

BİM539, Ders 2: Test Teorisi ve Terminoloji

Beklenen vs. Gözlemlenen Davranış

- *Beklenen Davranış*: Ne olmalı?
- *Gözlenen Davranış*: Ne oldu?
- *Test*: Beklenen ve gözlenen davranışların karşılaştırılması
- *Hata*: beklenen \neq gözlenen ise
- Beklenen davranış aynı zamanda gereksinim olarak da adlandırılmaktadır

Örnek

- Sqrt fonksiyonunu test ettiğimizi varsayalım:
`// returns the square root of num`
`float sqrt(int num) { ... }`
- sqrt fonksiyonu 42 parametresi ile çağırıldığında,
- `float ret = sqrt(42);`
Beklenen: `ret == 6.48074069841`
- `float ret = sqrt(9) ise;`
Beklenen: `ret == 3`
- `float ret = sqrt(-9) ise;`
Matematiksel olarak, -9 'un kökü reel sayı olamaz ancak gereksinimlerde bu davranış belirtilmelidir.

EXHAUSTIVE TEST İMKANSIZLIĞI

- `sqrt` fonksiyonunu tüm sayılar için test etmek istediğimizi varsayalım (negatif ve pozitif sayılar)
- Parametre Java dilinde `int` (signed 32-bit integer) olsun
- Kaç farklı sayı test edilmelidir?

4,294,967,296

Peki iki farklı parametre olsaydı ne olurdu?

- `add` fonksiyonunu test ettiğimizi varsayalım:

```
// return the sum of x and y  
int add(int x, int y) { ... }
```

- Kaç farklı test çalıştırılmalı? (İpucu: `x` ve `y`'nin bütün kombinasyonları)

4,294,967,296 \wedge 2

Parametre bir dizi ise?

- `add` fonksiyonunu test ettiğimizi varsayalım:

```
// return sum of elements in A  
int add(int[] A) { ... }
```

- Kaç test çalıştırılmalı? (Not: `A` dizisi keyfi boyutta olabilir)

4,294,967,296 \wedge Infinity

Tüm parametre kombinasyonlarını
test etmek, problem olmadığını
garanti eder mi?

Tabii ki HAYIR 😊

- Derleyici problemleri
- Paralel programlama problemleri (Ör: veri yarışı)
- Fonksiyonel olmayan durumlar (Ör: performans)
- Kayan noktalı sayı durumları (Ör. Hassasiyet kaybı)
- Sistem seviyesinde durumlar (Ör. İşletim sistemi/cihaz bağımlı problemler)
- Gereksinimlerin yanlış anlaşılması

Derleyici Problemleri

- Kaynak kod değil binary olarak derlenmiş kod bilgisayarda çalışır
- Derleyicide problem varsa ne olur? (Nadir görülür)
- Derleyici programınızda bir hata ortaya çıkarırsa ne olur? (daha sık)

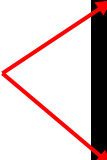
```
int add_up_to (int count) {  
    int sum, i; /* bazı C derleyicileri sum'ı 0 ile başlatırken bazıları  
başlatmaz */  
    for(i = 0; i <= count; i++) sum = sum + i;  
    return sum;  
}
```

- ☞ Bu kod bazı derleyicilerde çalışırken bazılarında çalışmaz
- Bundan kaçınmak için aynı derleyiciyi aynı derleyici ayarları ile kullanmalısınız ancak bu her zaman uygulanabilir olmayabilir

Paralel Programlama Durumları

```
class Main implements Runnable {  
    public static int count = 0;  
    public void run() {  
        for(int i=0; i < 1000000; i++) { count++; }  
        System.out.println("count = " + count);  
    }  
    public static void main(String[] args) {  
        Main m = new Main();  
        Thread t1 = new Thread(m);  
        Thread t2 = new Thread(m);  
        t1.start();  
        t2.start();  
    }  
}
```

Neden?



```
$ javac Main.java  
$ java Main  
count = 1868180  
count = 1868180  
$ java Main  
count = 1033139  
count = 1033139
```

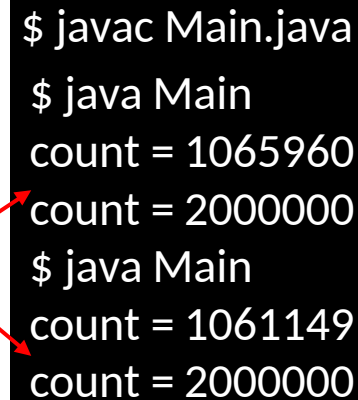
Paralel Programlama Durumları

- Bu neden gerçekleşti?
 - t_1 ve t_2 threadleri farklı CPU'larda çalıştı
 - İki thread count'ı aynı anda artırmaya çalıştı
 - Genelde data race adı verilen durum gerçekleşir
- Data race durumu varsa, sonuç tanımsızdır
 - Java dili özellikleri böyle söylüyor!
 - Her çalıştırdığınızda farklı sonuç elde edersiniz
 - Bir testin geçmiş olması doğruluğu garanti etmez
- Kötü kısmı: Genelde, 99% oranında test geçerli olacaktır
 - ☞ Bu hatayı bulmak için aynı test binlerce kere çalıştırılmalı

Paralel Programlama Durumları

```
class Main implements Runnable {  
    public static int count = 0;  
    public void run() {  
        for(int i=0; i < 1000000; i++)  
            synchronized(this) { count++; }  
        System.out.println("count = " + count);  
    }  
    public static void main(String[] args) {  
        Main m = new Main();  
        Thread t1 = new Thread(m);  
        Thread t2 = new Thread(m);  
        t1.start();  
        t2.start();  
    }  
}
```

Çözüldü mü?



```
$ javac Main.java  
$ java Main  
count = 1065960  
count = 2000000  
$ java Main  
count = 1061149  
count = 2000000
```

Paralel Programlama Durumları

- `synchronized` data race durumunu engeller
 - En sonda `count = 2000000` olur, beklenildiği gibi
 - Nasıl?
 - `Synchronized, count` artırılırken kodu kilitler ve diğer thread erişim sağlayamaz
 - Ancak `count` değeri çalışma sırasında non-deterministiktir. Neden?
 - `t1` ve `t2` threadlerinin hızları non-deterministiktir
- ☞ Data-race içermeyen programlar hala sorun yaratabilirler.

Bu dersin amacı...

- Aşağıdaki konuları görmezden gelerek 😊
 - Derleyici problemleri
 - Paralel programlama problemleri
 - Fonksiyonel olmayan durumlar
 - Kayan noktalı sayı durumları
 - Sistem seviyesinde durumlar
- Sıralı bir programı tek bir derleyici ve tek bir cihaz üzerinde test etmek olası test sayısının çok fazla olması (test explosion problem) sebebiyle zaten zordur
- Test sayısının fazla olması problemine odaklanacağız

Test = Sanat + Bilim

- Exhaustive test imkansızdır
- Hedef: “yeterli” kapsama miktarına erişmek
- Test kapsamı nasıl tanımlanabilir?
 - İdealde: $\text{bulunan_hata} / \text{toplam_hata}$
 - Ancak toplam_hata’yı ölçme yolu var mıdır?
(İpucu: Bütün hataları bilseydiniz, onu test etmezsiniz)
 - Bu yüzden aracı bir metriğe ihtiyaç bulunmaktadır
(Örnek. $\text{test_edilen_satır_sayısı} / \text{toplam_satır_sayısı}$)
- Bu aracı metriği kullanarak “yeterli” olup olmamasına karar vermek bir sanattır 😊
 - En sonunda, karar vermek için alan hakkında bilgiye ihtiyaç duyulur

Test Kapsamını Yükseltmek

- Kalite Güvencesi (QA) mühendislerinin test için kısıtlı bir zamanı bulunmaktadır
 - Test kapsamını maksimize edecek alt test setlerini seçmelidir
- Soru) Hangi test kapsamı maksimum yapar?
 - Yeni bir hata ortaya çıkaran testler
(Ancak bunu bilmek imkansızdır)
- Soru) Hangi testlerin yeni bir hata ortaya çıkarması daha olasıdır?
 - Kodda yeni bir davranışı test edenler
 - Bu fikrin arkasında denklik sınıfı bölümlleme (*equivalence class partitioning*) bulunur.

Denklik Sınıfı Bölümleme

- “Denklik sınıflarına” göre giriş değerleri bölünebilir
 - Denklik sınıfı = aynı davranışı gösteren değerler
- Ör., `sqrt` metodu için denklik sınıfları:
{pozitif sayılar, 0, negatif sayılar}
- Her bir sınıfın davranışı:
 - Pozitif sayılar: Pozitif sayı döner
 - 0: 0 döner
 - Negatif sayılar: Karmaşık sayı döner

Denklik sınıfları oldukça katı bir şekilde bölünmelidir

- *Katı*: Bir değer yalnızca bir sınıfa ait olabilir
- Eğer bir değer birden fazla sınıfa ait ise
 - Aynı girdinin iki farklı davranışa sahip olduğu anlamına gelir
 - Gereksinimde bir bug vardır veya anlaşılamamıştır
- Eğer bir girdinin sınıfı yoksa
 - Bu girdinin beklenen bir davranışı bulunmamaktadır
 - Yeni bir denklik sınıfı tanımlanabilir

Değerler sayısal olmak zorunda değildir

- Bir yazım denetleyicisi: Girdi değerleri string'tir
- Denklik sınıfı:
{string_sozlukte_var, string_sozlukte_yok}
- Davranış:
 - string_sozlukte_var: Hiçbir şey yapma
 - string_sozlukte_yok: altını kırmızı çiz

Değerler sayısal olmak zorunda değildir

- Girdi: ton balığı konservesi
- Denklik sınıfı:
{tarihi_gecmemis,
tarihi_gecmiş_ancak_kokmuyor,
tarihi_gecmis_kokuyor}
- Davranışlar:
 - tarihi_gecmemis: ye
 - tarihi_gecmiş_ancak_kokmuyor: kedileri besle
 - tarihi_gecmis_kokuyor: At

Bütün Denklik Sınıflarını Test Et

- Her bir denklik sınıfından en az bir değer al
- Programda beklenen bütün davranışları kapsamanızı sağlar
- Exhaustive test kullanmadan yüksek kapsama miktarına erişmenizi sağlar!
- Değer nasıl seçilir? Bu da sanatın bir başka parçası 😊
 - Ancak bazı rehberler bulunmaktadır

İç ve Sınır Değerler

- Deneysel Gerçek:
 - Hatalar denklik sınıfının sınır değerlerinde, orta değerlere göre daha yaygın olarak görülür.
- Neden?
 - Off-by-one hataların yaygınlığı sebebiyle

Off-by-one Hatalar

- Beklenen davranışın şöyle olduğu varsayılın:
 - Metot kişinin yaşını girdi olarak alsın
 - Metot kişinin ehliyet alıp alamayacağını belirliyor olsun
 - Kural: Kişi ehliyet alabilmek için 18 yaşında veya daha büyük olmalıdır

- Kod aşağıdaki gibi olabilir:

```
boolean ehliyetAlabilir(int yas)
{
    return yas > 18;
}
```

- Gözlenen ve beklenen davranış aynı mıdır?

Denklik Sınıfı Bölümleme

EHLIYET_ALAMAZ= [...5,6,7,8,9,10,11,12,13,14,15,16,17]

EHLIYET_ALABILIR = [18,19,20,21,22,23,24,25,26,27,28,...]

Sınır değerleri test et

EHLIYET_ALAMAZ= [...5,6,7,8,9,10,11,12,13,14,15,16,**17**]

EHLIYET_ALABILIR = [**18**,19,20,21,22,23,24,25,26,27,28,...]

- Sınır değerleri test et (**kırmızı** ile gösterilmiştir)
- Bug bulunmaktadır: `yas > 18`

Ayrıca ara değerleri de test et

EHLIYET_ALAMAZ= [...5,6,**7**,8,9,10,11,**12**,13,14,15,16,**17**]

EHLIYET_ALABILIR = [**18**,19,20,21,**22**,23,24,**25**,26,27,28,...]

- Ara değerleri (**yeşil** ile gösterilmiştir) test etmek de davranışı görmek açısından önemlidir

Peki yeterli midir?

EHLIYET_ALAMAZ= [...5,6,**7**,8,9,10,11,**12**,13,14,15,16,**17**]

EHLIYET_ALABILIR = [**18**,19,20,21,**22**,23,24,**25**,26,27,28,...]

- Şimdiye kadarki girdi değerleri: {**7**, **12**, **17**, **18**, **22**, **25**}

“Gizli” (IMPLICIT) sınır değerler

- Şu ana kadar eklediğimiz sınır değerler açık (explicit) değerlerdi – Bu değerler gereksinimlere göre tanımlanmaktadır
- Bazı sınır değerler ise gizlidir– Bu sınırlar programlama dili, donanım, etki alanı vs. ile tanımlanır:
 - Dil sınırları: MAXINT, MININT
 - Donanım sınırları:
Hafıza alanı, sabit disk alanı, vs.
 - Etki alanı sınırı:
Negatif olamaz, not 100’ü aşamaz vs.

Gizli sınırları da ekle

EHLİYET_ALAMAZ=

[**MININT**,...,**-1,0**, ...,5,6,**7**,8,9,10,11,**12**,13,14,15,16,**17**]

EHLİYET_ALABILIR =

[**18**,19,20,21,**22**,23,24,**25**,26,27,28,...,**MAXINT**]

- **MININT**, **MAXINT**: Dil sınırları
- **-1, 0**: Etki alanı sınırları (yaş negatif olamaz)
- Inputs: {**MININT**,**-1,0**,**7**,**12**,**17**,**18**,**22**,**25**,**MAXINT**}

Off-by-one Hatayı Tespit Etme

- Şimdi bu girdileri deneyelim:

```
boolean ehliyetAlabilir(int yas)
{
    return yas > 18;
}
```

Inputs: {MININT, -1, 0, 7, 12, 17, 18, 22, 25, MAXINT}

- Hatırlatma, Beklenen değer:
 - Kişi ehliyet alabilmek için 18 yaşında veya daha büyük olmalıdır
- **18** girdisi ile bir hata bulundu:
 - Beklenen davranış: Ehliyet alabilir
 - Gözlenen davranış: Ehliyet alamaz

Base, edge, ve corner durumlar

- **Base durum:** Bir iç değer veya beklenen durum
- **Edge durum:** Sınır değer veya beklenmeyen durum
- **Corner durum (veya patolojik durum):**
Normal çalışma parametrelerinin dışındaki değer
VEYA aynı anda meydana gelen birden fazla edge
durum

Base, edge, ve corner durumlar:

Örnek

- Bir kedi için aşağıdaki sınırlar olduğunu varsayalım:
 - Ağırlık aralığı 0 – 45 kg
 - Sıcaklık aralığı 0 – 50 Derece
- Base durumlar: (10 kg, 25 D), (20 kg, 45 D), ...
- Edge durumlar: (**45 kg**, 25 D), (10 kg, **0 D**), ...
- Corner durumlar: (**90 kg**, 45 D), (**45 kg**, **50 D**), ...
- Neden corner'lar test edilir?
 - Terazinin 90 kg için doğru çalışması beklenmese bile, kullanıcı yine de ne olduğunu umursar (yani teraziyi kırar mı?)

Kaynak Kodda Sınır Değerler

- Şimdiye kadar sınır değerleri kaynak kodun dışından elde etmiştik
 - Açık sınırlar: gereksinimlerden
 - Gizli sınırlar: dil bilgisinden, uygulama alanından, ...
- `ehliyetAlabilir` fonksiyonunun şöyle olduğunu varsayalım:

```
boolean ehliyetAlabilir(int yas) {  
    return yas >= 18 && yas <= 65;  
}
```
- 65 yaş sınırı gereksinimde bulunmazken kodda bulunmaktadır
 - Bu değeri test etmek istemez misiniz?
 - Buna *saydam kutu testi* denir

Kara-, saydam, ve gri-kutu testleri

- **Kara-kutu testi:**
 - İç yapının ve kaynak kodların bilinmedi test türüdür
 - Testler kullanıcı bakış açısı ile yapılır
 - Programlamayı bilmeyen sıradan kişiler tarafından yapılabilir.
- **Saydam-kutu testi:**
 - Yapı ve kodun içinin bilindiği test türüdür
 - Testler geliştirici bakış açısı ile gerçekleştirilir
 - Test girdileri kodun belirli satırlarını çalıştıracak şekilde oluşturulur
 - Test süreci belirli metotların çağırılması ile oluşturulabilir (birim test)
- **Gri-kutu testi:**
 - İç yapının ve kodun biraz bilindiği türdür
 - Bilgi kısmi kod incelemeden veya tasarım dokümanından gelir
 - Kullanıcı bakış açısıyla gerçekleşir ancak kodun iç yapısı hakkında da bilgi sahibidir

Kara-kutu test örneği

- Bir tarayıcı kullanarak web sayfasını test etme
- Bir oyunu oynayarak test etme
- Bir scripti API uç noktaları ile test etme
- Beta testlerin tamamı

Saydam-kutu test örnekleri

- Programın özel bir yolunu test etmek için test girdileri oluşturmak
 - Sınır değerleri seçmek gereksinimlerde bulunmaz ancak kaynak kodda bulunmaktadır
 - Hataya sebep olabilecek girdilerin seçimi (bunların doğru bir şekilde ele alındığını gözlemlemek için)
- Bir test scriptiyle metotları doğrudan çağırmak
 - Bir fonksiyonun doğru sonucu döndürdüğünü test etmek
 - Bir sınıfın oluşturulması durumunda geçerli bir objenin dönmesi

Gri-Kutu test örnekleri

- *Kodu gözden geçirdikten sonra bubble sort kullanıldığını görmek. Ardından büyük test girdileri ile test etmek.*
- *Kodu gözden geçirdikten sonra bir web uygulamasında kullanıcı girdilerin doğru bir şekilde test edilmediğini gözlemlemek ve ardından SQL Injection yapacak testler yazmak*
- *Tasarım dokümanını okuduktan sonra ağ üzerinde çok fazla veri transferi olduğunu görmek ve stres testi uygulamak*

Statik ve dinamik test

- Şimdiye kadar iyi test girdilerinin seçilmesi gerektiğinden söz ettik
 - Ancak test için yapılması gereken tek şey bu mudur?
- Dinamik test = kod çalıştırılır
 - İyi kapsama miktarı için iyi girdilere ihtiyaç vardır
- Statik test = kod çalıştırılmaz
 - Kod çalıştırılmadığı için girdilere ihtiyaç yoktur
 - Hataları tespit etmek için kodun analiz edilmesi gerekir

Dinamik Test

- Şu ana kadar söz ettiğimiz şeyler...
 - Kod belirli durumlar altında çalıştırılır
(ör: girdi değerleri, derleyici, işletim sistemi, çalışma zamanı kütüphaneleri vs.)
 - Gözlemlenen sonuç, beklenen sonuç ile karşılaştırılır
- Endüstride daha yaygın olarak kullanılmaktadır
 - Programcılar öğretilmeden nasıl yapılacağını bilebilir
 - Özel araçlar ve eğitim olmadan yapılabilir
 - Sınıftakilerin büyük bir çoğunluğu dinamik test ile ilgileniyor olacaktır

Statik Test

- Kod bir kişi tarafından veya test aracı tarafından analiz edilir
- Örnekler:
 - Bir kişi tarafından kod izlenir ve incelenir
 - Bir araç kullanılır
 - Linting
 - Model checking
 - Complexity analysis
 - Code coverage
 - Finite state analysis
 - ... COMPILING!

Textbook Chapters 2-4'yi Okuyunuz 😊