BS450 Yazılım Test Mühendisliği

Yazılım Testine Giriş

Burada kim yazılım testini sevmektedir?

Burada kim testin zor olduğunu düşünüyor?

Test etme ve hata ayıklamanın temelde aynı şey olduğunu kim düşünüyor?

Testin yazılımın işe yaradığını göstermek olduğunu kim düşünüyor?

Testin yazılımın
çalışmadığını göstermek
olduğunu kim düşünüyor?

Testin arkasındaki fikrin, yazılım kullanımıyla ilgili riskleri azaltmak olduğunu kim düşünüyor?

Testin daha kaliteli yazılımların geliştirilmesine yardımcı olacağını kim düşünüyor?

Beizer'in Olgunluk Modeli

Seviye 0:
Test, hata ayıklama ile aynıdır

Test etmenin en az olgun hali **Seviye 0'dır. Test, hata ayıklama ile aynı görünür.**

Seviye 0 düşüncesinde, programcılar programları derleyip ardından birkaç rastgele girişle programlardaki hataları ayıklar.

Bu bakış açısı, bir programın **yanlış davranışı** ile program içindeki bir **hata** arasında ayrım yapmaz. Ayrıca, güvenilir veya güvenli yazılım geliştirmeye çok az katkıda bulunur.

(**…** ve ileride gösterileceği üzere test etmek hata ayıklamakla **aynı şey değildir**.)

Seviye 1:
Yazılım testinin amacı, yazılımın çalıştığını göstermektir.

En basiti olan Seviye 0'a göre önemi bir adım yukarıda.

Ancak en önemsiz programların dışında hiçbir programda doğruluğu kanıtlamak neredeyse imkansızdır!

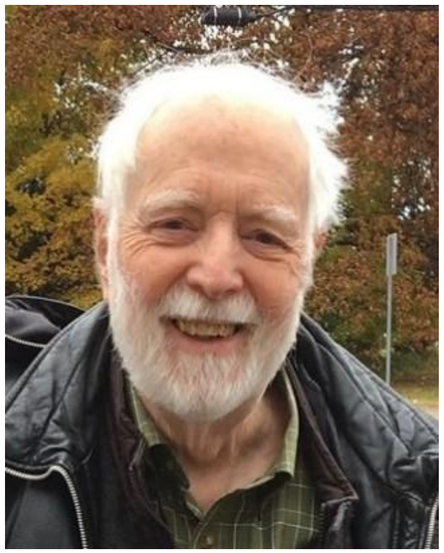
(… derste daha sonra döneceğimiz başka bir nokta.)

Seviye 2:
Yazılım testinin amacı yazılımın çalışmadığını göstermektir

Başarısızlıkları aramak kesinlikle geçerli bir hedef olmasına rağmen, aynı zamanda olumsuz bir hedeftir.

Test uzmanlarının ve geliştiricilerin farklı ekiplerde olduğu bir şirket organize edilmişse, test uzmanlarının sorun bulmaktan hoşlanabileceği ancak geliştiricilerin asla sorun bulmak istemediği bir durumla karşılaşabilirsiniz.

- yazılımın çalışmasını isterler!



“Excellent testing can make you unpopular with almost everyone!”

— Bill McKeeman

Seviye 2 testleri, test uzmanlarını ve geliştiricileri düşmanca bir ilişkiye sokar ve bu da ekibin morali açısından kötü olabilir.

Bunun da ötesinde, birincil amacımız hataları aramaktır ancak herhangi bir hata bulunamazsa;

Bütün iş bitmiş olur mu?

Veya iş tamamen bitmiş olsa bile …

Uygulamamız çok mu iyi *yoksa* **testimiz mi kötü?**

Seviye 3: Yazılım testinin amacı belirli bir şeyi göstermek değil, yalnızca yazılım kullanma riskini azaltmaktır.

Seviye 3 düşüncesi, yazılımı her kullandığımızda bazı risklere maruz kaldığımız gerçeğini kabul etmemizi sağlar.

Seviye 3 düşüncesi, yazılımı her kullandığımızda bazı risklere maruz kaldığımız gerçeğini kabul etmemizi sağlar.

Risk küçük olabilir ve sonuçları önemsiz olabilir veya risk büyük olabilir ve sonuçları felaket olabilir, ancak risk her zaman oradadır.

Bu, tüm geliştirme ekibinin aynı şeyi istediğini fark etmemizi sağlar: yazılımı kullanma riskini azaltmak.

Seviye 3 testinde, test etme ve geliştirme, riski azaltmak için el ele çalışır.

Seviye 4:

Test, tüm BT profesyonellerinin daha yüksek kaliteli yazılım geliştirmesine yardımcı olan zihinsel bir disiplindir.

Test uzmanları ve geliştiriciler aynı "ekipte" olduğunda veya test etme, geliştirme açısından eşit veya hatta daha önemli kabul edildiğinde, kuruluş **Seviye 4** testine geçebilir.

Seviye 4 testi, testi **kaliteyi artıran zihinsel bir disiplin olarak tanımlar.**

daha kaliteli yazılım geliştirmek

Test uzmanları ve geliştiriciler aynı "ekipte" olduğunda veya test etme, geliştirme açısından eşit veya hatta daha önemli kabul edildiğinde, kuruluş **Seviye 4** testine geçebilir.

Seviye 4 testi, testi **kaliteyi artıran zihinsel bir disiplin olarak tanımlar.**

Aynı şekilde Seviye 4 testi, testin amacının geliştiricilerin daha kaliteli yazılım üretme yeteneğini geliştirmek olduğu anlamına gelir.

Yazım denetleyicinin en iyi kullanımı yalnızca yanlış yazılan sözcükleri bulmak değil, aynı zamanda yazım yeteneğimizi geliştirmektir.

Yazım denetleyicinin yanlış yazılmış bir kelime bulduğu her seferde, kelimenin nasıl doğru yazılacağını öğrenme fırsatına sahip oluruz.

Yazım denetleyici, yazım kalitesi konusunda "uzmandır".

Beizer'in Olgunluk Modeli

0. Hata ayıklamayla aynı eylem
1. Amaç yazılımın çalıştığını göstermektir
2. Amaç yazılımın çalışmadığını göstermektir
3. Amaç yazılım kullanma riskini azaltmaktır
4. Amaç, tüm BT profesyonellerinin daha iyi yazılımlar geliştirmesine yardımcı olmaktır.

Seviye 1:
Yazılım testinin amacı, yazılımın çalıştığını göstermektir.

En basiti olan Seviye 0'a göre önemi bir adım yukarıda.

Ancak en önemsiz programların dışında hiçbir programda doğruluğu kanıtlamak neredeyse imkansızdır!
(... derste daha sonra döneceğimiz başka bir nokta.)

⟶ ***Neden?***

Neden TÜM hataları bulmak imkansız
veya
Yazılım testi zor?

Exhaustive Test ile ilgili problemler

```
public int daysBetweenTwoDates(int year1, int month1, int day1,
                               int year2, int month2, int day2)
```

Java'da **int** içindeğer aralığı **-2,147,483,648** ila **2,147,483,647** (veya 2^{32})

Altı int ile 2^{32x6} veya 2^{192} ($\approx 6 \times 10^{57}$) tekil girdi denemesi!

Her bir girdi denemesinin ≈ 1 nanosaniye sürdüğünü varsayalım.

Bütün değerleri denemek 10^{41} yıl alacaktır!


1 hour here is 7 years on earth
STILL NOT LONG ENOUGH
FOR EXHAUSTIVE TESTING
imgflip.com

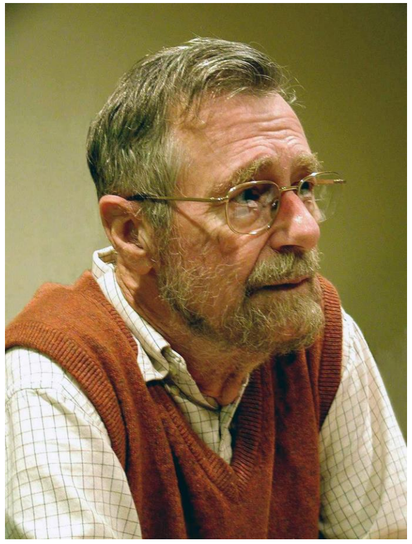
Halting (durma) Problemi ve Yazılım Testi

Bilgisayar Bilimlerinde Halting problemi temel olarak bir programın bazı girdiler verildiğinde sonlandırılıp sonlandırılmayacağını bilmeme sorunudur.

Eğer rastgele bir programa girdi verirsek, o orijinal programın sonlandırılıp sonlandırılmayacağını söyleyebilecek hiçbir programın yazılamayacağı kanıtlanmıştır.

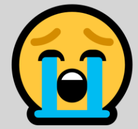
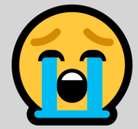
Yazılım testinde de bu doğrudur: Test girdilerimiz göz önüne alındığında, test edilen programın sonsuz bir döngüye takılıp takılmayacağını bilmiyoruz!

Bu, genel olarak kapsamlı testlerin sadece zorlu değil, aynı zamanda hesaplanamaz olduğu anlamına gelir.



> “Software testing can only show the presence, not the absence of bugs”

— Edsger Dijkstra

	Çözülebilir Problemler	İnatçı problemler	Hesaplanamayan problemler
Tanım	Verimli bir şekilde çözülebilir	Çözme yöntemi var, ancak oldukça zaman alır	Herhangi bir bilgisayar programı tarafından çözülemez
Teoride hesaplanabilir			
Pratikte hesaplanabilir			
Örnek	Haritada en kısa rotayı bulma	Şifre çözme	

Oracle Sorunu

Eğer şunlar gerçekleşebilir olsa bile:

1) **Bütün girdilerle yazılım çalıştırılabilse**

(ör., yazılım testi çözülebilir problem olsaydı)

2) **Yazılımın her bir girdi ile sonlanması garanti olsa**

(ör., yazılım testi hesaplanabilir bir problem olsaydı)

Yine de **oracle problemini** çözmemiz gerekirdi – **yazılımın verilen girdilerle ürettiği çıktının doğru olduğunu nasıl bilebilirdik?**

Oracle Problemi

Yazılım testinde bir **oracle** **yazılım çıktısının doğru olduğuna karar veren** **bir şey** veya **biri**

Ör: **JUnit'de** bir **assertion**

Manuel olarak karar veren bir insan

Hayır, şirket olan değil.

Ancak programın çalıştığından emin olmak için her bir girdiye ihtiyacımız yok... değil mi...?

Peki bu **girdi alt kümesini** nasıl seçeceğiz?

Yazılım test probleminin özü budur.

Yazılımın kalitesi hakkında mümkün olduğunca fazla bilgi ortaya çıkaracak bir dizi girdi seçmemiz gerekir.

Ancak tüm hataları ortaya çıkaracak tüm girdileri seçip seçmediğimizi **bilmiyoruz**.

TÜM Hataları Bulmak Neden İmkansız veya:

Yazılım Testi Neden Zordur?

1. Önemsiz olmayan herhangi bir program için tüm girdileri yürütmek **zorludur (inatçı)**.
2. Yazılımın her girdide sonlandırılmasının sağlanması **karar verilemez**
3. Karşılık gelen girdilere göre doğru/yanlış çıktıları tanımak, en azından ilk etapta yazılımı oluşturmak kadar zordur - **oracle problemi**

Yazılım Arızaları (failure) Nasıl Oluşur?

Bir Metot ve Testleri

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Bir Metot ve Testleri

```
public class StringUtilsTest {

    @Test
    public void shouldReturnRepeatedChar() {
        Set<Character> resultSet = duplicateLetters("software testing");
        assertTrue(resultSet.contains('t'));
    }

    @Test
    public void shouldNotReturnNonRepeatedChar() {
        Set<Character> resultSet = duplicateLetters("software debugging");
        assertFalse(resultSet.contains('t'));
    }
}
```

PASSED

FAILED

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Bu metot bir hata (debug) içeriyor. Daha doğrusu bir **kusur (defect)**.

Peki kusur nerede? Metotun hatalı (fail) olmasına nasıl neden olur?

Kusurlar (Defects)

Yazılım başarısızlıkları (failure) her zaman kodda bir veya daha fazla kusurun yürütülmesiyle başlar.

Kusur (defect), yalnızca bir **parça hatalı, yanlış koddur**.

Bir kusur, program ifadesinin (statement) bir parçası veya tamamı olabilir ya da mevcut olmayan ve olması gereken ifadelere karşılık gelebilir.

Her ne kadar programcılar kodda meydana gelen kusurdan sorumlu olsalar da teknik olarak her zaman hatalı olmayabilirler; sorun örneğin yetersiz belirlenmiş gereksinimlerden kaynaklanmış olabilir.

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Örneğimizdeki kusur ikinci döngü başlatıcıdadır.

`i + 1`'de başlamalıdır `i` 'de değil yoksa sürekli aynı elemanları kontrol edecektir.

Enfeksiyonlar (infections)

Enfeksiyon, kusur yürütüldüğünde meydana gelen olaydır ve programın durumu etkilenir.

Bir programın durumu enfekte olduğunda hatalı çalışmaya başlar:

- Değişkenler yanlış değerler almaya başlar
- Programda verilen kararlar yanlış değerlendirilir ve yürütme yolu doğru olandan sapar..

Ancak bu noktada programın çıktısını etkilememiştir (ve hatanın şu ana kadar gözlemlenebilir bir etkisi olmamıştır).

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Örneğimizdeki enfeksiyon, döngünün dizedeki bir dizini çok erken yinelemeye başlamasına neden oluyor.

Bu ayrıca dizedeki her karakterin `duplicates` kümesine eklenmesine neden olur. Ancak bu noktada programda gözle görülür bir yanlışlık yok.

Hatalar (Failures)

Enfeksiyon programın çıktısına yayıldığında bir **hata** oluşur.

Yani program gözle görülür şekilde yanlış davranır.

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Bu nedenle **hatalar**, programların ne zaman gözlemlenebilir çıktılar sağladığına bağlıdır. Örneğimizde testlerimizde metodun dönüş değerini sorgulayarak **test hatasına** neden oluyoruz.

Ancak tüm yazılımın çalıştırılması sırasında metot dahili olarak başka bir metot tarafından kullanılabilir ve çok daha sonra farklı bir yerde hata meydana gelebilir.

Hatalar vs Test Hataları

Bu nedenle başarısızlıklarla test başarısızlıklarını birbirinden ayırmamız gerekir.

Hatalar, yazılımın üretimde bir bütün olarak çalıştırıldığında hatalı davranmasıdır.

Test hataları, testlerin kendilerinin başarısız olduğu durumlardır çünkü:

(a) test bir yazılım hatasını ortaya çıkarmıştır

(b) testin kendisi hatalıdır; örneğin yazılımın davranışı hakkında yanlış bir iddiada (assert) bulunmuştur.

Testing vs Debugging

Artık test etme ve hata ayıklamanın aynı olduğu fikrini de çürütebiliriz ☺

Test, yazılımın çalışmasını gözlemleyerek değerlendirme sürecidir.

Hata ayıklama, hataları/test hatalarını ve nihayetinde onlardan sorumlu olan kusurlara kadar takip etme sürecidir.

Yazılım Hataları Nasıl Olur?

1. Yürütme sırasında bir **kusur** içeren program konumuna ulaşılır.

Defect

2. Kusur program durumunu **enfekte eder.**

Infection

3. Enfeksiyon programın çıktısına yayılarak bir **hataya** neden olur.

Failure

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Kusurlara her zaman ulaşılmaz (yürütülmez)

Giriş dizesi `s` boşsa ne olacağını düşünün.

İyi bir test paketinin mümkün olduğunca çok yazılım kullanması gerekir.

Buna daha sonraki derslerde geri döneceğiz.

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i > s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i + 1; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Kusurlar her zaman enfeksiyonlara neden olmayabilir

Döngü testinin tersine çevrildiği bir kusuru düşünün (örn. "<" yerine ">" kullanılır)

Boş string için kusur yürütülür, ancak hiçbir değişken yanlış değerleri almaz ve döngü gövdesi doğal olarak çalıştırılmaz.

Yani bu özel girdi için enfeksiyon bulunmaz.

```java
public static Set<Character> duplicateLetters(String s) {
    // lower case the string and remove all characters that are not letters
    s = s.toLowerCase().replaceAll("[^a-z.]", "");

    // initialise the result set
    Set<Character> duplicates = new TreeSet<>();

    // iterate through the string
    for (int i = 0; i < s.length(); i++) {
        char si = s.charAt(i);

        // iterate through the rest of the string, checking for the same letter
        for (int j = i; j < s.length(); j++) {
            char sj = s.charAt(j);

            if (si == sj) {
                // a match has been found, add it to the result set
                duplicates.add(si);
            }
        }
    }

    return duplicates;
}
```

Enfeksiyonlar her zaman çıktıya yayılmayabilir

Orijinal kusuru düşündüğümüzde girdinin `s="stst"` olduğunu varsayalım.

Kusur yürütülür ve `"s"` ile `"t"` biraz erken de olsa `duplicates'e` eklenir. Böylelikle çıktısı doğru olur.

Test senaryolarının hataları ortaya çıkarması gerekir

Metot her iki testte de hatalıdır ancak bu testlerden yalnızca biri hatayı ortaya çıkarır. (test hatasına neden olarak).

```java
public class StringUtilsTest {

    @Test
    public void shouldReturnRepeatedChar() {
        Set<Character> resultSet = duplicateLetters("software testing");
        assertTrue(resultSet.contains('t'));
    }

    @Test
    public void shouldNotReturnNonRepeatedChar() {
        Set<Character> resultSet = duplicateLetters("software debugging");
        assertFalse(resultSet.contains('t'));
    }
}
```

PASSED

FAILED

Yazılım Hataları Test Senaryolarıyla Nasıl *Tespit* Edilir?

RIPR model

Defect **R**eached

State **I**nfected

Infection **P**ropagated

Failure **R**evealed

Neleri ele almayacağız...

*Genel olarak, belirli teknolojileri ve ortamları ele **almayacağız**.*

Ele alacağımız uygulamalar ve teknikler, geliştirmekte olduğunuz herhangi bir yazılım sistemi için geçerlidir.

Ders, yapmanız gereken herhangi bir testin temelini öğrenmenize hizmet edecek şekilde tasarlanmıştır.

Bununla birlikte, her alanın kendi test uygulamaları ve araçları vardır, bu nedenle zamanı geldiğinde bunlara odaklanan ek kaynaklar aramanız gerekecektir.