

BSM462

Yazılım Testi

Hafta - 4

Birim Testi

Dr. Öğr. Üyesi M. Fatih ADAK

fatihadak@sakarya.edu.tr

İçerik

- ▶ Birim testi tanım
- ▶ Birim testi aşamaları
- ▶ Statik birim testi
- ▶ Dinamik birim testi
- ▶ Test sürücüsü
- ▶ Alt birim
- ▶ Test verisinin seçimi
- ▶ Kontrol akışı testi
- ▶ Veri akışı testi
- ▶ Domain testi
- ▶ Fonksiyonel program testi
- ▶ Mutasyon testi
- ▶ Ayıklama (debugging)
 - ▶ Kaba kuvvet
 - ▶ Nedenin ortadan kaldırılması
 - ▶ Geri izleme
- ▶ Extreme Programming'te birim testi
- ▶ Birim testi için kullanılan araçlar

Birim Testi - Tanım

- ▶ Geliştirme aşamasında üretilen birimlerin test edilmesidir.
- ▶ Test aşamalarının en alt seviyesidir.
- ▶ Birçok organizasyon bu sorumluluğu program geliştirme ekibine bırakır. Bunun için ayrı kişi veya kişiler bünyesinde barındırmaz.
- ▶ Birim'in tam olarak ne olduğu konusunda üzerinde anlaşılmış bir açıklama yoktur.
 - ▶ Bazıları birimi fonksiyon olarak alır.
 - ▶ Bazıları birimi sınıf olarak alır.
- ▶ Her ikisinde de bir modülün test edildiğini söylemek yanlış olmaz.

Yaklaşım

- ▶ Her bir birim tek başına ele alınmalıdır.
- ▶ Bunun iki önemli nedeni vardır.
 - ▶ Test aşamasında bulunan hatalar spesifik olarak nereden kaynaklandığı bulunabilir ve kolay onarılır.
 - ▶ Her birimin beklenen çıktıyı doğru olarak verdiği onaylanabilmelidir.

Programcının Takip Edeceği Yol

- ▶ Bir birim testinin başarıyla gerçekleşebilmesi için;
 - ▶ Her satır kodun çalıştırılması gerekir. Buda her satır kodun test edilmesi demektir.
 - ▶ Birimdeki her işlemi doğru ve yanlış olarak değerlendirebilmek için çalıştırılmalıdır.
 - ▶ Birimin amaçlanan işlevini yerine getirdiğine ve bilinmeyen hataların kalmadığına emin olunmalıdır.
- ▶ Yine de bilinmesi gereken, bütün hataların tespiti imkansız olduğudur. Bazı hatalar ilerleyen test aşamalarında ortaya çıkacak bazıları ise son kullanıcılar tarafından tespit edilecektir.

Birim Testi Aşamaları

- ▶ Birim testi iki tamamlayıcı aşamada yürütülür.
 - ▶ Statik Birim Testi
 - ▶ Dinamik Birim Testi



Statik Birim Testi

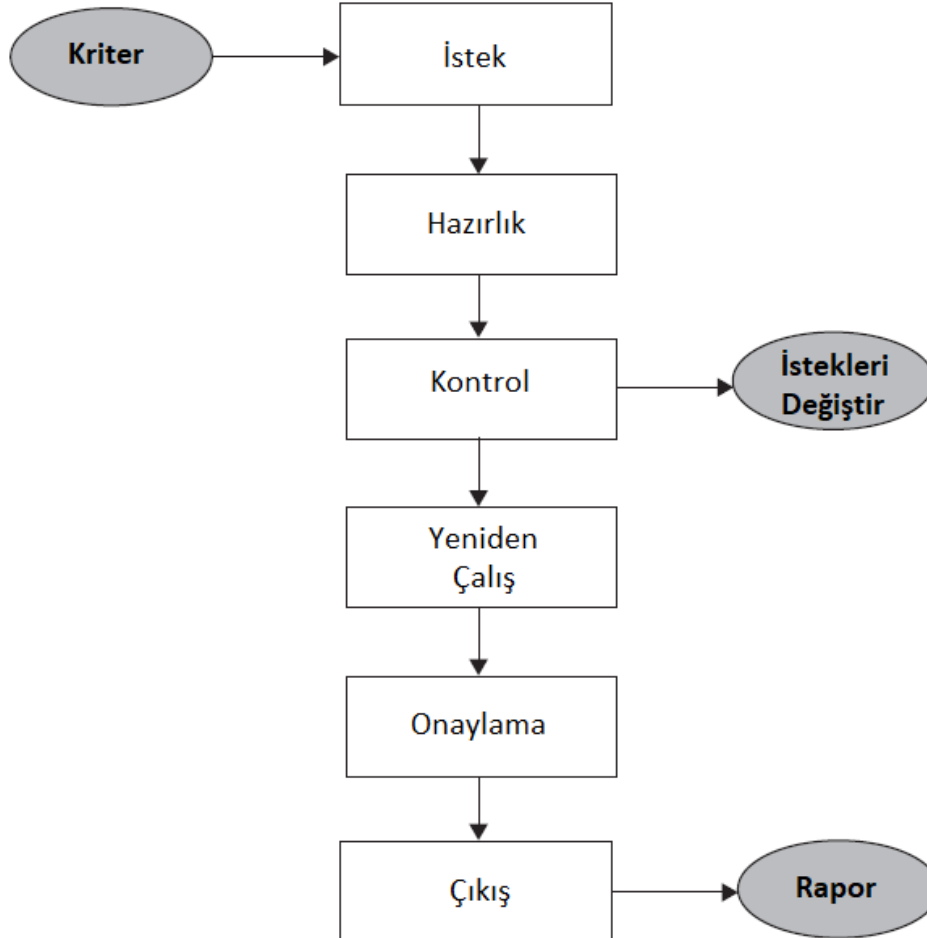
- ▶ Bu tür birim testinden programcı birimi çalıştırmaz.
- ▶ Onun yerine çalışma anında ortaya çıkabilecek mümkün olan bütün davranışları inceler.
- ▶ Her birimin kodu ihtiyaçlar karşısında onaylanır.
- ▶ Daha ucuz bir yöntemdir.
- ▶ Dinamik birim testinin bir alternatifi değildir.
- ▶ Genelde dinamik testten önce yapılır.
- ▶ Statik Birim testi genelde iki temel yöntemle uygulanır.
 - ▶ Kontrol (Muayene)
 - ▶ İzlenecek Yol

Statik Birim Testi - Kontrol Muayene

- Birimin adım adım bir akran grubu tarafından deęerlendirilmesidir.
- Her adımdan önce belirlenen kriterlere g re kontrol edilir.



Statik Birim Testinin Gerçekleştirilmesi



Statik Birim Testinin Gerçekleştirilmesi

1. Adım : İstek

- ▶ Testin gerçekleştirilmesi için istekte bulunma.
- ▶ Test yapabilme için birimin hazır olması, bunun için aşağıdaki kriterleri sağlamalıdır.
 - ▶ **Bütünlük:** Birime ait olan bütün kod ulaşılabilir olmalıdır.
 - ▶ **Temel İşlevsellik:** Kod derlenebilmeli ve temel görevlerini yerine getirmelidir.
 - ▶ **Okunabilirlik:** Kod bu aşamada başka programcılar tarafından okunabilir düzeyde olmalıdır.
 - ▶ **Karmaşıklık:** Çok basit düzeyde bir kod olmamalıdır.
 - ▶ **Gereksinim ve Tasarım Dokümanları:** iyi bir dokümantasyon
 - ▶ **Moderatör:** Kod üzerindeki incelemeleri yürütecek biri
 - ▶ **Programcı:** Kodu yazan kişinin hazır bulunması gerekir.
 - ▶ **Sunucu:** Kodu yazmayan sadece değerlendiricilere sunan kişi
 - ▶ **Değerlendirici:** Kodu değerlendirecek uzman kişiler.
 - ▶ **Kaydedici:** Değerlendiricilerin önerilerini ve değişiklikleri kaydeden kişi

Statik Birim Testinin Gerçekleştirilmesi

2. Adım : Hazırlık

- Her değerlendirici toplantıdan önce mutlaka kodu okumalı ne yapmaya çalışıldığı anlaşılmalıdır. Her değerlendirici aşağıdaki maddeleri hazırlar.
 - **Soru Listesi:** Her değerlendirici birim ile ilgili soru listesi hazırlar.
 - **Olası Değişiklik Talepleri:** Bu kusur raporundan ziyade birimdeki değişiklik talepleridir.
 - **Önerilen İyileştirmeler:** Değerlendirici problemlerin nasıl iyileştirileceği hakkında önerilerde bulunabilir.



Statik Birim Testinin Gerçekleştirilmesi

3. Adım : Kontrol

- Bu adım aşağıdaki aşamalardan oluşur. Bir toplantı düzeyindedir.
 - Kodu yazan programcı birimde yapılan işlevselliği genel anlamda anlatır.
 - Sunucu kodu satır satır okur. Bu aşamada değerlendirici soru sorarak olası çıkabilecek kusurların önüne geçmeye çalışır.
 - Kaydedici istek ve değişiklikleri not alır.



Statik Birim Testinin Gerçekleştirilmesi

4. Adım : Yeniden Çalış

- Toplantı sonunda kaydedici bir özet sunum yapar. Geliştirmeleri ve talepleri bildirir.
- Kodu yazanlar problemleri çözmek için ve yeni talepleri yerine getirmek için çalışırlar.



Statik Birim Testinin Gerçekleştirilmesi

5. Adım : Onaylama

- Bu aşama ya moderatör ya da bu amaç için belirlenen ayrı bir kişi tarafından gerçekleştirilir.
- Taleplerin yerine getirildiğini, belirlenen kusurların ortadan kaldırıldığını, problemlerin çözüldüğünü onaylar.



Statik Birim Testinin Gerçekleştirilmesi

6. Adım : Çıkış

- ▶ Eğer aşağıdaki maddeler tamamlanmış ise değerlendirme tamamlanmıştır.
 - ▶ Birimdeki her bir kod satırı denetlendi.
 - ▶ Eğer çok fazla kusur tespit edilmiş ise (kod satır sayısının %5'inden fazla ise) yeniden bir toplantı tertiplendi.
 - ▶ Kod yazarlar ve değerlendiriciler kod üzerinde anlaşma sağladılar.
 - ▶ Bütün talepler rapora yazıldı ve onaylandı
 - ▶ Özet rapor değerlendiricilere verildi.

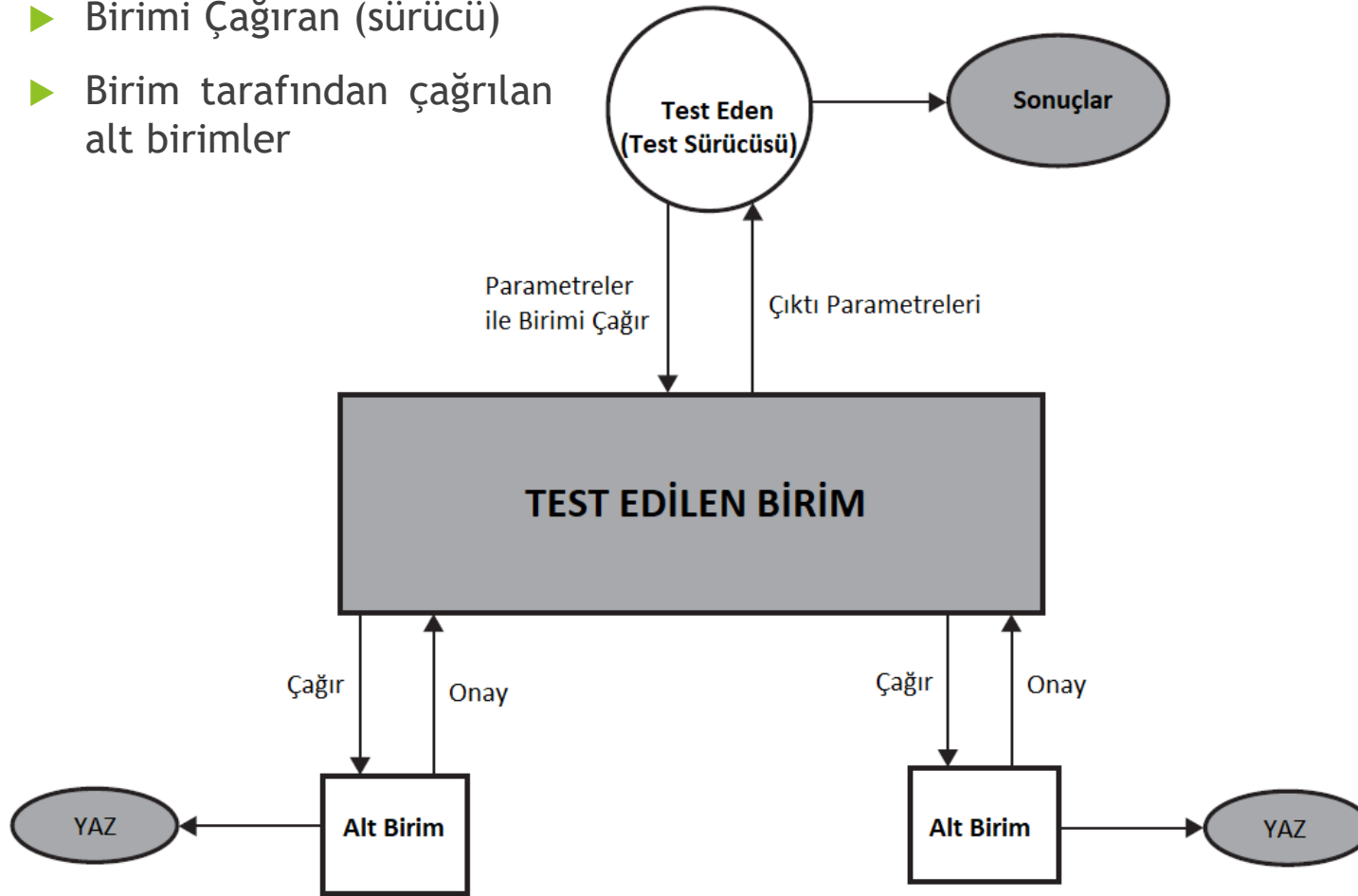


Dinamik Birim Testi

- Bu birim testi, birimi çalıştırma tabanlıdır.
- Girdiler birime verilerek birim çalıştırılır ve sonuç gözlemlenir.
- Beklenen çıktı ile asıl çıktı arasındaki herhangi bir farklılık bir başarısızlığa işaret eder ve bu kodda bir hata olduğunun göstergesidir.

Dinamik Birim Testi

- İki Ana Bölümden Oluşur
 - Birimi Çağır (sürücü)
 - Birim tarafından çağrılan alt birimler



Test Sürücüsü (Test Driver)

- Birimi teste sokan modüldür.
- Sürücü tarafından verilen girdiler ile birim çalıştırılır ve sonuçlar elde edilir.
- Üretilen sonuç ile beklenen sonucu karşılaştırır.

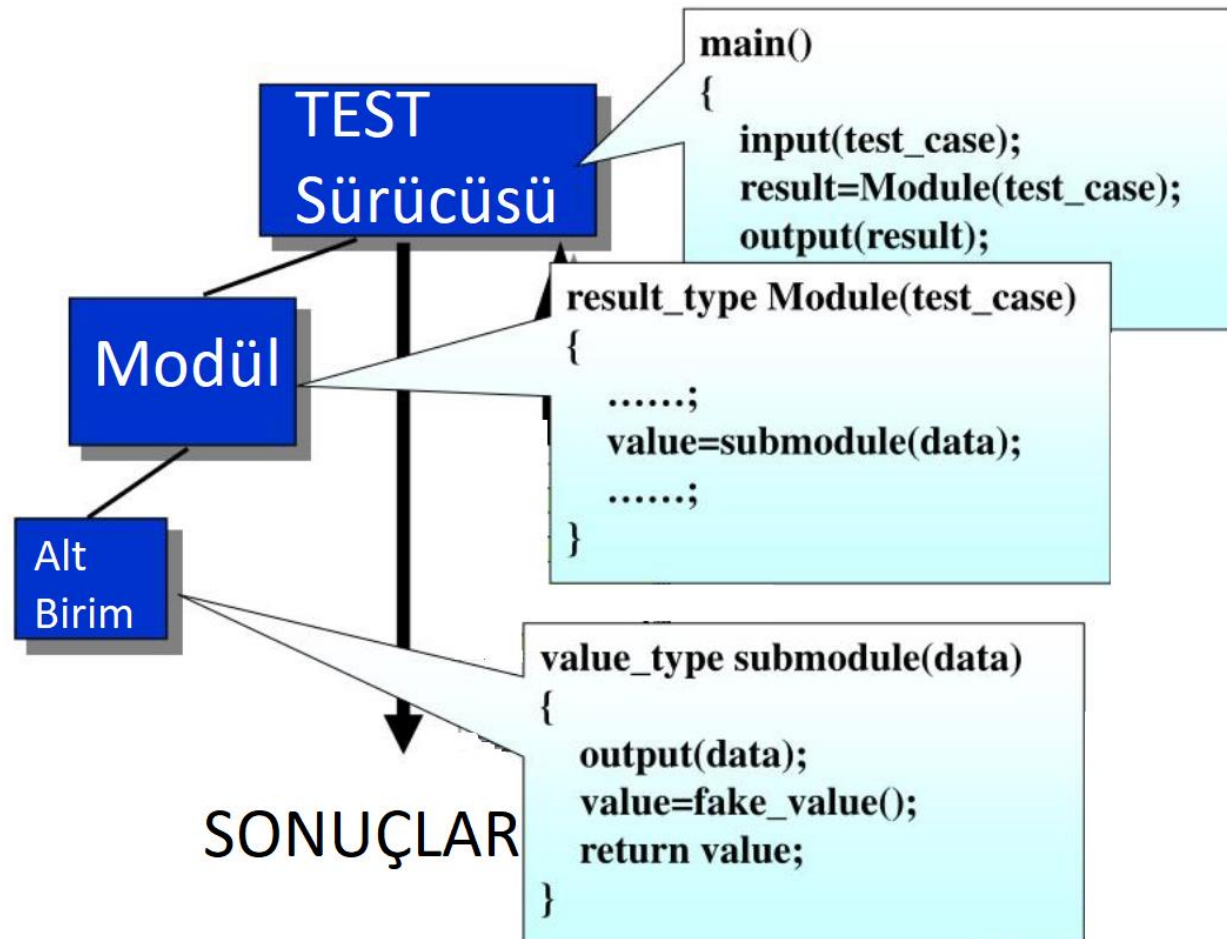


Alt Birim (Stub)

- ▶ Test altındaki birimin çağırdığı alt programdır.
- ▶ İki işlevi yerine getirir.
 - ▶ Kendisinin çalıştığını gösterir bu onun çağırılmasıdır. Ekrana bir mesaj yazarak ta bunu gösterebilir.
 - ▶ Önceden hesaplanmış değeri geri döndürür.



Dinamik Birim Testi



Test Verisinin Seçimi

- ▶ Hataları ortaya çıkarabilmek için test verisinin seçimi önemli bir işlemdir.
- ▶ Test veri seçiminin bağlı olduğu teknikler
 - ▶ Kontrol Akışı Testi
 - ▶ Veri Akışı Testi
 - ▶ Domain Testi
 - ▶ Fonksiyonel Program Testi

Kontrol Akışı Testi

- ▶ Bir akış takip edilir.
- ▶ Örnek bir akış
 - ▶ Bir program biriminden bir kontrol akış grafiği çiz
 - ▶ Birkaç kontrol akış test kriteri seç
 - ▶ Seçilen kriteri karşılamak için Kontrol akış grafiğindeki yolları tanımla
 - ▶ Seçilen yollardan ifadelere dayanan yol üret.
 - ▶ İlgili yolu test etmek için değerler üret.

Veri Akışı Testi

- ▶ Bir akış takip edilir.
- ▶ Örnek bir akış
 - ▶ Bir program biriminden bir veri akış grafiği çiz
 - ▶ Birkaç veri akış test kriteri seç
 - ▶ Seçilen kriteri karşılamak için Veri akış grafiğindeki yolları tanımla
 - ▶ Seçilen yollardan ifadelere dayanan yol üret.
 - ▶ İlgili yolu test etmek için değerler üret.

Domain Testi

- ▶ Kontrol ve Veri Akış testinde yakalamak için özel tür hatalar belirtilmez.
- ▶ Aksine Domain testinde, domain hataları tanımlanır.
- ▶ Test verisi bu domain hatalarını yakalamak üzere seçilir.

Fonksiyonel Program Testi

- ▶ Programın girdi ve çıktı domainin tanımla
- ▶ Verilen girdi domaini için bazı özel değerler seç ve çıktıyı hesapla
- ▶ Verilen çıktı domaini bazı özel değerler seç ve bu çıktıyı üretecek girdileri hesapla
- ▶ Yukarıda seçilen girdiler için farklı kombinasyonlar dene

Mutasyon Testi

- ▶ Mutasyon testi, test verilerinin yeterliliğinin ölçülmesine odaklanan bir testtir.
- ▶ Kontrol akışı, veri akışı gibi bir test etme stratejisi değildir.
- ▶ Küçük bir değişiklik ekleyerek program değiştirilir.
- ▶ Düzenlenmiş programa, değişikliğe uğrayan (mutant) denir.
- ▶ Test başarısız olduğu zaman mutant öldürülmüş sayılır.
- ▶ Mutasyon skoru **öldürülen mutantların** yüzdesi ile ifade edilir.
- ▶ Mutasyon skoru %100 ise mutasyon yeterli ifadesi kullanılır.
- ▶ İstenilen mutasyon skoruna erişene kadar yeni test durumları eklenir.

Mutasyon Testi Örnek

```
#include "stdio.h"

int program(int uzunluk,int *dizi){
    int i;
    int r=0;
    for(i=1;i<uzunluk;i++){
        if(dizi[i] > dizi[r]) r=i;
    }
    return r;
}

int main(){
    int test1[]={1,2,3};
    printf("Test 1 icin: %d \n",program(3,test1));

    int test2[]={1,2,1};
    printf("Test 2 icin: %d \n",program(3,test2));

    int test3[]={3,1,2};
    printf("Test 3 icin: %d \n",program(3,test3));
    return 0;
}
```

Program birimini test etmek için üç farklı test verisi oluşturuldu.

```
Test 1 icin: 2
Test 2 icin: 1
Test 3 icin: 0
```

Mutasyon Testi

Örnek

Mutasyon öncesi elde edilen sonuçlar

```
Test 1 için: 2  
Test 2 için: 1  
Test 3 için: 0
```

Program birimi mutasyona uğratıldı.

Mutant 1

for(i=0;i<uzunluk;i++) satırında i=0 yapıldı



```
Test 1 için: 2  
Test 2 için: 1  
Test 3 için: 0
```

Mutant 2

if(i > dizi[r]) r=i; satırında değer yerine i getirildi.



```
Test 1 için: 2  
Test 2 için: 2  
Test 3 için: 0
```

Mutant 3

if(dizi[i] >= dizi[r]) r=i; satırı >= olarak değiştirildi.



```
Test 1 için: 2  
Test 2 için: 1  
Test 3 için: 0
```

Mutant 4

if(dizi[r] > dizi[r]) r=i; satırında i yerine r getirildi.



```
Test 1 için: 0  
Test 2 için: 0  
Test 3 için: 0
```

Mutasyon Testi Sonuçları

- ▶ Mutant 1 ve Mutant 3, normal test ile aynı sonuçları üretti ve bu mutantlar öldürülmeyecek.
- ▶ Mutant 2 ve Mutant 4 farklı sonuçlar ürettiler ve bu mutantlar öldü kabul ediliyor.
- ▶ 4'te 2'si öldürüldüğü için Mutasyon Skoru: %50
- ▶ Skor düşük ve Mutant 1 ve 3 eşit olmadığı gösterilmeli. Bu durumda farklı bir input dizisi seçilerek mutasyon testi tekrarlanmalıdır.
- ▶ Buradaki amaç verinin test için yeterli olduğunu göstermektir.

Ayıklama (Debugging)

- ▶ Hatanın sebebini bulmak için yapılan işleme ayıklama denir.
- ▶ Genel anlamda üç farklı yaklaşım bulunmaktadır.
 - ▶ Kaba Kuvvet (Brute Force)
 - ▶ Nedenin Ortadan Kaldırılması (Cause Elimination)
 - ▶ Geri İzleme (Backtracking)



Kaba Kuvvet (Brute Force)

- ▶ Birçok programcı tarafından tercih edilir.
- ▶ Bilgisayar hatayı bulsun yaklaşımı kullanılır.
- ▶ Programın birçok yerinde print komutu ile değerler ekrana yazdırılarak hatanın nereden kaynaklandığı bulunmaya çalışılır.
- ▶ Günümüz gelişmiş debug araçları print yaklaşımı gereksiz kılmıştır.
- ▶ Gelişmiş debug araçları ile adım adım kod içerisinde ilerlenip bütün değişkenlerin değerleri gözlemlenebilmektedir.

Nedenin Ortadan Kaldırılması (Cause Elimination)

- ▶ Bu yaklaşım içerisinde iki bölümün bulunduğu bir işlemdir.
 - ▶ Induction - Tüme varım
 - ▶ Başarısızlığa neden olan tüm veriler toplanır.
 - ▶ Ardından toplanan veriler **davranış** ve **bulgular** olarak organize edilir.
 - ▶ Bir hipotez tasarlanır. Toplanan veriler bu hipotezi kanıtlamak veya çürütmek için kullanılır.
 - ▶ Deduction - Tümden gelim
 - ▶ Olası tüm nedenlerin listesi olasılıklarının sırasına göre oluşturulur.
 - ▶ Olasılıklarını düşürmek için listedeki her bir maddenin sebebini ortadan kaldırmak veya kanıtlamak için testler yapılır.

Geri İzleme (Backtracking)

- Bu yaklaşımda programcı hatanın olduğu noktadan itibaren başlayarak kod çalışmasında geriye doğru giderek hatayı bulmaya çalışır.
- Bu daha çok küçük programlarda kullanışlı bir yaklaşımdır.
- Çünkü program boyutu arttıkça geriye gidecek yol sayısı astronomik bir şekilde artabilir.



Ayıklamanın Önemi

- ▶ 1963, ABD'de Mars'a gönderilecek bir roket başarısızlıkla sonuçlandı.
- ▶ 10 milyon dolar kayba neden oldu.
- ▶ Hata Fortran ile yazılmış olan program kodundan kaynaklanmıştı.

- ▶ `DO 5 I = 1,3`
 döngü_gövdesi
 5 CONTINUE

yerine

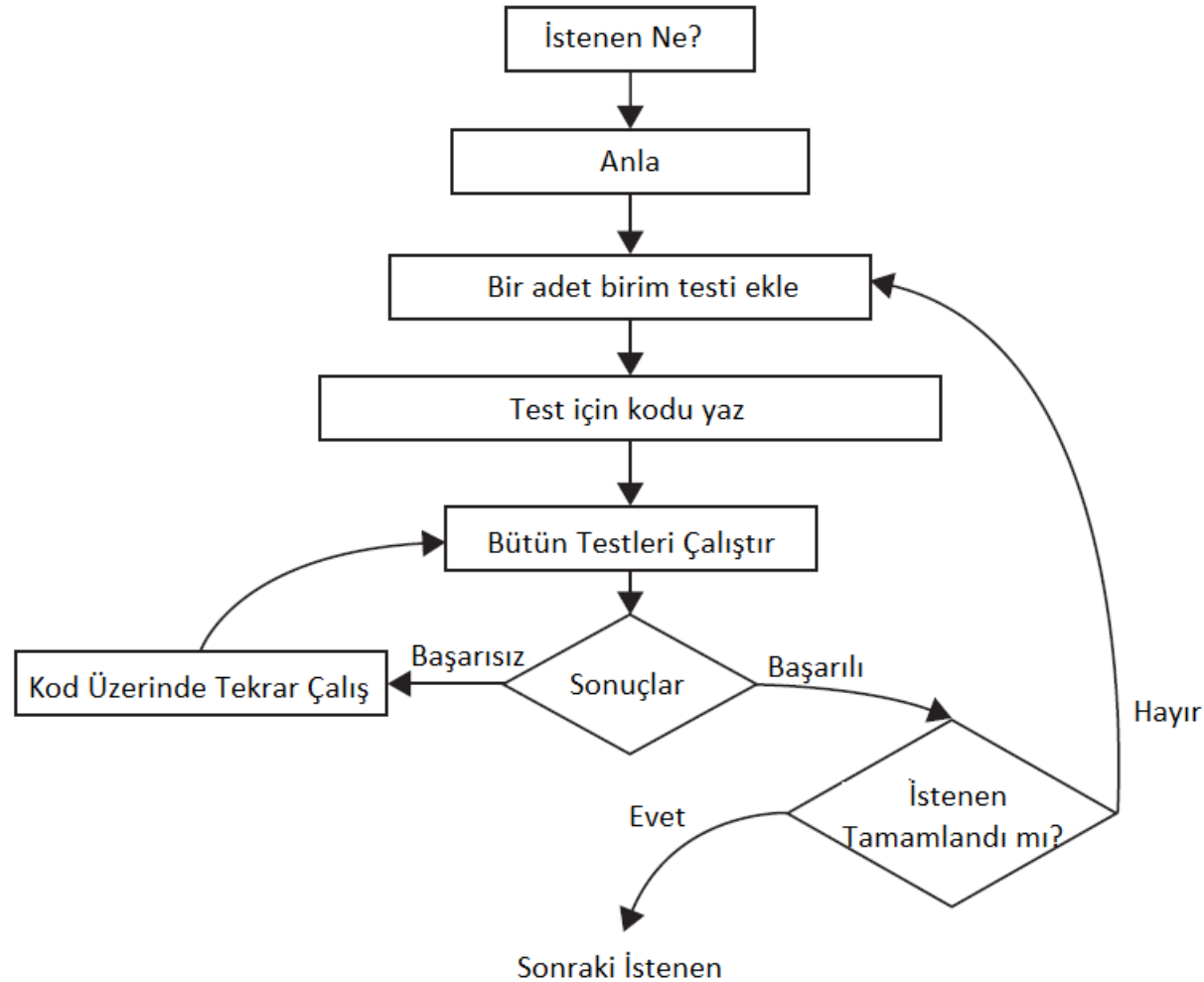
- ▶ `DO 5 I = 1.3`
 döngü_gövdesi
 5 CONTINUE

// 5 döngünün adı

Extreme Programming (XP) İçerisinde Birim Testi

- ▶ Test Driven Development (TDD) önce testi yaz yaklaşımıdır ve XP'de benimsenmiştir.
- ▶ Geliştirilecek kodun önce testi yazılır.
- ▶ İşleyiş
 - ▶ Birkaç tane birim testi yaz
 - ▶ Daha sonra testin geçebilmesi için basit olarak kodu yaz.
 - ▶ Bu işleyiş test edilecek hiçbir şey kalmayana kadar devam eder.

Extreme Programming (XP) İçerisinde Birim Testi



Birim Testi İçin Kullanılan Araçlar

- ▶ Kod Denetleyicisi (Code auditor)
 - ▶ Bu araç ile yazılımın bazı minimum kodlama standartlarını yerine getirip getirmediğini kontrol eder.
- ▶ Sınır Denetleyicisi (Bound checker)
 - ▶ Bu araç ile verilerin bellekte kapladığı yerlerin dışına veri yazımı var mı? kontrol eder.
- ▶ Doküman Oluşturucu (Documenters)
 - ▶ Bu araç kaynak kodu okuyup çağırın/ çağrılanların bulunduğu ağaç diyagramı ve veri modelini oluşturur.
- ▶ Statik Kod Analizcisi
 - ▶ Bazı kompleks ölçüm kriterlerini baz alarak kod yapısını analiz eder.

Birim Testi İçin Kullanılan Araçlar

- ▶ Test Verisi Üretici
 - ▶ Programın istenilen şekilde çalışabilmesi için test verisi seçmeye yardımcı olur.
- ▶ Test harness
 - ▶ Dinamik birim testlerinin yürütülmesini sağlar.
- ▶ Performans İzleyici
 - ▶ Yazılımı oluşturan modüllerin zamana dayalı karakteristiğini gösterir.
- ▶ Versiyon Kontrol Edici
 - ▶ Yazılımın sürümlerini kontrol eden, aradaki farklılıkları izlemeyi kolaylaştıran bir araç

Kod Denetleyicisi

Create the Expression - SSW Code Auditor

Regular Expression:

Test your regular expression pattern here. There are two test areas, so you can test one regular expression against two examples at once, this useful when you want to have a pattern that matches one example and not the other. The results from both tests will be shown below.

Test Area 1:

Test Area 2:

Results:

Results:

For more information on regular expressions see [this page](#).

SSW Code Auditor Report

File Edit View Favorites Tools Help

Address: C:\Documents and Settings\RyanTee\My Documents\My Code Auditor Data\Reports\Bad NorthwindWindowsCS - Standard Report.xml


SSW Code Auditor - Job Report

Tips:

- You can use the checkboxes to tick files that you have fixed. Note these checkboxes are not saved, so if you close and re-open the report they will all be unchecked again.
- Want to find the line easily in VS.NET, then turn on Line Numbering.
- Want to double click to find the line in VS.NET, then use the Output Window.
- Want to permanently skip a line of code on a file, then add a Code Auditor Page Directive before the line to skip. Eg.: **//SSW Code Auditor - Ignore next line** (C#) or **'SSW Code Auditor - Ignore next line** (VB.NET)
- Want to permanently skip a file, then add a Code Auditor Page Directive on the top of file. Eg.: **//SSW Code Auditor - Ignore this page** (C#) or **'SSW Code Auditor - Ignore this page** (VB.NET)

Job Name: NorthwindWindowsCS

Summary

Rules Passed:	52
Rules Failed:	10 
Warnings:	10
Replaced Warnings:	0
Rules:	62
Rules Run:	62
Rules Disabled:	0

No Good - your code is not healthy

Assembly Name: SSWCodeAuditor.exe
Assembly Version: 10.52
Started: 8/29/2005 5:24:35 PM
Ended: 8/29/2005 5:24:45 PM
Time Taken: 00:00:09 (hh:mm:ss)
Files Processed: 43 of 43 (100%)

☒ **Paths Scanned:**

☒ C:\Program Files\SSW Code Auditor\Samples\NorthwindWindowsCS\ (10)

View by:

- ☐ 1 C# UI- Button, CheckBox, GroupBox and RadioButton must have System FlatStyle (Tip: Fix in designer). (1)
C:\Program Files\SSW Code Auditor\Samples\NorthwindWindowsCS\RegexEditorForm.cs Line(s): 171
- ☐ 2 C# Code- Exception variables should be called ex (Tip: Fix in code.) (1)
C:\Program Files\SSW Code Auditor\Samples\NorthwindWindowsCS\RegexEditorForm.cs Line(s): 374
- ☐ 3 C#/VB.NET Code- XP Theme - Should not use Application.EnableVisualStyles(Tip: Comment out in code) (1)
C:\Program Files\SSW Code Auditor\Samples\NorthwindWindowsCS\Autoexec.cs Line(s): 35

My Computer

Sınır Denetleyicisi (Bound Checker)

The screenshot displays the Microsoft Visual Studio environment with the Memory Exerciser tool running. The main window shows the DevPartner Error Detection window, which lists various modules loaded and a warning about calling SuspendThread() while executing under Error Detection. The Solution Explorer on the right shows the project structure, including the Memory Exerciser Snapshots and the active session. The DevPartner Activity Monitor window in the bottom right corner shows a graph of memory usage over time, with a peak around 2:14:32 PM. The Memory Exerciser control panel in the bottom left corner shows the current allocation status, including the allocated memory (2048 MB) and the released memory (1024 MB). The System Stats window at the bottom shows the private bytes (1476.805 MB) and the working set - private (1277.074 MB).

MemoryExerciser (Running) - Microsoft Visual Studio

Quick Launch (Ctrl+Q)

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST DEVPARTNER ANALYZE WINDOW HELP

Process: [3020] MemoryExerciser.exe Suspend Thread: Stack Frame:

DevPartner Error De...xerciser.exe).DPbcl

18 Module Load mscf.dll
19 Module Load imm32.dll
20 Module Load dbghelp.dll
21 Module Load apphelp.dll
22 Module Load mscoreei.dll
23 Module Load shlwapi.dll
24 Module Load MSVCRT110_CLR0400.dll
25 Module Load clr.dll

Thread 2 Begins
Thread 3 Begins
Thread 4 Begins
Thread 0 Resumes

30 Module Load mscorlib.ni.dll
31 Calling SuspendThread() while executing under Error Detection may lead to spurious deadlocks.
32 Module Load ole32.dll
33 Module Load CRYPTBASE.dll
34 Module Load uxtheme.dll
35 Module Load System.ni.dll

Calling SuspendThread() while executing under Error Detection may lead to spurious deadlocks.

Current Call Stack - Thread 0 [0x0270]

Function	File	Line / Offset
clr.dll	clr.dll	0x00619F56
clr.dll	clr.dll	0x0063C469
clr.dll	clr.dll	0x00084EA0
clr.dll	clr.dll	0x0008358B
clr.dll	clr.dll	0x0008397E
clr.dll	clr.dll	0x000BE085
clr.dll	clr.dll	0x000BE00F
clr.dll	clr.dll	0x0018538C
clr.dll	clr.dll	0x00182720
clr.dll	clr.dll	0x001821A0
mscorlib.dll	mscorlib.dll	0x00000000
kernel32.dll	kernel32.dll	0x00000000

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'MemoryExerciser' (1 project)

- DevPartner Studio
 - MemoryExerciserSnap.dpcov
 - MemoryExerciserSnap1.dpcov
 - MemoryExerciserSnap.dpmrg
 - MemoryExerciser.dpcov
 - DevPartner Error Detection - Active Session (MemoryExerciser)
 - MemoryExerciser.dpmrg

DevPartner Activity Monitor

70000
60000
50000
40000
30000
20000
10000
0

2:13:31 PM 2:13:51 PM 2:14:11 PM 2:14:32 PM 2:14:52 PM

C:\opt\examples\QA\QA\Solutions\VS2012\Dot Net 4.0 Solutions\Memc

Options... Close

Memory Exerciser

Allocate

2048 MB

Start Allocation

Allocated Memory

Release 1024 MB

System Stats

Private Bytes: 1476.805 MB Working Set - Private: 1277.074 MB

Immediate Window Output Solution Explorer Team Explorer

Sınır Denetleyicisi (Bound Checker)

The screenshot displays the BoundsChecker 7 application window. The title bar reads "BoundsChecker 7 - [BoundsChecker - Active Session (C:\Program Files\Compuware\BoundsChecker\Ex...)". The menu bar includes File, Edit, Program, Window, and Help. Below the menu is a toolbar with icons for file operations and execution. The main interface is divided into several panes:

- Errors List:** A table showing detected errors.
- Details:** A text area providing details about the selected error.
- Call Stack:** A table showing the current call stack for Thread 0.
- Code Editor:** A text area showing the source code with the error location highlighted.
- Navigation:** A status bar at the bottom showing the current line number and event logging status.

Errors List:

Type	Quantity	Location
Bad Pointer Use	10	
COM Interface Method Failure	4	
Dangling Pointer	2	
Dangling pointer	2	Pointer_ExprUsesDanglPtr - [PTRERR.CPP, line 106 (main.bug)]
Dangling pointer	1	Pointer_ExprUsesDanglPtr - [PTRERR.CPP, line 106 (main.bug)]
Dangling pointer	1	Pointer_ExprUsesDanglPtr - [PTRERR.CPP, line 106 (main.bug)]

Details:

Dangling Pointer: Pointer 0x01AAAE20, allocated by malloc, has already been freed.

Current Call Stack - Thread 0 [0x0A04]

Function	File	Line / Offset
Pointer_ExprU...	PTRERR.C...	106
DoAllErrors	BugUtility.cpp	1547
DoAllErrors	main.cpp	101
OnAllDo	BugBench7...	713

Code Editor:

```
c:\program files\compuware\boundschecker\examples\bugbench7\main\pterrr.cpp

char *a = (char *)malloc( 10 ) ;
char b[ 10 ] ;

free ( a ) ;
if ( a > b )
    a = b ;
}
```

Status Bar: Line Number: 102 | Event Logging: ON | Program Events: 240

Test Verisi Üreteci

- ▶ <https://www.generatedata.com/>
 - ▶ Javascript
 - ▶ Perl
 - ▶ Php
 - ▶ Ruby
 - ▶ C#
 - ▶ HTML
 - ▶ SQL

Java için Test Verisi Üreteci

- ▶ Java Faker
 - ▶ <https://github.com/ghacupha/java-faker>

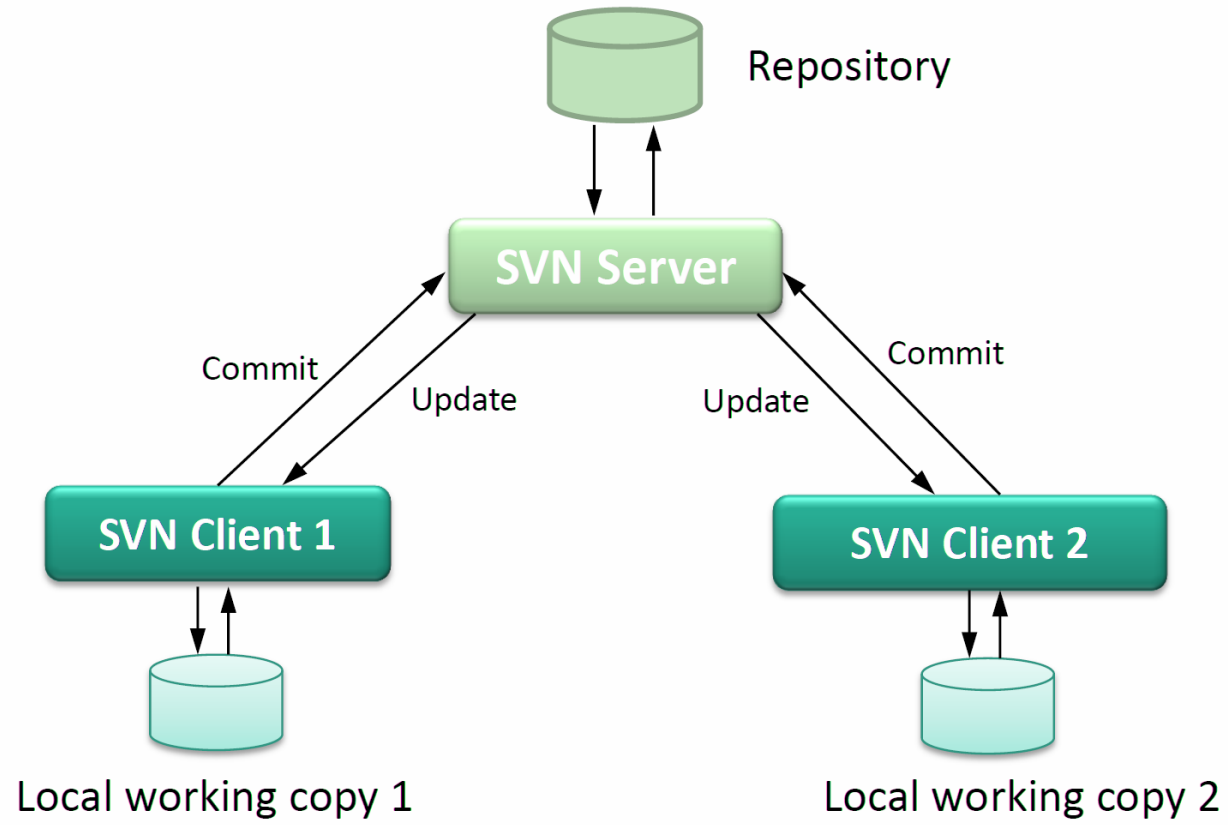


Test Harness

- ▶ JUnit
 - ▶ Java dilinde birim testlerin yazılması ve çalıştırılmasını sağlar.
- ▶ NUnit
 - ▶ .Net ortamında birim testlerin yazılması ve çalıştırılmasını sağlar.

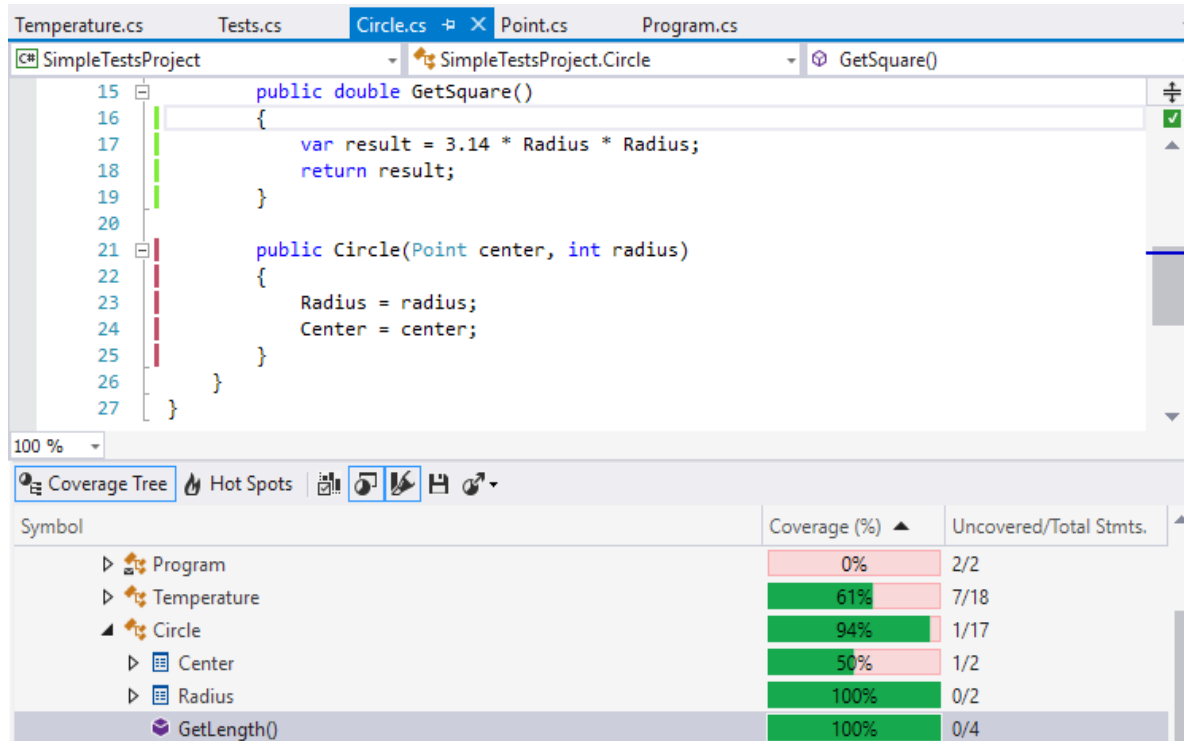


Versiyon Kontrol Edici



Bir Sınıf için Tüm Testler Yazıldı mı?

- Nasıl emin olabiliriz?
- Bu işi yapan bazı araçlar bulunmakta



The screenshot shows an IDE with a code editor and a coverage tool. The code editor displays the `Circle` class with two methods: `GetSquare()` and `Circle(Point center, int radius)`. The coverage tool at the bottom shows a tree view of the project and a table of coverage data.

Symbol	Coverage (%)	Uncovered/Total Stmt.
Program	0%	2/2
Temperature	61%	7/18
Circle	94%	1/17
Center	50%	1/2
Radius	100%	0/2
GetLength()	100%	0/4

DotCover

<https://www.jetbrains.com/dotcover/>

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.