BU NOTLAR ISTQB INGILIZCE SYLLABUS ESAS ALINARAK HAZIRLANMISTIR, ANCAK CEVIRILER KISIYE OZEL OLARAK YAPILMISTIR, BAGLAYICI OZELLIGI YOKTUR

Hazirlayanin Notu : Bu Syllabus ISTQB'ye Turkce hazirlananlar icin yapilmis bir calismadir. Ingilizceden Turkce'ye ceviri esnasinda Teknik tabirler mumkun oldugunca orjinal haliyle birakilmistir. Ornegin Syllabus kelimesi yerine ders ozeti veya mufredat kelimeleri kullanmaktansa orjinalini korumayi tercih ettim. Ayni sekilde tam karsiliginin tek kelime olarak turkcede olmadigini dusundugum Testing kelimesini de ingilizce asli ile kullandim. Yine error, failure,defect gibi turkce karsiliklari benzer olan kelimelerde turkce cevirinin yanina ingilizce orjinal halini de eklemeyi tercih ettim. Yani bu ceviri profesyonel bir ingilizce-Turkce ceviri olmaktan ziyade kendi derslerimde de yaptigim uzere teknik terimlerin ingilizce orijinal halleri kullanilarak yapilmis bir tercumedir. Ticari hic bir fayda gozetmeksizin, kendime gore yapilmis bir tercumedir. Yanlislarim varsa bana yazmaniz durumunda seve seve duzeltecegimi de bilmenizi isterim.

Ingilizce syllabus uzerinden vakit buldukca bu calismayi yapmaya devam edip sizlerle paylasacagim.

0 Introduction

0.1 Syllabus'un amaci

Bu syllabus, Temel Düzeyde Uluslararası Yazılım Test Kalifikasyonunun standartlarini belirler. ISTQB® bu syllabus'u aşağıdaki amaclari gerceklestirmek icin olusturmustur.

1. Üye kurullarin, yerel dillerine tercüme yapmasi ve eğitim sağlayıcıları akredite etmesi. Üye kurullar, müfredatı kendi dil ihtiyaçlarına göre uyarlayabilir ve yerel yayınlarına uyarlamak için referanslar ekleyebilir.
2. Belgelendirme kuruluşlarının, bu müfredatın öğrenme hedeflerine uyarlanmış kendi yerel dillerinde sınav soruları türetmesi.
3. Eğitim sağlayıcılarin, eğitim materyalleri üretmesi ve uygun öğretim yöntemlerini belirlemesi
4. Sertifika adaylarının, sertifika sınavına hazırlanması (bir eğitim kursunun parçası veya bağımsız olarak)
5. Uluslararası yazılım ve sistem mühendisliği topluluğunun, yazılım ve sistem testi mesleğini ilerletmesi ve yazilacak kitap ve makalelere bir temel olusturmasi

ISTQB®, önceden yazılı izin almak koşuluyla, diğer kuruluşların bu müfredatı başka amaçlar için kullanmasına izin verebilir.

0.2 Software Testing icin Foundation Level Sertifikasi

Foundation Level yeterliligi, yazılım testinde yer alan herkese yöneliktir. Bu kualifikasyon tester'larla birlikte test analyst, test engineer, test consultant, test manager, user acceptance tester ve software developerlari da icerir. Ayrica bu kalifikasyon software test'leri konusunda temel bir bilgi edinmek isteyen product owner, project manager, quality manager, software development manager, business analyst, IT director veya yonetim danismanlari icin de uygundur. Temel seviyedeki bu sertifikaya sahip olanlar ust duzey yazilim sertifikalari icin de muracaat edebilirler.
Bu Temel Müfredat 2018 V3.1 aşağıdaki bilgileri içerir:

- Müfredat için ticari sonuçlar
- Ticari sonuçlar ve öğrenme hedefleri arasındaki izlenebilirliği gösteren matris
- Bu syllabus'un özeti

0.3 Olcumlendirilebilir Ogrenme Hedefleri ve Bilişsel Bilgi Düzeyleri

Öğrenme hedefleri, ticari sonuçları destekler ve Sertifikalı Test Uzmanı Temel Düzey sınavlarını oluşturmak için kullanılır. Bu syllabus icerigi (giris ve ekler haric) K1 seviyesinde test icin gereklidir.

Yapilan sinavda adaylardan syllabus icerisindeki alti bolumde anlatilan konulardan herhangi bir keyword'u hatirlamasi veya tanimlamasi istenebilir. Ogrenme hedef seviyeleri her bolumun basinda belirtilmekle birlikte asagidaki uc seviye olarak siniflandirilmistir.

- K1: hatirlama
- K2: anlama
- K3: uygulama

Öğrenme hedeflerine ilişkin daha fazla ayrıntı ve örnekler Ek B'de verilmiştir.

Ogrenme hedeflerinde acik olarak yazmasa bile K1 seviyesi icin bölüm başlıklarının hemen altında anahtar kelime olarak listelenen tüm terimlerin tanımları hatırlanmalıdır .

0. 4 Foundation Level Sertifika Sinavi

Foundation Level sertifika sınavı bu syllabus iceriginden yapilacaktir. Sınav sorularının cevapları, bu syllabus'un birden fazla bölümüne dayalı materyal kullanımını gerektirebilir. Sinav'da giris ve ekler haric tum bolumlerden soru gelebilir. Standartlar, kitaplar ve diğer ISTQB® müfredatları referans olarak dahil edilmiş olsa da bu syllabus'da ozet olarak verilen kisimlar haric o kaynaklardan soru sorulmayacaktir.

Sınav, çoktan seçmeli 40 sorudan olusmaktadir. Sınavı geçmek için soruların en az %65'inin (yani 26 sorunun) doğru cevaplanması gerekir.

Sinava girmek icin bir kursa dahil olma mecburiyeti yoktur.

0. 5 Akreditasyon

Bir ISTQB® Kurulu, ders materyali bu müfredatı takip eden eğitim sağlayıcılarını akredite edebilir. Eğitim sağlayıcılar, akreditasyonu gerçekleştiren Üye Kurul veya organdan akreditasyon yönergeleri almalıdır. Akredite edilmiş bir kurs, bu müfredata uygun olarak kabul edilir ve kursun bir parçası olarak bir ISTQB® sınavına girmesine izin verilir.

0. 6 Ayrinti Seviyesi

Bu syllabus'daki ayrıntı düzeyi, uluslararası düzeyde kabul edilmis kurslar ve sınavlar gozonunde bulundurularak belirlenmistir. Bu amaca ulaşmak için syllabus şunlardan oluşur:

• Foundation Level'in amacını açıklayan genel öğretim hedefleri

• Öğrencilerin hatırlaması gereken terimlerin bir listesi

• Ulaşılacak bilişsel öğrenme sonucunu tanımlayan, her bilgi alanı için öğrenme hedefleri

• Kabul edilen literatür ve standartlar gibi kaynaklara yapılan referanslar dahil olmak üzere temel kavramların tanımı

Bu syllabus, yazılım testi ile ilgili tum konularin aciklanmasini degil, Foundation Level eğitim kurslarında kapsanacak konulari ve ayrıntı düzeyini icermektedir. Agile methodolojisine uygun yurutulen projeler de dahil olmak üzere tüm yazılım projelerine uygulanabilecek test kavram ve tekniklerine odaklanır. Bu syllabus, belirli bir yazılım geliştirme yaşam döngüsü veya yöntemiyle ilgili herhangi bir özel öğrenme hedefi içermez, ancak bu kavramların Agile projelerde veya diğer yinelemeli(iterative) ve artımlı(incremental) yaşam döngülerinde ve sıralı(sequential) yaşam döngülerinde nasıl uygulanacağını tartışır.

0.7 Bu Syllabus Hangi Bolumlerden Olusur?

Syllabus sinavda soru cikacak içeriğe sahip altı bölüme ayrilmistir. Her bölümün üst düzey başlığı, bölümün zamanını belirtir. Akredite eğitim kursları için, müfredat, aşağıdaki gibi altı bölüme dağıtılmış en az 16.75 saatlik eğitim gerektirir:

- Bölüm 1: 175 dakika Testing Temel Bilgileri
- Bölüm 2: 100 dakika Yazılım Geliştirme Yaşam Döngüsü ve Testing
- Bölüm 3: 135 dakika Statik Testing
- Bölüm 4: 330 dakika Test Teknikleri
- Bölüm 5: 225 dakika Test Yönetimi
- Bölüm 6: 40 dakika Testing için Tool Desteği

1 Testing'in Temelleri	175 minutes

coverage, debugging, defect, error, failure, quality, quality assurance, root cause, test analysis, test basis, test case, test completion, test condition, test control, test data, test design, test execution,

test implementation, test monitoring, test object, test objective, test oracle, test planning, test procedure, test process, test suite, testing, testware, traceability, validation, verification

Testing'in Temelleri bolumu ogrenme hedefleri:

1.1 Testing nedir?

FL-1.1.1 (K1) Testing'in hedeflerini tanimlama

FL-1.1.2 (K2) Testing ve debugging karsilastirilmasi

1.2 Testing Nicin Gereklidir?

FL-1.2.1 (K2) Testing'in nicin gerekli olduguna dair ornekler

FL-1.2.2 (K2) Testing and quality assurance(Kalite Guvence) arasindaki iliski ve Testing'in daha iyi bir kaliteye ulasmadaki rolunu gosteren ornekler

FL-1.2.3 (K2) Error, defect ve failure karsilastirilmasi

FL-1.2.4 (K2) Bir defect'i olusturan temel neden ve etkilerini tanimlama

1.3 Testing'in Yedi Prensibi

FL-1.3.1 (K2) Testing'in Yedi Prensibinin aciklanmasi

1.4 Test Asamalari

FL-1.4.1 (K2) Context'in test surecine etkileri

FL-1.4.2 (K2) Test asamalari boyunca yapilan test faaliyetlerini ve ilgili görevleri tanımlayın

FL-1.4.3 (K2) Test sureclerinde kullanilan work product (calisma urunleri)'nin tanimlanmasi

FL-1.4.4 (K2) Testin temel hedefi ile kullanilan work product arasinda izlenebilirligin onemi

1.5 Testing Psikolojisi

FL-1.5.1 (K1) Testin başarısını etkileyen psikolojik faktörleri hatırlama

FL-1.5.2 (K2) Bir test uzmanı ile yazılımcının bakış açılarını karşılaştırma

1.1 Testing Nedir ?

Yazılım sistemleri, ticari uygulamalardan (örn. bankacılık) tüketici ürünlerine kadar (örn. otomobiller), yaşamın ayrılmaz bir parçasıdır. Çoğu insan, beklediği gibi çalışmayan bir yazılımla karsilasmistir. Düzgün çalışmayan yazılımlar para, zaman veya iş itibarı kaybı, hatta yaralanma veya ölüm dahil olmak üzere birçok soruna yol açabilir. Software testing, yazılımın kalitesini değerlendirmenin ve çalışma sırasında yazılım hatası(failure) riskini azaltmanın bir yoludur.

Testing icin yaygın ama yanlış bir algı, testing'in yalnızca test amacli kodlarin calistirilmasindan ve sonuçların kontrol edilmesinden ibaret oldugudur. Bölüm 1.4'te açıklandığı gibi, yazılım testi birçok farklı aktiviteyi içeren bir süreçtir.Testleri calistirma(execution) (sonuçların kontrolü de dahil) bu faaliyetlerden sadece biridir. Test süreci(test process) testleri execute etme disinda, test planlama, analiz etme, test tasarlama(designing) ve uygulama(implementing tests), test ilerlemesini ve sonuçlarını raporlama ve yapilan testin kalitesini değerlendirme gibi faaliyetleri de içerir.

Bazı testler, testin isleyisine test edilen bileşen(component) ve testin gerceklestigi sistemi de dahil eder; bu tür testlere dinamik test denir. Bazi testler ise, test edilen bileşen ve sistemi calistirilan testlere dahil etmez; bu tür testlere statik test denir. Bu nedenle, testing aynı zamanda gereksinimler(requirements), user story ve kaynak kodu(source code) gibi calisma urunlerinin(work product) gözden geçirilmesini de içerir.

Testing icin var olan başka bir yanlış algı ise, Testing'in tamamen gereksinimlerin(requirements), kullanıcı hikayelerinin(user story) veya diger spesifikasyonlarin dogrulanmasina odaklandigidir. Testing, sistemin belirtilen gereksinimleri karşılayıp karşılamadığını kontrol etmenin yaninda, sistemin calisma ortaminda kullanıcı (user) ve diğer paydaşlarin(stakeholder) ihtiyaçlarını karşılayıp karşılamayacağını kontrol etmeyi de içerir.

Test faaliyetleri, farklı yaşam döngülerinde(life cycle) farklı şekilde düzenlenir ve yürütülür (Bölüm 2.1).

1.1.1 **Testing'in Temel Amaclari**

Herhangi bir proje için testin amaçları şunları içerebilir:

- Gereksinimler, user story, tasarım ve kodlar gibi calisma ürünlerini(work product) değerlendirerek kusurları önlemek
- Belirtilen tüm gereksinimlerin karşılanıp karşılanmadığını doğrulamak
- Testin tamamlanıp tamamlanmadığını kontrol etmek ve kullanıcılar ve stakeholder'larin beklediği gibi çalışıp çalışmadığını doğrulamak
- Testimizin kalite düzeyine güven oluşturmak
- Kusurları(defects) ve arızaları(failures) bulmak, böylece yazilimin "yetersiz yazılım kalitesine (inadequate software quality)" dusme riskini azaltmak
- Paydaşlara(stakeholder), bilinçli kararlar vermelerini sağlamak için yeterli bilgi vermek, ozellikle de testimizin kalite düzeyi ile ilgili bilgiler saglamak
- Sözleşmeye dayalı, yasal veya düzenleyici gereksinimlere veya standartlara uymak ve/veya testimizin bu tür gereksinimler veya standartlara uygunluğunu doğrulamak

Testin amaçları, test edilen bileşen(component) veya sistemin bağlamına(context), test düzeyine ve yazılım geliştirme yaşam döngüsü(lifecycle) modeline bağlı olarak değişebilir. Bu farklılıklar örneğin şunları içerebilir:

- Bileşen(component) testi sırasında, bir amaç mümkün olduğunca çok sayıda hata bulmak olabilir, böylece altta yatan kusurlar erken tespit edilip düzeltilir. Diğer bir amaç, bileşen testlerinin kodlarinin uzerinden gecmek ve kod kalitesini artırmak olabilir.

- Kabul testi(acceptance testing) sırasında, sistemin beklendiği gibi çalıştığını ve gereksinimleri karşıladığını doğrulamak bir amaç olabilir. Bu testin bir başka amacı da , paydaşlara(stakeholder), sistemi belirlenen zamanda yayınlama(release) durumunda olusabilecek riskler konusunda bilgi vermek olabilir.

1.1.2 Testing ve Hata Ayiklama(Debugging)

Testing ve debugging birbirinden farklidir.

Testlerin yürütülmesi(Execution), yazılımdaki kusurların(defect) neden olduğu arızaları(failure) gösterebilir. Hata ayıklama(debugging) ise, bu tür kusurları(defect) bulan, analiz eden ve düzelten geliştirme(development) etkinliğidir. Sonraki onay testi, düzeltmelerin(fixes) kusurları(defect) çözüp çözmediğini kontrol eder. Bazı durumlarda, geliştiriciler(developers) hata ayıklamayı(debugging), ilişkili bileşen(associated component) ve bileşen entegrasyonu testini(component integration testing) yaparken, test uzmanları(tester) ilk testten(initial test) ve son onay testinden(final confirmation test) sorumludur. Ancak, Agile geliştirmede ve diğer bazı yazılım geliştirme yaşam döngülerinde, tester'lar hata ayıklama(debugging) ve bileşen testine(component testing) dahil olabilir.

Yazılım testi kavramları hakkında daha fazla bilgiye ISO standardı (ISO/IEC/IEEE 29119-1 standardindan ulasilabilinir.

1.2 Testing Nicin Gereklidir ?

Bileşenlerin(component), sistemlerin ve bunlarla ilişkili belgelerin titiz bir şekilde test edilmesi, çalışma sırasında meydana gelen arıza(failure) riskinin azaltılmasına yardımcı olabilir. Kusurlarin(defects) tespit edilip giderilmesi, bileşenlerin veya sistemlerin kalitesine katkıda bulunur. Ayrica, sözleşmeye bağlı veya yasal gereklilikleri ve sektöre özel standartları karşılamak için de yazılım testi gerekebilir.

1.2.1 Testing'in Basariya Katkilari

Bilgi işlem tarihi boyunca, yazılım ve sistemler işletime alındıktan sonra kusurların varlığı(defect) nedeniyle, arızalara(failure) neden olması veya bir şekilde paydaşların(stakeholders) ihtiyaçlarını karşılamaması oldukça yaygındır. Ancak uygun test teknikleri, gerekli test uzmanlığı düzeyinde, uygun test seviyelerinde ve yazılım geliştirme yaşam döngüsünün uygun noktalarında uygulandığında bu tür sorunlarin sıklığını azaltabilir. Örnekler şunları içerir:

- Gereksinimlerin(requirements) gözden geçirilmesine veya user story'nin iyileştirilmesine tester'larin dahil edilmesi bu work product'daki kusurları tespit edilmesine katki saglar. Gereksinim kusurlarının tanımlanması ve kaldırılması, yanlış veya gereksiz özelliklerin geliştirilmesi riskini azaltır.
- Sistem tasarlanırken tester'larin sistem tasarımcıları ile yakın bir şekilde çalışması, her iki tarafın birbirinin yaptigi islemleri daha iyi anlamasini saglayabilir. Bu artan anlayış, temel tasarım kusurları riskini azaltabilir ve testlerin daha erken bir aşamada tanımlanmasını sağlayabilir.
- Kod geliştirme aşamasındayken tester'larin developer'lar ile yakın bir şekilde çalışması, tarafların kodu ve kodun nasıl test edileceğini daha iyi anlamasını saglayabilir. Bu artan anlayış, kod ve testler içindeki kusur(defect) riskini azaltabilir.
- Test edenlerin yazılımı piyasaya sürmeden önce doğrulamasını(verify) ve onaylamasını(validate) sağlamak, bunlar yapilmadiginda gözden kaçabilecek hataları tespit edebilir ve hatalara(failure) neden olan kusurlarin(defects) kaldırılması sürecini destekleyebilir(Ornegin debugging). Bu da, yazılımın paydaş(stakeholders) ihtiyaçlarını karşılama ve gereksinimleri karşılama olasılığını artırır.

Bu örneklere ek olarak, tanımlanmış test hedeflerine (bkz. bölüm 1.1.1) ulaşılması, genel yazılım geliştirme(development) ve bakım(maintenance) başarısına katkıda bulunur.

1.2.2 Kalite Guvencesi(Quality Assurance) ve Testing

İnsanlar testing'e atıfta bulunmak için genellikle kalite güvencesi (veya sadece KG / QA) ifadesini kullansa da, kalite güvencesi ve test aynı değildir, ancak ilişkilidir. Daha kapsamli bir kavram olan **kalite yönetimi** onları birbirine bağlar. Kalite yönetimi, bir kuruluşu kalite açısından yönlendiren ve kontrol

eden tüm faaliyetleri içerir. Diğer faaliyetlerinin yanı sıra kalite yönetimi, hem kalite güvencesini hem de kalite kontrolünü içerir. Kalite güvencesi, uygun kalite seviyelerine ulaşılacağına dair güven sağlamak için tipik olarak uygun süreçlere bağlı kalmaya odaklanır. Prosesler düzgün bir şekilde yürütüldüğünde, bu prosesler tarafından oluşturulan iş ürünleri(work products) genellikle daha yüksek kalitede olur ve bu da kusurların önlenmesine katkıda bulunur. Ek olarak, kusurların nedenlerini tespit etmek ve ortadan kaldırmak için kök neden(root cause) analizinin kullanılması ve süreçleri iyileştirmek için geriye dönük toplantılarındaki(retrospective meetings) tespitlerin doğru uygulanması, etkin kalite güvencesi için önemlidir.

Kalite kontrol, urunun uygun kalite seviyelerine ulaşmasını destekleyen test faaliyetleri de dahil olmak üzere çeşitli faaliyetleri içerir. Test faaliyetleri, genel yazılım geliştirme(development) veya bakım(maintenance) sürecinin bir parçasıdır. Kalite güvencesi, tüm sürecin uygun şekilde yürütülmesiyle ilgili olduğundan, kalite güvencesi, uygun testler yapilmasini destekler. 1.1.1 ve 1.2.1 bölümlerinde açıklandığı gibi, test etme, kalitenin elde edilmesine çeşitli şekillerde katkıda bulunur.

1.2.3 Hata(Error), Kusur(Defect) ve Ariza(Failure)

Bir kişi bir hata (error / mistake) yapabilir ve bu da yazılım kodunda veya diğer ilgili iş ürününde bir kusurun(defect) (arıza/fault veya hata/bag) ortaya çıkmasına neden olabilir. Bir iş ürününde bir kusurun(defect) ortaya çıkmasına neden olan bir hata(error), ilgili baska bir iş ürününde bir kusurun ortaya çıkmasına yol açan baska bir hatayı tetikleyebilir. Örneğin, bir gereksinim belirleme hatası gereksinim hatasına yol açabilir ve bu da kodda bir hataya(defect) yol açan bir programlama hatasıyla(error) sonuçlanır.

Koddaki bir kusur calistirilirsa(execute), her zaman olmasa da bir arizaya(failure) neden olabilir. Örneğin, bazı kusurlar(defect), nadiren veya hiç meydana gelmeyebilecek bir arızayı(failure) tetiklemek için çok özel girdiler(input) veya ön koşullar(precondition) gerektirir.

Hatalar(Error), aşağıdakiler gibi birçok nedenden dolayı oluşabilir:

• Zaman baskisi

• İnsan hatalari

• Deneyimsiz veya yetersiz vasıflı proje calisanlari

• Gereksinimler ve tasarım hakkında yanlış bilgilendirme dahil olmak üzere proje calisanlar arasındaki iletişimsizlik

• Kodun, tasarımın, mimarinin, çözülmesi gereken temel sorunun ve/veya kullanılan teknolojilerin karmaşıklığı

• Dahili sistem(intra-system) ve harici sistemlerin(inter-system) arayuzleri arasinda yasanabilecek yanlış anlamalar (ozellikle bu tür sistem içi ve sistemler arası etkileşimlerin sayıca fazla olduğu durumlarda)

• Yeni, tam bilinmeyen teknolojiler

Koddaki kusurlardan(defects) kaynaklanan arızalara(failure) ek olarak, arızalar çevresel koşullardan da kaynaklanabilir. Örneğin radyasyon, elektromanyetik alanlar ve kirlilik, aygıt yazılımında kusurlara neden olabilir veya donanım koşullarını değiştirerek yazılımın calismasini(execute) etkileyebilir.

Beklenmeyen(unexpected) test sonuçlarının tümü başarısızlık(failure) değildir. Testlerimizin gorevi hatayi bulmak olsa da bazen testler hatali bir urundeki hatayi tespit edemeyip, hatasiz urun sonucu verebilir(False positives). Bu durum testlerin calistirilma yontemlerindeki hatalardan veya test verilerindeki, test ortamındaki veya diğer test yazılımındaki kusurlardan veya başka nedenlerden

kaynaklanabilir. Bazen de bunun tersi bir durumda hatasiz olan bir urun hatali olarak tanimlanabilir(false negatives).

1.2.4 Kusurlar(Defects), Temel Nedenler(Root Causes) ve Etkiler(Effects)

Kusurların(defect) temel nedenleri(root causes), kusurların oluşmasına katkıda bulunan ilk eylem veya koşullardır. Kusurlar, gelecekte benzer kusurların oluşmasını azaltmak için temel nedenlerini tespit etmek amaciyla analiz edilebilir. Kök neden analizi, en önemli kök nedenlere odaklanarak, gelecekte önemli sayıda kusurun ortaya çıkmasını önleyen süreç iyileştirmelerine imkan tanir.

Örneğin, tek bir yanlış kod satırı nedeniyle yanlış faiz ödemelerinin müşteri şikayetleriyle sonuçlandığını varsayalım. Arızalı(defective) kod, product owner'in faizin nasıl hesaplanacağını yanlış anlaması nedeniyle, tam anlasilamayan bir user story neticesinde yazilmistir. Faiz hesaplamalarında büyük oranda kusur varsa ve bu kusurların temel nedeni benzer yanlış anlamalara dayanıyorsa, product owner'lar gelecekte bu tür kusurları azaltmak için faiz hesaplamaları konusunda eğitilebilir.

Bu örnekte, müşteri şikayetleri etkilerdir(effect). Yanlış faiz ödemeleri başarısızlıktır(failure). Koddaki yanlış hesaplama bir kusurdur(defect) ve koddaki yanlis hesaplama, orijinal kusurdan yani user story'deki belirsizlikten kaynaklanmaktadır. Orijinal kusurun temel nedeni ise, product owner'in user story yazarken bir hata yapmasına neden olan bilgi eksikliğiydi. Kök neden analizi süreci hakkinda tum fikirler ISTQB-CTEL-TM ve ISTQB-CTEL-ITP'de karsilastirilmistir.

1.3 Testing'in Yedi Prensibi

Geçen 50 yılda birçok test prensibi ortaya atılmış ve tartışılmıştır. Aşağıda listelenen maddeler, bu prensiplerin ortak noktalarını biraraya getiren ve özetleyen bir özelliğe sahiptir

1. Test, kusurların varlığını gösterir, yokluğunu değil

Testing, kusurların mevcut olduğunu gösterebilir, ancak kusur olmadığını kanıtlayamaz. Test etme, yazılımda kalan keşfedilmemiş hataların olasılığını azaltır, ancak hiçbir hata kalmasa bile testing, yazilimin tamamen dogru oldugunun bir **kanıtı** olamaz.

2. Yazilimi butunuyle test etmek imkansızdır

Bir yazilimin tamamini test etmek (tüm girdi ve önkoşul kombinasyonları dahil) kucuk projeler dışında mümkün değildir. Bir yazilimin tamamini test etmeye çalışmak yerine, risk analizi, test teknikleri ve öncelikler kullanilarak test faaliyetlerini gerekli yerlere yogunlastirmak gerekir.

3. Testlerin erken baslamasi zaman ve para kazandirir

Hataları erken bulmak için, yazılım geliştirme yaşam döngüsünde statik ve dinamik test faaliyetleri mümkün olduğunca erken başlatılmalıdır. Testing'e erken baslamaya bazen sola kaydırma(shift left) denir. Yazılım geliştirme yaşam döngüsünün başlarında test yapmak, gec farkedildigi icin maliyeti artacak değişiklikleri azaltmaya veya ortadan kaldırmaya yardımcı olur (bkz. bölüm 3.1).

4. Kusurlar belirli alanlarda yogunlasir (cluster together/kumelesme)

Bir uygulama icerisindeki az sayıda modül release öncesi testler sırasında keşfedilen kusurların çoğunu

içerir veya operasyonel arızaların çoğundan sorumludur. Öngörülen hata kümeleri ve test veya operasyonda gerçek gözlemlenen kusur kümeleri, test çabasına odaklanmak için kullanılan bir risk analizine önemli bir girdidir (ilke 2'de belirtildiği gibi).

5. Pestisit paradoksuna dikkat edin(Turkcede antibiyotik direnci tabiri uygun olabilir)

Aynı testler defalarca tekrarlanırsa, bu testler artık yeni bir kusur bulamaz. Yeni kusurları tespit etmek için mevcut testlerin, test verilerinin değiştirilmesi veya yeni testlerin yazılması gerekebilir.(Tıpkı pestisitlerin bir süre sonra böcekleri öldürmede artık etkili olmadığı gibi, testler de artık kusurları bulmada etkili olmazlar.) Standart tekrarlanan(automated) regresyon testi gibi bazı durumlarda, pestisit paradoksuna dikkat edilmesi, nispeten daha düşük sayıda regresyon kusuru olusmasi gibi faydalı bir sonuca sahiptir.

6. Testing, kosullara(context) bağımlidir (Test yaklaşımı ve aktiviteleri yazılım projesinin koşullarına göre değişiklik gösterir)

Test aktiviteleri, yazılımın özelliklerine, bağlamına ve içeriğine göre farklı biçimlerde ele alınmalıdır. Örnek olarak, bir e-ticaret yazılımı ile nükleer santral için yazılmış güvenlik tehlikesi taşıyan bir uygulama farklı şekillerde, farklı test teknikleri ve metodolojileri kullanılarak test edilmelidir.

7. Hataların olmaması bir yanılgıdır

Bazı kuruluşlar, tester'larin tüm olası testleri çalıştırmasını ve olası tüm kusurları bulmasını bekler, ancak sırasıyla 2 ve 1 numaralı ilkeler bize bunun imkansız olduğunu söyler. Ayrıca, sadece çok sayıda kusuru bulup düzeltmenin bir sistemin başarısını garantileyeceğini beklemek bir yanılgıdır (yanlış bir inançtır). Örneğin, belirtilen tüm gereksinimlerin kapsamlı bir şekilde test edilmis ve bulunan tüm kusurlar düzeltilmis bir yazilim, hatasiz olmasina karsin kullanımı zor, kullanıcıların ihtiyaç ve beklentilerini karşılamayan veya diğer rakip sistemlere kıyasla daha düşük seviyede olan bir sistem üretebilir.

Bu ve diğer test ilkelerinin örnekleri için Myers 2011, Kaner 2002, Weinberg 2008 ve Beizer 1990'a bakılabilir.

1.4 Test Surecleri

Tek bir evrensel yazılım test süreci yoktur, ancak bunlar olmadan testin belirlenen hedeflere ulaşma olasılığının daha düşük olacağı ortak test faaliyet grupları vardır. Bu test faaliyet gruplari da bir test sürecidir. Herhangi bir durum icin hazirlanmis uygun ve spesifik yazılım test süreci birçok faktöre bağlıdır. Bu test sürecinde hangi test faaliyetlerinin yer alacagi, bu faaliyetlerin nasıl uygulanacagi ve bu faaliyetlerin ne zaman gerçekleştirilecegi gibi konular kuruluşun test stratejisi belirlenirken muzakere edilebilir.

1.4.1 Test Process in Context

Bir kuruluş için test sürecini etkileyen bağlamsal faktörler asagida listelenen maddeleri içerir, ancak bunlarla sınırlı değildir:

- Yazılım geliştirme yaşam döngüsü(Software development lifecycle) modeli ve kullanılan proje metodolojileri
- Dikkate alinan test seviyeleri ve test türleri
- Ürün ve proje riskleri
- Iş alanı

• Aşağıdaki maddeleri içeren ancak bunlarla sınırlı olmayan operasyonel kısıtlamalar:

- o Bütçeler ve kaynaklar
- o Zaman çizelgeleri
- o Karmaşıklık
- o Sözleşme ve düzenleyici gereksinimler

• Sirket politikalari ve uygulamalari

• Ilgili sirket ici veya harici standartlar

Asagida genis olarak aciklanacak 3 bolum, organizasyonlar icin genel test sureclerini tanimlamaktadir

- Test faaliyetleri ve görevleri
- Test çalışma ürünleri
- Test esası ve test çalışması ürünleri arasında izlenebilirlik

Test esasının(test basis) (göz önünde bulundurulan herhangi bir seviye veya test türü için) tanımlanmış ölçülebilir kapsam kriterlerine sahip olması çok yararlıdır. Kapsam kriterleri(coverage criteria), yazılım test hedeflerine ulaşıldığını gösteren faaliyetleri yönlendirmek için temel performans göstergeleri (key performance indicators- KPI) olarak etkin bir şekilde hareket edebilir (bkz. bölüm 1.1.1).

Örneğin, bir mobil uygulama için test temeli, bir gereksinimler listesi ve desteklenen mobil cihazların bir listesini içerebilir. Her gereksinim, test esasının bir unsurudur. Desteklenen her cihaz aynı zamanda test esasının bir unsurudur. Kapsam kriterleri, test esasının her bir unsuru için en az bir test senaryosu gerektirebilir. Bu testler bir kez calistirildiginda, test sonuçları, paydaşlara(stakeholders) belirtilen gereksinimlerin karşılanıp karşılanmadığını ve desteklenen cihazlarda arızaların gözlemlenip gözlemlenmediğini gosterecektir.

Test süreçleri hakkında daha fazla bilgi icin, ISO standardı (ISO/IEC/IEEE 29119-2),gozden gecirilebilir.

1.4.2 Test Faaliyetleri ve Görevleri

Soru ile ilgili oldugu icin asagidaki bolumler eklendi. Normal calisma yukaridaki kisimdan devam edecek

2.2.4 Acceptance Testing (Kabul Testi)

Kabul testinin amaçları

Kabul testi(acceptance testing), sistem testi gibi, tipik olarak tüm sistemin veya ürünün(product) davranışına(behaviour) ve yeteneklerine(capability) odaklanır.

Kabul testinin yapilis amaclari asagidakileri maddeleri icerir;

• Bir bütün olarak sistemin kalitesine güven olusturmak

• Sistemin tamamlandığının ve beklendiği gibi çalışacağının doğrulanması

• Sistemin işlevsel(functional) ve işlevsel olmayan (non-functional) davranışlarının belirtildiği gibi olduğunun doğrulanması

Kabul testi, sistemin dağıtıma(deployment) ve müşteri (son kullanıcı) tarafından kullanıma hazır olup olmadığını değerlendirmek için bilgi üretebilir. Kabul testi sırasında kusurlar bulunabilir, ancak kusurları bulmak genellikle bir amaç değildir(a) ve kabul testi sırasında önemli sayıda kusur bulmak bazı durumlarda büyük bir proje riski olarak kabul edilebilir.

Kabul testleri, yasal veya düzenleyici gereklilikleri veya standartları da karşılayabilir.

Kabul testinin yaygın biçimleri aşağıdakileri içerir:

- User acceptance testing : Kullanici Kabul Testi
- Operational acceptance testing : Operasyonel Kabul Testi
- Contractual and regulatory acceptance testing : Sözleşmeye dayalı ve düzenleyici kabul testleri
- Alpha and beta testing: Alfa ve beta testi.

Her biri aşağıdaki dört alt bölümde açıklanmıştır.

User acceptance testing (UAT) : Kullanici Kabul Testi

Sistemin kullanıcı kabul testi, tipik olarak, gerçek veya simüle edilmiş bir işletim ortamında hedef kullanıcılar tarafından sistemin kullanımına uygunluğun doğrulanmasına odaklanır. Temel amaç, kullanıcıların ihtiyaçlarını karşılamak, gereksinimleri karşılamak ve iş süreçlerini minimum zorluk, maliyet ve riskle gerçekleştirmek için sistemi kullanabilecekleri konusunda güven oluşturmaktır.

Spesifik yaklaşımlar ve sorumluluklar

Kabul testi genellikle müşterilerin, iş kullanıcılarının, ürün sahiplerinin veya bir sistemin operatörlerinin sorumluluğundadır ve diğer paydaşlar da dahil olabilir.

Kabul testi genellikle sıralı geliştirme (sequential development) yaşam döngüsündeki son test düzeyi olarak düşünülür, ancak başka zamanlarda da gerçekleşebilir.

Yinelemeli geliştirmede(iterative development), proje ekipleri, kabul kriterlerine göre yeni bir özelliği doğrulamaya odaklananlar ve yeni bir özelliğin kullanıcıların ihtiyaçlarını karşıladığını doğrulamaya odaklananlar gibi, her yineleme sırasında ve sonunda çeşitli kabul testi biçimleri kullanabilir. Ayrıca, alfa testleri ve beta testleri, her yinelemenin sonunda, her yinelemenin tamamlanmasından sonra veya bir dizi yinelemeden sonra gerçekleşebilir. Kullanıcı kabul testleri, operasyonel kabul testleri, yasal kabul testleri ve sözleşmeye dayalı kabul testleri de, her yinelemenin sonunda, her yinelemenin tamamlanmasından sonra veya bir dizi yinelemeden sonra gerçekleşebilir.

2.3.5 Islevsel (Functional) kullanılabilirlik, performans verimliliği veya güvenlik gibi sistemlerin ve yazılımların özelliklerini değerlendirir.Testing

Islevsel Testing (Functional Testing), sistemin gerçekleştirmesi gereken işlevleri değerlendiren testleri içerir. Fonksiyonel gereksinimler, iş gereksinimleri spesifikasyonları(business requirements specifications), alt bolumler(epics), kullanıcı hikayeleri(user stories), kullanım senaryoları(use cases) veya fonksiyonel spesifikasyonlar gibi iş ürünlerinde tanımlanabilir veya dokumante edilmemis de olabilir. **Fonksiyonlar, sistemin yapması gerekenlerdir**.

Islevsel testler tüm test seviyelerinde gerçekleştirilmelidir (örneğin, bileşenlere(component) yönelik testler, bir bileşen spesifikasyonuna dayalı olabilir), ancak her düzeyde odak noktası farklıdır. (bkz. bölüm 2.2).

Islevsel Testing, yazılımın davranışını dikkate alır, bu nedenle, bileşen veya sistemin işlevselliğini denetleyecek test koşulları ve test senaryoları türetmek için kara kutu(black box) teknikleri kullanılabilir. (bkz. bölüm 4.2).

İşlevsel testin hassasiyeti, işlevsel kapsam(Functional coverage) yoluyla ölçülebilir. İşlevsel kapsam, hangi işlevlerin testler tarafından ne ölçüde uygulandığının belirlenmesidir ve kapsanan öğe türlerinin yüzdesi olarak ifade edilir. Örneğin, testler ve işlevsel gereksinimler(functional requirements) incelenerek, test tarafından ele alınan gereksinimlerin karsilanma yüzdesi hesaplanabilir ve bu inceleme ile muhtemel kapsam boşlukları belirlenebilir.

Fonksiyonel test tasarımı ve uygulaması, yazılımın uygulandigi spesifik iş alanina ait özel beceri veya bilgileri içerebilir(örneğin, petrol ve gaz endüstrileri için jeolojik modelleme yazılımı bilgisi gerekeblir).

2.3.6 Islevsel Olmayan(Non-functional) Testing

Islevsel Olmayan(Non-functional) Testing, sistemlerin ve yazılımların kullanılabilirlik(usability), performans verimliliği(performance efficiency) veya güvenlik(security) gibi özelliklerini değerlendirir. Yazılım kalite özelliklerinin sınıflandıılrması için ISO standardına (ISO/IEC 25010) bakılabilir. **İşlevsel olmayan test, sistemin "ne kadar iyi" calistiginin(behaves) test edilmesidir**.

Yaygın yanlış algılamaların aksine, işlevsel olmayan testler tüm test seviyelerinde yapılabilir hatta yapılmalıdır ve mümkün olduğunca erken yapılmalıdır. İşlevsel olmayan kusurların geç keşfedilmesi, bir projenin başarısı için son derece tehlikeli olabilir.

Kara kutu teknikleri (bkz. bölüm 4.2), işlevsel olmayan testler için test koşulları ve test senaryoları türetmek için kullanılabilir. Örneğin, performans testleri için stres koşullarını tanımlamak için sınır değer analizi kullanılabilir.

İşlevsel olmayan testlerin hassasiyeti, işlevsel olmayan kapsamin belirlenmesi yoluyla ölçülebilir. İşlevsel olmayan kapsam, bazı işlevsel olmayan öğe türlerinin testler tarafından uygulanma derecesidir ve kapsanan öğe türlerinin yüzdesi olarak ifade edilir. İşlevsel olmayan kapsam, bazı işlevsel olmayan öğe türlerinin testler tarafından uygulanma derecesidir ve kapsanan öğe türlerinin yüzdesi olarak ifade edilir. Örneğin, bir mobil uygulama için testler ve desteklenen cihazlar arasında izlenebilirlik kullanılarak, uyumluluk testiyle ele alınan cihazların yüzdesi hesaplanabilir ve potansiyel olarak kapsama boşlukları belirlenebilir.

İşlevsel olmayan test tasarımı ve yürütmesi, bir tasarımın veya teknolojinin doğal zayıflıkları (örneğin, belirli programlama dilleriyle ilişkili güvenlik açıkları) veya belirli kullanıcı tabanı (örneğin, sağlık tesisi yönetim sistemleri kullanıcıların kişilikleri) gibi özel beceri veya bilgileri içerebilir.

İşlevsel olmayan kalite özelliklerinin test edilmesine ilişkin daha fazla ayrıntı için ISTQB-CTAL-TA, ISTQB-CTAL-TTA, ISTQB-CTAL-SEC ve diğer ISTQB® uzman modüllerine bakılabilir.

2.3.8 Degisim Tabanli (Change-related) Testing

Bir sistemde, bir kusuru düzeltmek icin ya da yeni veya değişen işlev(feature) nedeniyle bir değişiklik yapıldığında, değişikliklerin kusuru düzelttiğini veya işlevi doğru bir şekilde uyguladığını ve öngörülemeyen olumsuz sonuçlara neden olmadığını doğrulamak için test yapılmalıdır.

- Onay testi(Confirmation testing): Bir hata giderildiginde, yeni yazılım sürümünde yeniden çalıştırılacak ve onceki hata nedeniyle başarısız olan tüm test senaryoları kullanilarak yeni surum test edilebilir. Yazılımı, kusuru düzeltmek için gereken değişiklikleri kapsayacak şekilde test edilebilmek icin yeni testler de eklenebilir. En azından, kusurun neden olduğu arızayı/arızaları yeniden üretme adımları yeni yazılım sürümünde yeniden calistirilmalidir. Onay testinin amacı, orijinal kusurun başarılı bir şekilde giderildiğini teyit etmektir.
- Regresyon testi: İster düzeltme ister başka nedenle , kodun bir bölümünde yapılan değişikliğin, aynı bileşen(component) içinde, ayni sistemin farkli bilesenlerinde ve hatta diğer bazi sistemlerin bileşenlerindeki bazi bölümlerinin davranışını yanlışlıkla etkilemesi mümkündür. Yapilan degisiklik bazen kodun kendisinde degil, bir işletim sisteminin veya veritabanı yönetim sisteminin yeni bir sürümünun kullanilmasi gibi ortamdaki değişiklikleri de içerebilir. Bu tür istenmeyen yan etkilere(side-effects) regresyon denir. Regresyon testi, bu tür istenmeyen yan etkileri tespit etmek için testler yapmayı içerir.

Onay testi ve regresyon testleri tüm test seviyelerinde gerçekleştirilir.

Özellikle yinelemeli(iterative) ve artımlı(incremental) geliştirme yaşam döngülerinde (örneğin, Agile), yeni özellikler, mevcut özelliklerdeki değişiklikler ve kodun yeniden düzenlenmesi, kodda sık sık değişikliklere neden olur ve bu da yapilan değişikliklerle ilgili testler yapilmasini gerektirir. Sistemin gelişen doğası nedeniyle, onay ve regresyon testleri çok önemlidir. Bu, özellikle bireysel nesnelerin (örneğin cihazlar) sıklıkla güncellendiği veya değiştirildiği Nesnelerin İnterneti(Internet of Things) sistemleri için geçerlidir.

Regresyon testi paketleri birçok kez çalıştırılır ve genellikle yavaş gelişir, bu nedenle regresyon testi otomasyon için güçlü bir adaydır. Bu testlerin otomasyonu projede erken başlamalıdır (bkz. Bölüm 6).

4.2 Black-box Test Teknikleri

4.2.1 Esdeger Aralik (Equivalence Partitioning)

Esdeger Aralik Teknigi, belirli bir bölümdeki tüm üyelerin aynı sonuclari uretmesi beklenecek şekilde bölümlere (eşdeğerlik sınıfları olarak da bilinir) ayrilmasina dayanir. (bkz. Kaner 2013 ve Jorgensen 2014). Hem geçerli hem de geçersiz değerler için esdeger araliklar bulunur.

• Geçerli değerler, bileşen veya sistem tarafından kabul edilmesi gereken değerlerdir. Geçerli değerler içeren bir esdeger araliga "geçerli esdeger aralik(valid equivalence partition)" denir.

• Geçersiz değerler, bileşen veya sistem tarafından reddedilmesi gereken değerlerdir. Geçersiz değerler içeren bir esdeger araliga "geçersiz esdeger aralik(invalid equivalence partition)" denir.

• Girişler, çıkışlar, dahili değerler, zamanla ilgili değerler (örneğin bir olaydan önce veya sonra) ve arayüz parametreleri (örneğin entegrasyon testi sırasında test edilen entegre bileşenler) dahil olmak üzere test nesnesiyle ilgili herhangi bir veri öğesi için bölümler tanımlanabilir.

• Gerekirse herhangi bir bölüm alt bölümlere ayrılabilir.

• Her değer sadece bir esdeger araliga ait olmalıdır.

• Test senaryolarında geçersiz esdeger aralik kullanıldığında, hataların maskelenmediğinden emin olmak için esdeger araliklar ayrı ayrı test edilmelidir, baska esdeger aralikla birleştirilmemelidir. Aynı anda birkaç hata oluştuğunda iclerinden ancak biri görünür ve digerleri maskelenebilir, bu da diğer hataların algılanmamasına neden olur.

Bu teknikle %100 kapsama elde etmek için, test senaryoları, her bir esdeger aralikdan en az bir değer kullanarak tanımlanan tüm esdeger araliklari(geçersiz esdeger araliklar dahil) kapsamalıdır. Bu teknikte tum durumlari kapsama(coverage) yuzdesi, en az bir değerle test edilen eşdeğer aralik sayısının, tanımlanmış eşdeğer aralik toplam sayısına bölünmesiyle hesaplanir. Esdeger Aralik tüm test seviyelerinde uygulanabilir.

4.2.2 Sınır Değer Analizi (Boundary Value Analysis)

Sınır değer analizi (BVA), Esdeger Aralik Tekniginin bir uzantısıdır, ancak yalnızca sayısal veya sıralı verilerden oluşan araliklar olusturmak mumkun oldugunda kullanılabilir. Bir araliktaki minimum ve maksimum değerler(veya ilk ve son değerleri) sınır değerleri olarak kabul edilir (bkz. Beizer 1990).

Örneğin, bir programin, tamsayı olmayan girişleri engellemek için girişleri sınırlamak amaciyla bir tuş takımı kullanarak tek bir tamsayı değerini giriş olarak kabul ettiğini varsayalım. Geçerli aralık 1'den 5'e kadar olsun(sinirlar dahil). Dolayısıyla, üç esdeger aralik vardır: geçersiz (çok düşük); geçerli; geçersiz (çok yüksek). Geçerli esdeger bolge için sınır değerleri 1 ve 5'tir. Geçersiz (çok yüksek) bolge için sınır değeri 6'dır. Geçersiz (çok düşük) bolge için sınır değeri 0'dır.

Yukarıdaki örnekte, sınır başına iki sınır değeri belirledik. Geçersiz (çok düşük) ve geçerli arasındaki sınır, 0 ve 1 test değerlerini verir. Geçerli ve geçersiz (çok yüksek) arasındaki sınır, 5 ve 6 test değerlerini verir. Bu tekniğin bazı varyasyonları, sınır başına üç sınır değeri tanımlar: sınırdan önceki, sınırdaki ve sınırın hemen üzerindeki değerler. nceki örnekte, üç noktalı sınır değerleri kullanılarak, alt sınır test değerleri 0, 1 ve 2'dir ve üst sınır test değerleri 4, 5 ve 6'dır (bkz. Jorgensen 2014).

Esdeger araliklarda sınırlardaki degerlerin hata verme olasiligi, aralik içindeki diger degerlerin hata verme olasiligindan daha yüksektir. Belirlenen sinirlar uzerinde degisiklik yapilabiecegini hatirlamak onemlidir. Hem onceden tanimlanan hem de uygulanan sınırların, amaçlanan konumlarının altina veya ustune kaydırılmasi, sinirin tamamen kaldirilmasi veya daha onceden belirtilmeyen yeni ek sınırlar tanimlanmasi mumkundur. Sınır değer analizi ve testi, kullandigi sinir degerleri sayesinde yazılımı kullanilan değerin ait olması gereken araliktan farklı bir aralikta olabilecek davranışlar göstermeye zorlayarak bu tür kusurların neredeyse tamamını ortaya çıkaracaktır.

Sınır değer analizi tüm test seviyelerinde uygulanabilir. Bu teknik genellikle bir dizi sayıyı (tarihler ve saatler dahil) gerektiren gereksinimleri(requirements) test etmek için kullanılır.

Bu teknikte tum durumlari kapsama(coverage) yuzdesi, test edilen sınır değerlerinin sayısının, tanımlanmış sınır testi değerlerinin toplam sayısına bölünmesiyle hesaplanir.

5.1.2 Test Lideri ve Tester'in Gorevleri

Bu syllabus'ta iki test pozisyonundan bahsedilmektedir; test lideri ve test uzmanı(tester). Bu iki rol tarafından gerçekleştirilen faaliyetler ve görevler, proje ve ürün bağlamına(context), rollerdeki kişilerin yeteneklerine ve organizasyona bağlı olarak degisiklikler gosterebilir.

Test yöneticisi, test sürecinin genel sorumluluğunu ve test faaliyetlerinin başarılı sekilde yurutulmesini saglayacak liderlik gorevini ustlenir. Test yöneticiligi rolü, profesyonel bir test yöneticisi, bir proje yöneticisi, bir development yöneticisi veya bir kalite güvence yöneticisi tarafından gerçekleştirilebilir. Daha büyük projelerde veya organizasyonlarda, birkaç test ekibi bir test yöneticisine, test koçuna veya test koordinatörüne rapor verebilir, her takımın başında bir test lideri(test leader) veya lider test uzmanı(lead tester) bulunur.

Genel olarak test lideri görevleri şunları içerebilir:

- Organizasyon için bir test politikası ve test stratejisi olusturmak veya var olanlari kontrol altinda tutmak
- Proje kapsamini(context) dikkate alarak ve test amaçlarını ve risklerini gozonunde bulundurarak test faaliyetlerini planlamak. Bu planlama, test yaklaşımlarını seçmeyi, yaklasik test süresini hesaplamayi, gerekli is gucu ve maliyeti tahmin etmeyi, kaynakları hazirlamayi, test seviyelerini ve test döngülerini tanımlamayı ve hata(defect) yönetimini planlamayı içerebilir.
- Test plan(lar)ını yazmak ve güncellemek
- Test plan(lar)ını proje yöneticileri, product owner ve diğer paydaslarla ile koordine etmek
- Entegrasyon planlaması gibi diğer proje faaliyetlerini test perspektifleri ile uyumlu hale getirmek icin iletisimde bulunmak
- Testlerin analizini, tasarımını, uygulamasını ve yürütülmesini başlatmak, test ilerlemesini ve sonuçlarını izlemek ve çıkış kriterlerinin(exit criteria) durumunu (veya definition of done) kontrol etmek ve test tamamlama faaliyetlerini kolaylaştırmak
- Toplanan bilgilere dayalı olarak test ilerleme raporları ve test özet raporları hazırlamak ve sunmak

- Test sonuçlarına ve ilerlemeye gore planlamayı uyarlamak (bazen test ilerleme raporlarında ve/veya projede tamamlanmış diğer testler için test özet raporlarında belgelenir) ve test kontrolü için gerekli tüm önlemleri almak
- Hata yönetim sisteminin(defect management system) kurulmasını ve test yazılımlari icin uygun konfigürasyon yönetimini desteklemek
- Test ilerlemesini ölçmek ve testin ve ürünün kalitesini değerlendirmek için uygun olcum sistemleri tanimlamak
- Test araçlarinin(tool) seçimi (ve gerekirse satın alma ve/veya destek) için bütçe önermek, pilot projeler için zaman ve isgucu ayırmak ve test aracının kullanımına sürekli destek sağlamak dahil, test sürecini desteklemek için test araçların seçimini ve uyarlanmasini(implementation) desteklemek
- Test ortamının/ortamlarının uyarlanmasina(implementation) karar vermek
- Organizasyon içinde test uzmanlarını(Tester), test ekibini ve test mesleğini tanıtmak ve savunmak
- Test uzmanlarının becerilerini ve kariyerlerini geliştirmek (eğitim planları hazirlamak, performans değerlendirmeleri yapmak, koçluk vb. gibi yontemlerle)

Test yöneticisi rolünün gerçekleştirilme şekli, yazılım geliştirme yaşam döngüsüne bağlı olarak değişebilir. Örneğin, Agile development'da, yukarıda bahsedilen görevlerden bazıları Agile team tarafından gerçekleştirilir, özellikle zaten grup icinden bir tester'in gunluk yaptigi testlerle ilgili gorevler grup uyelerinden birine verilebilir. Birden fazla takımı veya tüm organizasyonu kapsayan veya personel yönetimi ile ilgili olan görevlerden bazıları, geliştirme ekibinin dışındaki ve bazen test koçları olarak adlandırılan test yöneticileri tarafından yapılabilir. Test sürecini yönetme hakkında daha fazla bilgi için Black 2009'a bakılabilir.

Genel olarak tester'in görevleri şunları içerebilir:
- Test planlarını gözden geçirmek ve katkıda bulunmak
- Test edilebilirlik(testability) için gereksinimleri(requirement), kullanıcı hikayelerini(user story), kabul kriterlerini(acceptance criteria), spesifikasyonları ve modelleri analiz etmek, kontrol etmek ve değerlendirmek (yapilan test cercevesinde)
- Test koşullarını(test conditions) tanımlamak ve dokumantasyonunu yapmak ve test senaryoları(test case), test koşulları(test conditions) ve test temeli(test basis) arasındaki izlenebilirliği ortaya koymak
- Genellikle sistem yönetimi ve ağ yönetimi ile koordineli olarak test ortam(lar)ını tasarlamak, kurmak ve doğrulayın(verify)
- Test senaryoları ve test prosedürleri tasarlamak ve uygulamak
- Test verilerini hazırlamak veya var olan verileri almak
- Ayrıntılı test yürütme cizelgesi oluşturmak
- Testleri calistirmak(execute), sonuçları değerlendirmek ve beklenen sonuçlardan sapmaları dokumante etmek
- Test sürecini kolaylaştırmak için uygun araçları kullanmak
- Testleri gerektiği gibi otomasyon yapmak (bir developer veya test otomasyon uzmanı tarafından desteklenebilir)
- Performans verimliliği(performance efficiency), güvenilirlik(reliability), kullanılabilirlik(usability), güvenlik(security), uyumluluk(compatibility) ve taşınabilirlik(portability) gibi non-functional ozellikleri değerlendirmek
- Başkaları tarafından geliştirilen testleri gözden geçirmek

Test analizi, test tasarımı, belirli test türleri veya test otomasyonu üzerinde çalışan kişiler bu rollerde uzman olabilir. Ürün ve proje ile ilgili risklere ve seçilen yazılım geliştirme yaşam döngüsü modeline bağlı olarak, farklı kişiler farklı test seviyelerinde tester rolünü üstlenebilir. Örneğin, bileşen

testi(component testing) düzeyinde ve bileşen entegrasyonu testi(component integration testing) düzeyinde, tester rolü genellikle developer tarafından gerceklestirilir. Kabul testi(acceptance test) düzeyinde, tester rolü genellikle iş analistleri(business analyst), konu uzmanları ve kullanıcılar(user) tarafından gerceklestirilir. Sistem testi(system test) düzeyinde ve sistem entegrasyonu(system integration) test düzeyinde, tester rolü genellikle bağımsız bir test ekibi tarafından gerceklestirilir. Operasyonel kabul(operational acceptance) testi seviyesinde, tester rolü genellikle operasyon ve/veya sistem yönetim personeli tarafından yapılır.

5.8.3 Giris(Entry) ve Cikis (Exit) Kriterleri (Definition of Ready, Definition of Done)

Yazılımın ve testin kalitesi üzerinde etkili bir kontrol uygulamak için, belirli bir test faaliyetinin ne zaman başlaması gerektiğini ve faaliyetin ne zaman tamamlandığını tanımlayan kriterlerin olması tavsiye edilir. Giriş kriterleri(Entry criteria) (daha tipik olarak Agile geliştirmede hazır tanımı “definition of ready” olarak adlandırılır), belirli bir test faaliyetini üstlenmek için ön koşulları tanımlar. Giriş kriterleri karşılanmadan baslayan testing faaliyetinin, daha zor, daha fazla zaman alan, daha maliyetli ve daha riskli olması muhtemeldir. Çıkış kriterleri(Exit Criteria veya daha tipik olarak Agile geliştirmede “definition of done” olarak adlandırılır), bir test seviyesi veya tamamlanmış bir dizi test beyan etmek için hangi koşulların elde edilmesi gerektiğini tanımlar. Her test seviyesi ve test türü için giriş ve çıkış kriterleri tanımlanmalıdır ancak bu kriterler test hedeflerine göre farklılık gösterebilir.

Genelde giriş kriterleri şunları içerir:

• Test edilebilir gereksinimlerin(requirements), kullanıcı öykülerinin(user stories) ve/veya modellerin mevcudiyeti (örn., model tabanlı bir test stratejisi izlerken)

• Bir önceki test seviyesi için çıkış kriterlerini(exit criteria) karşılayan test öğelerinin mevcudiyeti

• Uygun test ortamının(environment) varligi

• Test verilerinin(test data) ve diğer gerekli kaynakların(resources) mevcudiyeti

Genelde çıkış kriterleri şunları içerir:

• Planlanan testlerin uygulanmis olmasi

• Tanımlanmış bir kapsam düzeyine (örn. gereksinimler, kullanıcı öyküleri(user stories), kabul kriterleri(acceptance criteria), riskler, kod) ulaşılmış olmasi

• Çözülmemiş kusurların sayısının, üzerinde anlaşmaya varılan bir limit dahilinde olmasi

• Tahmini kalan kusur sayısının yeterince düşük olmasi

• Değerlendirilen güvenilirlik(reliability), performans verimliliği(efficiency), kullanılabilirlik(usability), güvenlik ve diğer ilgili kalite özelliklerinin yeterli seviyede olmasi.

Çıkış kriterleri karşılanmasa bile, harcanan bütçe, planlanan sürenin tamamlanması ve/veya ürünü piyasaya sürme baskısı nedeniyle test faaliyetlerinin kısıtlanması da yaygındır. Proje paydaşları(stakeholders) ve işletme sahipleri, daha fazla test yapmadan yaşama geçme riskini gözden geçirip kabul ederse, bu tür koşullar altında testin sona erdirilmesi kabul edilebilir.