**Erciyes Üniversitesi**
**Bilgisayar Mühendisliği Bölümü**

# BZ 313 Yazılım Mühendisliği

## 20. Doğrulama, Test ve Bug'lar

# Güvenilir Sistemler Oluşturma: İki İlke

**Bir yazılım sisteminin güvenilir olması için:**

- Geliştirmenin her aşaması, artımlı doğrulama ve test ile düzenli olarak yapılmalıdır.
- Test ve düzeltme kaliteyi garanti etmez, ancak kapsamlı testler yapılmadan güvenilir sistemler mümkün değildir.

# Statik ve Dinamik Doğrulama

**Statik doğrulama:**

Yazılımın yürütülmesini içermeyen doğrulama teknikleridir.

- Manuel olabilir veya bilgisayar araçları kullanılabilir.

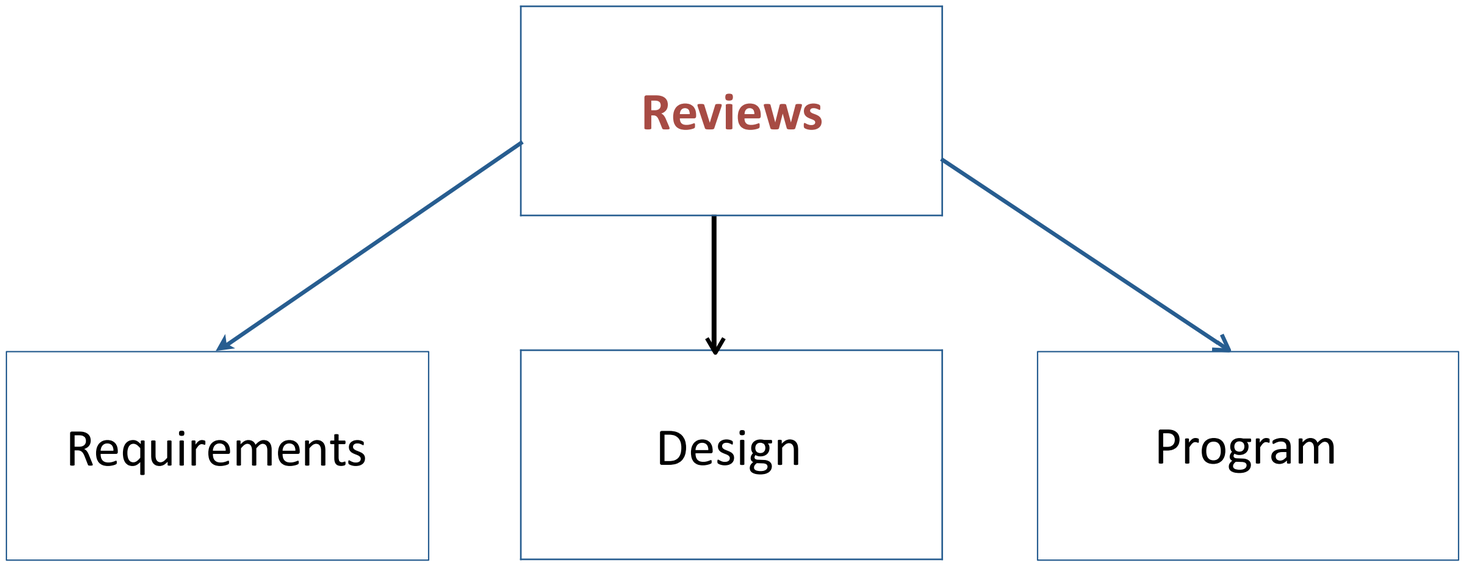
**Dinamik doğrulama:**

- Yazılımın deneme verileriyle test edilmesi.
- Hataları kaldırmak için hata ayıklama.

| Özellik | Statik Doğrulama | Dinamik Doğrulama |
|---|---|---|
| **Kod Çalıştırma** | Gerekmez | Gereklidir |
| **Test Aşaması** | Erken (kodlama sırasında) | Geç (çalışma zamanı) |
| **Hedef** | Söz dizimi, yapı ve standartlar | Fonksiyonel hatalar ve performans |
| **Kapsam** | Yüzeysel, mantık hatalarını bulamaz | Derinlemesine, mantık ve çalışma hataları |
| **Maliyet** | Daha düşük | Daha yüksek |

# Statik Doğrulama: İncelemeler (Review)

**İncelemeler,** yazılım geliştirme süreci boyunca gerçekleştirilen bir statik doğrulama biçimidir.



# Review

İncelemeler, iyi bir yazılım geliştirmenin temel bir parçasıdır

**Kavram**

İş arkadaşları birbirlerinin çalışmalarını gözden geçirir:

- yazılım geliştirmenin herhangi bir aşamasına uygulanabilir, ancak program tasarımını veya kodunu gözden geçirmek için özellikle değerlidir
- resmi veya gayri resmi olabilir

**Hazırlık**

Geliştirici(ler) iş arkadaşlarına belgeler (ör. modeller, teknik özellikler veya tasarım) veya kod listesi sağlar.

Katılımcılar materyalleri önceden inceler.

**Toplantı**

Geliştirici, gözden geçiren kişileri materyaller aracılığıyla yönlendirir, her bölümün ne yaptığını açıklar ve soruları teşvik eder.

# İnceleme (Review) Toplantısı

İnceleme, yapılandırılmış bir toplantıdır

**Katılımcılar ve rolleri:**

- Geliştirici(ler): Çalışmaları incelenmekte olan kişi(ler)
- Moderatör: Toplantının istikrarlı bir şekilde ilerlemesini sağlar
- Kâtip: tartışmayı yapıcı bir şekilde kaydeder
- İlgili taraflar: Aynı projedeki diğer geliştiriciler
- Dış uzmanlar: Bu proje üzerinde çalışmayan bilgili insanlar
- Müşteri: Sürecin bu kısmı hakkında bilgi sahibi olan müşteri temsilcileri

# İncelemelerin Faydaları

**Faydalar:**

- Ekstra gözler hataları tespit eder, iyileştirmeler önerir.
- Meslektaşlar uzmanlıklarını paylaşırlar; eğitime yardımcı olur.
- Bileşenler arasındaki uyumsuzluklar tanımlanabilir.
- Geliştiricilere yarım kalmış işleri toparlamaları için bir teşvik verir.
- Zamanlama ve yönetim kontrolüne yardımcı olur.

# Başarılı İncelemeler

**Bir incelemeyi başarılı kılmak için:**

- Kıdemli ekip üyeleri liderlik göstermelidir.
- İyi incelemeler herkes tarafından iyi bir hazırlık gerektirir.
- Herkes yardımsever olmalı, tehdit edici değil.

Bolca zaman tanıyın ve başka bir gün devam etmeye hazır olun.

# Statik Doğrulama: Çift (Pair) Tasarımı ve Çift (Pair) Programlama

**Kavram: Paylaşılan geliştirme ile gözden geçirmenin faydalarını elde edin**

İki kişi ekip olarak birlikte çalışır:

- Tasarım ve/veya kodlama
- Test ve sistem entegrasyonu
- Dokümantasyon ve devir teslim

Avantajlar şunları içerir:

- İki kişi daha az hatayla daha iyi yazılımlar oluşturur
- Çapraz eğitim

Birçok yazılım evi mükemmel üretkenlik bildirmektedir.

# Statik Doğrulama: Program Denetimleri

**Amacı hataları tespit etmek olan resmi program incelemeleri:**

- Kod satır satır okunur veya gözden geçirilir
- 2 saatlik toplantıda 150 ila 250 satır kod
- Yaygın hatalardan oluşan bir kontrol listesi kullanın
- Ekip bağlılığı gerektirir, örneğin eğitimli liderler

O kadar etkili ki, birim testinin yerini alabileceği iddia ediliyor.

# Statik Doğrulama: Analiz Araçları

**Program analistleri, olası** hatalar ve anormallikler için bir programın kaynağını tarar.

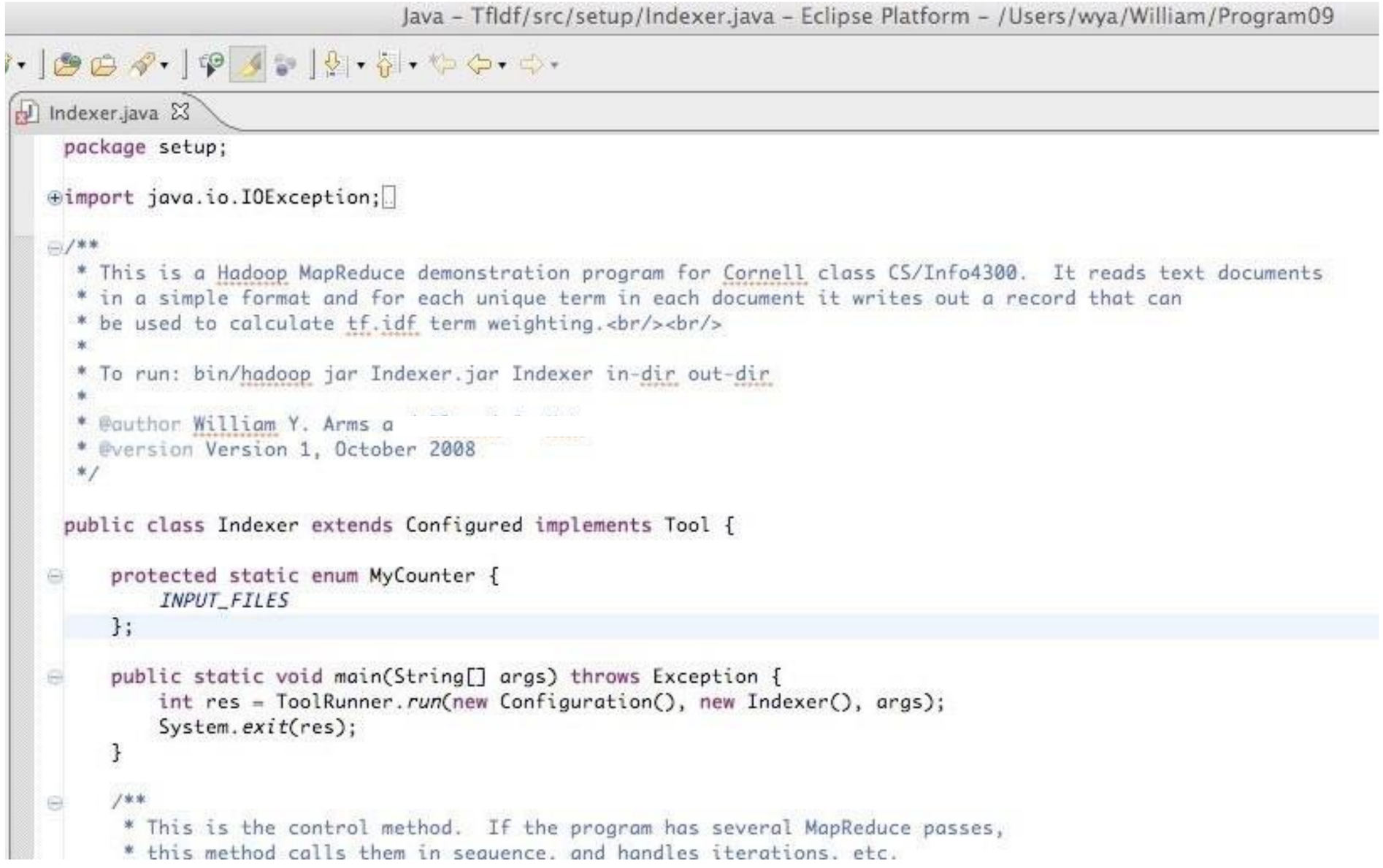
- Denetim akışı: Kodun kontrol yapılarında (if, for, while gibi) birden fazla çıkış veya giriş noktası bulunan döngülerin tespiti yapılır. Bu tür yapılar, karmaşıklığa neden olabilir ve hata olasılığını artırır.
- Veri kullanımı: Bildirilmemiş veya başlatılmamış değişkenler, kullanılmayan değişkenler, birden çok atama, dizi sınırları
- Arayüz hataları: Parametre uyuşmazlıkları, fonksiyonların kullanılmaması sonuçları, çağrılmamış prosedürler
- Depolama yönetimi: Atanmamış işaretçiler, işaretçi aritmetiği

# Statik Analiz Araçları (devamı)

**Statik analiz araçları**

- Çapraz referans tablosu: Bir değişkenin, prosedürün, nesnenin vb. her kullanımını gösterir.
- Bilgi akış analizi: Bir çıktının bağlı olduğu girdi değişkenlerini tanımlar.
- Yol analizi: Program boyunca tüm olası yolları tanımlar.

# Programlama Araç Setlerinde Statik Analiz Araçları



# Programlama Araç Setlerinde Statik Analiz Araçları

```
public void map(LongWritable key, Text value, OutputCollector<Text, WordinDoc> output, Reporter rep1)

    /*  Read the input document one token at a time and build the wordTree */

    String doc = "";                                                    // Doc ID
    String line = value.toString();

    StringTokenizer itr = new StringTokenizer(line);
    rep1.incrCounter(MyCounter.INPUT_FILES, 1);

    if (itr.hasMoreTokens()) doc = itr.nextToken();                     // First word of d

    String word;                                                        // Next word from
    int counter = 0;                                                    // Count of words
    int maxF = 1;                                                       // Max frequency o
    int lengthSq = 0;                                                   // Length of docum

    while (itr.hasMoreTokens() && counter <= MAXWORDS) {
        word = itr.nextToken().toLowerCase().replaceAll("[^a-z]", "");  // Get next word a

        if (word.length() > 1 && !stopwords.contains(word)){            // Ignore one char
            WordinDoc tempNode = new WordinDoc();
            int tempF = 1;
```

Problems | Javadoc | Declaration

1 error, 15 warnings, 0 others

| Description | Resource |
|---|---|
| Errors (1 item) | |
| Warnings (15 items) | |
| The field GraphClean.MapClean.fromURL is never read locally | GraphClean.java |
| The field GraphClean.MapClean.hashFromURL is never read locally | GraphClean.java |
| The field GraphClean.MapClean.hashToURL is never read locally | GraphClean.java |

# Dinamik Doğrulama: Test Aşamaları

**Test, aşamalara bölündüğünde etkilidir**

Kullanıcı arayüzü testi (ayrı olarak gerçekleştirilir)

**Birim testi**

- Birim test

**Sistem testi**

- Entegrasyon Testi
- Fonksiyon testi
- Performans testi
- Kurulum testi

Kabul testi (ayrı olarak yapılır)

# Test Stratejileri

### Aşağıdan yukarıya test

Sistemin alt seviyesinden başlanarak, her birim kendi test ortamında test edilir. Tüm sistemler için uygundur. **Avantaj:** Alt bileşenler test edilerek entegrasyon sürecinde hatalar erken tespit edilir.

### Yukarıdan aşağıya test

Büyük bileşenler test edilirken eksik olan alt bileşenler dummy (sahte) stub'lar ile taklit edilir. Özellikle şunlar için kullanışlıdır:

- Kullanıcı arayüzleri
- İş akışı
- Müşteri ve Yönetim Gösterileri

### Stres testi

Sistemi sınırlarında ve sınırlarının ötesinde test eder. Özellikle şunlar için kullanışlıdır:

- Gerçek Zamanlı Sistemler
- Transaction süreçleri

Çoğu sistem, üç stratejinin bir kombinasyonuna ihtiyaç duyar.

# Test Yöntemleri

**Saydam kutu testi**

Test, test ettikleri şeyin içini bilen kişiler tarafından gerçekleştirilir.

Örnek: Grafik paketindeki onay işaretleri

**Kara kutu testi**

Test, test ettikleri şeyin içini bilmeyen kişiler tarafından gerçekleştirilir.

Örnek: İç işleyişi bilmeden RESTful web servislerin postman ile testi

# Test: Birim Testi

- Bir sistemin küçük bölümleri, örneğin tek bir sınıf üzerinde testler.
- Vurgu, spesifikasyona göre gerçek kodun doğruluğu üzerinedir.
- Test verileri genellikle geliştirici(ler) tarafından spesifikasyon anlayışlarına ve birim bilgilerine göre seçilir.
- Çeşitli ayrıntı düzeylerinde olabilir.
- **Saydam kutu veya kara kutu olabilir:** birimin geliştiricisi (geliştiricileri) veya özel test uzmanları tarafından.

Birim testi kapsamlı değilse, sistem testi neredeyse imkansız hale gelir. Zamanlamanın gerisinde kalan bir proje üzerinde çalışıyorsanız, birim testini aceleye getirmeyin.

# Test: Sistem ve Alt Sistem Testi

- Kapsamlı bir şekilde test edilmiş birimleri birleştiren bileşenler veya komple sistem üzerinde testler.
- Entegrasyon ve arayüzlere vurgu.
- Gerçek veriler için tipik olan ve/veya sistemin sınırlarını zorlayan deneme verileri, örneğin arızalar, yeniden başlatma.
- Sistematik olarak gerçekleştirilir, tüm sistem monte edilene kadar bileşenler eklenir.
- **Saydam veya kara kutu olabilir:** geliştirme ekibi veya özel test uzmanları tarafından.

Sistem testi, bir sonrakini monte etmeden önce her bileşende tamamen hata ayıklanırsa en hızlı şekilde tamamlanır.

# Dinamik Doğrulama: Test Tasarımı

**Test, bir sistemin doğru olduğunu asla kanıtlayamaz.**

Yalnızca (a) bir sistemin tek bir durumda doğru olduğunu veya (b) bir hataya sahip olduğunu gösterebilir.

- Testin amacı, hataları bulmak veya programın belirli durumlarda doğru olduğunu göstermektir.
- Testler hiçbir zaman kapsamlı değildir.
- Test pahalıdır.

# Test Senaryoları

**Test senaryoları, belirli sorunları bulma olasılıkları yüksek olduğu** için seçilen belirli testlerdir.

Test senaryoları, ciddi hatalar bulma şansına karşı masrafı dengelemek için seçilir.

- Geliştirme ekibi tarafından seçilen senaryolar, bilinen savunmasız alanların test edilmesinde etkilidir.
- Deneyimli yabancılar ve müşteriler tarafından seçilen senaryolar, geliştiricilerin bıraktığı boşlukları bulmada etkili olacaktır.
- Deneyimsiz kullanıcılar tarafından seçilen senaryolar başka hatalar bulacaktır.

# Test Setlerindeki Varyasyonlar

**Test paketi,** bir sistem veya sistemin bir bileşeni için geçerli olan tüm test durumlarının kümesidir.

Testleri çalıştırırken, yalnızca belirli koşullar altında meydana gelen bazı hatalar vardır, örneğin, aynı makinede başka bir yazılım çalışırken, görevler belirli sıralarda planlandığında veya olağandışı veri kümeleriyle vb.

Bu nedenle, her test çalışmasının bazı test senaryolarını sistematik olarak değiştirmesi ve testlerin yapılma sırasını değiştirmesi vb. gelenekseldir.

# Artımlı Test (örneğin, Günlük Test)

**Spiral geliştirme ve artımlı test**

- Son sistemin yapısına ve bazı temel işlevlere sahip bir ilk yineleme oluşturun.
- İlk test senaryoları kümesini oluşturun.
- Sistemdeki değişiklikleri günlük olarak kontrol edin, tüm sistemi günlük olarak yeniden oluşturun.
- Her gün kapsamlı bir test senaryosu seti çalıştırın, yeni hataları belirleyin ve bunlarla ilgilenin.
- Sürekli olarak yeni test senaryoları ekleyin.

Microsoft gibi birçok büyük yazılım evi, tüm sistemin günlük yapısı ve kapsamlı test senaryoları setleri ile bu prosedürü takip eder. Gerçekten büyük bir sistem için bu, yüzlerce hatta binlerce test bilgisayarı ve çok büyük bir test uzmanı kadrosu gerektirebilir.

# Dinamik Doğrulama: Regresyon Testi

Yazılım değiştirildiğinde, değişikliklerin amaçlandığı gibi davrandığını ve değiştirilmemiş kodun davranışını olumsuz etkilemediğini kontrol etmek için **regresyon testi** kullanılır.

Her değişiklikten sonra, ne kadar küçük olursa olsun, tüm test paketini yeniden çalıştırın.

# Regresyon Testi: Program Testi

- Her biri beklenen davranışa sahip **bir dizi test paketi** oluşturun.
- Tüm test senaryolarını çalıştırmak ve beklenen davranışla karşılaştırmak için komut dosyaları oluşturun. (Komut dosyaları otomatik olabilir veya insan etkileşimine sahip olabilir.)
- Sistemde ne kadar küçük olursa olsun bir değişiklik yapıldığında (örneğin, bir hata düzeltildiğinde), değişikliği gösteren yeni bir test senaryosu ekleyin (örneğin, hatayı ortaya çıkaran bir test senaryosu).
- Değiştirilen kodu yayımlamadan önce, tüm test paketini yeniden çalıştırın.

# Testin Dokümantasyonu

Her projenin, eksiksizlik, görünürlük ve **gelecekteki bakım** için test prosedürlerini belgeleyen bir **test planına** ihtiyacı vardır.

Test planı şunları içermelidir:

- Test yaklaşımının açıklaması.
- Test senaryolarının ve ilgili hataların listesi.
- Testleri çalıştırma prosedürleri.
- Test analiz raporu.

# Hataları Düzeltme

## Hatayı izole edin

- Arada karşılaşılan hatalarda --> Tekrarlanabilir
- Karmaşık örnek --> Basit bir örnek

## Hatayı ve bağlamını anlayın

- Kök neden
- Bağımlılıklar
- Yapısal etkileşimler

## Hatayı düzelt

- Tasarım değişiklikleri
- Dokümantasyon değişiklikleri
- Kod değişiklikleri
- Yeni test senaryolarının oluşturulması

# Geri Dönen Bug’lar

## Hataları düzeltmek hataya açık bir süreçtir

- Bir hatayı düzelttiğinizde, ortamını da düzeltin.
- Hata düzeltmeleri statik ve dinamik test gerektirir.
- En ufak bir ilgisi olan tüm testleri tekrarlayın (regresyon testi).

## Bugların tekrardan oluşma alışkanlığı vardır

Bir hata düzeltildiğinde, gelecekteki regresyon testi için hata durumunu test paketine ekleyin.

## Devamlılık

Çoğu insan bir sorun etrafında çalışır. En iyi insanlar temel nedeni bulur ve sonsuza dek düzeltir!

# Zor Hatalar

Bazı hataların izlenmesi ve izole edilmesi son derece zor olabilir. Bu özellikle arada meydana gelen arızalar için geçerlidir.

- Büyük bir merkezi bilgisayar, döküm veya başka bir tanılama olmadan her ay birkaç kez durur.
- Veritabanı yükü, tanılama olmadan birkaç gün çalıştıktan sonra durur.
- Bir görüntü işleme sistemi doğru çalışır, ancak çok büyük miktarda bellek kullanır.

Bu tür sorunların izini sürmek için aylarca çaba gerektirebilir.

# Sistem Yazılımındaki Hatalar

**İyi üreticilerin sistem yazılımları bile hatalar içerebilir:**

- Fortran'da çalışma zamanı ortamında yanlış çalışan dahili işlev ($e^0 = 0$)
- Tamsayılarla çok yavaş olan dize-sayı işlevi
- Bellek sızıntısı olan ön yükleme sistemi

# Donanımdaki Hatalar

**Çeşitli donanım hatalarıyla karşı karşıya kalınabilir:**

- Eksik bayta sahip film çizicisi (1:1023)
- Sanal bellek yönetimi için mikro kod

Her sorun aslında gömülü yazılım/firmware'inde bir hataydı.

# Bir hatayı düzeltmek müşteriler için bir sorun yaratırken

Bazen müşteriler bir hataya dayanan uygulamalar oluşturur. Hatayı düzeltmek uygulamaları bozacaktır.

- Z ekseni etrafında yanlış yönde dönen grafik paketi.
- Bir uygulama, emülatör hatasız olsa bile bir emülatör ile çökmesi
- Microsoft'un Internet Explorer'ında 3 piksel oluşturma sorunu.

Bu hataların her biri ile kodun düzeltilmesi kolaydı, ancak onu yayımlamak mevcut programlar için sorunlara neden olabilirdi.

# Kullanıcı Arayüzlerini Test Etme Üzerine Bir Not

**Kullanıcı arayüzleri birkaç test kategorisine ihtiyaç duyar.**

- **Tasarım aşamasında,** tasarımın kullanılabilir olduğundan emin olmak için deneme kullanıcıları ile kullanıcı testi yapılır. Tasarım testi, grafik öğeler geliştirmek ve gereksinimleri doğrulamak için de kullanılır.
- **Uygulama aşamasında**, kullanıcı arayüzü, uygulamanın güvenilirliğini kontrol etmek için standart birim ve sistem testi adımlarından geçer.
- Son olarak, tüm sistem üzerinde kullanıcılarla **kabul testi** yapılır.

**Erciyes Üniversitesi**
**Bilgisayar Mühendisliği Bölümü**

# BZ 313 Yazılım Mühendisliği

## 20. Doğrulama, Test ve Bug'lar

Ders Sonu