BS450 Yazılım Test Mühendisliği

# Birim Test – Bölüm 1

# Birim Testleri – Özet

1. Kapsamı daralt
2. Tek bir sınıf veya yöntemle sınırlıdır
3. Küçük boyutlu

# Birim Testleri Neden Yazılır?

**Hataları önlemek için** (tabii ki de!)

Ayrıca **geliştiricilerin üretkenliğini artırır** çünkü birim test:

1. **Uygulama konusunda yardımcı olur** – Kodlama sırasında testler yazmak, yazılan kod hakkında hızlı geri bildirim sağlar.

2. **Başarısız olduklarında anlaşılması kolay olmalı;** her test kavramsal olarak basit olmalı ve sistemin belirli bir bölümüne odaklanmalıdır.

3. Mühendislere sistemin test edilen kısmının **nasıl kullanılacağına dair dokümantasyon** ve örnek olarak hizmet verir (çünkü yazılı dokümanlar çok hızlı bir şekilde güncelliğini yitirir).

**Google'da testlerin %80'i birim testleridir. Test yazma kolaylığı ve çalıştırma hızı, mühendislerin günde birkaç bin birim test koşması anlamına gelir.**

# İyi Birim Testleri Nasıl Yazılır

board `Piece` enum tipinde 2D bir dizidir.

```
public enum Piece { RED, YELLOW; }
```

```
public class Connect4 {

    private static final int WINNING_SEQUENCE = 4;

    final Piece[][] board;
    final int cols, rows;
    Piece turn;
    boolean gameOver;
    Piece winner;

    public Connect4(int cols, int rows) {
        this.cols = cols;
        this.rows = rows;
        board = new Piece[cols][rows];
        turn = Piece.RED;
        gameOver = false;
        winner = null;
    }
```

**“package private” üyeler**

**public metotlar**

**“package private” metotlar**

```
    public boolean isGameOver();

    public Piece winner();

    public Piece whoseTurn();

    public Piece getPieceAt(int col, int row);

    public void makeMove(int col);

    int firstAvailableRow(int col);

    boolean isValidCol(int col);

    boolean isValidRow(int row);

    boolean isBoardFull();

    boolean isGameWon();

    boolean isGameWon(int col, int row, int dCol, int dRow);
}
```

# Bir Test Problemi

To: hoca@erciyes.edu.tr
From:ersoy_babannemiyediler@erciyes.edu.tr
Subject: Test ile ilgili problem - Lütfen yardım edin!!!

Merhaba Hocam

Üçüncü yıl tez projemin koduna bazı birim testleri yazıyorum. Java ile yazılmadı, ancak son derste söylediğiniz gibi, tüm ilkeler hala geçerlidir ve eşdeğer araçlar mevcuttur, bu nedenle tüm tavsiyelerinize uyabilirim!

Ama sonra projeme yeni bir özellik ekledim, birçok testim bozuldu. Bir hata yoktu ancak tüm testlerin güncellenmesi gerekiyordu. Ayrıca testlerimin çoğu dün yazıldığı için çoğunun ne için olduğunu veya ne işe yaradığını aslında hatırlayamadım. Bu yüzden çoğunu çöpe atmak zorunda kaldım.

Otomatik testlerin geliştirmeyi hızlandırmaya yardımcı olduğunu söylediniz, ancak bunun çözülmesi uzun zaman aldı. Bütün bunları tekrar yaşamak istemiyorum.

Ne yapmalıyım?

Saygılarımla,
Ersoy

# Bir Test Çözümü

To: ersoy_babannemiyediler@erciyes.edu.tr
From:hoca@erciyes.edu.tr
Subject: Re: Test ile ilgili problem - Lütfen yardım edin!!!

Merhaba Ersoy,

Korkmaya gerek yok.

Muhtemelen testleriniz kodunuzun iç yapısı ve nasıl çalıştığı hakkında çok fazla varsayımda bulunmuştur. Bu, kod her değiştiğinde testleri güncellemeniz gerektiği anlamına gelir.

Bir sonraki derste testlerin **uygulama yerine davranışa nasıl odaklanması gerektiğini** ve **anlaşılır testlerin nasıl yazılacağını** anlatacağım.

Saygılarımla,
Hoca

# Sürdürülebilirliğin Önemi

**Bu senaryoda iki önemli sorun vardır:**

1. Birim testleri **kırılgandı**. Gerçek bir hata yaratmayan, zararsız ve ilgisiz bir değişikliğe tepki olarak bozuldular.

2. Birim testleri **belirsizdi**. İlk etapta testlerin ne yaptığı belli olmadığından testlerin nasıl düzeltileceğini anlamak zordu.

   Bu durum koda ve testlerine birden fazla katkıda bulunan olduğunda kolayca gerçekleşebilir (gerçek hayattaki yazılım projelerinde aynı anda birçok kişinin üzerinde çalışması muhtemeldir).

# Kırılgan (Brittle) Birim Testleri Nasıl Yazılmaz?

# Connect4

**İyi ve kötü birim testi örneklerini göstermek için** GitHub deposunun `bs450.connect4` paketinde `Connect4` sınıfı için yazılan testlere bakacağız.

`Connect4` sınıfının örnekleri, ızgaradaki sayaçların konumları, sırasının kimde olduğu vb. dahil olmak üzere Connect4 oyununun durumunu temsil eder.

*Herkes oyunun nasıl çalıştığını biliyor mu?*

# Değişmeyen Testler İçin Çabalayın

**Kırılgan testleri önlemenin temel stratejisi, projenin gereksinimleri değişmedikçe değişmesi gerekmeyecek testler yazmaya çalışmaktır:**

- Dahili yeniden düzenlemeler testi değiştirmemelidir.
- Yeni özellikler mevcut özellikleri etkilememelidir.
- Hata düzeltmeleri testlerde güncelleme gerektirmemelidir.
- Davranış değişiklikleri: bunlar testlerde değişiklik gerektirebilir.

board `Piece` enum tipinde 2D bir dizidir.

```
public enum Piece { RED, YELLOW; }
```

```
public class Connect4 {

    private static final int WINNING_SEQUENCE = 4;

    final Piece[][] board;
    final int cols, rows;
    Piece turn;
    boolean gameOver;
    Piece winner;

    public Connect4(int cols, int rows) {
        this.cols = cols;
        this.rows = rows;
        board = new Piece[cols][rows];
        turn = Piece.RED;
        gameOver = false;
        winner = null;
    }
```

**“package private” üyeler**

**public metotlar**

**“package private” metotlar**

```
    public boolean isGameOver();

    public Piece winner();

    public Piece whoseTurn();

    public Piece getPieceAt(int col, int row);

    public void makeMove(int col);

    int firstAvailableRow(int col);

    boolean isValidCol(int col);

    boolean isValidRow(int row);

    boolean isBoardFull();

    boolean isGameWon();

    boolean isGameWon(int col, int row, int dCol, int dRow);
}
```

# İki Farklı Test

Connect4 uygulamamızın yerçekimine uyması gerekir. Ayrıca eğer bir oyuncu bir sütuna bir taş bırakırsa, bunun doğru sırada olmasını sağlamamız gerekir.

İlk parça ilk sıraya düşecektir. Aynı sütundaki ikinci parça ikinci sıraya vb. düşecektir.

**Şimdi bunu test edeceğiz. Size iki farklı test göstereceğim.**

Biri uygulamayı test eder, diğeri davranışı test eder.

**Hangisinin gelecekte değişmesi (yani kırılgan olması) daha muhtemeldir?**

# Uygulama Testi

Değişkeni doğrudan değiştirir

```
@Test
public void testFirstAvailableRow() {
    Connect4 c4 = new Connect4(7, 6);
    c4.board[0][3] = Piece.RED;
    assertThat(c4.firstAvailableRow(0), equalTo(4));
}
```

Package-private metodu kullanarak uygulamayı doğrular

# Davranış Testi

```java
@Test
public void shouldPlaceCounterAboveLast() {
    Connect4 c4 = new Connect4(7, 6);

    c4.makeMove(0); // RED
    assertThat(c4.getPieceAt(0, 0), equalTo(Piece.RED));

    c4.makeMove(0); // YELLOW
    assertThat(c4.getPieceAt(0, 1), equalTo(Piece.YELLOW));

    c4.makeMove(0); // RED
    assertThat(c4.getPieceAt(0, 2), equalTo(Piece.RED));

    c4.makeMove(0);
    assertThat(c4.getPieceAt(0, 3), equalTo(Piece.YELLOW));
}
```

Bu test, neredeyse bir Connect4 oyunu oynayacak kadar genel API'yi kullanır.

Ortaya çıkan davranış (panoda bir değişiklik) **public** bir yöntem kullanılarak kontrol edilir.

# Uygulama Testi

Değişkeni doğrudan değiştirir

```
@Test
public void testFirstAvailableRow() {
    Connect4 c4 = new Connect4(7, 6);
    c4.board[0][3] = Piece.RED;
    assertThat(c4.firstAvailableRow(0), equalTo(4));
}
```

Package-private metodu kullanarak uygulamayı doğrular

**Board'u farklı şekilde uygulamaya karar verirsek bu teste ne olur?** (Örneğin, dizideki satırları ve sütunları değiştirin, panoyu tamamen ayrı bir sınıf kullanarak yeniden düzenleyin, vb.)

# Kırılgan Testlerin Önlenmesi

Aşağıdakileri yaparak **değişmeyen testler** için çabalayın:

1. Yalnızca **public** metotları çağırarak test edin.
2. Nasıl **elde edildiğini** değil, sonuçların **ne olduğunu** doğrulayın.

**Davranışın aksine uygulamayı test etmeye odaklanırsanız kırılgan testler elde edersiniz.**

**Bu yüzden her zaman davranışa karşı test yapmayı tercih edin.**