### Slide 1

Bu noktaya kadar test etme mekaniğini ve test edilen programın boyutunu büyük ölçüde göz ardı ettik.

Ancak, büyük programların (örneğin 500'den fazla ifade veya 50'den fazla sınıf içeren programlar) testi için ne yapmalıyız?

Bu bölümde büyük bir programın testini yapılandırmada ilk adımı ele alacağız: modül veya birim testi.

Kitabınızın takip eden 6. ve 7. bölümlerinde ise kalan adımları işleyeceğiz.

### Slide 2

Modül testi, bir programın alt bileşenlerini, örneğin alt programlar, sınıflar veya prosedürler gibi, bireysel olarak test ederiz.

Bunu yapmanın üç motivasyonumuz vardır.

Programı bir bütün olarak ele almak yerine, daha küçük birimlere odaklanarak süreci daha yönetilebilir hale getiriyoruz. Ayrıca her bir modüle ayrı ayrı odaklanmamız daha geniş bir test kapsamına ulaşmamızı sağlıyor.

Büyük programların test edilmesini basitleştirir ve hata ayıklama sürecini hızlandırır.

Son olarak birden fazla bileşeni aynı anda test etmemize olanak tanır ve süreçte paralellik sağlar.

### Slide 3

Birim testinin amacı modülün işlevselliğini, belirlenen işlevsel spesifikasyonlarla karşılaştırmaktır.

Burada amaıcn, modülün doğru çalıştığını kanıtlamak olmadığına, amacın belirtilen gerekliliklere uymadığı durumları tespit etmek olduğuna dikkat edelim.

Modül testi, hataların erken dönemde belirlenmesini sağlar ve genel yazılım kalitesini artırır.

Ayrıca sonraki entegrasyon testleri için temel oluşturur.

### Slide 4

İlk olarak, modülün gereksinimlerine ve kaynak koduna dayanarak kapsamlı test senaryoları tasarlarız.

Daha sonra, modüllerin hangi sırayla test edileceğini ve entegre edileceğini belirleriz.

Hem beyaz kutu hem de siyah kutu test tekniklerini birleştirerek daha etkili bir test süreci oluştururuz.

Buradaki asıl hedefimiz, hataları belirlemek ve yazılımın doğruluğunu test etmektir.

Konuya burada listelenen 1 ve 2 numaları başlıkları (Test durumlarının tasarlanması ve Test sırası) detaylı anlatarak işleyeceğiz. Son olarak 3. bir alt bölüm daha işlenecek. Orada da testleri gerçekleştirirken size faydalı olabilecke bazı öneriler olacak.

### Slide 5

Test tasarımı için iki ana kaynağa ihtiyacımız var:

Spesifikasyon: Modülün giriş, çıkış ve işlevlerini tanımlar.

Kaynak kod: Beyaz kutu analizi için iç mantığı sağlar.

Çıktılar, beklenen işlevsel davranışlarla uyumlu olmalıdır.

Aynı zamanda hata kodlarının ve sınır koşullarının doğrulanması gerekir.

### Slide 6

Modül testi büyük ölçüde beyaz kutu odaklıdır.

Bunun için iki sebep gösterebiliriz:

Bir sebep, daha büyük varlıkları, örneğin tüm programları (sonraki test süreçleri için geçerli olacak) test ettikçe beyaz kutu testinin daha az uygulanabilir hale gelmesidir.

İkinci bir sebep, sonraki test süreçlerinin farklı hata türlerini (örneğin, programın mantığıyla mutlaka ilişkili olmayan hatalar, örneğin programın kullanıcılarının gereksinimlerini karşılayamaması) bulmaya yönelik olmasıdır.

### Slide 7

Bu nedenle, bir modül testi için test durumu tasarım prosedürü şu şekildedir:

Modülün mantığını bir veya daha fazla beyaz kutu yöntemi kullanarak analiz edin ve ardından bu test durumlarını modülün spesifikasyonuna kara kutu yöntemlerini uygulayarak tamamlayın.

### Slide 8

Kullanacağımız test vakası tasarım yöntemleri Bölüm 4'te anlatılıyordu ve sizlere son haftalarda anlattık.

Şimdi bu yöntemleri modül testinde kullanımını bir örnekle göstereceğiz.

BONUS adlı bir modülü test etmek istediğimizi ve işlevinin en büyük satış geliri olan departman veya departmanlardaki tüm çalışanların maaşına 2.000$ eklemek olduğunu varsayalım.

Ancak, uygun bir çalışanın mevcut maaşı 150.000$ veya daha fazlaysa veya çalışan bir yöneticiyse, maaş yalnızca 1.000$ artırılacaktır.

Çıktılar:

Hata kodu 0: Başarılı çalışma.

Hata kodu 1: Çalışan veya departman tablosu boş.

Hata kodu 2: Yüksek satışlı bir departmanda uygun çalışan bulunamadı.

### Slide 9

Slayttaki tablolar BONUS programı için girdi verilerini temsil etmektedir.

Employee Table (Çalışan Tablosu) çalışanların blilgilerini tutmak için kullanılıyor.

Name (İsim): Çalışanın adını temsil eder.

Job Code (İş Kodu): Çalışanın iş pozisyonunu veya rolünü gösterir. Örneğin, bir çalışan yönetici (Manager) olarak kodlanmış olabilir.

Dept. (Departman): Çalışanın hangi departmanda çalıştığını belirtir.

Salary (Maaş): Çalışanın mevcut maaşını içerir.

Department Table (Departman Tablosu) ise hangi bölümün ne kadar satış yaptığı bilgisini tutuyor.

Bu tablo sayesinde hangi bölümdeki çalışanların maaşının arttırılacağı belirleniyor.

Ardından ise Çalışan tablosu kullanılarak çalışanlar bu bölümdeki çalışanlar bulunuyor ve maaşları tabloyu güncelleyerek güncelleniyor.

### Slide 10

1-6: Giriş parametresi olan EMPTAB tanımlanır. Bu tablo çalışan bilgilerini içerir: isim (NAME), iş kodu (CODE), departman (DEPT), ve maaş (SALARY).

7-9: DEPTTAB tanımlanır. Bu tablo departman bilgilerini içerir: departman kodu (DEPT) ve satış rakamları (SALES).

10-11: ESIZE ve DSIZE giriş tablolarındaki kayıt sayılarını temsil eder. ERRCODE ise işlemin durumunu göstermek için kullanılan çıkış parametresidir.

12: MAXSALES değişkeni, DEPTTAB tablosundaki maksimum satış değerini tutar. Başlangıç değeri 0’dır.

13: I, J, K sayaç değişkenleri olarak kullanılır.

14: FOUND bayrağı, ele alınan bir departmanın çalışanı olup olmadığını takip etmek için kullanılacak.

15-17: Bonus miktarları (SINC ve LINC) ve maaş sınırı (LSALARY) tanımlanır.

18: MGR değişkeni, yönetici pozisyonunu temsil eder.

### Slide 11

1: ERRCODE başlangıç değeri olarak 0 atanır (başarı durumu).

2-3: Eğer ESIZE veya DSIZE sıfır ya da negatifse, ERRCODE 1 yapılır. ESIZE ve DSIZE giriş tablolarındaki kayıt sayılarını tutuyordu. Bu nedenletablolar boş olduğundan prosedür 1 hata kodunu döndürür.

5-7: Bu satır aralıklarında ilk döngümüz var. Bu döngü ile DEPTTAB tablosundaki en yüksek satış (MAXSALES) değeri bulunur.

8-11: İkinci döngü başlıyor. Bu döngüde en yüksek satış yapan departmanlar belirleniyor ve bu deprartmanlarda çalışanların maaşları güncellenecek. Öncelikle 9. satırda o dönüşte ele alınan departmaın satış rakamı MAXSALES mi o kontrol ediliyor. Eğer öyleyse 11. satırda FOUND değişkeni 0 yapılıyor.

FOUND bayrağı, ele alınan bir departmanın çalışanı olup olmadığını takip etmek için kullanılıyordu. Böylece uygun çalışan olmadığını varsayarak sonraki döngüye başlayacağız.

12: Üçüncü döngü, tüm çalışanların bulunduğu EMPTAB çalışan tablosu üzerinde dönüyor. Her br çalışan tek tek inceleniyor.

13: İlgili çalışanın departman kodu ile J. departmanının eşleşip eşleşmediğine bakıyoruz. Yani bu çalışanın daha önceki döngüde seçilen çok satış yapan departmanda çalışıp çalışmadığına bakıyoruz.

15: Eğer çalışan bulunduysa, FOUND bayrağı 1 yapılır. Yani bu departmanda en az bir çalışan var demek oluyor. Bunu yapmamızın nedenini ileride göreceğiz. Çok satış yapan deparmanda çalışan yoksa 2 hata kodunu döndüreceğiz.

16-19: Çalışanın maaşı ve iş koduna göre bonus miktarı eklenir:

150.000 $ veya daha fazla maaşı olanlar ve yöneticiler LINC (1.000 $) alır.

Diğer çalışanlar SINC (2.000 $) alır.

21: Eğer uygun departmanda çalışan bulunamazsa, ERRCODE 2 yapılır.

22-25: Program sonlanır.

### Slide 12

Böylece testini yapacağımız modülü açıkladık. Şimdi testine odaklanalım.

Hatırlarsak bir modülü test derken, modülün mantığını bir veya daha fazla beyaz kutu yöntemi kullanarak analiz edip beyaz kutu test durumlarını oluşturmamız ve ardından kara kutu yöntemlerini de uygulayarak testi tamamlamalıydık.

Şimdi modülün mantığını beyaz kutu testleri ile analiz etme adımı ile başlayalım.

Modül mantığını analiz ederken tüm koşullu kararları inceleriz.

Örneğin:

Tablo boş mu kontrolü.

Satışları karşılaştırma ve uygun departmanı belirleme.

Çalışan uygunluk kontrolleri.

Kasama kriteri: Bu adımlar, kapsamlı bir test süreci oluşturmak için gereklidir.

### Slide 13

Hatırlarsak beyaz kutu testleir için farklı yöntemlerimiz vardı. Bunlar bu slaytta hatırlatma amacıyla listelendi.

Karar Kapsamı: Her kararın doğru/yanlış sonuçlarını test eder.

Koşul Kapsamı: Kararlardaki her koşulun doğru/yanlış sonuçlarını test eder.

Çoklu Koşul Kapsamı: Kararlardaki tüm koşulların tüm olası kombinasyonlarını test eder.

Bir sonraki slaytta bu modülde az sayıda kararımız var.

Bu yüzden de muhtemelen bu yöntemler arasından çok koşullu kapsamı (multicondition coverage) seçmeliyiz.

Ancak seçimin etkilerini görmek için mantık kapsamı kriterlerinin hemen hemen hepsini inceleyeceğiz.

Sadece ifade kapsamını (statement coverage) hemen hemen her zaman kullanışlı olmak için çok sınırlı olduğundan eliyoruz.

### Slide 14

Hangi mantık kapsamı tekniğini kullanırsanız kullanın, ilk adım programdaki koşullu kararları listelemektir.

Bu programdaki adaylar tüm IF ve DO ifadeleridir.

Programı incelediğimizde, tüm DO ifadelerinin basit yinelemeler olduğunu görürüz.

Her yineleme sınırının başlangıç ​​değerine eşit veya ondan büyük olacağını (yani her döngü gövdesinin her zaman en az bir kez yürütüleceği anlamına gelir) ve her döngüden çıkmanın tek yolunun DO ifadesi olduğunu görebiliriz.

Bu nedenle, bu programdaki DO ifadelerinin özel bir ilgiye ihtiyacı yoktur.

DO ifadelerini dışarıda brıraktığımızda, analiz etmek için geriye şu 6 IF ifadeleri kalır.

IF ifadelerinin solunda satır numaraları verilmiştir.

Az sayıda IF ifadesi olduğundan, muhtemelen çok koşullu kapsamı seçmeliyiz, ancak etkilerini görmek için tüm mantık kapsamı kriterlerini (her zaman kullanışlı olmak için çok sınırlı olan ifade kapsamı hariç) inceleyeceğiz.

### Slide 15

Karar kapsamı kriterini karşılamak için, altı kararın her birinin her iki sonucunu da çağırmak için yeterli test vakasına ihtiyacımız var.

Tüm karar sonuçlarını çağırmak için gereken girdi durumları slayttaki tabloda listelenmiş durumda.

Sonuçlardan ikisi her zaman gerçekleşeceğinden, test vakaları tarafından zorlanması gereken 10 durum var.

Bu tabloyu oluşturmak için, karar-sonuç durumlarının uygun karşılık gelen girdi koşullarını belirlemek ve bunun içinde program mantığının geriye doğru izlenmesi gerektiğine dikkat ediniz.

### Slide 16

Bir öncki slaytta verilen girdi durumları bu slaytta gösterilen iki test durumu tarafından çağrılabilir.

Her test durumunun, 2. Bölüm'de tartışılan ilkelere bağlı kalarak beklenen çıktının bir tanımını içerdiğine dikkat edin.

Bu iki test durumu karar kapsamı kriterini karşılasa da, modülde bu iki test durumu tarafından tespit edilemeyen birçok hata türü olabileceği açıktır.

Örneğin, test durumları hata kodunun 0 olduğu, bir çalışanın yönetici olduğu veya departman tablosunun boş olduğu (DSIZE<=0) durumları incelemez.

### Slide 17

Koşul kapsamı kriterini kullanarak daha tatmin edici bir test elde edilebilir.

Bu yöntemde, bildiğiniz gibi, kararlardaki her koşulun her iki sonucunu da çağırmak için yeterli test vakasına ihtiyacımız var.

Tüm sonuçları çağırmak için koşullar ve gerekli girdi durumlarını bu ve bundan sonraki slaytta gösterilen tabloda listeledim.

Tüm koşullar tek slayta sığmayacağından iki slayta bölüştürüldü.

İlk 3 IF cümlesine ait koşullar veya girdi durumları bu slaytta gösteriliyor.

### Slide 18

Kalan 3 IF ifadesi için üretilen girdi durumları ise burada.

8 girdi durumu tanımlandığından toplam 16 durum göz önüne laınmalı.

Ancak bu durumlardan 2 tanesi gene her zaman gerçekleştiriğnden onları çıkardığımızda toplam 14 durumu kapsayan test durumlarını oluşturmalıyız.

### Slide 19

Bu durumlar daha önce olduğu gibi yalnızca iki test vakası tarafından çağrılabilir.

Bu slaytta bu yeni test vakalarını görüyorsunuz.

Bu test vakaları bir önceki tabloda belirlediğimiz tüm sonuçları çağırdıkları için, kapsam kriterini karşılıyorlar.

Ancak bu test vakaları özellikle bir sorunu göstermek için tasarlanmış.

Aslında bu test vakaları "karar kapsam kriterini" (decision coverage) karşılaması için hazırladığımız test vakalarına(3 slayt önce) göre daha zayıf bir test vakası kümesi.

Yani normal şartlarda daha kapsamlı test yapmanızı sağlaması beklenen condiiton coverage seçtiğimiz test vakaları nedeniyle decision coverage'a göre daha az olası sorunu test ediyor.

Çünkü bu yeni test vakaları, her ifadeyi yürütmüyorlar.

### Slide 20

Örneğin, ifade 18 asla yürütülmez.

Gene her iki yöntemin de test edemediği durumlar bulunmakta.

Daha önce verdiğimiz örnk burada da geçerli. Bu test vakaları ERRORCODE=0 çıktı durumuna neden olmazlar.

Elbette, alternatif bir test vakası seti bu sorunları çözebilir, ancak burada önemli olan iki test vakasının koşul kapsamı kriterini karşıladığı gerçeğidir.

Yani seçeceğniz test durumlarına gre de yöntemin verimliliği değişebilmekte.

Ve ne kadar akıllı olursanız olun bu durumla karşılaşabilirsiniz.

### Slide 21

Karar/durum kapsamı kriterini kullanmak, bir önceki test vakası kümesindeki önemli zayıflığı ortadan kaldıracaktır.

Burada, tüm koşulların ve kararların tüm çıktılarının en az bir kez çağrılacağı yeterli test vakası sağlanması gerekir.

Burada 2 slayt önceki test case'leri gösteren tabloyu veriyorum.

Bunu decision/condition şartını sağlayacka şekilde dönüştürmek ve ifade 18'in de çalışmasını sağlamak için kullandığımız EMPTAB tablosunu güncellememiz gerekiyor.

Örneğin Jones'u yönetici yapmamız ve Lorin'i normal (yönetici olmayan) çalışan yapmamız ifade 18'in çalışmasını sğalar.

Çünkü bu değişiklik karar 16'nın her iki çıktısını da üretme sonucunu doğurur ve böylece ifade 18'i yürütmemize neden olur.

### Slide 22

Araştırılacak son ölçüt çoklu koşul kapsamıdır.

Bu ölçüt, her kararda tüm olası koşul kombinasyonlarının en az bir kez çağrılacağı yeterli test vakası gerektirir.

Bu, Tablo 5.2'den çalışılarak gerçekleştirilebilir. 6, 9, 13 ve 21 numaralı kararların her birinin iki kombinasyonu vardır; 2 ve 16 numaralı kararların her birinin dört kombinasyonu vardır.

### Slide 23

Gene test vakalarını tasarlarken şu yöntemi kullandığımızı hatırlatayım.

Önce mümkün olduğunca çok sayıda kombinasyonu kapsayan test vakalasını seçiyoruz.

Ardından kalan kombinasyonların mümkün olduğunca çoğunu kapsayan başka bir test vakası seçiyoruz ve tüm kombinasyonlar bitene kadar devam ediyor.

Bu durumda "Multicondition coverage" için 4test vakaıs seçmemiz yeterli.

Tüm test vakaları slayta sığmadığından, ilk slaytta 3 tane test vakası var. Kalan test vakası ise sonraki slaytta.

### Slide 24

Test vakalarının kümesi, önceki kümelerden daha kapsamlıdır; bu da başlangıçta bu kriteri seçmemiz gerektiği anlamına geliyor.

### Slide 25

BONUS modülünün çok sayıda hataya sahip olabileceğini ve çoklu koşul kapsamı kriterini karşılayan testlerin bile hepsini tespit edemeyeceğini fark etmek önemlidir.

Örneğin, hiçbir test vakası ERRORCODE'un 0 değeriyle döndürüldüğü durumu üretmez; bu nedenle, 1. satır eksik ya da hatalı olsaydı, hata tespit edilemezdi.

LSALARY yanlışlıkla $150.000,01 olarak başlatılırsa, hata fark edilmezdi.

16 ifadesi SALARY(K)>LSALARY yerine SALARY(K)>LSALARY ifadesini belirtmiş olsaydı, bu hata bulunmazdı.

Ayrıca, çeşitli off-by-one hatalarının tespit edilip edilmeyeceği (örneğin DEPTTAB veya EMPTAB'deki son girdinin doğru şekilde işlenmemesi) büyük ölçüde şansa bağlı olurdu.

Şimdi iki nokta belirgin olmalı:

Birincisi, çoklu koşul kriteri diğer kriterlerden daha üstündür.

İkincisi, herhangi bir mantık kapsamı kriteri modül testlerini türetmenin tek yolu olarak yeterince iyi değildir.

Dolayısıyla, önceki iki slayttaki testleri bir dizi kara kutu testiyle destekleyeceğiz.

### Slide 26

Beyaz kutu testleri yeterli değil dedik.

Bu nedenle sonraki adım test çalışmamızı kara kutu testiyle desteklemektir.

Bunu yapmak için öncelikl BONUS'un arayüz arayüz spesifikasyonuna ihtiyacımız var.

Bu spesifikasyon aşağıdaki slaytta gösteriliyor.

Modül, dört tane parametre/argüman tanımlıyor: EMPTAB, DEPTTAB ...

Ardından bu parametrelerin niteliklerini spesifik olarak belirtiyor.

EMPTAB tablosu hem input hem de output paramtresi. Employee bilgilerini tutuyor.

Her employee'nin ismi,iş tanımına ait kod, bölüm kodu ve maaşı tutuluyor.

...

### Slide 27

ESIZE ve DSIZE, sırasıyla EMPTAB ve DEPTTAB'daki giriş sayısını gösterir.

EMPTAB ve DEPTTAB'daki girişlerin sırası hakkında hiçbir varsayımda bulunulmamalıdır.

Modülün işlevi, en büyük satış tutarına sahip olan departman veya departmanlardaki çalışanların maaşını (EMPTAB.SALARY) artırmaktır (DEPTTAB.SALES).

Uygun bir çalışanın mevcut maaşı 150.000$ veya daha fazlaysa veya çalışan bir yöneticiyse (EMPTAB.CODE=‘M’), artış 1.000$’dır; değilse, uygun çalışan için artış 2.000$’dır.

Modül, artan maaşın EMPTAB.SALARY alanına sığacağını varsayar.

ESIZE ve DSIZE 0’dan büyük değilse, ERRCODE 1 olarak ayarlanır ve başka bir işlem yapılmaz.

Diğer tüm durumlarda, işlev tamamen gerçekleştirilir.

Ancak, maksimum satış departmanında çalışan olmadığı bulunursa, işleme devam edilir ancak ERRCODE değeri 2 olur; aksi takdirde, 0 olarak ayarlanır.

### Slide 28

Bu şartname neden-sonuç grafiklerine (cause-effect graphing) uygun değildir (kombinasyonları üretilebilecek ayırt edilebilir girdi koşulları kümesi yok).

Bu nedenle, sınır değer analizi (boundary valu analysis) kullanılacaktır.

Bu modül için toplam 19 adet girdi sınırları belirlenebilir.

İlk 12 tanesi bu slaytta listeleniyor.

### Slide 29

Kalan 7 girdi koşulu ise bu slaytta listeleniyor.

### Slide 30

Bu slaytta da çıktı sınırlarını (output boundaries) görüyorsunuz.

20-23 arası çıktı sınırları.

24 ise Hhata tahmin (error gussing) tekniğine dayalı üretilen bir başka test koşulu.

Bu, modülün ERRCODE=2 durumuyla karşılaştığında girdinin işlenmesini hatalı bir şekilde sonlandırıp sonlandırmadığını belirlemek için önerilmiş.

### Slide 31

Bu 24 koşulu gözden geçirdiğimizde, 2, 5 ve 8 numaralılar pratik olmayan test vakaları gibi görünüyor.

Ayrıca asla gerçekleşmeyecek koşulları da temsil ettiklerinden (genellikle test ederken tehlikeli bir varsayımdır, ancak burada güvenli görünüyor), bunları hariç tutuyoruz.

### Slide 32

Bir sonraki adım, kalan 21 koşulu mevcut test vakaları kümesiyle (beyaz kutu testinden elde ettiklerimiz) karşılaştırarak hangi sınır koşullarının henüz kapsanmadığını belirlemektir.

Bunu yaparken, 1, 4, 7, 10, 14, 17, 18, 19, 20, 23 ve 24 numaralı koşulların beyaz kutu testlerinin ötesinde test vakaları gerektirdiğini görüyoruz.

Bir sonraki adım, 11 sınır koşulunu kapsayacak ek test vakaları tasarlamaktır.

Bir yaklaşım, bu koşulları mevcut test durumlarına birleştirmektir (yani, beyaz kutu test vakalarını değiştirerek), ancak bu önerilmez çünkü bunu yapmak, mevcut test durumlarının çoklu koşul kapsamını istemeden bozabilir.

Bu nedenle, en güvenli yaklaşım yeni test durumları eklemektir.

Bunu yaparken, amaç sınır koşullarını kapsayacak şekilde gereken en az sayıda test durumu tasarlamaktır.

### Slide 33

Üç yeni test durumu bunu başarır.

Slayta sığmadığı için ilk iki test durumunu bu slaytta görüyorsunuz.

Test durumu 5, koşullar 7, 10, 14, 17, 18, 19 ve 20'yi kapsar

Test durumu 6, koşullar 1, 4 ve 23'ü kapsar.

Test durumu 7 ise, koşul 24'ü kapsar.

### Slide 34

Bu slaytta da son test durumunu, 7 . test durumunu, görüyorsunuz.

Test durumu 7, koşul 24'ü kapsamakta.

Böylece tek bir modülü hem beyaz kutu hem de kara kutu teknikleriyle nasıl testinin gerçekleştirildiğini gördük.

### Slide 35

Şu ana kadar bu sunumda öğrendiklerimizi en üst seviyeden özetleyecek olursak...

Beyaz kutu ve siyah kutu test tekniklerini birleştirerek kapsamlı bir testler oluşturmalıyız. Çünkü bir tekniğin yakalaymadığı hataları diğer bir teknik yakalayabiliyor.

Kenar durum analizi kullanmak olağan dışı durumları tespit etmek için çok faydalı bir yöntem.

Hata tespitine öncelik verin; doğru çalışmayı kanıtlamakla yetinmeyin.

Çeşitli senaryoları ele almak için birden fazla test durumu oluşturun.

### Slide 36

Modül test etme sürecini gerçekleştirirken iki temel husus olduğunu dersin başında belirtmiştik

Birinci husus şu ana kadar anlatılmış olanetkili bir test vakası setinin tasarımıdır.

İkinci husus ise birden fazla modüllü programlarda modüllerle çalışma programının nasıl olduğu.

Bu önemli bir mevuzudur.

Burada ele alnan soru şudur: Bir programı her modülü bağımsız olarak test ederek ve ardından programı oluşturmak için modülleri birleştirerek mi test etmelisiniz, yoksa test edilecek bir sonraki modülü test edilmeden önce daha önce test edilmiş modüller kümesiyle mi birleştirmelisiniz?

İlk yaklaşıma artımlı olmayan veya "büyük patlama" testi veya "büyük patlama" entegrasyonu denir.

İkinci yaklaşıma artımlı test veya artımlı entegrasyon denir.

İlk aşamada, artımlı (incremental) ve artımsız (non-incremental) test olmak üzere iki yaklaşımı ele alacağız.

Sonrasında, yukarıdan aşağıya (top-down) ve aşağıdan yukarıya (bottom-up) geliştirme veya test olmak üzere iki artımlı yaklaşımı inceleyeceğiz.

### Slide 37

...

Bu etkilerikonuyu anlattıkça daha iyi anlayacağız.

### Slide 38

### Slide 39

Slayttakii program örnek olarak kullanılmıştır.

Dikdörtgenler programdaki altı modülü (alt rutinler veya prosedürler) temsil eder.

Modülleri birbirine bağlayan çizgiler programın kontrol hiyerarşisini temsil eder.

Yani, modül A, modül B, C ve D'yi çağırır; modül B, modül E'yi çağırır; vb.

Geleneksel yaklaşım olan artımlı olmayan test şu şekilde gerçekleştirilir.

İlk olarak, altı modülün her birinde bir modül testi gerçekleştirilir ve her modül bağımsız bir varlık olarak test edilir.

Modüller, duruma bağlı olarak aynı anda veya art arda test edilebilir.

Modüllerin tamamının modül testleri tamamlandığında, modüller birleştirilerek program oluşturulur.

Burada ayrıca Stub ve Driver kavramlarından bahsetmek istiyorum.

Her modülün test edilmesi özel bir sürücü (driver) modülü ve bir veya daha fazla (stub) taslak modül gerektirir.

Örneğin, modül B'yi test etmek için önce test vakaları tasarlanır ve ardından modül B'ye, test altındaki modül aracılığıyla test vakalarını 'sürmek' veya iletmek için kodlanması gereken küçük bir modül olan sürücü modülünden giriş argümanları geçirilerek beslenir (test araçlarında sürücü modüller hazır gelir veya kolayca ayarlanabilir).

Sürücü modülü ayrıca, test eden kişiye B tarafından üretilen sonuçları göstermelidir.

Ek olarak, modül B, modül E'yi çağırdığından, B, E'yi çağırdığında kontrolü almak için bir şey mevcut olmalıdır.

Modül E'nin işlevini simüle etmek için kodlanması gereken, 'E' adı verilen özel bir modül olan taslak modül bunu başarır.

### Slide 40

Alternatif yaklaşım artımlı testtir.

Her modülü ayrı ayrı test etmek yerine, test edilecek bir sonraki modül öncelikle daha önce test edilmiş modül kümesiyle birleştirilir.

Slayttaki programı artımlı olarak test etmek için bir prosedür vermek için henüz erken, çünkü çok sayıda olası artımlı yaklaşım var.

Ama artımlı testi genel hatlarıyla anlatabilmek için önemli bir seçim yapmalıyız.

O da programın en üstünden mi yoksa en altından mı modül testlerini yapmaya başlayacağımız seçimidir.

Ancak, bu konuyu ilerleyen slaytlarda tartıştığımız için, şimdilik en alttan başladığımızı varsayalım.

İlk adım, E, C ve F modüllerini paralel olarak (üç kişi tarafından) veya seri olarak test etmektir.

Her modül için bir sürücü hazırlamamız gerektiğini, ancak bir taslak hazırlamamamız gerektiğini unutmayın.

Bir sonraki adım, B ve D'yi test etmektir; ancak bunları ayrı ayrı test etmek yerine, sırasıyla E ve F modülleriyle birleştirilirler.

Başka bir deyişle, B modülünü test etmek için, test durumlarını içeren bir sürücü yazılır ve B-E çifti test edilir.

Bir sonraki modülün daha önce test edilen modüllerin kümesine veya alt kümesine eklenmesi olan artımlı süreç, son modül (bu durumda modül A) test edilene kadar devam eder.

Bu prosedürün alternatif olarak yukarıdan aşağıya doğru ilerleyebileceğini unutmayın.

### Slide 41

Bu noktada dahi artımlı ve artımlı olmayan test ile ilgili birkaç gözlemortaya çıkarılabilir.

Toplam 6 tane gözlemimiz var.

Bunlar ilk dördü artımlı testin avantajı olarak düşünülebilir.

Bunlara bu slaytta değineceğiz.

Artımlı testin dezavantajı olarak görebileceğimiz son 2 gözlemimiz ise diğer slaytta olacak.

(1) Artımlı olmayan test daha fazla çalışma gerektirir.

Bir önceki slaytta üzerinde konuştuğumuz program için konuşalım.

Artımlı olmayan test için beş sürücü ve beş taslak hazırlanmalıdır (en üst modül için bir sürücü modülüne ihtiyacımız olmadığını varsayarak).

Aşağıdan yukarı artımlı test beş sürücü gerektirir ancak taslak gerektirmez.

Yukarıdan aşağı artımlı test beş taslak gerektirir ancak sürücü gerektirmez.

Daha az çalışma gerekir çünkü artımlı olmayan yaklaşımda ihtiyaç duyulan sürücü modülleri (üstten başlarsanız) veya taslak modülleri (alttan başlarsanız) yerine daha önce test edilmiş modüller kullanılır.

(2) Artımlı test kullanıldığında, uyumsuz arayüzlerle veya modüller arasındaki yanlış varsayımlarla ilgili programlama hataları daha erken tespit edilir.

Bunun nedeni, modül kombinasyonlarının erken bir zamanda birlikte test edilmesidir.

Ancak, artımlı olmayan test kullanıldığında, modüller sürecin sonuna kadar ``birbirlerini görmezler''.

(3) Önceki maddenin sonucu olarak, artımlı test kullanılırsa hata ayıklama daha kolay olmalıdır.

Modüller arası arayüzlerle ilgili hataların ve varsayımların var olduğunu varsayarsak (deneyime göre iyi bir varsayım), o zaman, eğer artımlı olmayan test kullanıldıysa, hatalar tüm program birleştirilene kadar yüzeye çıkmayacaktır.

Bu sırada, programın herhangi bir yerinde olabileceğinden hatayı saptamakta zorluk çekebiliriz.

Tersine, artımlı test kullanılırsa, bu tür bir hata saptamak daha kolay olmalıdır, çünkü hatanın en son eklenen modülle ilişkili olma olasılığı yüksektir.

(4) Artımlı test daha kapsamlı testlerle sonuçlanabilir.

B modülünü test ediyorsanız, modül E veya A (alttan mı yoksa üstten mi başladığınıza bağlı olarak) yürütülür.

E veya A daha önce kapsamlı bir şekilde test edilmiş olsa da, belki de B'nin modül testinin sonucu olarak yürütülmesi yeni bir koşulu, belki de E veya A'nın orijinal testindeki bir eksikliği temsil eden bir koşulu çağıracaktır.

Öte yandan, artımlı olmayan test kullanılırsa, B'nin testi yalnızca B modülünü etkileyecektir.

Başka bir deyişle, artımlı olmayan testte ihtiyaç duyulan taslaklar veya sürücülerin yerine, artımlı testte, daha önce test edilmiş modüller alır.

Bu modüller ise stub ve sürücülere göre daha detaylı olduklarında ve gerçek dünyayı daha gerçekçi şekilde yansıttıklarından daha fazla hata üretme potansiyeline sahiptirler.

### Slide 42

(5) Artımlı olmayan yaklaşım daha az makine zamanı kullanıyor gibi görünür.

Daha önceki slaytlarda incelediğimiz modül hiyerarşisi şeklinde A modülü aşağıdan yukarıya yaklaşım kullanılarak test ediliyorsa, B, C, D, E ve F modülleri A'nın yürütülmesi sırasında yürütülür.

A'nın artımlı olmayan bir testinde, yalnızca B, C ve E için taslaklar yürütülür.

Aynısı yukarıdan aşağıya artımlı bir test için de geçerlidir. F modülü test ediliyorsa, A, B, C, D ve E modülleri F'nin testi sırasında yürütülebilir.

F'nin artımlı olmayan testinde ise, yalnızca F sürücüsü ve F'nin kendisi yürütülür.

Bu nedenle, artımlı yaklaşım kullanılarak bir test çalışması sırasında yürütülen makine talimatlarının sayısı, artımlı olmayan yaklaşım için olandan açıkça daha fazladır.

Bunu telafi eden şey, artımlı olmayan testin artımlı testten daha fazla sürücü ve taslak gerektirmesidir; sürücüleri ve taslakları geliştirmek için makine zamanı gerektirmesidir.

(6.) Modül test aşamasının başlangıcında, artımlı olmayan test kullanıldığında paralel faaliyetler için daha fazla fırsat vardır (yani, tüm modüller aynı anda test edilebilir).

Bu, büyük bir projede (birçok modül ve kişi) önemli olabilir çünkü bir projenin personel sayısı genellikle modül test aşamasının başlangıcında en yüksek seviyededir.

Artımlı olmayan test yaklaşımında her modülün testi aynı anda oluşturulduğundan tüm çalışanlar bu işe dağıtılarak paralel şekilde testleri geliştirebilirler.

Özetle, 1'den 4'e kadar olan gözlemler artımlı testin avantajları iken, 5 ve 6 numaralı gözlemler dezavantajlarıdır.

Bilgisayar endüstrisindeki mevcut eğilimler göz önüne alındığında (donanım maliyetleri azalıyor ve azalmaya devam edecek gibi görünüyor, donanım kapasitesi artıyor ve işçilik maliyetleri ve yazılım hatalarının sonuçları artıyor) ve bir hata ne kadar erken bulunursa, onu onarmanın maliyetinin o kadar düşük olduğu gerçeği göz önüne alındığında...

1'den 4'e kadar olan gözlemlerin önem kazandığını, 5 numaralı gözlemin ise daha az önemli hale geldiğini görebilirsiniz.

6 numaralı gözlem, varsa bile, zayıf bir dezavantaj gibi görünüyor.

Sonuç olarak, bu gözlemler artımlı testin daha üstün olduğu sonucuna varmamızı sağlıyor.

### Slide 43

### Slide 44

Yukarıdan aşağıya strateji, programdaki en üstteki veya ilk modülle başlar.

Bundan sonra, kademeli olarak test edilecek bir sonraki modülü seçmek için tek bir "doğru" prosedür yoktur.

Tek kural, bir sonraki modül olmaya hak kazanmak için, modülün alt (çağrı yapan) modüllerinden en az birinin daha önce test edilmiş olmasıdır.

Slayttaki şekliu stratejiyi göstermek için kullanalım.

A'dan L'ye kadar olan kısım programdaki 12 modüldür.

Modül J'nin programın G/Ç okuma işlemlerini ve modül I'in yazma işlemlerini içerdiğini varsayalım.

İlk adım modül A'yı test etmektir.

Bunu başarmak için, B, C ve D'yi temsil eden stub modülleri yazılmalıdır.

Ne yazık ki, stub modüllerinin üretimi sıklıkla yanlış anlaşılır.

Yanlış yazılan stub modüller sıklıkla şunu yaparlar: ‘buraya kadar geldik’ diyen bir mesaj yazması hiçbir iş yapmadan çıkar.

Çoğu durumda, bu ifadeler yanlıştır. Modül A, modül B'yi çağırdığından, A, B'nin bir miktar iş yapmasını bekler

Bu çalışma büyük olasılıkla A'ya döndürülen bir sonuçtur (çıktı argümanları).

Eğer taslak basitçe denetimi döndürürse veya anlamlı bir sonuç döndürmeden bir hata mesajı yazarsa, modül A başarısız olur.

Bu hata A'daki bir hatadan dolayı değil, stuba karşılık gelen modülü simüle etmedeki başarısızlığından dolayı.

Bu nedenle, stub ların üretimi önemsiz bir görev değildir.

### Slide 45

Üst modülü test ettikten sonra, çok sayıda dizi mümkündür.

Örneğin, tüm test dizilerini gerçekleştiriyorsak, çok sayıda olası modül dizisinin dört örneği slaytın altında var.

Her biri A modülünü bitirdikten sonra farklı seçimlerle yazılım tüm modüllerini test ediyor.

Eğer paralel test gerçekleştirilirse, diğer alternatifler mümkün.

Örneğin, A modülü test edildikten sonra, bir programcı B modülünü alıp A-B kombinasyonunu test edebilir.

Aynı zamanda başka bir programcı A-C'yi test edebilir; ve üçüncüsü A-D'yi test edebilir.

Peki bu seçimlerdne hangisi daha iyi?

### Slide 46

Genel olarak en iyi sıralama diye bir şey yoktur, ancak dikkate alınması gereken iki kılavuz vard. Bunlar slaytta gösteriliyor.

(1) Programın kritik bölümleri varsa (belki de G modülü), bu bölümlerin mümkün olduğunca erken eklenmesi için diziyi tasarlayın.

‘‘Kritik bölüm’’ karmaşık bir modül, yeni bir algoritmaya sahip bir modül veya hataya meyilli olduğundan şüphelenilen bir modül olabilir.

(2) G/Ç modüllerinin mümkün olduğunca erken eklenmesi için diziyi tasarlayın.

İlkinin motivasyonu açık olmalı, ancak ikincisinin motivasyonu daha fazla tartışmayı hak ediyor.

### Slide 47

Eğer J ve I modülleri G/Ç modülleriyse ve eğer G modülü bazı kritik işlevleri yerine getiriyorsa, artımlı sıra slaytta üstte gösterildiği gibi olabilir.

A B F J D I ...

Bu sırada, altıncı artıştan sonraki test edilmiş program biçimi slaytın altında gösterilen şekilde olacaktır.

Bu durumun ilk avantajı: Test edilmiş program bu ara duruma ulaşıldığında, test vakalarının gösterimi ve sonuçların incelenmesi basitleşir.

Çünkü tüm diğer nod ların testinde ihtiyaç duyulacak temel veriler I/O nodları sayesinde sistemin içinde vardır. Artı tekrar tekrar ihtiyaç duyulabilecek kritik fonksiyonlar da gerçekleştirilmiştir. Stub gerçekleştirimleri çok daha basitleşecektir.

Başka bir avantajı da, programın çalışan bir iskelet versiyonuna sahip olmanızdır, yani gerçek giriş ve çıkış işlemlerini gerçekleştiren bir versiyon.

### Slide 48

Bu erken iskelet versiyonu:

(1) Erkenden insan faktörü hatalarını ve sorunlarını bulmanızı sağlar.

(2) Programı nihai kullanıcıya göstermeyi mümkün kılar.

(3) Programın genel tasarımının sağlam olduğuna dair kanıt görevi görür.

(4) Moral yükseltici görevi görür.

Bu noktalar yukarıdan aşağıya test stratejisinin en önemli avantajlarıdr.

### Slide 49

Bir sonraki adımımız, aşağıdan yukarıya artımlı test stratejisini incelemek.

Genellikle, aşağıdan yukarıya test, yukarıdan aşağıya testin tam tersidir.

Bu nedenle, yukarıdan aşağıya testin avantajları aşağıdan yukarıya testin dezavantajları haline gelir ve yukarıdan aşağıya testin dezavantajları aşağıdan yukarıya testin avantajları haline gelir.

Bu nedenle, aşağıdan yukarıya testin anlatımı daha kısa sürecek.

Aşağıdan yukarıya strateji, programdaki terminal modülleriyle başlar (diğer modülleri çağırmayan modüller).

Bu modüller test edildikten sonra, yine artımlı olarak test edilecek bir sonraki modülü seçmek için en iyi prosedür yoktur.

Ancak tek kural, bir sonraki modül olmaya uygun olmak için, modülün tüm alt modüllerinin (çağırdığı modüller) daha önce test edilmiş olmasıdır.

Daha öncek örnek olarak seçtiğimiz çeşitli modüllerden oluşan programımız slaytta yeniden gösterildi.

Bu programın aşağıdan yukarıya testinde ilk adım, E, J, G, K, L ve I modüllerinin bir kısmını veya tamamını seri veya paralel olarak test etmektir.

Bunu yapmak için her modülün özel bir sürücü modülüne ihtiyacı vardır. bu sürücü: test girişlerini içerir, test edilen modülü çağırır ve çıktılarını görüntüler (veya gerçek çıktıları beklenen çıktılarla karşılaştırır)

Stub'lardaki durumun aksine, sürücü modülünün test edilen modülü yinelemeli olarak çağırabilmesi nedeniyle sürücünün birden fazla sürümüne ihtiyaç duyulmaz.

Çoğu durumda, sürücü modülleri stub modüllerden daha kolay üretilir.

### Slide 50

Burada da modüllerin hangi sırada test edileceği önemli bir seçimdir.

Daha önce olduğu gibi, test sırasını etkileyen önemli bir faktör hangi modülü ne kadar kritik olduğudur.

Bu test yaklaşımında test edilmiş modülleri gösteren bir ara durum slaytta gösterilmekte.

Burada D ve F modüllerinin daha en kritik olduğuna karar verilmiş. O nedenle onlara en kısa zamanda erişecek şekilde bir modül test sırası seçilmiş.

D'yi açmak için direk J ile başlanmış.

Aynı zamanda D yi açmak için K ve L'den başlanarak H ve I üzerinden D'ye erişilmiş.

Sonraki adımlar önce E'yi test etmek ve ardından B'yi test etmek, sonra B'yi test etmek için de daha önce test edilen E, F ve J modülleriyle birleştirmek olabilir.

Aşağıdan yukarıya stratejinin bir dezavantajı, erken iskelet programı kavramının olmamasıdır.

Bu durumu slayttaki şekilden de görebilirsiniz.

Aslında, çalışan program son modül (modül A) eklenene kadar mevcut değildir ve bu çalışan program tam programdır.

G/Ç işlevleri tüm program entegre edilmeden önce test edilebilse de, erken iskelet programının avantajları mevcut değildir.

### Slide 51

Yukarıdan aşağıya ve aşağıdan yukarıya sorunu artımlı ve artımsız sorun kadar net olsaydı kullanışlı olurdu, ancak ne yazık ki öyle değil.

Bu ve sonraki slayttaki tablolar bu iki yaklaşımın avantajlarını ve dezavantajlarını listeliyor.

Karar vermenin en güvenli yolu, bu tablolardaki faktörleri test edilen program açısından tartmaktır.

Ayrıca, yukarıdan aşağıya ve aşağıdan yukarıya testlerin tek olası artımlı stratejiler olmadığının da farkında olunmalıdır.

Avatajlar

(1) Programın en üstüne doğru büyük hatalar oluştuğunda avantajlıdır.

(2) G/Ç işlevleri eklendiğinde, vakaların temsili daha kolaydır.

(3) Erken iskelet programı gösterimlere izin verir ve morali yükseltir.

Dezavantajlar

(1) Taslak modüller üretilmelidir.

(2) Taslak modüller genellikle ilk göründüklerinden daha karmaşıktır.

(3) G/Ç işlevleri eklenmeden önce, taslaklardaki test vakalarının temsili zor olabilir.

(4) Test koşullarının oluşturulması imkansız veya çok zor olabilir.

(5) Test çıktısının gözlemlenmesi daha zordur.

(6) Tasarım ve testin örtüşebileceği sonucuna varır.

(7) Belirli modüllerin test edilmesinin tamamlanmasını erteler.

### Slide 52

Avantajlar

(1) Programın alt kısmında büyük hatalar oluştuğunda avantajlıdır.

(2) Test koşullarının oluşturulması daha kolaydır.

(3) Test sonuçlarının gözlemlenmesi daha kolaydır.

Dezavantajlar

(1) Sürücü modülleri üretilmelidir.

(2) Program bir varlık olarak son modül eklenene kadar var olmaz.

### Slide 53

(1)

Modül testinin kalan kısmı, testi gerçekten gerçekleştirme eylemidir.

Bunu yapmak için bir dizi ipucu ve kılavuz burada yer almaktadır.

Bir test vakası, modülün gerçek sonuçlarının beklenen sonuçlarla uyuşmadığı bir durum ürettiğinde, iki olası açıklama vardır: ya modül bir hata içerir ya da beklenen sonuçlar yanlıştır (test vakası yanlıştır).

Bu karışıklığı en aza indirmek için, test vakaları kümesi test yapılmadan önce gözden geçirilmeli veya denetlenmelidir (yani, test vakaları test edilmelidir).

(2)

Otomatik test araçlarının kullanımı, test sürecinin sıkıcılığının bir kısmını en aza indirebilir.

Örneğin, sürücü modüllerine olan ihtiyacı ortadan kaldıran test araçları mevcuttur.

Akış analizi araçları, bir programdaki yolları sıralar, asla yürütülemeyen ifadeleri bulur (‘‘ulaşılamaz’’ kod) ve bir değişkenin bir değer atanmadan önce kullanıldığı örnekleri belirler.

(3)

Bu bölümün önceki kısımlarında olduğu gibi, beklenen sonucun tanımının bir test durumunun gerekli bir parçası olduğunu unutmamız gerekir.

Bir test yürütürken, yan etkileri (bir modülün yapmaması gereken bir şeyi yaptığı durumlar) aramayı unutmayın.

(4)

Bir kişinin kendi programlarını test etmeye çalışmasıyla ilişkili psikolojik sorunlar modül testine de uygulanır.

Programcılar kendi modüllerini test etmek yerine onları değiştirebilirler.

(5)

Son olarak, tek kullanımlık test vakalarından kaçınılmalıdır; test vakaları gelecekte yeniden kullanılabilecek bir biçimde düşünülmelidir.

### Slide 54

Modül testinin önemini şu şekilde özetleyebiliriz.

...

Son olarak: Modül test yazılımı, kapsamlı bir test prosedürünün yalnızca başlangıcıdır. Daha yüksek seviyeli testlere geçmeniz gerekecektir, ki bunları 6. Bölümde ele alacağız ve kullanıcı testlerini 7. Bölümde ele alacağız.