BSM462
Yazılım Testi

Hafta - 1

Temel Kavramlar ve Ön Hazırlıklar

Dr. Öğr. Üyesi M. Fatih ADAK

fatihadak@sakarya.edu.tr

Değerlendirme Sistemi

- 1 Vize %15
- 1 Sunum %24
  - Sınıfta Sunum (Konu verilecek)
- 1 Rapor %12
  - Sunum konusu ile ilgili detaylı rapor (Sadece sunum yapan verebilecek)
- 1 Ödev %9
  - C++ ya da Java dilinde yazılım testi gerçekleştirimi (Konu verilecek)
- 1 Final %40

Haftalık Konu Akışı

Hafta	Konular
1	Temel Kavramlar ve Ön Hazırlık
2	Program Testi Teorisi
3	Tasarım ve Test Edilebilirlik
4	Test Araçlarının Tasarımı
5	Birim Testi
6	Java'da Yazılım Testinin Uygulanması - 1
7	Java'da Yazılım Testinin Uygulanması - 2
8	Google C++ Test Framework - 1
9	Google C++ Test Framework - 2
10	Raporlama Türleri
11	Hata Durum Doğrulaması ve Hata Kodları
12	Kabul Testleri
13	Test Ekibi Organizasyonu
14	Yazılım Testinin Kaliteye Etkileri

Programlama Araçları

- Java NETBEANS
  - Junit
- Visual Studio C++
  - Google C++ Test Framework

İçerik

- Yazılım testi ve rolü
- Yazılım kalitesi ve rolü
- Test mühendisinin rolü ve özellikleri
- Doğrulama ve Onaylama
- Yazılım testi terimleri
- Yazılım test seviyeleri
- Yazılımın paydaşları
- Yazılım testinin hedefleri
- Test durumu ve beklenen çıktılar
- Beyaz ve Siyah kutu testleri
- Testin planlanması ve tasarımı
- Test odaklı geliştirme
- Testi izleme ve ölçme
- Test takımı organizasyonu

Yazılım Testi

- Bütün yazılımsal problemler böcek (bug) olarak ifade edilebilir.
- Bir bug genelde yazılımın istenilen şeyi yapmadığı ya da istenmeyen şeyleri yaptığı durumlarda ortaya çıkar.
- İstenmeyen bu durumların yazılımın geliştirilme sürecinde yakalanabilmesi oldukça önemlidir.
- Yazılım testinin amacı bakım maliyetlerini düşürmek ve ürünün piyasaya sürüldükten sonra ortaya çıkacak hataları en aza indirmek için yazılımın geliştirilme ve piyasaya sürülmeden önceki süreçte bug tespiti yapmak, test etmek ve bunun önüne geçmektir.

Yazılım Testi devam

- Yazılım testi içerik tabanlı ve risk odaklı gitmelidir.
- Metodolojik ve disiplinli bir yaklaşım gerektiren yazılım testi, arama, sorun giderme, yaratıcı ve ikna edici gibi özellikleri barındıran kişiler tarafından gerçekleştirilmelidir.

Yazılım Kalitesi

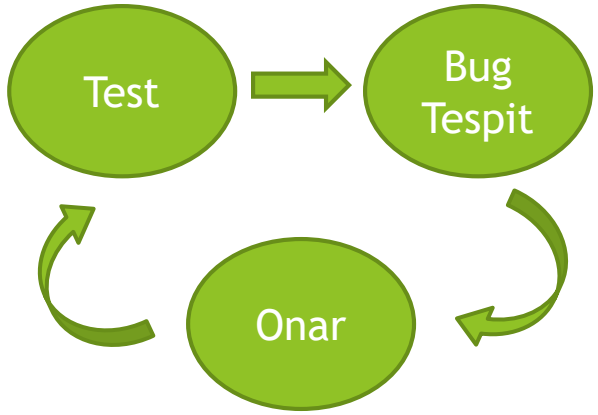
- Kalite kısaca mükemmellik derecesi olarak ifade edilebilir.
- Yazılım kalitesinin amacı geliştirilen ya da geliştirilecek yazılımı mükemmele götürebilmektir.
- Kalitesi düşük yazılım kusur içeren yazılım anlamına gelmektedir.
- Kalitenin ölçümü için bir standart oluşturulduğu söylenemese de birçok kalite ölçüm metriği tanımlanmış durumdadır.

Bug ve Kusur

- Bug ve kusur birçok alanda aynı kareye dahil edilebilir.
- Fakat birebir aynı ifadeler değildir.
- Yazılım testinin amacı bug bulma iken, Yazılım kalitesinin amacı kusurları tespit etmektir.
- Kullanıcının yazılımı rahatça kullanamaması yine bir kusur sayılmakta ve yazılım kalitesi alanına girmektedir.

Testin Rolü

- Yazılım ürünlerinin kalitesinin değerlendirilmesinde test büyük rol oynar.
- Yazılım geliştirme sürecinde genel bir test çevrimi kullanılır.
- Yazılım sürecinde ve ürün oluştuktan sonra ki testler farklı değerlendirilebilir.

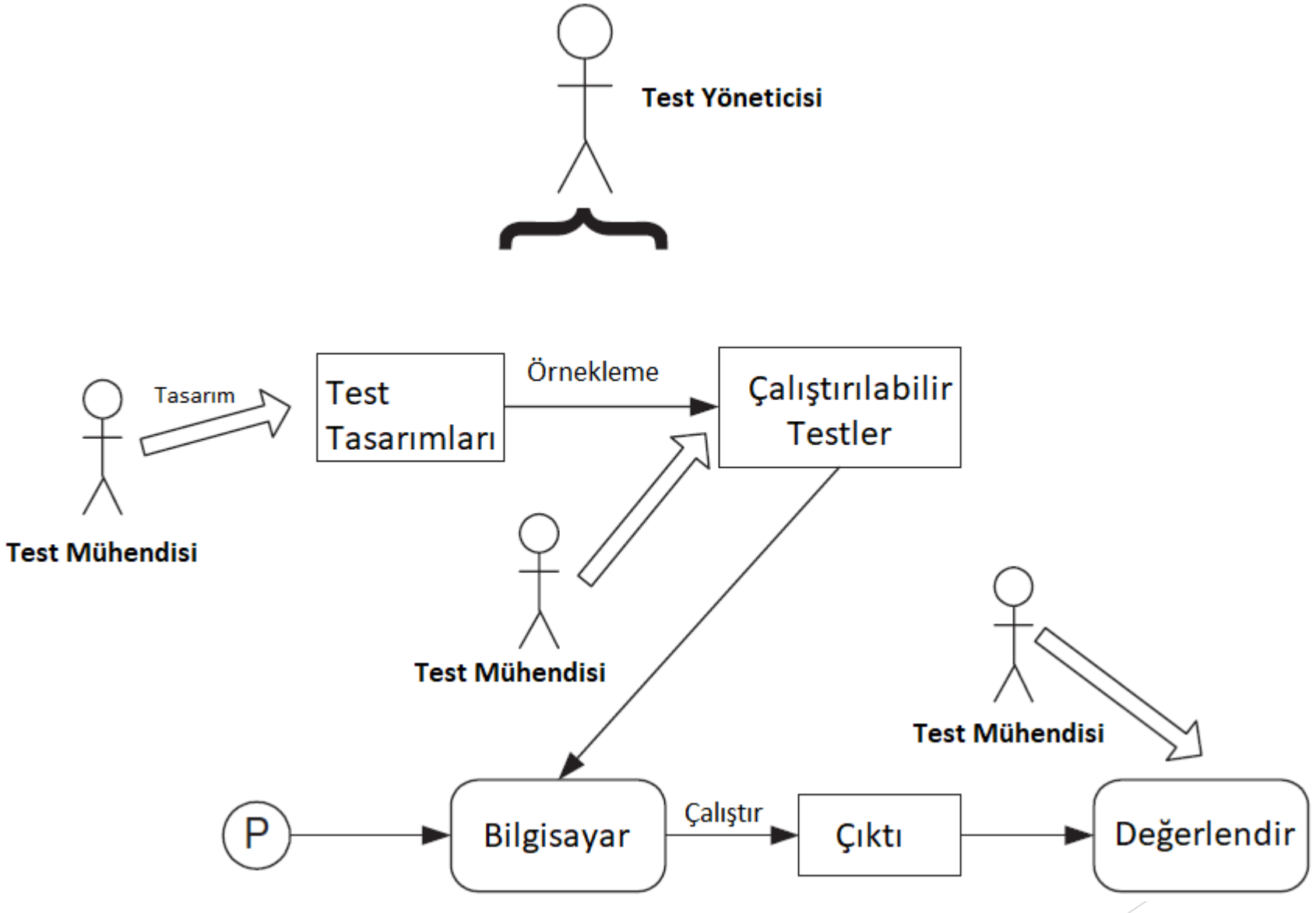


Kalitenin Rolü

- İki ana kategoride incelenebilir.
  - Statik Analiz
    - Yazılım modülleri
    - Tasarım dokümanları
    - Yazılım kodu, algoritmanın analizi

    inceler
  - Dinamik Analiz
    - Olası program hatalarını ortaya çıkarmak için program yürütme süreçlerini içerir.
    - Programın performans ve davranışsal özellikleri incelenir.
    - Program çeşitli girdiler ile test edilir.

Test Mühendisinin Rolü



İyi Bir Test Mühendisinin Özellikleri

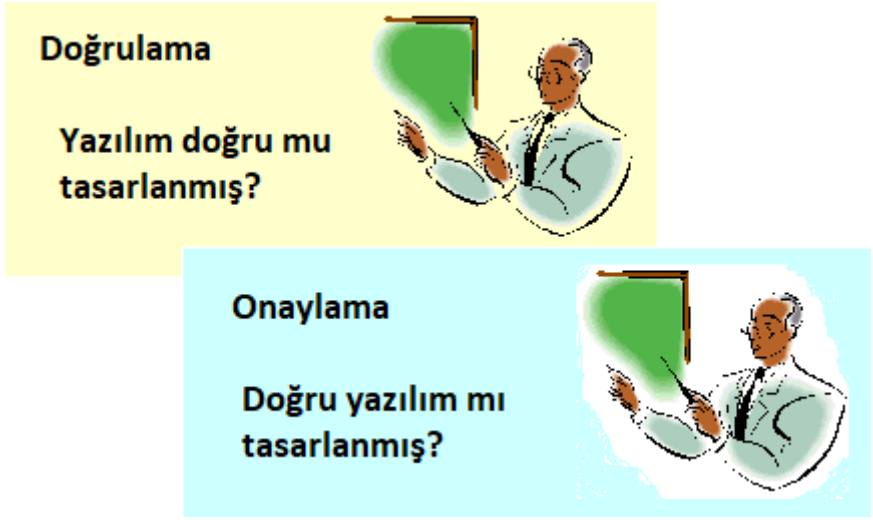
- Teknolojiyi bilir
  - Uygulamanın geliştirildiği teknolojiye hakim
  - Daha iyi tasarım daha güçlü test uygulamaları
- Mükemmeliyetçi ve gerçekçi
  - Hangi problemlerin onarıma ihtiyacının olup hangisinin olmadığını bilir.
- Takıntılı, diplomatik ve ikna edici
  - Hataya sebep olan geliştiriciyi ikna edip hataları giderir.
- Kaşif ruhlu
  - Bilinmeyen durumlara girişmesini ve risk almasını sever.
- Gideren
  - Problemin neden kaynaklandığını bilir böylelikle geliştirici ile daha kolay iletişim kurar

İyi Bir Test Mühendisinin Özellikleri devam

- İnsanların yetenek ve dirençlerine karşı durabilen
  - Test mühendisleri programcılardan çok fazla direnç ile karşılaşabilirler.
  - Bu noktada test mühendisleri usta ve inatçı olmalıdırlar.
- Düzenli
  - Ellerinde kontrol listesi bulunur.
  - Kendini desteklemek için gerçekleri ve rakamları kullanır.
- Objektif ve doğru
  - Doğru olmayan bilginin rapor edilmesi güvenirliğini düşürür.
- Kusurlar değerlidir
  - İyi test ediciler kusurlardan bir şeyler öğrenirler.
  - Erken tespit edilen bir kusur, çok geç tespit edilmiş bir kusurdan çok daha az maliyetlidir.

Doğrulama ve Onaylama (Verification and Validation)

- Birbirlerine yakın gibi görünen ifadeler yazılım testi ile ilgili iki önemli adımdır.



Doğrulama ve Onaylama (Verification and Validation)

- Doğrulama
  - Gereksinimlerin tanımlanması
  - Kullanım kılavuzu
  - Tasarımın analizi
- Onaylama
  - Müşterinin beklentilerini karşılıyor mu?
  - Yazılım geliştirme sürecinin sonuna doğru yapılması geliştirme maliyeti açısından tehlikelidir.

Yazılım Test Çıktısında Karıştırılabilen Terimler

- Failure (Başarısızlık)
  - Yazılımın özelliklerinde belirtilmeyen bir tavır sergilemesi
- Error (Hata)
  - Yazılımın verdiği bir durumdur.
- Fault (Arıza)
  - Yazılımdaki hataya (error) sebep olan ifadedir.
  - Bir olay aktive etmediği için uzun bir süre arıza (fault) tespit edilemeyebilir.
- Defect (Kusur)
  - Yazılım alanında Arıza ile Kusur aynı terimdir. Birbirleri yerine kullanılabilir.

Yazılım Test Çıktısında Karıştırılabilen Terimler devam

- Örnek
  - Ondalık sayılarda bölme işlemi yapan bir yazılımın geliştirilmesi isteniyor.

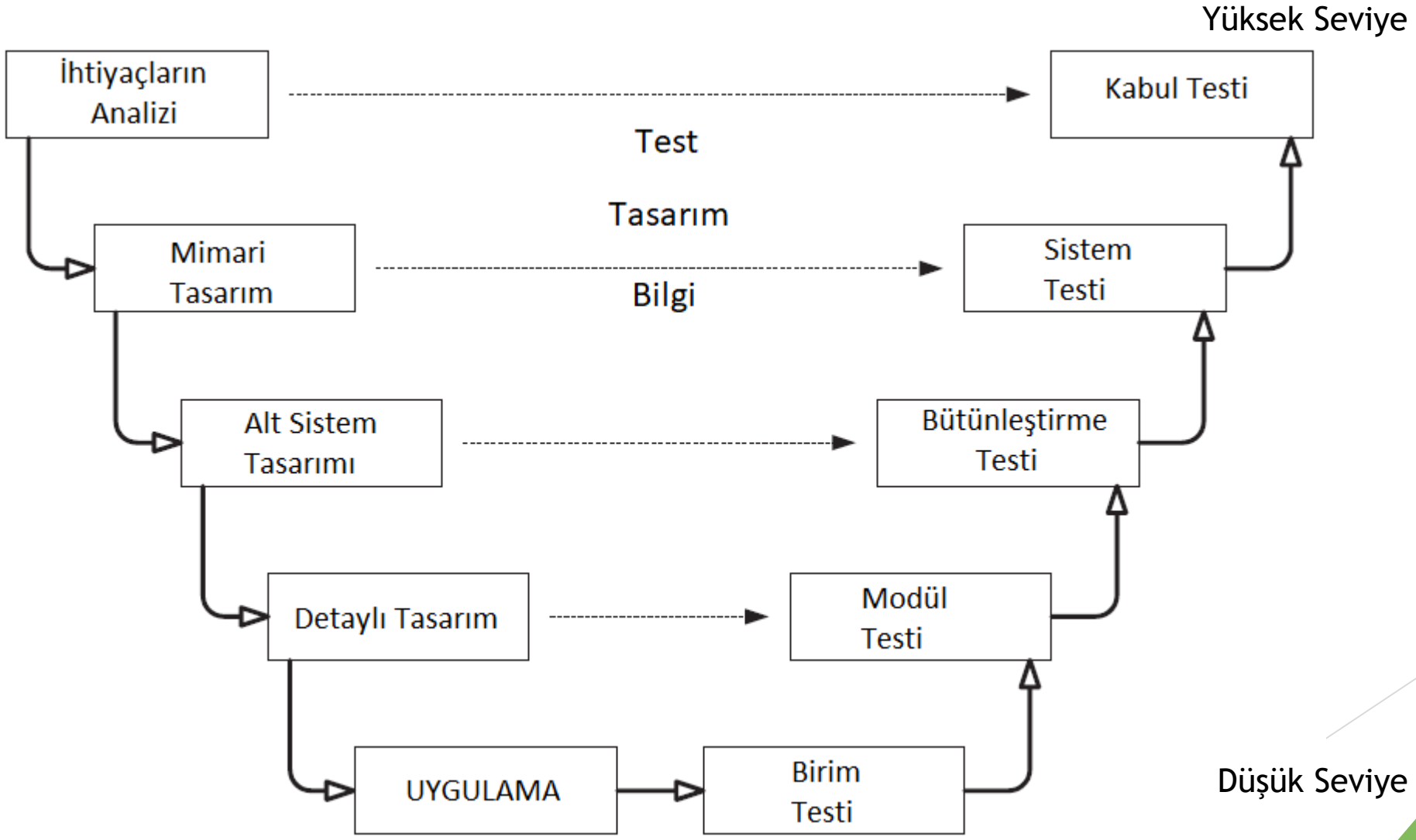
```
#include <iostream>
usign namespace std;

int main(){
    int x,y;
    cout<<"x:";
    cin>>x;
    cout<<"y:";
    cin>>y;
    cout<<"x/y="<<x/y;
    return 0;
}
```

- Failure (Başarısızlık) : Yazılımın sadece tam sayılarda bölme yapması
- Error (Hata) : Kullanıcı ikinci değere sıfır girdiğinde sıfıra bölünme hatası verip program sonlanacaktır.
- Fault (Arıza) veya Kusur : Sıfıra bölünme hatasının programın verebiliyor olması

Yazılım Etkinliğine Dayalı Test Seviyeleri

- Kabul Testleri
  - Gereksinimlere göre yazılımı değerlendirmek
- Sistem Testleri
  - Yazılımı mimari tasarım açısından değerlendirmek
- Bütünleştirme (Integration) Testleri
  - Yazılımı alt sistem tasarımına göre değerlendirmek
- Modül Testleri
  - Yazılımı ayrıntılı tasarıma göre değerlendirmek
- Birim Testleri
  - Yazılımı uygulamaya göre değerlendirmek

Yazılım Etkinliğine Dayalı Test Seviyeleri



Yazılımın Paydaşları

Yazılım Testinin Hedefleri

- Çalıştığını göstermek
- Çalışmadığını göstermek
  - Modülün ya da modüllerin çalışmadığını göstermek için denemeler yapmak
- Olası arıza riskini azaltmak
  - Ne kadar fazla test o kadar düşük hata oranı
- Bakım maliyetlerini düşürmek

Bir Test Durumu (Test Case)

- En temel tanım
  - <girdi, beklenen çıktı>

- Bazı durumlarda çıktı o anki sistem ve girdiye bağlı olabilir.
- Kişi 300 TL çekmek istiyor beklenen çıktı 300 TL
- Bakiye sorgulamada beklenen çıktı 400 TL'dir.

TB_1: $< 0, 0 >$,
TB_2: $< 25, 5 >$,
TB_3: $< 40, 6.3245553 >$,
TB_4: $< 100.5, 10.024968 >$.

TS_1: < **Bakiye Sorgula, 700 TL** >, < **Kart İade, "Miktar"** >,
< **300 TL, "300 TL"** >, < **Bakiye Sorgula, 400 TL** > .

Beklenen Çıktı

- Kompleks bir nitelik olan çıktı aşağıdakilerden bir veya bir kaçını içerebilir.
  - Program tarafından üretilen değerler
    - Tam sayı, yazı, ses, görüntü
    - Uzak kaynaklar için mesaj veya gözlemleme
  - Durum değişiklikleri
    - Programın durum değişikliği
    - Veritabanının durum değişikliği
  - Birlikte düşünülmesi gereken bir dizi değer
- Beklenen çıktı program çalıştırılmadan önce test edici tarafından hesaplanmalıdır.

Beklenen Çıktı devam

- Bazı durumlarda beklenen çıktının hesaplanması çok zor olabilir. Bu durumda aşağıdaki adımlar uygulanır.
  - Seçilen girdi ile programı çalıştır.
  - Çalışma sonucunda programın üretmiş olduğu çıktıyı gözlemle
  - Üretilmiş çıktının beklenen çıktı olduğunu doğrula
  - Doğrulanmış üretilen çıktıyı test durumlarında beklenen çıktı olarak kullan

Eksiksiz (Complete, Exhaustive) Test Kavramı

- Bir programın tamamıyla test edilmesi pek mümkün değildir.
- Tam veya Ayrıntılı (Complete, Exhaustive) testin anlamı
  - Test sürecinin sonucunda ortaya çıkarılmamış hata kalmamıştır.
  - Bütün problemler tam testin sonucundan bilinmelidir.
- Aşağıdaki nedenlerden dolayı böyle bir durum pek mümkün değildir.
  - Tam testi gerçekleştirebilmek için gerekli girdi hacmi çok büyüktür.
  - Bazı zamanlarda geçerli olan girdi bazı zamanlarda geçersiz olabilir.
  - Tasarım başlıkları çok kompleks olacaktır.
    - Örneğin global ya da statik değişkenin kullanılması
  - Sistemin mümkün olan bütün çalışma ortamlarını oluşturmak çok zor bir durumdur.

Test Yapma Faaliyetleri

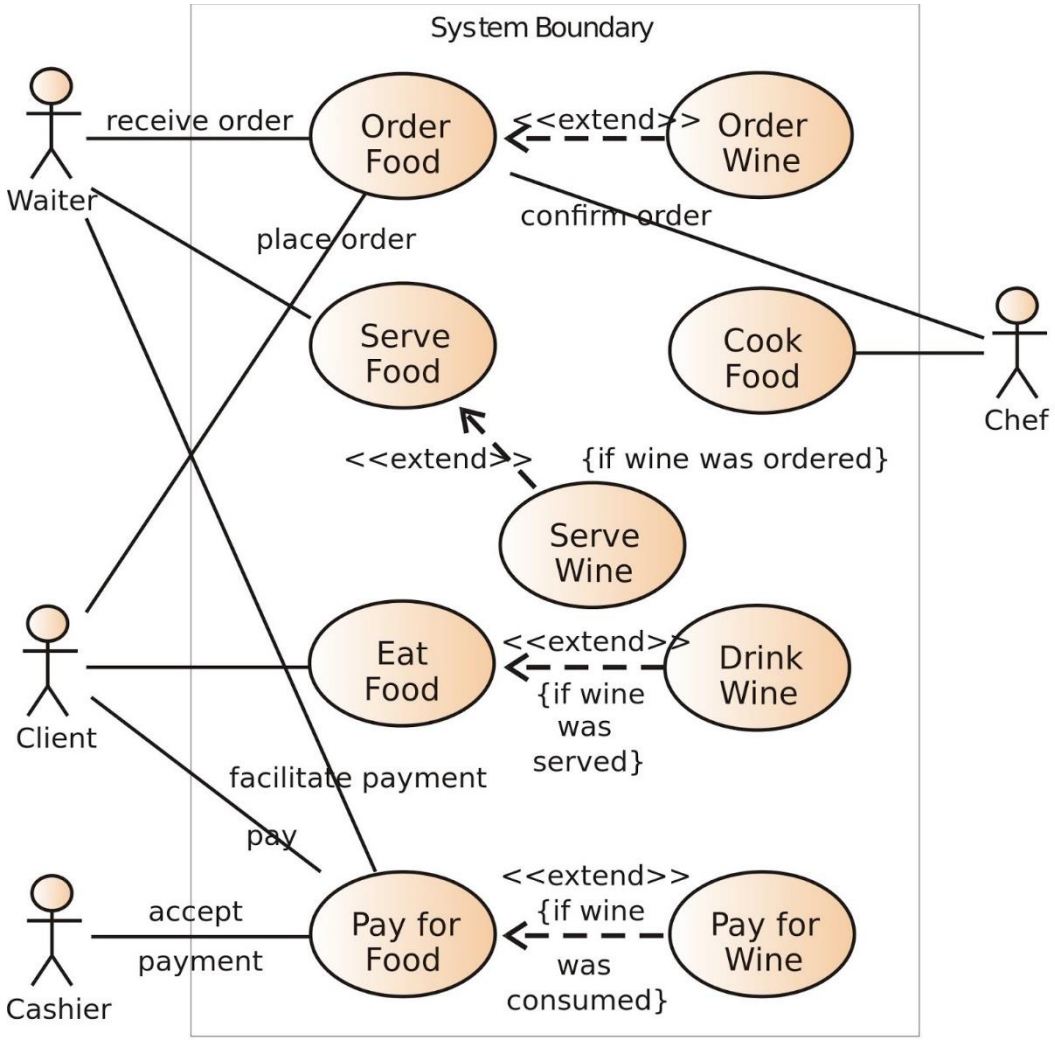
- Düşük maliyetli ve etkili test yazma için test tasarımcılarının analiz etmesi gereken bilgi kaynakları
  - Gereksinimler ve fonksiyonel özellikler
  - Kaynak kod
  - Girdi ve çıktı parametreleri
  - İşlevsel profil
  - Fault (Arıza) Modeli

Test Yapma Faaliyetleri

- Yazılım geliştirme kullanıcı ihtiyaçlarını belirleme ile başlar.
- Bir test mühendisi de programın bütün ihtiyaçlarını bilmesi yazılımın geliştirme döngüsünü tamamlamak için önemlidir.
- İhtiyaçlar
  - Düz metin ile yazılmış ifadeler
  - Denklemler
  - Şekiller
  - Akış diyagramları

  Şeklinde olabilir.

İhtiyaç Belirlemede Diyagramlar



Kaynak Kod

- Her ne kadar ihtiyaçların belirlenmesi sistemin istenen davranışını belirtse de
- Sistemin gerçek davranışını kaynak kod tanımlar.

Örnek

Büyük bir öğrenci verisi sıralanması gerekiyor.

Detaylı Tasarım -> Öğrenci dizisini sırala

Birçok sıralama algoritması bulunmakta ve

Bunlar farklı karakteristikler içermekte

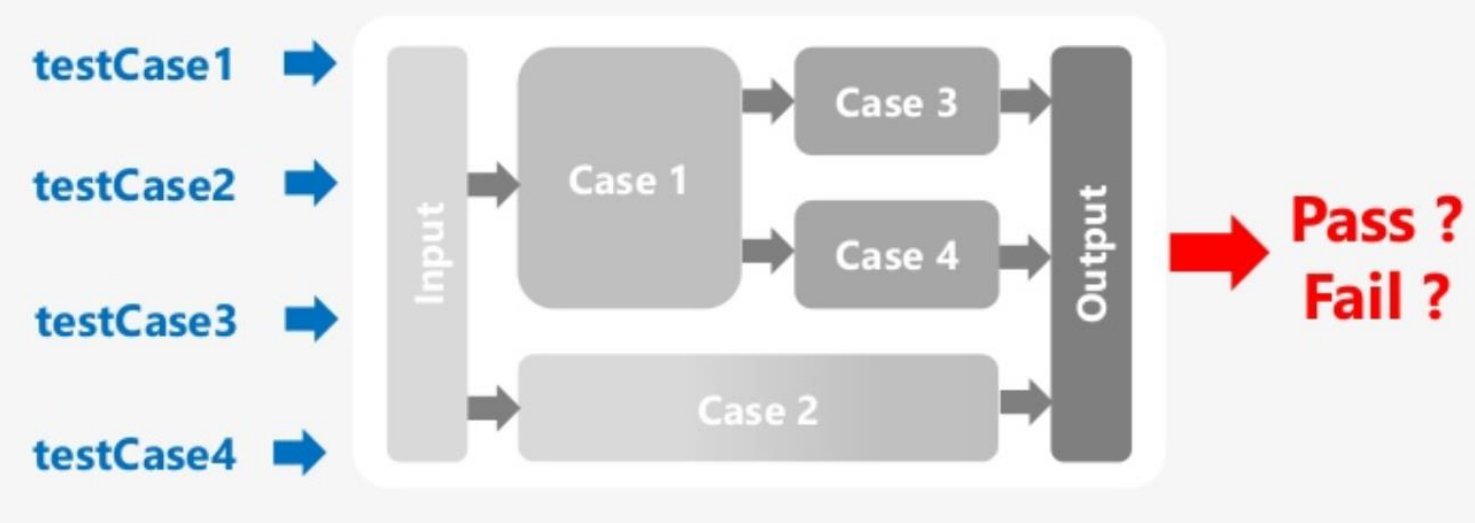
- İterasyon kullanan
- Özyineleme
- Başka dizi kullanan

Beyaz Kutu (White Box) Testi

- Yapısal test tekniklerini içerir
- Kaynak koddaki kontrol ve veri akışını inceler
- Kontrol Akışı
  - Kod satırında bir bir ilerleyen ifadeler
  - Fonksiyon çağrımları
  - Mesaj geçirmeler
  - Kesmeler (Interrupts)
- Veri Akışı
  - Sabit veya değişken içerisindeki değerlerin başka değişken veya sabite akışı

Beyaz Kutu (White Box) Testi

- Yapısal test tekniklerini programın tekil birimlerine uygulanır.
- Bireysel programcılar kaynak kodun detayını bildikleri için beyaz kutu testi uygularlar.



Siyah Kutu (Black Box) Testi

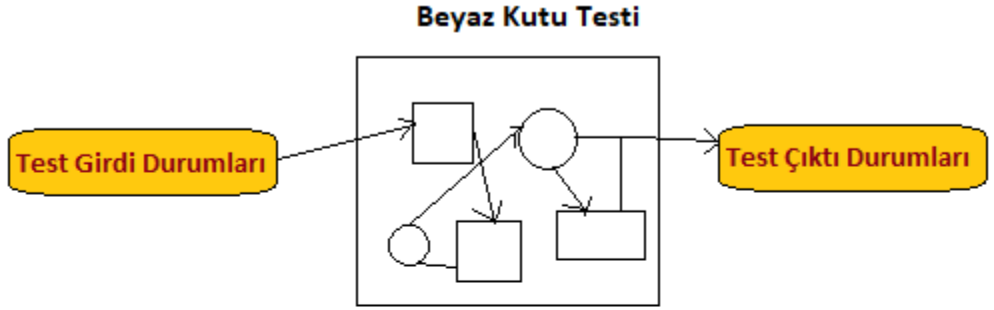
- Fonksiyonel test tekniklerini içerir
- Bir fonksiyonel testte programın detayına erişilemez.
- Programa kara kutu gibi davranılır.
- Parametre verip döndürdüğü değer incelenir.
- Bu test şeklinde mühendis sadece programın dışından erişilebilen kısmı ile ilgilenir.

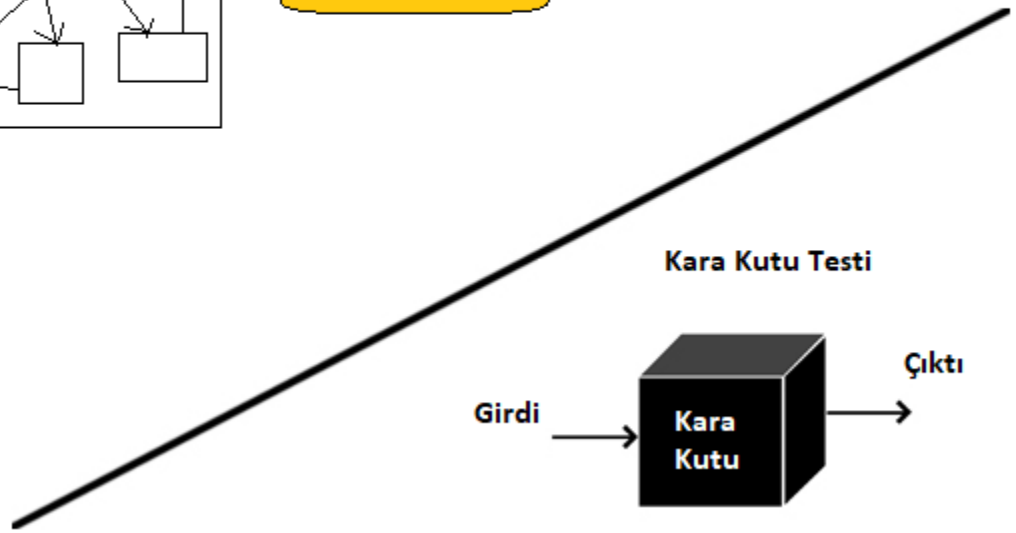
Siyah Kutu (Black Box) Testi

- Fonksiyonel test teknikleri programın tekil birimlerine uygulanabileceği gibi bütün bir sisteme de uygulanabilir.
- Siyah kutu testi genelde yazılım kalite grubu tarafından sistemin dış arayüzünde gerçekleştirilir.

Beyaz ve Siyah Kutu Testi

- Yapısal ve fonksiyonel testler yazılımın yaşam süreci boyunca farklı gruplar tarafından farklı zamanlarda kullanılır.
- Beyaz veya Siyah kutu testleri programcılara veya test mühendislerine verilmiş alternatif seçenekler değildir.





Testin Planlanması ve Tasarımı

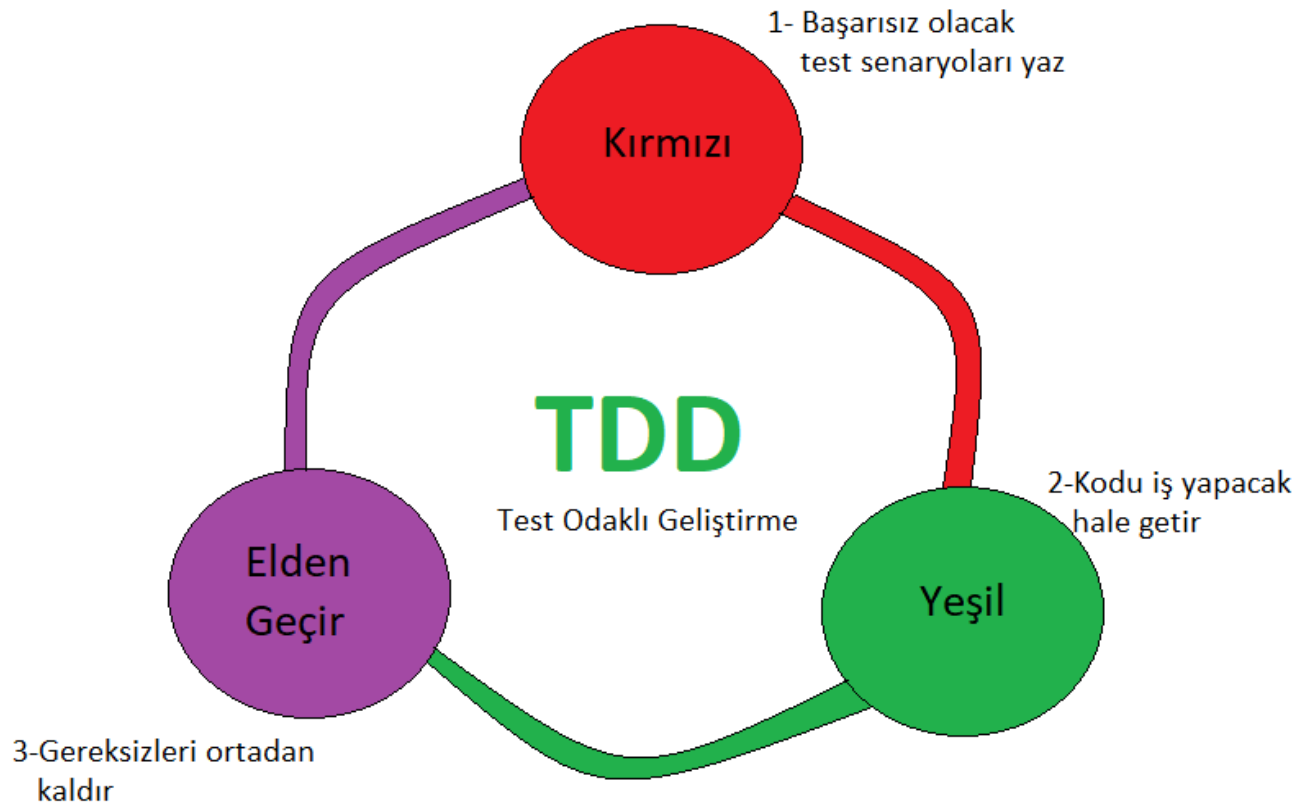
- Test planlama ve tasarımın amacı testin çalıştırılması için hazır hale getirmektir.
- Bir test planı
  - Bir çerçeve (framework) sağlar
  - Kapsamı gösterir
  - İhtiyaç duyulan kaynakların detayını belirtir
  - İhtiyaç duyulan efor
  - Bütçe ve aktivitelerin takvimi

Testin Planlanması ve Tasarımı

- Test Tasarımı
  - Yazılım testinin kritik safhasıdır.
  - Sistem gereksinimleri incelenir.
  - Sistem özellikleri çıkarılır.
  - Test senaryolarının amaçları belirlenir.
  - Davranışlar tanımlanır.
- Her test amacı için bir veya daha fazla test örneği tasarlanır.

Test-Driven Development (TDD)
Test Odaklı Geliştirme

- Yazılım geliştirmede test merkezli bir yaklaşım popüler olmaya başlamıştır.



Testi İzleme ve Ölçme

- İzleme ve ölçme mühendisliğin anahtar prensipleridir.
- Aynı prensipler yazılım testinde de uygulanır.
- Sistemin kalite seviyesini ortaya çıkarmak için bazı metriklerin izlenmesi önemlidir.
- Testin çalıştırılmasındaki metrikler
  - Testin çalıştırılmasını izlemek için metrikler

  Test senaryolarının çalıştırılma süreci ile ilgilenir.
  - Kusurların izlenmesi için metrikler

  Testin çalıştırılması sonucu bulunan kusurlar ile ilgilenir.

Testi İzleme ve Ölçme

- Metriklerin izlenmesi ve analiz edilmesi belli periyotlarla devam etmelidir.
- Test süreci devam ederken bir yönetim takımı metriklerin tanımlanması ve izlenmesini yönetmelidir.
- Test sürecinde iki önemli metrik
  - Test Senaryosunun Etkinliği

  Kusurun ortaya çıkarılma yeteneğini ölçer. Test tasarım sürecini geliştirmeye çalışır.

  - Test Başarısının Etkinliği

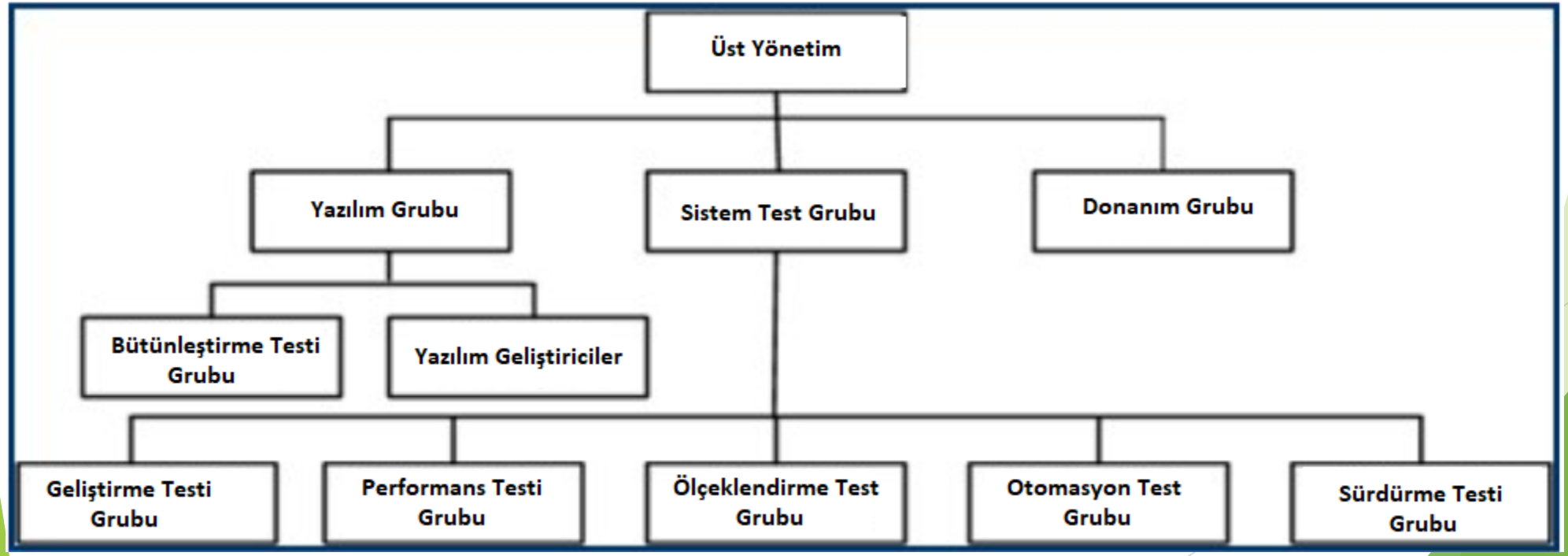
  Son kullanıcılar tarafından bulunan kusurların sayısı ile ilgilenir. Bu test mühendislerinin gözünden kaçmış olması anlamına gelir.

Test Takımının Organizasyonu ve Yönetimi

- Test, yazılımın yaşam döngüsüne dağıtılmış bir süreçtir.
- Bu süreçler
  - Birim testi - Programcılar
  - Bütünleştirme testi – Test Mühendisleri
  - Sistem testi – Yazılım geliştirme grubundan ayrı olmalıdır.
  - Kabul testi – Müşterilerden oluşabilecek özel bir grup
- Her bir test süreci için ayrı bir test grubunun bulunması her ne kadar kulağa hoş gelse de birim testlerinin programcılar tarafından gerçekleştirilmesi daha mantıklıdır.

Test Takımının Organizasyonu ve Yönetimi

- Diğer test süreçleri farklı gruplar tarafından yönetilebilir.
- Burada en önemli nokta bu test süreçleri birbirleri ile yakından ilgilidir.
- Bundan dolayı bu grupların yönetim ve organizasyonu gereklidir.



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.