BSM462
Yazılım Testi

Hafta - 3

Tasarım ve Test Edilebilirlik

Dr. Öğr. Üyesi M. Fatih ADAK

fatihadak@sakarya.edu.tr

İçerik

- Test Edilebilirlik
- Test Edilebilirlik Amaç
- Teste Başlama Zamanı
- Tasarım İkilemi
- Tasarımın Zayıflığı
- Soyutlama ve Arayüzler
- Sınıf Tasarım Prensipleri
  - Tek Sorumluluk İlkesi
  - Açık Kapalılık İlkesi
  - Liskov Değiştirme İlkesi
- Test Edilebilirlik Özellikleri
- Program Geliştirme Teknikleri
- Defansif Programlama
- Sözleşme Odaklı Tasarım
- Test Odaklı Geliştirme
- Taklit Nesneleri (Mock Objects)
- Mockito
- Eclipse Uygulama

Test Edilebilirlik

- Test Edilebilirlik = Kontrol Edilebilirlik + Görünürlük

- Kontrol Edilebilirlik : Girdileri testte uygulayabilme ve belirtilen durumlara yerleştirme yeteneği.

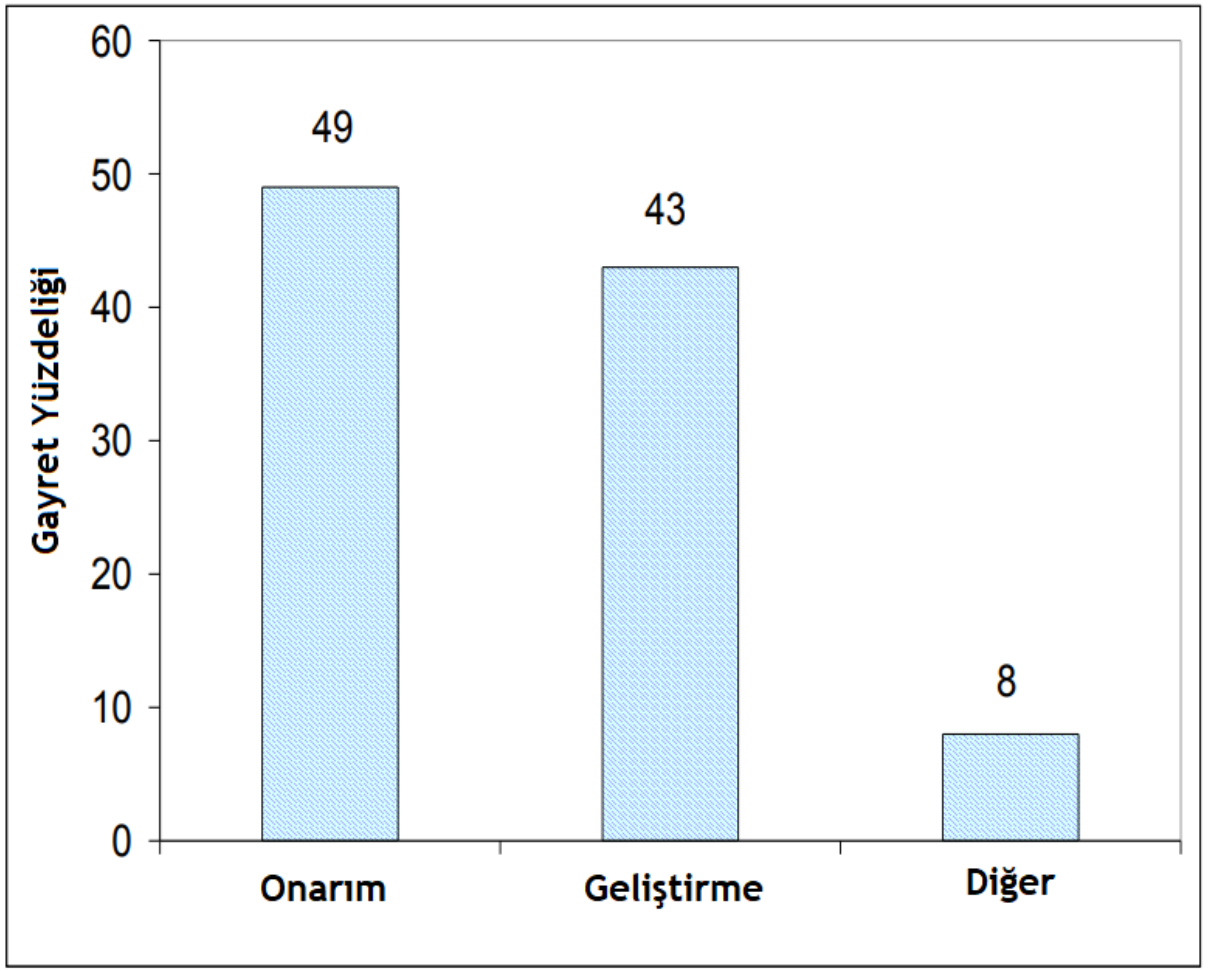
- Görünürlük : Durumları gözlemleme yeteneği.

«Görebildiğimiz test edebildiğimizdir.»

Test Edilebilirlik Amaç

- Yazılım kalitesini arttırmak
- Süreci test etmek için kolaylık sağlar.
- Test otomasyonu için destek verir.
- Maliyeti düşürür.
  - Yazılım şirketlerinin bir çoğu %60-80 arası bütçelerini destek verebilmek için harcarlar.

Test Edilebilirlik Amaç



Nasıl Daha İyi Test Edilebilirlik

- Tasarıma daha fazla zaman harcanmalıdır.
  - İyi tasarım, kolay ve daha iyi test edilmeyi sağlar.
- Test edilebilirlik özellikleri eklenmelidir.
- Geliştirme süreci boyunca özel teknikler kullanılmalıdır.

Teste Başlama Zamanı

- İlk günden başlanmalıdır.
- Test mühendisleri daha ilk günden geliştirme sürecine dahil olmalıdırlar.
- Tasarım değişikliklerine bir yazılım açık olmalıdır. Böylelikle test ihtiyaçlarını karşılayabilir.

Tasarım İkilemi

- Tasarım için verilen süre
- Mükemmel araçları kullanmak

Yazılım Projeleri

Planlanan

Ortaya Çıkan

Tasarımın Zayıflığı

- Kötü tasarımın belirtileri
  - Değişikliğe olan direnç
  - Dayanıklı değil
  - Taşınabilir değil
- Kötü tasarımın nedenleri
  - Soyutlamanın eksikliği
  - Kapsüllemenin tutarsızlığı
  - Nesneler ile katmanlar arası bağlılık

Değişikliğe Olan Direnç için ADKAR Modeli

awareness
Değişime olan ihtiyaç bilinci

desire
Değişime katılma arzusu

knowledge
Nasıl değiştirileceği hakkında bilgi sahibi

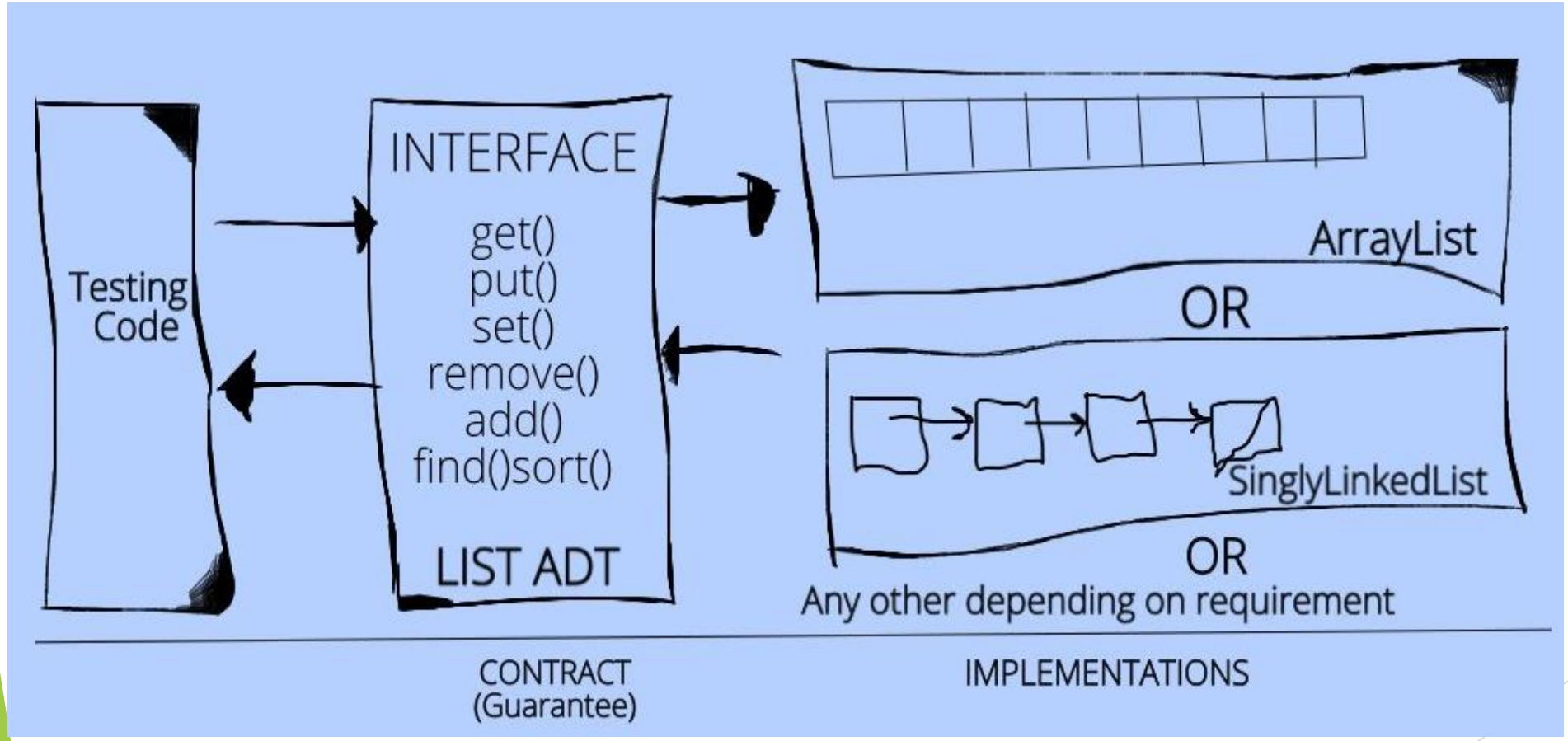
ability
Gerekli beceri ve davranışları uygulama yeteneği

reinforcement
Değişimi sürdürmek için pekiştiren

Soyutlama ve Arayüzler

- Gerçekleştirmeden ayrı bir arayüz
  - Böylelikle kalıtımda sadece arayüz kullanılabilir.
- Çok biçimliliği desteklemek için override kullan
- Böylelikle daha modüler bir yazılım testi gerçekleştirmek mümkündür.

Testte Arayüz Kolaylığı



Sınıf Tasarım Prensipleri

- SRP (Single Responsibility Principle) Tek Sorumluluk İlkesi
  - Her sınıfın yerine getireceği sadece bir yükümlülüğü vardır.
- OCP (Open-Closed Principle) Açık-Kapalılık İlkesi
  - Bir modül genişletilmeye açık olmalı, düzenlemelere kapalı olmalıdır.
- LSP (Liskov Substitution Principle) Liskov Değiştirme İlkesi
  - Kalıtım alınan sınıftan referans kullanan fonksiyonlar kalıtım alınan sınıfların nesnelerini bilgileri olmadan kullanabilmeliler.
  - Fakat bir modülün fonksiyonelliğini etkilememeliler.

SRP (Single Responsibility Principle) Tek Sorumluluk İlkesi

- Kötü Örnek

```
interface IEmail{
    public void setGonderen(String gonderen);
    public void setAlan(String alan);
    public void setMesaj(String mesaj);
}

class Email implements IEmail{
    public void setGonderen(String gonderen){ /*Göndereni ayarlar*/ }
    public void setAlan(String alan){ /*Alanı ayarlar*/ }
    public void setMesaj(String mesaj){ /*Mail mesajını ayarlar*/ }
}
```

- İyi Örnek

```
interface IEmail{
    public void setGonderen(String gonderen);
    public void setAlan(String alan);
    public void setMesaj(IIcerik mesaj);
}
interface IIcerik{
    public String StringOlarakIcerik();
}
class Email implements IEmail{
    public void setGonderen(String gonderen){ /*Göndereni ayarlar*/ }
    public void setAlan(String alan){ /*Alanı ayarlar*/ }
    public void setMesaj(IIcerik mesaj){ /*Mail mesajını ayarlar*/ }
}
```

OCP (Open-Closed Principle)
Açık-Kapalılık İlkesi

- Kötü Tasarım

```
class Sekil{
    public:
        int Tur;
};
class Kare : public Sekil{
    public:
        Kare(){
            Tur=2;
        }
};
class Dikdortgen : public Sekil{
    public:
        Dikdortgen(){
            Tur=1;
        }
};
class Editor{
    public:
        void Ciz(Sekil *sk){
            if(sk->Tur == 1)
                DikdortgenCiz((Dikdortgen*)(sk));
            else if(sk->Tur == 2)
                KareCiz((Kare*)(sk));
        }
        void DikdortgenCiz(Dikdortgen *d){  }
        void KareCiz(Kare *k){  }
};
```

- İyi Tasarım

```
class Sekil{
    public:
        virtual void Ciz();
};
class Kare : public Sekil{
    public:
        void Ciz(){
            //Kare çizer
        }
};
class Dikdortgen : public Sekil{
    public:
        void Ciz(){
            //Dikdortgen çizer
        }
};
class Editor{
    public:
        void SekilCiz(Sekil *sk){
            sk->Ciz();
        }
};
```

OCP (Open-Closed Principle)
Açık-Kapalılık İlkesi – Başka Kötü Örnek

- Kötü Tasarım

```
enum Renk{
    SIYAH, BEYAZ, SARI
}
enum Beden{
    DAR, ORTA, BOL
}

public class Urun {
    public String isim;
    public Renk renk;
    public Beden beden;

    public Urun(String isim, Renk renk, Beden beden) {
        this.isim = isim;
        this.renk = renk;
        this.beden = beden;
    }
    @Override
    public String toString() {
        // TODO Auto-generated method stub
        return isim + " - " + renk + " - " + beden;
    }
}
```

```
public class Filtre {
    public Stream<Urun> renkIleFiltrele(List<Urun> urunler,Renk renk){
        return urunler.stream().filter(u -> u.renk == renk);
    }
    public Stream<Urun> bedenIleFiltrele(List<Urun> urunler,Beden beden){
        return urunler.stream().filter(u -> u.beden == beden);
    }
}
```

```
public class Program {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Urun kazak = new Urun("Sarar", Renk.SIYAH, Beden.DAR);
        Urun etek = new Urun("Ipek", Renk.SARI, Beden.ORTA);
        Urun gomlek = new Urun("IGS", Renk.SARI, Beden.ORTA);

        List<Urun> urunler = List.of(kazak, etek, gomlek);
        Filtre filtre = new Filtre();
        System.out.println("Sarı Giysiler:");
        filtre.renkIleFiltrele(urunler, Renk.SARI).forEach(
                    u -> System.out.println(u)
                );
    }

}
```

OCP (Open-Closed Principle)
Açık-Kapalılık İlkesi – İyileştirilmiş Hali

- İyi Tasarım

```
interface Tarif<T>{
    boolean Uygunmu(T urun);
}

public interface Filtreleme<T> {
    Stream<T> filtrele(List<T> urunler, Tarif<T> tarif);
}

public class RenkTarif implements Tarif<Urun>{
    private Renk renk;

    public RenkTarif(Renk renk) {
        this.renk = renk;
    }
    @Override
    public boolean Uygunmu(Urun urun) {
        return urun.renk == renk;
    }
}
```

```
public class Filtre implements Filtreleme<Urun> {
    @Override
    public Stream<Urun> filtrele(List<Urun> urunler, Tarif<Urun> tarif) {
        return urunler.stream().filter(u -> tarif.Uygunmu(u));
    }
}
```

```
public class BedenTarif implements Tarif<Urun> {
    private Beden beden;

    public BedenTarif(Beden beden) {
        this.beden = beden;
    }
    @Override
    public boolean Uygunmu(Urun urun) {
        return urun.beden == beden;
    }

}
```

OCP (Open-Closed Principle)
Açık-Kapalılık İlkesi – İyileştirilmiş Hali

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    Urun kazak = new Urun("Sarar", Renk.SIYAH, Beden.DAR);
    Urun etek = new Urun("Ipek", Renk.SARI, Beden.ORTA);
    Urun gomlek = new Urun("IGS", Renk.SARI, Beden.ORTA);

    List<Urun> urunler = List.of(kazak, etek, gomlek);
    Filtre filtre = new Filtre();
    System.out.println("Sarı Giysiler:");
    filtre.filtrele(urunler, new RenkTarif(Renk.SARI)).forEach(
                u -> System.out.println(u)
            );
}
```

LSP (Liskov Substitution Principle)
Liskov Değiştirme İlkesi

```java
public class Dikdortgen {
    protected int genislik;
    protected int yukseklik;

    public void setGenislik(int genislik){
        this.genislik = genislik;
    }
    public void setYukseklik(int yukseklik){
        this.yukseklik = yukseklik;
    }
    public int getGenislik(){
        return genislik;
    }
    public int getYukseklik(){
        return yukseklik;
    }
    public int Alan(){
        return genislik * yukseklik;
    }
}
```

```java
public class Kare extends Dikdortgen {
    @Override
    public void setGenislik(int genislik){
        this.genislik = genislik;
        this.yukseklik = genislik;
    }
    @Override
    public void setYukseklik(int yukseklik){
        this.genislik = yukseklik;
        this.yukseklik = yukseklik;
    }
}

public class LSP {

    private static Dikdortgen getYeniDikdortgen(){
        return new Kare();
    }

    public static void main(String[] args) {
        Dikdortgen d = LSP.getYeniDikdortgen();
        d.setGenislik(5);
        d.setYukseklik(10);

        System.out.println(d.Alan());
    }

}
```

Test Edilebilirlik Özellikleri

- Olayların log kaydını tutmak
- Ayrıntılı mod desteği
- Testler için belli noktalar kayıt eklemek
- Script şeklinde yükleme işlemleri

```
Select
    EventType = EVENTDATA().value('(EVENT_INSTANCE/EventType)[1]', 'sysname'),
    PostTime = EVENTDATA().value('(EVENT_INSTANCE/PostTime)[1]', 'datetime'),
    LoginName = EVENTDATA().value('(EVENT_INSTANCE/LoginName)[1]', 'sysname'),
    UserName = EVENTDATA().value('(EVENT_INSTANCE/UserName)[1]', 'sysname'),
    DatabaseName = EVENTDATA().value('(EVENT_INSTANCE/DatabaseName)[1]', 'sysname'),
    SchemaName = EVENTDATA().value('(EVENT_INSTANCE/SchemaName)[1]', 'sysname'),
    ObjectName = EVENTDATA().value('(EVENT_INSTANCE/ObjectName)[1]', 'sysname'),
    ObjectType = EVENTDATA().value('(EVENT_INSTANCE/ObjectType)[1]', 'sysname'),
    CommandText = EVENTDATA().value('(EVENT_INSTANCE//TSQLCommand[1]/CommandText)[1]', 'nvarchar(max)')
```

Test Edilebilirlik Özellikleri devam

- Avantajları
  - Hatalar daha fazla yayılmadan tespit
  - Problemin kaynağını ve yerini kolayca bulmak
  - Bug'ları yeniden üretebilme yeteneği
- Ayrıca
  - Zaman kayıtları log işleminde faydalı olabilir.
  - Log olayının iyi açıklanarak kaydı önemlidir.
  - Bir log formatında buluşulmalıdır.
  - Uyarıyı log kaydına yerleştirmekten daha önemlisi gerekli yerde hatayı fırlatmak.

Program Geliştirme Teknikleri

- Defansif programlama
  - Yazılımda her ayrıntı için test düşünme ve yazma
- Sözleşme odaklı tasarım
- Test odaklı geliştirme
- Taklit Nesneleri (Mock Objects)

Defansif Programlama

```
void test( int *p ) {
    assert( p != 0 );
    if (p == 0)
        return;
    // p artık kullanılabilir
}
```

Defansif Programlama

- Değer sabitlerini tanımlama kısmı hariç değer sabitlerinin kullanımından kaçın.

```
procedure TTeletextGenerator.GetPreviousPage(client: TSTTNetClient; var pageName:
  TPageName);
begin
  pageName.pg := client.data.lt.lastgetpg;
  pageName.sb := client.data.lt.lastgetspg;
  if pageName.pg > LowestReadPgno(client) then
  begin
    dec (pageName.pg);
    while ((pageName.pg > HighestReadPgno(client))
      or (DSubp (pageName) = 0)) and (pageName.pg > LowestReadPgno(client)) do dec (pageName.pg);
    if (pageName.pg < LowestReadPgno(client)) or (DSubp (pageName) = 0) then client.ErrorCode := 24
    else
    begin
      pageName.sb:=1;
      FState.lchr:=7;
    end;
  end
  else client.ErrorCode:=24;
end;
```

Sözleşme Odaklı Tasarım

- Ön Koşullar
  - Bir metot çağrıldığında neler doğru olmalıdır.
- Hedef Şartlar
  - Metot işlemini başarıyla sonuçlandırdığında neler doğru olmalıdır.
- Sınıf Değişmezleri
  - Her bir sınıf örneği için neler doğru olmalıdır.

Test Odaklı Geliştirme

- Önce testi yaz, daha sonra bu testin geçmesi için asgari gereken kodu yaz.
- Test, geliştiriciye gerekli olan özellikleri sağlar.
- Geliştirici için hızlı bir geribildirim sağlar.
- Yazılımda gerekli yerleri vurgulaması hızlıdır.
- Geliştirme sürecini hızlandırır.
- Bütün küçük parçalara fonksiyonellik kazandırır.
- %100'lük birim testinden emin olur.

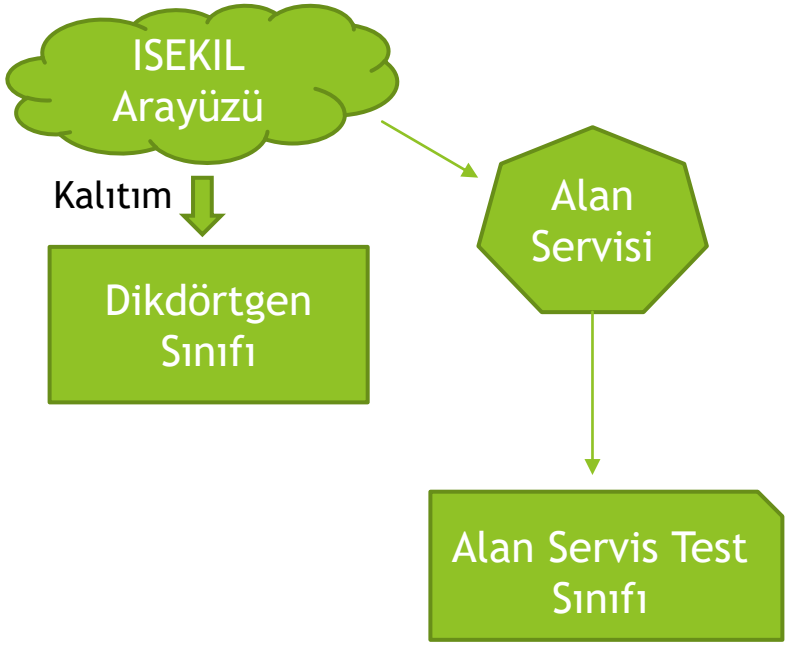
Taklit Nesneleri (Mock Objects)

- Mock nesneleri, gerçek olanların davranışlarını taklit eden simule nesnelerdir.
  - Eğer gerçek nesneler çok kompleks ve birim testi için uygun değilse mock nesneleri kullanılır.
  - Gerçek nesnenin verdiği sonuç kararsız ise mock nesneleri kullanılır.
  - Gerçek nesne halen oluşturulamamışsa mock nesneleri kullanılır.
- Günümüzde birçok programlama dili mock nesneleri oluşturmak için bir framework sağlamıştır.

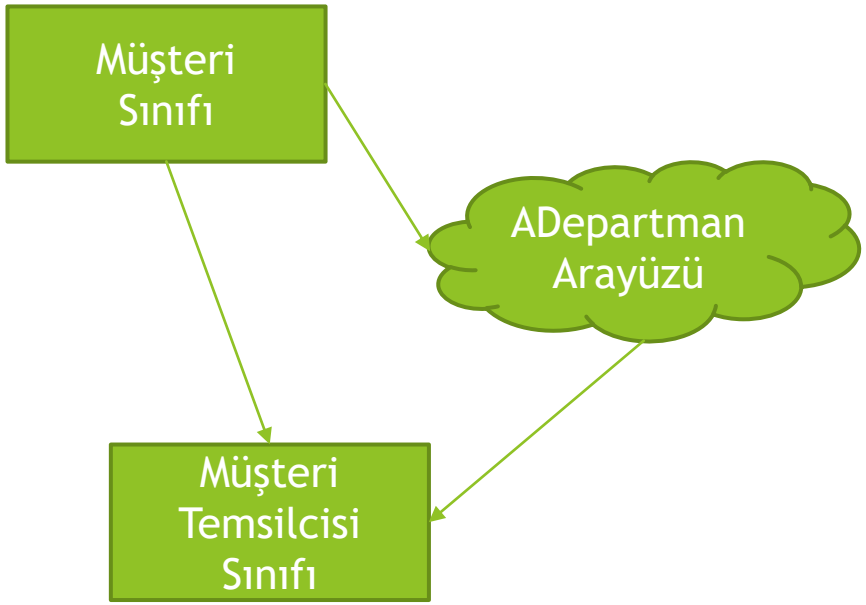
Mockito
http://mockito.org

- Mockito bir test çerçevesidir (Framework).
- Öğrenilmesi kolaydır.
- Mock nesneleri oluşturmanızı sağlar.
- Java için mock nesneleri kütüphanesidir.
- İndirilebilir tam link
  - https://repo1.maven.org/maven2/org/mockito/mockito-all/1.10.19/mockito-all-1.10.19.jar

Eclipse Java Örnek Uygulama



Eclipse Java Kompleks Örnek



Referanslar

- Naik, Kshirasagar, and Priyadarshi Tripathy. *Software testing and quality assurance: theory and practice*. John Wiley & Sons, 2011.
- Ammann, Paul, and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.
- Padmini, C. "Beginners Guide To Software Testing." (2004).
- Archer, Clark, and Michael Stinson. *Object-Oriented Software Measures*. No. CMU/SEI-95-TR-002. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1995.
- Pandey, Ajeet Kumar, and Neeraj Kumar Goyal. *Early Software Reliability Prediction*. Springer, India, 2015.