D ile Güvenli Programlar

Zafer ÇELENK

http://www.zafercelenk.net
http://www.ddili.org

Güvenli Programlar

- Güvenli programlama ne demektir ?
- Programlamada güvenlik neden önemlidir ?
- Somut bir örnek; Therac-25 Tıp Kazası

Therac-25 Cihazı

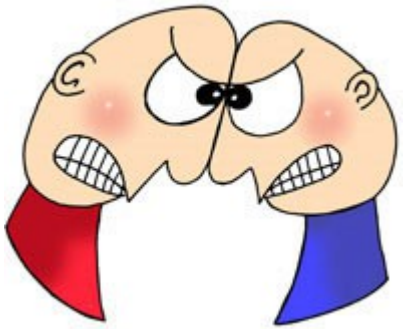
Assert

- Assert (iddia ediyorum) öyleyse doğrudur ;)

  Kodun sağlamlığını arttıran ve programın yanlış sonuçlar doğuracak işlemlerle devam etmesini önleyen bir olanak.

- Assert felsefesi nedir ?

  Bir dosyanın her zaman var olduğuna güvenmek!



- Assert, teste dayalı programlamanın atası.

  Bir varsayımda bulunuyorsanız bunu kodunuzda açıkça belirtiniz. Assert birim testleri ve sözleşmeli programlama olanaklarının da temelini oluşturur.

Assert

- Assert kullanımı için örnek bir durum;

  Derleyici seçeneği -release, assert denetimlerinin sanki programa hiç yazılmamışlar gibi gözardı edilmelerini sağlar. Sadece assert(false) için bu kural geçersizdir.

```
double NotOrtalamasi(double araSinavNotu, double yilSonuNotu)
{
    double ortalama = (araSinavNotu + yilSonuNotu) / 2;

    return ortalama;
}
```

```
double NotOrtalamasi(double araSinavNotu, double yilSonuNotu)
{
    // Böyle olduğunu varsayıyorum
    assert(araSinavNotu >= 0);
    assert(yilSonuNotu >= 0, "Sınav notu hatalı!");

    double ortalama = (araSinavNotu + yilSonuNotu) / 2;

    return ortalama;
}
```

Assert

- Bir assert ifadesi oluşturmak.

```
assert(mantıksal_ifade);
assert(mantıksal_ifade, mesaj);
```

- Assert'in çalışması ve sonuçları

```
assert(araSinavNotu >= 0, "Sınav notu hatalı!"); //false
```

- AssertError yapısını anlamak

```
core.exception.AssertError@.\assert.d(16): Sınav notu hatalı!
----------------
417E44
417CCF
402021
402DA2
4029C3
434145
----------------
```

Assert

- Bir assert çıktısının anotamisi

```
core.exception.AssertError@.\assert.d(16): Sınav notu hatalı!
```

1 2 3 4

1 – AssertError olanağının bulunduğu modül
core.exception.AssertError

2 – Assert hatasının oluştuğu kod dosyası
Assert.d

3 – Assert ifadesinin bulunduğu satır numarası
16

4 – Eğer varsa ilgili hata için girilmiş hata mesajı
Sınav notu hatalı!

Enforce

- Programın akışını sonlandırma gereksinimi

  if-throw hata atma düzeneğini sarmalar.

- Geliştiricinin acil durum freni *throw*

  throw new Exception("Hata var!")

- Durum kontrolü ve hata fırlatmak

  Eğer hata varsa tüm kapsamları terk et ve programı sonlandır.

Enforce

- Bir hata fırlatma örneği;

```
string VerCihazKodu(string cihazKodu)
{
     if (cihazKodu == null)
     {
          throw new Exception("Hata, Cihaz Kodu değeri null olamaz!");
     }

     return cihazKodu;
}
```

- Daha iyi bir olanak olarak *enforce*

```
string VerCihazKodu(string cihazKodu)
{
     //Şart koşuyorum
     enforce(cihazKodu != null, "Hata, Cihaz Kodu değeri null olamaz!");

     return cihazKodu;
}
```

Scope

- İstisna yakalama olanağı try-catch
  - catch ve finally blokları try bloğu olmadan kullanılamaz.
  - Bu bloklarda kullanılmak istenen bazı değişkenler o noktalarda geçerli olmayabilirler.
  - Bir kapsamdan çıkılırken kesinlikle işletilmesi gereken ifadelerin hepsinin bir arada en aşağıdaki finally bloğuna yazılmaları, ilgili oldukları kodlardan uzakta kalacakları için istenmeyebilir.
- Tüm bunlar için yeni bir olanak *scope*
  - scope(success) : Kapsamdan başarıyla çıkılırken işletilecek olan ifadeleri belirler.
  - scope(failure) : Kapsamdan hatayla çıkılırken işletilecek olan ifadeleri belirler.
  - scope(exit) : Kapsamdan başarıyla veya hatayla çıkılırken işletilecek olan ifadeleri belirler.

Scope

- D dilinde scope örneği;

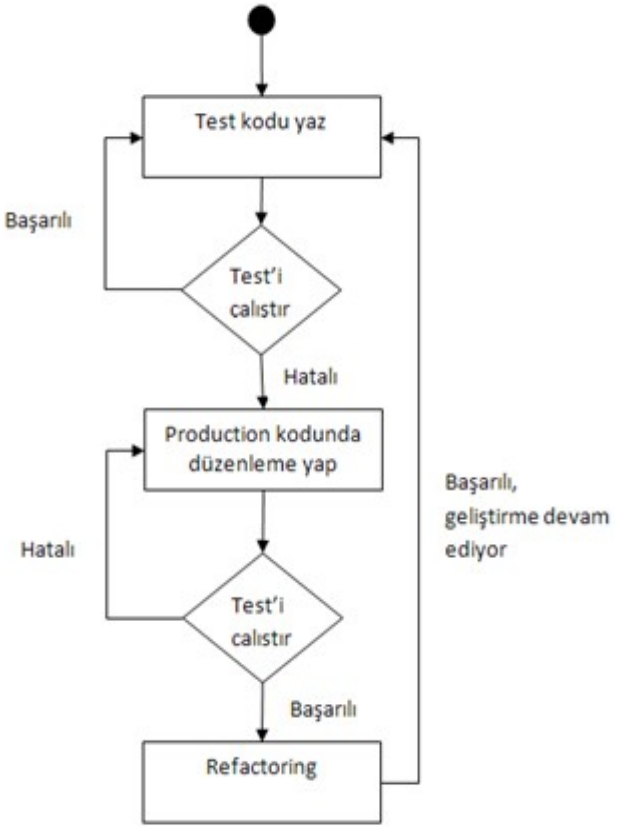
```
void deneme()
{
    scope(exit) writeln("çıkarken 1");

    scope(success)
    {
        writeln("başarılıysa 1");
        writeln("başarılıysa 2");
    }

    scope(failure) writeln("hata atılırsa 1");
    scope(exit) writeln("çıkarken 2");
    scope(failure) writeln("hata atılırsa 2");

    HataAtanBirMetot();
}
```

Unittest

- Teste dayalı programlama (TDD) nedir?

Test-Driven Development (TDD), Birim testler (unittest)

- Sonradan ortaya çıkabilecek hataların oluşturacağı maliyeti en aza indirger.
- Yeniden ele alınan kod bloklarında bozulma olup olmadığını anlamak amacıyla ilgili kod bloğu için önceden oluşturulmuş birim testi çalıştırılarak sonucun doğruluğu kolayca kontrol edilebilir.
- Uzun vadede birim testi olmadan kod yazmaktan daha hızlı ve verimli kod yazmayı sağlar çünkü sonrasında oluşacak hatalara dönüp bakma ihtimalini en aza indirger.
- Yazılan kodun en dip seviyede anlaşılmasını sağlar çünkü çok net anlaşılmayan kodun birim testini yazmak imkansızdır. Bu da çok daha temiz ve hatasız kod bloklarının ortaya çıkmasını sağlar.
- Bir kod bloğunun işlevi birim testine bakılarak çok daha kolay anlaşılabilir, bu da dokümantasyon niteliğinde olabilir.



Unittest

- Birim testleri ve başka birim testi olanakları

  Birim testi bir yazılım projesindeki metotların, fonksiyonların doğru çalışıp çalışmadığını anlamak için oluşturulan testtir.

  JUnit, NUnit, CppUnit, UnitTest++ v.s.

- D dili ve tümleşik unittest olanağı.

  dmd test.d -ofTest -unittest -w -wi

Unittest

- D ile yazılmış bir unittest örneği;

```
int HarfKacinciSirada(string metin, char harf)
{
     int konum = metin.indexOf(harf);

     return konum;
}
unittest
{
     string metin = "zafer";

     assert(HarfKacinciSirada(metin, 'f') == 2);
     assert(HarfKacinciSirada("zafer", 'r') == 4, "Hata Oluştu");
}
```

- Unittest olanağının hatayı yakalaması

```
core.exception.AssertError@unittest(19): unittest failure
```

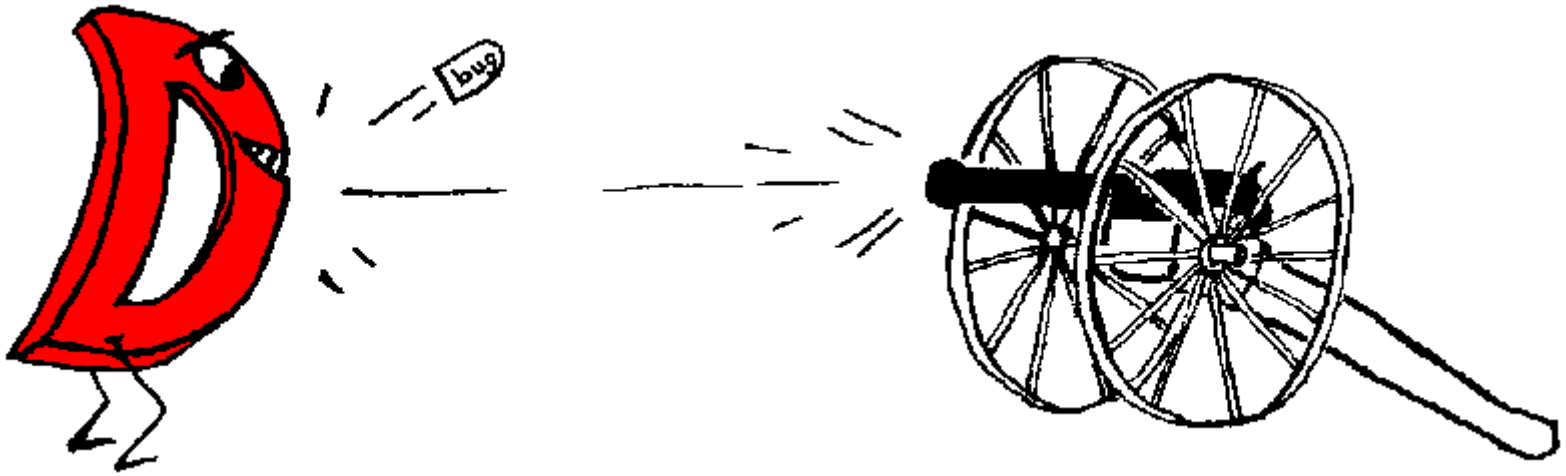
Sözleşmeli Programlama

- Sözleşmeli programlama nedir?

  Metotlar ve onları çağıran kodlar arasında yazısız bazı anlaşmalar vardır. Sözleşmeli programlama, bu anlaşmaları dil düzeyinde belirlemeye yarayan olanaktır.

- Sözleşmeli programlamanın doğuşu ve D

  Sözleşmeli programlama, ticari bir dil olan Eiffel tarafından "design by contract (DBC)" adıyla yayılmıştır. Bu yöntem D dilinde "contract programming" olarak geçer.



Sözleşmeli Programlama

- D ile sözleşmeli programlama yapısı

Metotlar ve onları çağıran kodlar arasında yazısız bazı anlaşmalar vardır.
Sözleşmeli programlama, bu anlaşmaları dil düzeyinde belirlemeye yarayan olanaktır.

```
void BirMetot(in int birParametre, out int başkaParametre)
in
{
    // Metot girişinde yapılacak kontroller
}
out
{
     // Metot çıkışında yapılacak kontroller
}
body
{
    // Metot içinde yapılacak işlemler ...
}
```

Sözleşmeli Programlama

- in bloğu

  Metoda gönderilen parametler denetlenir, ön kontroller yapılır. Bir assert denetiminin başarısız olması sözleşmeyi işlevi çağıran tarafın bozduğunu gösterir.

- out bloğu

  Metoto çıkışında metodun sözünü tutup tutmadığı denetlenir.

- invariant bloğu

  Bir nesnenin üye değişkenlerinin beklenen şekilde olması iyi bir nesne olduğu anlamına da gelecektir.

- body bloğu

  Metodun esas görevini yerine getiren kodların bulunduğu kısım

Sözleşmeli Programlama

- D ile sözleşmeli programlama örneği;

```
void bölüştür(in int toplam, out int birinci, out int ikinci)
in
{
    assert(toplam >= 0);
}
out
{
    assert(toplam == (birinci + ikinci));
}
body
{
    birinci = (toplam >= 7) ? 7 : toplam;
    ikinci = toplam - birinci;
}
```

Birim testlerinin tersine, sözleşmeli programlama normalde etkilidir; etkisizleştirmek için özel bir derleyici veya geliştirme ortamı ayarı gerekir. Bunun için dmd derleyicisinde ***-release*** *seçeneği kullanılır:*

Sözleşmeli Programlama

- Sözleşmeli programlamaya direk destek veren diller
  - Ada 2012
  - D
  - Eiffel
  - Fortress
  - Lisaac
  - Mercury
  - Vala

- Sözleşmeli programalamaya dolaylı destek veren diller

Java	iContract2, Contract4J, jContractor, Jcontract, C4J
JavaScript	Cerny.js, ecmaDebug or jsContract
Perl	the CPAN modules Class::Contract
C# (.NET languages)	Code Contracts
Python	PyDBC, Contracts for Python
Ruby	DesignByContract, contracts.ruby

Son söz

- Güvenli programlama bir koruyucu önlemdir.

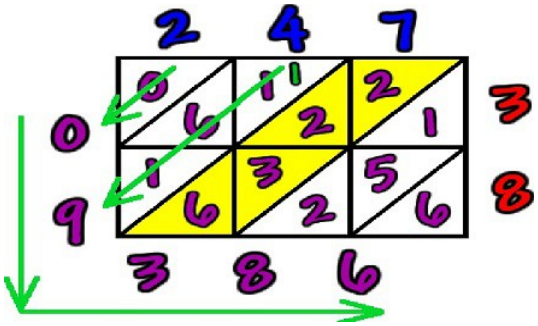
  Programcının bilgisi herhangi bir konuda yetersiz veya yanlış olabilir. Örneğin, kesirli sayıların eşitlik karşılaştırmalarında kullanılmalarının güvensiz olduğunu bilmiyordur.

- Güvenli programlama zaman kazandırır.

Birim testleri ve sözleşmeli programlama olanakları bizi zorlu hatalardan koruyan çok etkili araçlardır. Böylece zamanımızı hata ayıklamak yerine, ondan çok daha zevkli ve verimli olan kod yazmaya ayırabiliriz.

- Kesinlikle doğru olan varsayımları bile denetleyin

"Kesinlikle doğru olan" varsayımları bile denetleyin. Hiçbir varsayım bilerek yanlış olmayacağı için, zaten çoğu hata kesinlikle doğru olan varsayımlara dayanır.



TEŞEKKÜRLER

Ali Çehreli
Salih Dinçer
Mert Ataol
ve
Tütev Ankara
ayrıca
Tüm Katılımcılara