Verification (Doğrulama) ve Validation (Geçerlilik) Arasındaki Farklar Nelerdir?

-- *Are you building it right? (Ürünü doğru üretiyor musun?)* Bu soru Verification kavramını anlatır.

-- *Are you building the right thing? (Doğru ürünü mü üretiyorsun?)* Bu soru ise Validation kavramını anlatır.

Verification (Doğrulama) Nedir?

Yazılım süreçlerinde verification işlemini yapan rolü yazılımcılar üstlenir. Çıktının, ürünün şartlarında belirtilen özelliklere uygun olup olmadığını doğrulamak için yapılan bir işlemdir. Yani yazılımcının, elde ettiği kod çıktısıyla dokümanda belirtilen standart ve gerekliliklere uygun olup olmadığının kontrolünü yaptığı işlemler bütünüdür. Yazılım süreçlerinde genellikle ilk olarak verification işlemi yapılır. Verification işlemi yapılarak daha sonra yapılacak olan validation işleminde gözden kaçan bir durum tespit edilebilir.

Validation (Geçerlilik) Nedir?

Yazılım süreçlerinde validation işlemini yapan rolü testçiler üstlenir. Çıktının veya ürünün, **müşterinin beklentilerini karşılayıp karşılamadığını denetlemek içindir.** Yani testçinin, yazılımcıdan gelen kodu müşterinin istediği asıl ürün ile karşılaştırıp doğru ürün üretilip üretilmediğinin kontrolünü yaptığı işlemler bütünüdür. Yazılım süreçlerinde genellikle validation işlemi, verification işleminden sonra ikinci işlem olarak yapılır. Validation işlemi yapılarak daha önce yapılan verification işleminde gözden kaçan bir durum tespit edilebilir.

12. Projede Test Yapmaya Ne Zaman Başlarız ve Bunun Faydaları Nelerdir?

Yazılım Geliştirme Yaşam Döngüsünün (SDLC) erken aşamalarında yazılım testleri başlamalıdır. Buna sola kaydırma adı verilir. Burada amaç yazılımdaki hataların olabildiğince erken bulunmasıdır. Bu durum, SDLC'nin gereksinim toplama ve tasarım aşamalarında sorunların tespit edilmesine ve ortadan kaldırılmasına yardımcı olur. Teste erken başlamak hataların miktarını ve nihayetinde yeniden işleme maliyetini azaltır. Yazılım testinin yedi prensiplerindne birine göre "Erken test, zaman ve para tasarrufu sağlar."

13. Kullanıcı Gereksinimleri (User Requirements) Açık Bir Şekilde Yazılmamışsa Testimizi Nasıl Yaparız?

1. Öncelikle elime geçen tüm belgeleri testi sağlıklı yapmak için kullanmaya çalışırım.
2. Benzer bir uygulamanın sürümü ile varsa mevcut programın eski sürümünü referans olarak kullanıp testler yapabilirim. Böylece benzer uygulamalardan faydalanarak kendi test donelerimi oluştururum.
3. Proje ekibi üyeleri ile uygulama hakkında konuşur ve onların fikirlerini alarak ilerlerim.
4. Yeterli tecrübeye sahipsem keşif testi yapabilirim. Tüm bunların sonunda ortaya bir proje dökümanı çıkarmış olurum.

14. Keşif Testi Nedir, Hangi Durumlarda Yapılır ve Kapsamı Nedir?

Keşif Testi (Exploratory Testing) test senaryolarının önceden oluşturulmadığı, testçilerin sistemi test anında kontrol ettiği bir tür yazılım testidir. Herhangi bir test senaryosuna bağlı kalmaksızın yazılımları serbestçe keşfederek buldukları bug'ları raporlamasına dayanır. Test uzmanlarının minimum planlama ve maksimum test uygulamasına katıldığı uygulamalı bir yaklaşımdır. Testin yürütülmesinden önce neyin test edileceğine dair fikirleri not edebilirler.

Test tasarımı ve test execution'ın eş zamanlı bir şekilde yürütüldüğü yazılım test yaklaşım türüdür. Keşif testi gereksinimlerin az veya muğlak olduğu durumlarda ve zaman konusunda baskı olduğu zamanlarda yapılır.

Sistemin yapması gereken temel fonksiyonları yerine getirip getirmediğini, Özellikle test edilmesi gereken yerlerin hayati derecede önemli olduğu ve farklı cihazlarda testler ile uygulamanın aynı tepkiyi verip vermediğini ölçmek bu testin kapsamı dahilindedir. Bu bağlamda sistemde en çok kullanılan yerlerde çökmenin olup olmadığı, geçersiz datalar ile sisteme giriş yapılıp yapılamadığı, yeni çıkacak versiyonun test edilmesi ve farklı tarayıcılar ile cihazlarda uygulamanın verdiği tepkiyi keşif testi ile yapmak mümkündür. Bu sebeple bu test türü rastgele ve herhangi bir şekilde yapılandırma yapmadan uygulanır.

15. Onaylama Testi ile Regresyon Testi Arasındaki Farkı Açıklayınız.

• **Onaylama testleri:** Bir hata çözüldükten sonra, hata nedeniyle başarısız olmuş tüm test senaryoları tekrar test edilebilir; bu testler yazılımın yeni versiyonunda yeniden koşturulmalıdır. Hatanın fonksiyonalite eksikliğinden kaynaklanması durumunda, yazılımın test edilmesi için yeni testler de yazılabilir. En azından hatadan kaynaklanan arızayı/arızaları veya eksikliği yeniden oluşturmak için gereken test adımları, yazılımın yeni versiyonunda tekrar koşturulmalıdır. **Onaylama testinin amacı, asıl hatanın başarıyla çözülüp çözülmediğini onaylamaktır.**

• **Regresyon testleri:** Kodun bir bölümünde yapılan bir değişikliğin (bir düzeltme veya başka bir değişiklik çeşidi olabilir) kazara kodun diğer bölümlerinin (aynı birim içinde, aynı sistemin diğer birimlerinde ve hatta diğer sistemlerde) davranışını olumsuz bir şekilde etkilemesi olasıdır. Değişiklikler, işletim sisteminin veya veritabanı yönetim sisteminin yeni bir versiyonu gibi ortamda yapılan değişiklikler de olabilir. Bu istenmeyen yan etkilere regresyon denir. **Regresyon testleri, bu gibi istenmeyen yan etkileri bulmak için yapılan testleri içerir.**

Not: Onaylama testleri ve regresyon testleri tüm test seviyelerinde (birim, entegrasyon, sistem ve kabul) yapılabilir.

16. Black-box, White-box ve Grey-box Testleri Arasında Farklar Nelerdir?

Black-box: Test edenin test edilen nesnenin iç yapısından haberdar olmadığı ve sistemin fonksiyonlarına odaklanan bir yazılım testi yaklaşımıdır.

White-box: test uzmanının test edilen nesnenin iç yapısından (kod vs.) haberdar olduğu bir yazılım testi yaklaşımıdır. White-box testinin alt kategorisi olan Unit (Birim) Test, daha ziyade developerlar tarafından yapılır. Ancak gerekli kaynak kodlarının verilmesi halinde QA'ler de bu testi yapabilme becerisine sahiptir. Unit Test, yaptığımız tüm fonksiyonların tek tek birbirinden bağımsız bir şekilde test edilmesidir. Entegrasyon testinden önce gelir ve yazılım testinin ilk seviyesidir.

Grey-box: Beyaz kutu testinde testin yapılması için gerekli olan kaynak kodlarının bilinmesi durumu ile buna gerek olmayan kara kutu testlerinin bir arada yapılması anlamına gelen bir test türüdür. Bu test türü tester'ın yaptığı ve developer'ın yükünü azaltan bir test türüdür. Ancak tester'ın bu testi yapabilmesi için developerın geliştirdiği kaynak kodlara sahip olması ve bunları kullanabilecek beceriye sahip olması gerekir. Bir tester gerekli datalar elinde olduğunda bu test türünü yapabilecek yetkinliğe sahiptir. Özetle bu test ile hem sistemin iç gereksinimleri hem de girdiler ile çıktıların uyumu test edilir. Gri kutu testinin amacı, zayıf kod yapısı veya uygulama kullanımından kaynaklanan hataları aramak ve tespit etmektir.

17. Static Test Nedir, Faydaları Nelerdir ve Dinamik Test İle Arasındaki Farklar Nelerdir?

Tanım: Kaynak kodun çalıştırılmasından önce gereksinimlerin analizine dayanan bir test türüdür. yazılımın veya diğer çalışma ürünlerinin manuel incelenmesine veya kodun veya diğer çalışma ürünlerinin araç kullanılarak değerlendirilmesine (statik analizlere) dayanır. Bu testte mantıksal tutarsızlıklar, cümle düşükşükleri, uyumsuzluklar, yazım ve dilgisi ile okunabilirlik denetimi gibi durumlar gerçekleştirilir.

Fayda: Statik test ile hataların erken bulunup maliyetin azaltılması sağlanabilir. Dinamik testlerde bulunamayacak hatalar bu aşamada bulunarak verimli bir sistem elde edilebilir. Tasarımdaki ve kodlamaki hatalar giderilmiş olur. **Statik test sayesinde sürdürülebilir bir kod yapısı elde edilmiş olur.**

Kapsam: Statik test yaparken sadece gereksinimlerin analizini incelemeyiz aynı zamanda koddaki yanlışlıklara da bakarız örneğin kodu çalıştırmadan değer atanmamış variable'lar varsa, kod gereksiz yere tekrarlanmışsa, ulaşılmayan kod varsa ya da değer atanmış olmasına rağmen kod kullanılmamışsa bunu statik test ile bulmak mümkündür. Manuel bir inceleme ile ifade edilen bu durumları bulabiliriz. Bunun dışında tasarım hataları, güvenlik açıkları ve kabul kriterleri bağlamında eksik test yazımlarını da yine statik test ile bulmak mümkündür.

18. Test Plan'ın Tanımı, içeriği ve faydaları Hakkında Ne Biliyorsunuz?

Tanım ve İçerik: Planlanan test işlemlerinin kapsamını, stratejisini, kaynaklarını ve zaman çizelgesini açıklayan bir doküman. Diğer hususların yanı sıra test öğelerini, test edilecek özellikleri, test görevlerini, her bir görevi kimin yapacağını, test uzmanının bağımsızlık derecesini, test ortamını ve test prosedürünü belirtir.

Kullanılacak tasarım metodolojileri ve giriş ve çıkış kriterlerinin yanı sıra bunların seçimine ilişkin gerekçeler ve acil durum planlaması gerektiren tehlikeler sınav hazırlık prosedürünün bir kaydıdır. Proje ve test hazırlığı devam ettikçe daha fazla bilgi elde edilebilir ve test planına daha fazla ayrıntı eklenebilir. Test planlaması, ürünün kullanım ömrü boyunca devam eden bir süreçtir.

Fayda: QA ekipleri dışındaki kişilerin (geliştiriciler, işletme yöneticileri, müşteriye dönük ekipler) web sitesinin veya uygulamanın tam olarak nasıl test edileceğini anlamalarına yardımcı olurlar. QA mühendislerinin test faaliyetlerini yürütmeleri için net bir kılavuz sunarlar. Test kapsamı, test tahmini, strateji vb. hususları detaylandırırlar. Tüm bu bilgilerin tek bir belgede toplanması, yönetim personeli tarafından incelenmesini veya diğer projeler için yeniden kullanılmasını kolaylaştırır.

19. Test Özet ve Test İlerleme Raporlarının İçeriği Hakkında Bilgi Veriniz?

Test raporlamasının amacı, hem test faaliyetleri sırasında hem de tamamlandıktan sonra (örneğin, bir test seviyesi) test faaliyetleri hakkındaki bilgileri sentezlemek ve paylaşmaktır. Bir test faaliyeti sırasında oluşturulan bir test raporu, test ilerleme raporu olarak bilinirken, bir test faaliyetinin sonunda hazırlanan bir test raporu, test özet raporu olarak bilinir.

- Test raporunun içeriği, proje ortamına ve raporun hedeflenen okuyucu kitlesine bağlı olarak değişir.

20. Risk Temelli Test Deyince Aklınıza Ne Geliyor?

Öncelikle riskin tanımı yapmak gerekir. Bu bağlamda gelecekte olumsuz sonuçlara sebep olacak bir olayın ya da durumun gerçekleşme ihtimaline risk denir. Bu test türü, risk olasılığına dayanan yazılım testidir. Bazı durumlarda yazılım karmaşık olabilir, iş kritikliği yapmak zor olabilir ya da üründeki bazı özelliklerin kullanım sıklığı ile ürünün hassasiyetinden kaynaklı potansiyel sorunlar meydana gelebilir. Bu ve benzer, durumların analiz edilmesi ve buna dair gerekli önlemlerin alınması gereklidir. Bu doğrultuda risk tabanlı test, yazılımın daha önemli ve kusurlu olması muhtemel yönlerinin ve işlevlerinin test edilmesine öncelik verir.

Projeye katkısı bağlamında ise şunları söyleyebiliriz: Problemlerin olabildiğince erken aşamada fark edilmesine, testlerin daha az sorun ile koşulmasına, olası riskleri en aza indirme ve projenin zamanında bitiriilmesine katkı sağlamasına imkan tanır.

Risk bazlı testler ile kullanılacak test teknikleri, koşulacak testlerin seviyeleri ve çeşitleri, koşulacak testlerin kapsamı, Kritik hataların mümkün olduğunca erken belirlenmesi için

testlerin önceliklendirilmesi, riski azaltmak için testlere ek olarak yapılabilecek potansiyel aktivitelerin belirlenmesi gibi amaçlar gözetilir.

21. Alfa ve Beta Testleri hakkında Bilgi Verip Farklarını İzah Ediniz

Alfa ve beta testleri genellikle ürünün piyasaya sürülmesinden önce potansiyel veya mevcut kullanıcılardan geri bildirim almak isteyen **ticari paket yazılımın** (COTS) geliştiricileri tarafından kullanılır.

Alfa testleri yazılım geliştiren firmanın kendi ortamında sadece yazılım geliştirme ekibi tarafından değil, potansiyel-mevcut müşteriler ya da bağımsız bir test ekibi tarafından gerçekleştirilir. **Beta testleri** ise potansiyel-mevcut müşteriler ya da operatörler tarafından kendi ortamlarında yapılır. Alfa testleri, Beta testlerinden önce yapılabildiği gibi hiç alfa testi yapılmadan sadece beta testleri de yapılabilir.

Alfa ve beta testlerinin hedeflerinden biri, kullanıcılarda sistemin normal koşullar altında ve operasyonel ortamlarda asgari zorluk, maliyet ve riskle ulaşmak için kullanabileceklerine dair güven oluşturmaktır. Diğer bir hedef ise sistemin kullanılacağı şartlar ve ortamlar ile ilgili hataların belirlenmesidir.

22. Test uzmanları Genellikle Hangi Hataları Yapmaya Eğilimlidir.

1. İletişim kuramadan çekinme,
2. Soru sormaktan korkma ya da çekinme,
3. Kapsamı ve gereksinimleri tam olarak anlamadan teste başlama,
4. Yetersiz içerikli hata raporları üretme,
5. Test senaryoları yazılırken bazı gereksinimlerin göz ardı edilmesi,
6. Herhangi bir planlamanın olmadan çalışmaya başlama,
7. Yanlış pozitifler ve negatifler.

23. Test Bağımsızlığının Faydaları ve Avantajları Nelerdir?

-Farklı deneyimleri, teknik bakış açıları ve önyargıları nedeniyle bağımsız test uzmanlarının farklı hata türlerini fark etme olasılığı geliştiricilere göre daha yüksektir.

-Bağımsız bir test uzmanı, sistem spesifikasyonu ve uygulaması sırasında paydaşlar tarafından yapılan varsayımları doğrulayabilir, bunlara meydan okuyabilir veya bunları çürütebilir.

-Bir tedarikçinin bağımsız test uzmanları, kendilerini işe alan şirketin (siyasi) baskısı olmaksızın test edilen sistem hakkında dürüst ve objektif bir şekilde rapor verebilir.

24. Bir Bug Raporunda olmazsa olmaz hangi başlıklar vardır?

1. Başlık (Title)
2. Yeniden üretme adımları (Steps to reproduce)
3. Beklenen sonuç (Expected result)
4. Gerçekleşen sonuç (Actual result)
5. Priority (Öncelik)
6. Ekran görüntüsü veya video

25. Test bağımsızlığının potansiyel dezavantajları nelerdir?

-Developer ekibinden izole olma durumu iki taraf arasındaki işbirliği eksikliğine, geliştirme ekibine girdi sağlamada gecikmelere veya geliştirme ekibiyle düşmanca bir ilişkiye neden olabilir.

- Developerlar kalite noktasında sorumluluk duygusunu kaybedebilir.
- Bağımsız test uzmanları developerlar tarafından bir engel olarak görülebilir.

26. Test teknikleri ve test araçları arasındaki farklar nelerdir?

Herhangi bir test tekniğinin amacı, test koşullarının, senaryolarının ve verilerinin tanımlanmasına yardımcı olmaktır.

- Kara Kutu Teknikleri (Eşdeğerlik Bölümleme)
- White-box Yöntemleri (İfade Kapsamı)
- tecrübeye dayalı test teknikler (Hata tahmini)

- Yazılım testi bağlamında bir tool ise, planlama, gereksinimler, derleme oluşturma, test yürütme, hata izleme ve test analizi gibi bir veya daha fazla test faaliyetini destekleyen bir üründür.

Test Yönetim Araçları (Google Sheets-Trello-Jira)
Test Otomasyon Araçları (Selenium Webdriver - Cypress - Robot Framework)
Performans Test Araçları (Jmeter - HP Loadrunner)
API Test Araçları (Postman - Soap UI - Rest Assured)

27. Denklik Sınıfı (equivalence partitioning) ve Sınır Değer Analizi (Boundary Value) Test Tekniklerini Arasındaki Farklar Nelerdir?

Denklik sınıfı belirli bir bölümün tüm üyelerinin aynı şekilde ele alınması, beklenilecek şekilde verilerin bölümlere yani denklik sınıflarına ayrılması anlamına gelir.

- Sınır değer analizi (denklik sınıfının devamı niteliğindedir. Ancak yalnızca bölüm sıralı olduğunda ve sayısal veya sıralı verilerden oluştuğunda kullanılabilir) Bir bölümün sınır değerleri, minimum ve maksimum değerleridir (veya başlangıç ve son değerleridir).
- Denklik sınıfının sınırlarına yakın davranışların yanlış olma olasılığı, bölümlerin içindeki davranışlardan daha olasıdır.

28. Random/Monkey Testing Nedir Ne Zaman İhtiyaç Duyarız?

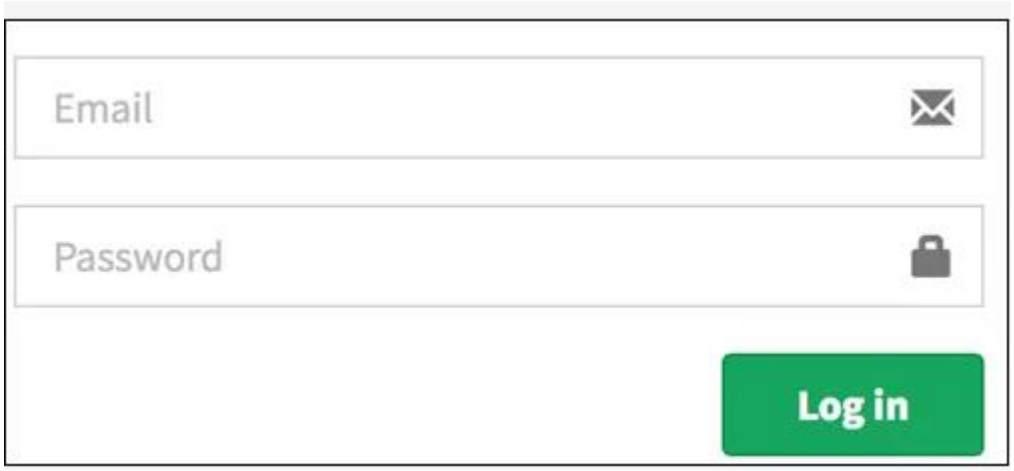
Random testin bir diğer adı Monkey yani maymun testidir. Bu tür testlerde veriler, genellikle bir araç veya otomatik bir sistem kullanılarak rastgele oluşturulur. Sistem rastgele oluşturulan bu girdi ile test edilir ve sonuçlar buna göre değerlendirilir. Bu testler daha az güvenilirdir. Bu nedenle genellikle sistemin kötü sonuçlara dayanıp dayanmayacağını belirlemek için yeni başlayanlar tarafından gerçekleştirilir.

29. Karar Tablosu (Decision Table) Testi Nedir Ne Zaman Kullanırız?

Karar tablosu testi, özellikleri kurallar veya neden-sonuç kombinasyonları şeklinde olan sistemleri test etmek için kullanılan bir test tekniğidir. Bir karar tablosunda, girdiler bir sütunda listelenir ve sonuçlar aynı sütunda ancak girdilerin altında listelenir. Tablonun geri kalanında, üretilen çıktıları tanımlamak için girdi kombinasyonları incelenir. Daha iyi test kapsamı elde etmemize imkan sağlar. **Neden-Sonuç tablosu** ismini de verebiliriz.

-- Örnek bir tablo ile ne denilmek istendiğine bakalım:

Kullanıcı adı ve şifre girerek login olacağımız bir sistem düşünelim.



Şu şekilde bir karar tablosu çıkarabiliriz;

KOŞULLAR	KURAL 1	KURAL 2	KURAL 3	KURAL 4
Kullanıcı Adı (T/F)	F	T	F	T
Şifre (T/F)	F	F	T	T
Çıkış (E/H)	E	E	E	H

* T= TRUE, F= FALSE , E= EVET, H= HAYIR

30. Negatif ve Pozitif Test Arasındaki Farklar Nelerdir? Projenizden Örnek Verebilir misiniz?

Negatif test senaryoları, yazılımın beklenmeyen durumlarda nasıl davranacağını kontrol etmek için kullanılır. Bu testler, kullanıcının yanlış girdiği veriler, sistem arızaları, sistem kaynaklarına erişim yetkisi olmayan kullanıcılar gibi beklenmeyen durumları içerebilir. Örnek olarak, bir kullanıcının yanlış şifre girerek bir hesaba erişmeye çalışması veya bir uygulamanın bir veritabanına bağlanmak için hatalı bir URL kullanması gibi senaryolar verilebilir.

Pozitif test senaryoları ise yazılımın doğru şekilde çalıştığını doğrulamak için kullanılır. Bu testler, yazılımın beklenen davranışlarını kontrol etmek için tasarlanmıştır. Örnek olarak, bir kullanıcının doğru şifre kullanarak bir hesaba erişmeye çalışması veya bir uygulamanın doğru URL kullanarak bir veritabanına bağlanması gibi senaryolar verilebilir.

Örnek bir proje için, bir e-ticaret web uygulamasının test senaryolarını ele alalım:

Negatif test senaryoları:

- Kullanıcının hesabına hatalı şifreyle giriş yapmaya çalışması
- Kullanıcının ödeme sayfasına geçmeden önce stokta olmayan bir ürünü sepete eklemesi
- Kullanıcının ödeme işlemi sırasında kredi kartı bilgilerini eksik veya hatalı girmesi

Pozitif test senaryoları:

- Kullanıcının doğru şifreyle hesabına giriş yapması
- Kullanıcının sepete ürün ekleyip, ödeme sayfasına geçmesi ve başarılı bir şekilde ödeme yapması
- Kullanıcının sipariş durumunu takip edebilmesi ve doğru ürünü doğru zamanda teslim alması

31. Waterfall Modeli Hakkında Ne Biliyorsunuz?

Waterfall modeli, yazılım geliştirme sürecinde sık kullanılan bir yöntemdir. Bu model, tüm sürecin doğrusal olarak işlendiği ve her aşamanın tamamlanması gerektiği sırayla çalışılan bir yöntemdir.

Bu modelde süreç aşağıdaki aşamalardan oluşur:

Gereksinim Analizi: Bu aşamada, müşterilerin talepleri ve ihtiyaçları toplanır ve analiz edilir. Gerekli sistem özellikleri, işlevler ve kullanılacak teknolojiler belirlenir.

Tasarım: Bu aşamada, önceki aşamada belirlenen gereksinimler doğrultusunda sistem mimarisi oluşturulur. Bu aşamada, veritabanı yapısı, kullanıcı arayüzü, veri akışı, işlevsellik gibi konular belirlenir.

Geliştirme: Bu aşamada, yazılım kodlaması gerçekleştirilir ve sistem işlevselliği oluşturulur.

Test Etme: Yazılım testlerinin yapılacağı aşamadır. Bu aşamada, yazılımın istenen özelliklerini yerine getirdiği ve istikrarlı çalıştığından emin olmak için test edilir.

Uygulama: Yazılımın hedef sisteme uygulanması ve kullanıma hazır hale getirilmesidir.

Bakım: Bu aşama, yazılımın işlevselliğini ve performansını optimize etmek için yapılan düzeltme ve iyileştirmeleri içerir.

32. SDLC'nin Son Aşaması Olan Bakım Aşamasındaki Aktiviteler Nelerdir?

1. Sorun Giderme: Bakım aşamasında, yazılımın kullanımı sırasında ortaya çıkan hatalar ve sorunlar giderilir. Bu işlem, yazılımın doğru şekilde çalışmasını sağlamak için önemlidir.

2. Değişiklik Yönetimi: Bakım süreci, yazılımda yapılacak değişiklikleri yönetmek için kullanılır. Bu değişiklikler, müşteri geri bildirimleri, güvenlik açıkları veya yeni iş gereksinimleri gibi faktörlere bağlı olarak olabilir.

3. Güncelleme: Bakım süreci, yazılımın güncel kalmasını sağlamak için yapılan güncellemeleri içerir. Bu güncellemeler, yeni teknolojiler, iş gereksinimleri veya güvenlik açıkları gibi faktörlere bağlı olarak yapılabilir.

4. Dokümantasyon Güncelleme: Bakım süreci, yazılımın dokümantasyonunu güncellemek için kullanılır. Bu, kullanıcı kılavuzları, işletme talimatları ve yardım dosyalarını içerebilir.

Bakım süreci, yazılımın kullanıcılarına en iyi deneyimi sunmak için önemlidir. Yazılımın bakımı, kullanıcıların yazılımı kullanırken karşılaşabilecekleri sorunları minimize etmeye yardımcı olur ve yazılımın işlevselliğini ve performansını artırır.

33. Waterfall ile Agile Arasındaki Farkları İzah Eder Misiniz?

Waterfall ve Agile, yazılım geliştirme sürecinde iki farklı yöntemdir. Waterfall, eski bir yöntemdir ve geleneksel olarak kullanılmıştır, ancak Agile daha yeni bir yaklaşımdır ve daha modern yazılım geliştirme süreçleri için uygundur.

Waterfall, yazılım geliştirme sürecinin lineer bir yol izlediği bir yöntemdir. Bu, her aşamanın tamamlanması gerektiği ve bir sonraki aşamanın başlayamayacağı anlamına gelir. Yani, bir aşama tamamlandıktan sonra, sonraki aşamaya geçilir. Örneğin, gereksinimler analizi,

tasarım, kodlama, test etme ve sürüm aşamaları gibi belirli aşamaları izler. Bu yöntem, genellikle büyük ölçekli projelerde kullanılır.

Diğer taraftan, Agile, yazılım geliştirme sürecinin daha esnek ve dinamik olduğu bir yöntemdir. Bu yöntemde, tüm aşamalar eş zamanlı olarak gerçekleştirilir ve yazılım geliştirme süreci bir döngü şeklinde devam eder. Her döngü, bir önceki döngüden öğrenilen bilgilere dayalı olarak iyileştirilir. Agile yöntemi, küçük ve orta ölçekli projelerde daha yaygın olarak kullanılır.

İki yöntem arasındaki temel farklar şunlardır:

Waterfall yöntemi, belirli bir sırayla aşamaları izlerken, Agile yöntemi, aşamaları eşzamanlı olarak gerçekleştirir.

Waterfall yöntemi daha az esnektir ve projeler için uzun süreli bir planlama yapılmasını gerektirirken, Agile yöntemi daha esnektir ve projelerin ihtiyaçlarına göre hızlı bir şekilde değiştirilebilir.

Waterfall yöntemi, bir sonraki aşamaya geçilmeden önce bir aşamanın tamamlanması gerektiği için geri dönülmesi zordur, ancak Agile yöntemi, geri dönülebilir ve her döngüde yapılan hatalar ve yanlış anlamalar düzeltilebilir.

Waterfall yöntemi, değişiklikleri ele almak için daha fazla zaman gerektirirken, Agile yöntemi daha hızlı değişiklikler yapmak için uygundur.

Özetlemek gerekirse, Waterfall yöntemi geleneksel bir yaklaşımken, Agile yöntemi daha yeni ve modern bir yaklaşımdır. Her iki yöntem de farklı projeler için farklı avantajlar sunar. Waterfall yöntemi, büyük ölçekli projeler için daha uygundurken, Agile yöntemi, küçük ve orta ölçekli projeler için daha uygundur.

34. En İyi Test Case'i Ortaya Çıkarmak İçin Hangi Noktalara Dikkat Etmeliyiz?

- Son kullanıcının bakış açısından test senaryoları yazarım
- Test adımlarını herkesin kolayca takip edebileceği basit bir şekilde yazarım
- Test senaryolarını yeniden kullanılabilir formatta dizayn ederim.
- Priority son derece önemlidir. Bu bağlamda önceliğe dikkat ederim.
- Bir test durumu açıklaması, test verileri, beklenen sonuç, ön koşul, son koşul verilerine test case'imde yer veririm.
- Geçerli test senaryoları ile birlikte geçersiz test senaryoları da yazarım.
- Uygun isimlendirmeler yaparak anlaşılır olmasını sağlarım
- Test senaryolarını düzenli olarak gözden geçirir ve gerekmesi halinde güncellerim.

NOT: Bütün bu maddelerden anşlılacağı üzere özetle anlaşılır, sade ve girdiler ile çıktıların net olduğu bir test case yazımı ideale yakın olacaktır.

35. Derleme (build) ile Yayınlama (release) Arasındaki Farklar Nelerdir?

Build, programın test edilmesi için geliştiriciler tarafından test ekibine sunulan yürütülebilir bir dosyadır. Program düzgün çalışana kadar birkaç onarım ve test aşamasından geçer. Program kararlı ve son kullanıcılar için hazır hale geldikten sonra piyasaya sürülür.

Release ise, test ekibi tarafından onaylandıktan sonra son kullanıcıların kullanımına sunulan yüklenebilir bir programdır. Herhangi bir program müşteriye yayınlandığında, hala bekleyen kusurların sayısını, kapsanan kullanıcı hikayelerini, değişiklik ihtiyaçlarını ve yayın sürümünü açıklayan sürüm notları eklenir.

36. Test Datası Nedir ve Neden Önemlidir?

Test datası, yazılım testleri için kullanılan girdi verileridir. Bu veriler, yazılımın belirli bir işlevselliği veya senaryoyu test etmek için kullanılır. Test datası, bir test senaryosunun gerçek dünya koşullarını simüle etmek için kullanılır ve yazılımın gerçek bir ortamda nasıl davranacağını gözlemlememizi sağlar.

Test datası önemlidir çünkü yazılımın doğru çalışması için doğru girdi verilerinin kullanılması gerekir. Yanlış verilerle yapılan testler sonuçları yanıltıcı olabilir ve yazılımın gerçek dünya koşullarında nasıl çalışacağı hakkında yanıltıcı bilgiler verebilir. Örneğin, bir hesaplamayı test eden bir yazılımda, test datası olarak sadece küçük sayılar kullanırsak, yazılımın büyük sayılarla nasıl başa çıkacağı hakkında bilgi edinemez ve büyük sayılarla çalışırken beklenmedik sonuçlar alabiliriz.

UI testleri için bir örnek test datası verelim: Örneğin sisteme login işlemi yapmak istiyorsunuz size geçerli kullanıcı adı ve şifre verilmelidir. Bunlar sizin sisteme login olmanız için kullancağınız geçerli test datalarıdır.

37. Kalite kontrol (Quality Control) ve kalite güvencesi (Quality Assurance) arasındaki fark nedir?

Kalite güvence (QA) ve kalite kontrol (QC), yazılım testinin hedefleri, odak noktaları ve metodolojileri bakımından farklılık gösteren iki önemli bileşendir.

Kalite güvencesi, yazılım geliştirme sürecinin tanımlanmış standartlara ve süreçlere bağlı kalmasını ve son ürünün gerekli kalite gereksinimlerini karşılamasını sağlayan proaktif bir tekniktir. QA, geliştirme süreci boyunca sorunları keşfedip ele alarak ve yazılımın müşteri ihtiyaç ve beklentilerini karşıladığını doğrulayarak hataları önlemeyi amaçlar. Bu, yazılımın amaçlanan standartlara uygun olarak üretildiğini garanti etmek için test stratejilerinin belirlenmesi ve test prosedürlerinin uygulanması gibi eylemleri içerir.

Buna karşılık kalite kontrol, yazılım ürününün gerekli kalite gereksinimlerini karşılamasını sağlamanın reaktif bir yoludur. Kalite kontrol, programdaki hataları bulmak için test senaryolarının yürütülmesini ve sonuçların değerlendirilmesini içerir. Programın kalite standartlarını karşıladığını garanti etmek için işlevsel, performans, güvenlik ve diğer test türleri gerçekleştirilir. Kalite kontrol ayrıca iyileştirme alanlarını keşfetmek için yazılım metriklerinin ölçülmesini ve değerlendirilmesini de içerir.

Özetlemek gerekirse, kalite güvencesi geliştirme sürecinin yüksek kaliteli yazılım üretmesini sağlamakla ilgiliyken, kalite kontrolü kalite gereksinimlerini karşıladığını garanti etmek için yazılım çıktısını test etmek ve gözden geçirmekle ilgilidir. QA proaktif bir yöntemken, QC reaktif bir yöntemdir.

NOT: Proaktif, öngörüp önlem alma anlamına gelir. Bu davranış tarzı, gelecekte ortaya çıkabilecek sorunları önceden tahmin etmek ve buna göre önlemler almak anlamına gelir. Reaktif ise, sorunların meydana gelmesi sonrasında müdahale etmek anlamına gelir. Bu yaklaşım, acil durumlarda hızlı tepki verilmesi gereken durumlarda kullanılır.

Özet olarak, proaktif yaklaşım öngörüp önlem alma anlamına gelirken, reaktif yaklaşım ise sorunlar ortaya çıktıktan sonra müdahale etme anlamına gelir.

38. Test Execution Hangi Durumlarda Durdurulur?

Ciddi bir sorun, önemli bir işlevsellik parçasının veya önemli bir sistem bileşeninin tamamen bozulduğunu fark ettiğim zaman test yürütümesini durmam gerekir. Örneğin, bir modüldeki bir hata nedeniyle diğer modülleri test edemiyorsam testi yürütmeyi durdururu. Bu tarza kritik hatalar, applikasyon üzerinde olumsuz etkisi olan hatalardır.

1. Gmail Uygulamasındaki "Oturum Aç" düğmesi çalışmıyor ve Gmail kullanıcıları hesaplarına erişemiyor.
2. Bir tüketici bir bankacılık web sitesindeki para transferi düğmesine tıkladığında bir hata mesajı görüntüleniyor.

39. Smoke ve Sanity Test Arasındaki Fark Nedir?

Smoke testi, geliştirme ekibinden elde edilen yapının test edilebilir olup olmadığını belirlemek için yapılır. "0. Gün" kontrolü olarak da bilinir. "Yapı seviyesinde" gerçekleştirilir. Kritik işlevler çalışmadığında veya önemli kusurlar henüz çözülmediğinde sadece tüm programı değerlendirerek test zamanının boşa harcanmasını önlemeye yardımcı olur.

Sanity testi, uygulamanın temel işlevselliğini çok fazla derinlemesine incelemeden doğrulamak için sürüm sürecinde gerçekleştirilir. Bazen regresyon testinin bir alt kümesi olarak da adlandırılır. ***"Sürüm"*** seviyesinde tamamlanır.

İki test arasındaki farkı E- commerce projesi üzerinden örnek bir senaryo ile de izah edelim:

Örnek Test Senaryosu: Ana sayfanın yüklenmesi (Smoke)

Step 1: Uygulama ana sayfası açılır.

Step 2: Sayfanın yüklenmesi beklendiği kadar hızlı mı gerçekleştiği kontrol edilir.

Step 3: Sayfa yüklemesi başarılı olduysa, sayfa üzerindeki ana işlevler (ürün kategorileri, arama kutusu, favoriler listesi) kontrol edilir.

Step 4: Tüm ana işlevlerin çalıştığı doğrulanır.

Örnek Test Senaryosu: Ürün Arama (Sanity)

Step 1: Uygulama ana sayfası açılır.

Step 2: Arama kutusuna bir ürün adı girilir.

Step 3: Arama sonuçları sayfası açılır.

Step 4: Sayfada gösterilen ürünlerin, arama sorgusuna göre filtrelenip filtrelenmediği kontrol edilir.

Step 5: İstenilen ürün bulunduysa, ürün sayfasına geçilir ve sayfadaki özelliklerin doğru olduğu doğrulanır.

40. Ad-Hoc Test Nedir?

Ad-hoc testler resmi testlerle taban tabana zıttır. Bir tür gayri resmi testtir. Ad hoc test, test uzmanlarının herhangi bir dokümantasyon veya test tasarım metodolojisi gözetmeksizin programı rastgele test etmesini içerir. Bu test genellikle test uzmanları test edilen program hakkında kapsamlı bir anlayışa sahipse gerçekleştirilir. Test uzmanları, herhangi bir test senaryosu veya iş gereksinimi dokümantasyonu olmadan programı rastgele test eder.

41. Entegrasyon Test Yaklaşımları Nelerdir?

Entegrasyon testi, bir yazılım sisteminin çeşitli modüllerinin, bileşenlerinin ve alt sistemlerinin etkileşimini değerlendiren yazılım testinde kritik bir adımdır. Entegrasyon testini gerçekleştirmek için çok sayıda yöntem vardır, örneğin:

Büyük Patlama Entegrasyon Testi: Bu yöntem, tüm bileşenlerin üretildikten sonra birlikte test edilmesini gerektirir. Bu strateji ile test ekibi test yapmadan önce tüm bileşenlerin hazır olmasını bekler. Bu yöntem test süreci boyunca zaman kazandırabilir, ancak hataları tespit etmek ve düzeltmek zor olabilir.

Yukarıdan Aşağıya Entegre Test: Bu yöntemde önce en üst düzey modüller test edilir, ardından bunları destekleyen alt düzey modüller test edilir. Bu yöntem, alt düzey modüllerin

davranışını taklit etmek için taslakların veya sürücülerin kullanılmasını gerektirir. Bu faydalı olabilir.

Aşağıdan Yukarıya Entegrasyon Testi: Bu yöntem, yukarıdan aşağıya entegrasyon testinin tersidir; test alt düzey modüllerle başlar ve daha üst düzey modüllere doğru ilerler. Bu yöntemde, hataların erken tespit edilmesine yardımcı olabilecek taslaklar veya sürücüler yerine gerçek modüller kullanılır. Bu strateji ile kodun veya modüllerin küçük parçaları bir seferde test edilir ve ardından aşamalı olarak sisteme entegre edilir. Bu strateji, test ekibinin kusurları erkenden bulmasına ve tüm sistemi birleştirmeden önce bunları düzeltmesine olanak tanır.

Sandviç Entegrasyon Testi: Bu test yöntemi yukarıdan aşağıya ve aşağıdan yukarıya entegrasyonu karıştırır. Önce en üst düzey modüller test edilir, ardından alt düzey modüller ve son olarak tekrar en üst düzey modüller test edilir. Bu yöntem zaman alıcıdır, ancak kapsamlı sonuçlar sunar.

NOT: Bu yaklaşımlardan hangisi projeye uygunsa o seçilerek uygulanır.

42. Yazılım testinde, global ve lokal testler nedir ne anlama gelir.

Global testler, yazılımın farklı diller, kültürler ve bölgeler için teknik veya işlevsel sorunlar olmadan uygun şekilde adapte edilebileceğinden emin olmak için yapılan testlerdir. Global testler aynı zamanda **uluslararasılaştırma** testleri olarak da adlandırılır. Global testlerin amacı, yazılım ürününün birden fazla dil ve kültürel gelenekleri (tarih ve saat formatları, sayı formatları ve para birimi simgeleri gibi) destekleyebilmesini sağlamaktır.

Lokal testler ise, yazılım ürününün belirli bir bölge için uyarlandığından emin olmak için yapılan testlerdir. Lokal testlerin amacı, yazılım ürününün hedef pazar için kültürel olarak uygun ve ilgili olduğundan ve bu bölgede kullanılan belirli dil, biçimlendirme ve diğer geleneklerle doğru şekilde çalıştığından emin olmaktır.

Özetle, global testler yazılım ürününün dünya genelinde farklı kültürler ve bölgeler için kolayca adapte edilebilmesini sağlamaya odaklanırken, lokal testler yazılım ürününün belirli bir hedef pazar için tamamen işlevsel ve kültürel olarak uygun olduğundan emin olmaya odaklanır.

43. Ne kadar Test Yapmak Gerekir?

Bu sorunun kesin bir cevabı yoktur. Bu doğrultuda test yapma sayısınız mutlak bir sınırı olduğu söyleyemeyiz. Burada projeye odaklı aplikasyona en büyük zararı verebilecek en olası durumları veya programın en sık kullanılan alanlarını belirlemek en doğru uygulama olacaktır. Burada **risk tabanlı test** kullanabiliriz. Böylece zamanımızı ve enerjimizi en kritik bölümlere odaklayabiliriz. Testler, ayrıca bir uygulamanın durumu veya sağlığı hakkında paydaşların

programı serbest bırakma veya test için ek zaman harcama konusunda eğitimli bir seçim yapmaları için yeterli bilgi sunmalıdır.

44. Gereksinimler Sık Sık Değiştiğini Varsayalım Tavrınız Ne Olurdu?

Gereksinimlerin amacı değişmez ancak uygulamada bir takım teknik detaylar değişebilir. Bu durumla başa çıkmak için öncelikle uygulamanın amacına odaklanmak gerekir.

Öncelikle gereksinimlerin ne kadar değiştiğini belirlemek gerekir. Bu bağlamda değişikliklerin kapsamı ve önemini belirlemek gerekecektir.

Müşteriler veya kullanıcılarla düzenli olarak iletişim halinde kalmak ve ihtiyaçlarına odaklanmak önemlidir. Ayrıca, proje paydaşlarıyla işbirliği yaparak, yeni gereksinimleri mümkün olan en iyi şekilde yönetebilirsiniz.

Değişen ihtiyaçların önceliğini belirleyin ve bu önceliklere göre işleri planlayın. Bu, müşteri veya kullanıcıların en önemli ihtiyaçlarının karşılanmasını sağlar.

Yapılan değişikliklerin doğru çalıştığından emin olmak için sık sık test etme yapılmalıdır. Böylece değişikliklerin herhangi bir istenmeyen etkisi olup olmadığı kontrol edilebilir.

Değişen gereksinimlere esnek bir şekilde yanıt verin. Projenin yönünü değiştirmek veya yeni özellikler eklemek gerektiğinde, bu değişikliklere hızlı ve etkili bir şekilde uyum sağlayabilecek bir ekip kurun.

Sonuç olarak, değişen gereksinimlerle başa çıkmak için esnek ve açık fikirli bir yaklaşım benimsemek önemlidir. Müşterilerin veya kullanıcıların ihtiyaçlarının değişebileceği ve bu değişikliklere hızlı bir şekilde yanıt vermek gerektiği bilinmelidir.

45. Requirement Traceability Nedir ve Neden Önemlidir?

Requirement Traceability (Gereksinim izlenebilirliği), yazılım testinde gereksinimlerin karşılandığını doğrulamak için tasarlanan testlerle ilişkilendirilmesini içeren bir süreçtir. Gereksinim izlenebilirliğinin amacı, tüm gereksinimlerin test edilmesini ve test sırasında bulunan herhangi bir kusurun orijinal gereksinime kadar izlenebilmesini sağlamaktır.

Gereksinim izlenebilirliği için adımlar şu şekilde sıralanabilir:

Gereksinimlerin tanımlanması: Gereksinim izlenebilirliğinin ilk adımı, test edilen yazılım sistemi için tüm gereksinimleri belirlemektir.

Bir izlenebilirlik matrisi oluşturma: İzlenebilirlik matrisi, gereksinimleri bunları doğrulamak için tasarlanmış test senaryolarına bağlayan bir belgedir. Her gereksinime benzersiz bir tanımlayıcı atanır ve her test senaryosu doğruladığı gereksinim(ler)e bağlanır.

Test senaryolarının yürütülmesi: Test senaryoları belirlendikten ve gereksinimlere bağlandıktan sonra, yazılım sisteminin belirtilen gereksinimleri karşıladığını doğrulamak için çalıştırılabilirler.

Hataların izlenmesi: Test sırasında, düzeltilmesi gereken hatalar keşfedilebilir. Geliştirme ekibi bu hataları orijinal gereksinime kadar takip ederek, hata giderildikten sonra gereksinimin karşılandığından emin olabilir.

İzlenebilirlik matrisinin güncellenmesi: Test süreci ilerledikçe, izlenebilirlik matrisi gereksinimlerde veya test senaryolarında yapılan değişiklikleri veya güncellemeleri yansıtacak şekilde güncellenmelidir.

Gereksinim izlenebilirliği, tüm gereksinimlerin test edilmesini sağlamaya yardımcı olur ve bu da yazılım sisteminin genel kalitesini artırabilir. Ayrıca, kusurları kaynağına kadar izlemenin bir yolunu sağlar, bu da kalıpları belirlemeye ve gelecekteki projeler için geliştirme sürecini iyileştirmeye yardımcı olabilir.

46. Bir Requirement Traceability Örneği'nde Neler Olmalıdır?

İzlenebilirlik matrisi olarak da bilinen gereksinim izleme çizelgesi, yazılım testlerinde gereksinimler ve test senaryoları arasındaki ilişkiyi izlemek için kullanılan bir araçtır.

Bir Requirement Traceability Örneği'nde aşağıdaki tabloda görüleceği gibi Requirement ID, Description, Test Case ID, Test Case Description ve Test Result olmalıdır.

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Test Result
REQ-001	Sistem, kullanıcıların oturum açmasına izin vermelidir	TC-001	Oturum açma sayfasının görüntülendiğini doğrulayın	Pass
REQ-001	Sistem, kullanıcıların oturum açmasına izin vermelidir	TC-002	Kullanıcının kullanıcı adını ve parolasını girebildiğini doğrulayın	Pass
REQ-001	Sistem, kullanıcıların oturum açmasına izin vermelidir	TC-003	Kullanıcı yanlış bir parola girdiğinde bir hata mesajının görüntülendiğini doğrulayın	Fail
REQ-002	Sistem, kullanıcıların ürün aramasına izin vermelidir	TC-004	Arama sayfasının görüntülendiğini doğrulayın	Pass
REQ-002	Sistem, kullanıcıların ürün aramasına izin vermelidir	TC-005	Kullanıcının bir arama terimi girebildiğini doğrulayın	Pass
REQ-002	Sistem, kullanıcıların ürün aramasına izin vermelidir	TC-006	Arama sonuçlarının görüntülendiğini doğrulayın	Pass

Bu örnekte, ilk sütun gereksinim kimliğini, ikinci sütun gereksinimi, üçüncü sütun test senaryosu kimliğini, dördüncü sütun test senaryosunu ve beşinci sütun test senaryosunun sonucunu listeler.

Her gereksinim bir kez listelenir ve onu doğrulamak için tasarlanmış tüm test senaryolarına bağlanır. Grafik, hangi test senaryolarının yürütüldüğünü ve her bir test senaryosunun sonuçlarını gösterir. Bir test senaryosu başarısız olursa, hangi gereksinime bağlı olduğunu belirlemek kolaydır ve geliştirme ekibinin sorunu hızla ele almasına olanak tanır. Grafik, test süreci boyunca gereksinimlerde veya test senaryolarında yapılan değişiklikleri veya güncellemeleri yansıtacak şekilde güncellenebilir.