

# TEST MÜHENDİSLİĞİNE GİRİŞ EĞİTİMİ

## Kaynaklar:

Takip ettiğim ve eğitimi hazırlarken bazı yerlerde çok kısa bilgi, bazı yerlerde grafik olarak yararlandığım kaynaklar aşağıdaki gibidir.

- <http://www.istqb.org/downloads/finish/16/15.html>
- <http://www.istqb.org/downloads/finish/16/16.html>
- <http://www.istqb.org/downloads/finish/20/14.html>
- <http://www.istqb.org/downloads/finish/41/87.html>
- <http://www.istqb.org/downloads/finish/6/88.html>
- [http://www.amazon.com/Testing-Computer-Software-2nd-Kaner/dp/0471358460/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1348123327&sr=1-1&keywords=0471358460](http://www.amazon.com/Testing-Computer-Software-2nd-Kaner/dp/0471358460/ref=sr_1_1?s=books&ie=UTF8&qid=1348123327&sr=1-1&keywords=0471358460)
- [http://www.amazon.com/Software-Testing-Real-World-Improving/dp/0201877562/ref=sr\\_1\\_1?s=books&ie=UTF8&qid=1348123373&sr=1-1&keywords=software+testing+in+real+world](http://www.amazon.com/Software-Testing-Real-World-Improving/dp/0201877562/ref=sr_1_1?s=books&ie=UTF8&qid=1348123373&sr=1-1&keywords=software+testing+in+real+world)
- <http://www.slideshare.net/onsoftwaretest/istqb-iseb-lecture-notes-presentation>
- <http://www.slideshare.net/yogindernath/istqb-iseb-foundation-exam-practice-2>
- <http://www.slideshare.net/yogindernath/istqb-iseb-foundation-exam-practice>
- <http://www.slideshare.net/yogindernath/istqb-iseb-foundation-exam-practice-4>
- Ivar Jacobson: <http://blog.ivarjacobson.com/ivarblog/>
- W. Edwards Deming: [http://www.youtube.com/watch?v=HBW1\\_GhRkTA](http://www.youtube.com/watch?v=HBW1_GhRkTA)
- James O. "Jim" Coplien: <http://users.rcn.com/jcoplien/>
- Mel Conway: <http://www.melconway.com/research/committees.html>

## 2. ISTQB Müfredatı

-  **Testin Temelleri**
-  **Geliştirme Boyunca Test**
-  **Statik Test Teknikleri**
-  **Test Tasarım Teknikleri**
-  **Test Yönetimi**
-  **Test Araçları**



1	2	3
4	5	6

## 1. Testin Temelleri



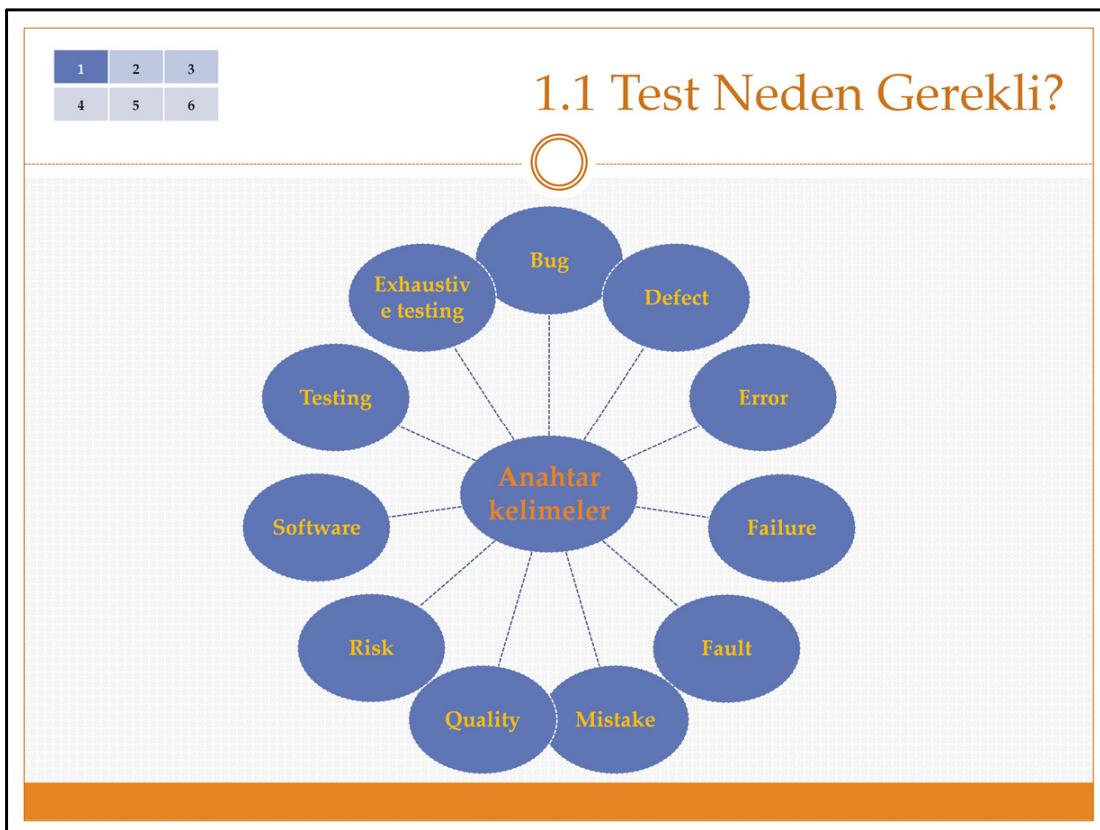
Test Neden Gerekli?

Test Nedir?

7 Test Prensibi

Temel Test Süreçleri

Test Psikolojisi



Türkçesi:

- **Bug, Defect, Error, Failure, Fault, Mistake:** Hata (Malesef herbir terim için kelime üretmemiştir), dilimizde de İngilizceleri kullanılmaktadır.
- **Quality:** Kalite
- **Risk:** Risk, hata çıkma olasılığı
- **Software:** Yazılım
- **Testing:** Yazılım testi
- **Exhaustive Testing:** Eksiksiz, detaylı test

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?



- Neler test edilir?

- Yazılımdan istenilenler yerinde ve yapılmış mı? (validation)
- Yazılım istenilen işlevleri yerine getiriyor mu? (verification)
- Yazılım işlevleri yaparken hata veriyor mu? (reliability)
- Yazılım istenilen hızda yapıyor mu? (performance)
- Yazılım istenilen kadar işlev yapabiliyor mu? (load)
- Yazılım istenilen işlevleri en çok ne kadar yapıyor? (stress)
- Yazılım istenilen kolay yapıyor mu? (usability)
- Yazılım istenilen işlevleri güvenli yapıyor mu? (security)
- Yazılım işlevleri her zaman yapabiliyor mu? (compatibility)

Yazılım bir hobi çalışması olmadığı ve vadedilen bir takım işleri; hızlı, güvenilir, kolay, arzu edilen her zaman yerine getirmesi planlarıyla yapıldığından üstte sıralanan özelliklere sahip olması gereklidir.

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?



- “Bug” nedir?

- **Error:**

- ✖ İnsan tarafından üretilen yanlış bir sonuç doğuran aktiviteler

- **Fault, Defect, Bug:**

- ✖ **Error**'den dolayı kodda ortaya çıkan yanlışlıklar.

- **Failure:**

- ✖ **Bug**'dan dolayı yazılımın verdiği hatalar.

- **Örnek**

- **Failure** bir hadise; **fault** ise **error** sonucunda yazılımda ortaya çıkan istenmedik bir durum

İngilizce'de hata denildiğinde bir çok terim kullanılırken Türkçe'de tek bir kelime ile bunu anlatmak zor hatta imkansızdır. Bu yüzden bu kavramların ingilizcelerini açıklamak daha kolay günün birinde her kelimenin Türkçe karşılığı olduğunda onları kullanmak iletişimim ve anlıyı kolaylaştırır.

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?

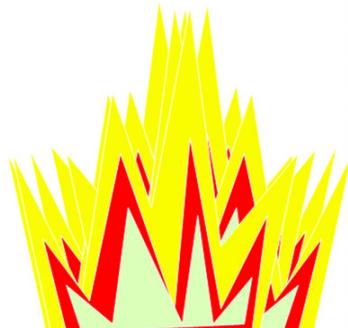


- **Error – Fault – Failure**

A person makes  
an error ...



... that creates a  
fault in the  
software ...

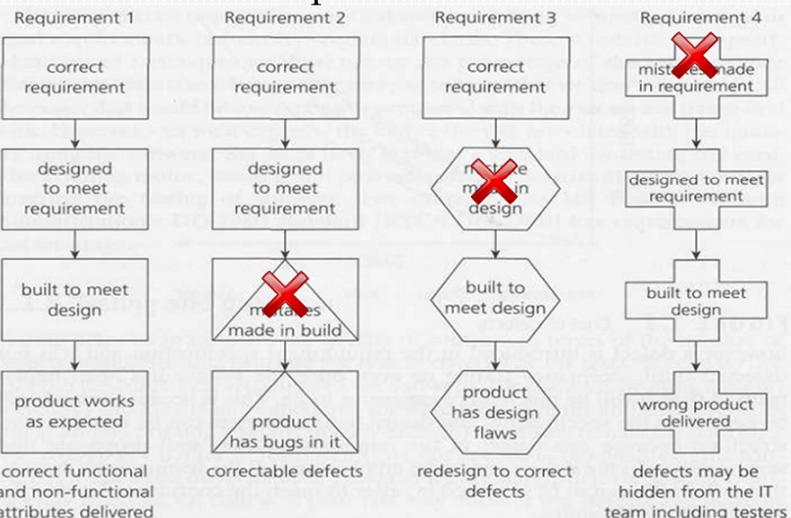


... that can cause  
a failure  
in operation

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?

- **Error ve Defect tipleri**



Hata SLDC göz önüne alındığında, yazılım hayat döngüsünün her yerinden yapılabilir. En büyük ve dönülmesi olmayan hatalar gereksinimlerin toplanması aşamasında yapılan hatalardır. Bu tip hataların düzeltilmesi için, tekrar saha çalışması yapılarak düzeltmeler yapılmalı ve daha sonraki süreçler başarılı işletilmeli. En az soruna neden olan hatalar ise yazılım aşamasında yapılan ve bulunan hatalardır. Test mühendisleri tarafından bulunan defectler ilgili yazılımcıya haber iletişime geçilerek çabucak düzeltilebilir.

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?

The diagram illustrates the exponential increase in the cost of finding and fixing defects over time. A graph plots 'COST' on the vertical axis against 'TIME' on the horizontal axis. A curve starts at the origin and rises steeply, labeled with the text 'cost of finding and fixing defects increases over time'. Below the graph, the timeline is divided into phases: Requirements, Design, Build, Test, and Live use.

cost of finding and fixing defects increases over time

COST

TIME

Requirements      Design      Build      Test      Live use

SDLC

Gereksinim

Tasarım

Geliştirme

Test

Yayınlama

- Hatanın Maliyeti

### Yazılım Geliştirme Hayat Döngüsü (SDLC)

Gereksinimlerin müşterilerden toplanarak analiz edilip daha sonra tasarım, geliştirme, test ve yayına gibi evrelerinden oluşan bir döngüdür. Bu döngü içerisinde hatanın bulunduğu yere göre yazılıma maliyeti farklılık göstermektedir. Son evrelerde yakalanan hatanın maliyeti, erken evrelerde yakalanan hatanın maliyetine zamanın karesiyle (exponential) orantılı olarak artmaktadır. Bu testin önemini artırmaktadır.

Kısaca **iyi test** Ciddi hataları erken bulmaktır! – Özay Civelek, *Testurk.com*.

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?



- Neden hatalar oluşur
  - Yazılım ürünü insanlar tarafından yazılan kodlardan oluşur
    - ✖ İnsanlar bazı şeyler bilebilir fakat herşeyi değil
    - ✖ İnsanlar yetenekli olabilir fakat kusursuz değildir
    - ✖ İnsanlar hataya meyillidir
  - Zaman kaygısı kaliteyi düşürür
    - ✖ Kontroller için zaman olmayabilir
    - ✖ Tamamlanmamış yapılar kalmış olabilir

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?



- Büyük yazılım hataları
  - Ekomik boyutu
    - Yazılım hatalarının Amerikan ekonomisine zararı yıllık yaklaşık olarak **60 Milyar Dolar** olduğu tahmin edilmekte.
  - Büyük projeler
    - Ariane 5: 7 Milyar Dolar
    - Marine space probe to Venus: 250 Milyon Dolar
    - Hartford Coliseum Collapse: 90 Milyon Dolar
  - **Büyük Canlı Kaybı Olabilirdi**
    - World War III... Almost (1983): Tüm insanlık tehlikede

### Ariane 5:

4 Haziran 1996'da Avrupa Hava Ajansının (ESA) ilk roket havalandıktan kısa bir süre sonra düştü. Düşme sebebinin daha sonradan yazılımdaki bir hatadan kaynaklandığı anlaşıldı. Hatanın 64 bit ondalıklı integerin 16 bit işaretli integere dönüştürülürken ondalık kısımda oluşan hatayı kontrol eden **exception handling** yapısının olmamasından kaynaklandığı anlaşıldı.

### Marine Space Probe to Venus:

28 Temmuz 1962'de Mariner1 isimli hava taşıtin roketinin yönü belirlenen yönden aniden çıkararak teklikeli yola girdiğinde başka bir roket Atlantik okyanusundan imha etti. Roketin yönündeki değişime neden olan hata, mühendislerin kağıt üzerine yazdıkları formülün koda aktarılırken yanlışlık yapmasından kaynaklanıyordu.

### Hartford Coliseum Collapse :

1978'de Hartford, Connecticut'da bulunan spor salonunun çatısında biriken sulu karın ağırlığını çatının çökmesine neden olmuş. Çökme sebebi projenin hesaplanmasında kullanılan CAD programının çatı yük hesabını yanlış yapmasından kaynaklanıyordu.

### World War III... Almost (1983)

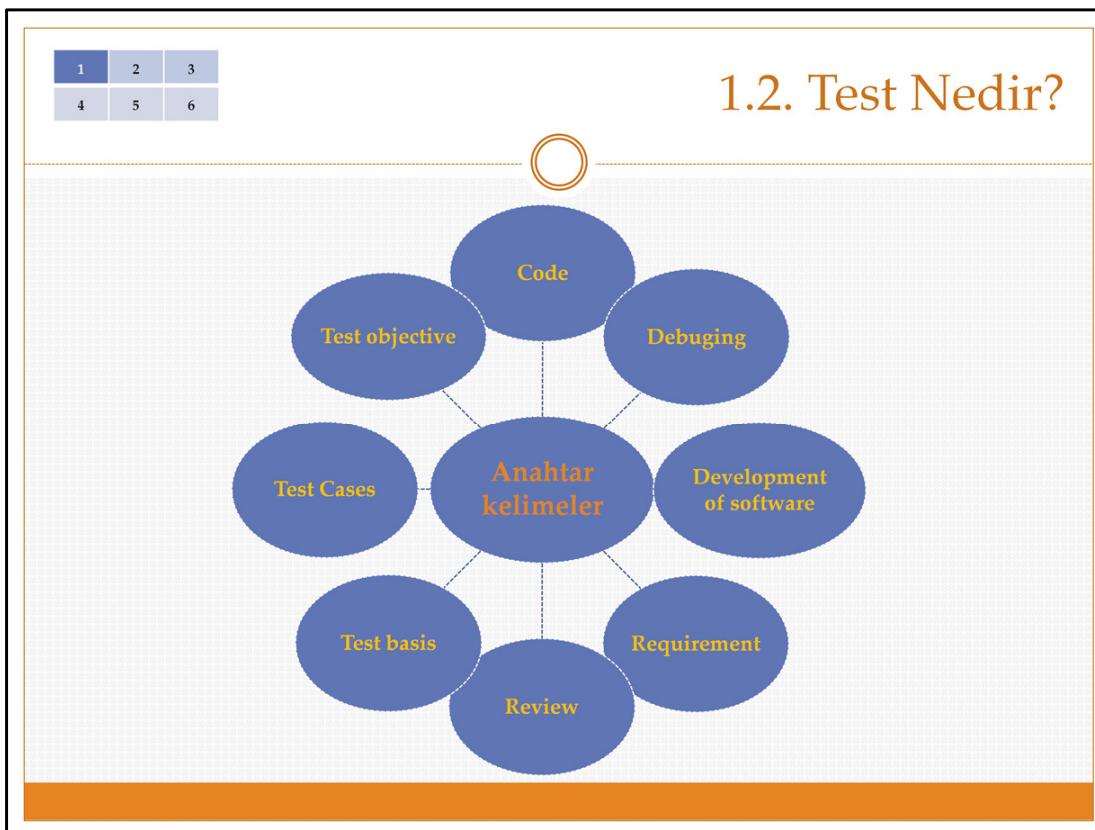
Radar sistemini Amerika tarafından 5 adet bazistik füzesinin üzerine gönderildiğini raporlayınca, savunmak amaçlı nukleer füzelerin gönderilmesi son anda iptal edildi. Sorumlu asker Amerikanın böyle bir saldırıyı yapmak için 5 adet füzeden çok daha fazla füze göndermesi gerektiğini düşünerek savunma füzelerinden vazgeçti. Hata radarın filtrelemesini yapan kodda olduğu anlaşıldı, güneş ışığı yansımاسının formule dahil edilmemiş.

1	2	3
4	5	6

## 1.1 Test Neden Gerekli?



- Özette neden test gerekli:
  - Yazılımda hata olabilir?
  - Yazılımın güvenirliliği öğrenilmek istenebilir?
  - Yayınlanması ile geliştirme arasındaki zamanı doldurmak içindir
  - Yazılımın hatasız olduğu belgelemek içindir
  - Proje planında olduğu için test yapılmalıdır
  - Hatanın sonucu çok pahalı olabilir
  - Müşterilerin dava açmalarını engellemek
  - Piyasada başarılı olabilmek



## Anahtar kelimeler

**Code:** Program parçaları

**Debuging:** yazılım geliştirme süresince yazılan program parçalarının çalıştırılması

**Development of software:** Yazılım geliştirme

**Requirement:** Yazılım gereksinimleri, neden yazılım gerekli, kullanıcı ihtiyaçları

**Review:** Gözden geçirme

**Test basis:** Tüm birim veya sistemlerin est için gerekli dökümanlar

**Test Cases:** Test case; testin tanımı, adımları, amaçlarını içeren dökümanlar

**Test objective:** Test yapmaktaki amaç nedir.

1	2	3
4	5	6

## 1.2 Test Nedir?



### • Sürücü Testi

- Test süreçlerini anlamak için kullanılır
- Nedir?
  - ✖ Eğer sürücü çeşitli yolları deneyerek bir rotayı takip eder ve çeşitli manevralar yaparak rotasını güvenli bir şekilde tamamlar ise sürücü testi başarılıdır denir.
  - ✖ Sürüş sırasında rotanın tamamlanmasına engel olabilecek **tek bir ciddi hata** tüm sürüşün başarısız olmasına sebep olur fakat sürüsüz sırasında **birçok küçük hata** sürüşün başarılı sayılmasına engel değildir.

### Sürücü Testi Nedir?

Eğer sürücü çeşitli yolları deneyerek bir rotayı takip eder ve çeşitli manevralar yaparak rotasını güvenli bir şekilde tamamlar ise sürücü testi başarılıdır denir.

Sürüş sırasında rotanın tamamlanmasına engel olabilecek **tek bir ciddi hata** tüm sürüşün başarısız olmasına sebep olur fakat sürüsüz sırasında **birçok küçük hata** sürüşün başarılı sayılmasına engel değildir. Yazılım testi açısından bakıldığından, yazılım birçok küçük hatalar olabilir hatta bunları bulmak uzman test mühendisleri için dahi zor olabilir, yazılım testi bu tip hatalardan ziyade yazılımlının konusuna ve önemine göre dikkat alınacak olan ciddi hataların bulunması ve raporlanması olarak açıklanabilir.

Hatalardan tamamen ayıklanmış bir yazılım sadece hayal ürünüdür fakat hataların kontrol edildiği, risk değerlendirilmesi yapıldığı ve raporlandığı test anlayışı daha uygulanabilir bir test anlayışıdır.

1	2	3
4	5	6

## 1.2 Test Nedir?

### • ISTQB'ye Göre Test

- Sürücü analojisinden yola çıkarak testin açıklamasını şu parçalara bölmek gereklidir:
  - ✖ Proses (process)
  - ✖ SLDC boyunca Test Yapılır
  - ✖ Statik (static) ve Dinamik (dynamic) testler vardır
  - ✖ Planlama (planning)
  - ✖ Hazırlanma (preparation)
  - ✖ Değerlendirme (evaluation)

#### **Proses (process)**

Test tek bir aktiviteden ziyade bir süreçtir. Belli kaideleri vardır ve herbir olgu bu kurallar içinde değerlendirilir. Sürecin kalitesi aslında kullanılan testin de kalitesini verir. Test kalite içerisinde en önemli başlığı fakat test bir süreç dahilinde işletiliyorsa.

#### **SLDC boyunca Test Yapılır**

Test sadece geliştirme tamamlandıktan sonra yapılan ve burada biten bir aktivite değil SDLC'nin her bir evresinde değişik test tipleri işletilerek yazılım sürecine katkıda bulunulur. Unutmayın ki iyi SDLC'nin erken evrelerinde bulunan hatalar kolay çözülebilir ve daha düşük maliyetler harçanır.

#### **Statik (static) ve Dinamik (dynamic) testler vardır**

Test sadece kara kutu (**black box**) değil aynı zamanda beyaz kutu (**white box**) ve gri kutu (**gray box**) testleride kapsamaktadır. Testin alanı sadece ekran karşısından bilenin fonksiyonlarının doğru çalışmasını kontrol etmek değil ilerde değinileceği gibi daha geniş bir kapsamı vardır.

#### **Planlama (planning)**

Test planı teste hazırlık kısmında yapılır ve buna bağlı kalmak şarttır.

#### **Hazırlanma (preparation)**

Testle ilgili her şey bu evrede hazırlanır ve teste başlama kriterlerinin (start entrance criteria) oluşmasıyla teste başlanır. Test sonlandırma kriterleri (test exit criteria) ile de test sonlandırılır.

#### **Değerlendirme (evaluation)**

Test sonlandırılması ile test sonuçları değerlendirilir.

1	2	3
4	5	6

### 1.3. 7-Test Prensibi?



- Test hatanın varlığını kanıtlar**
- Detaylı (exhaustive) test imkansızdır**
- Erken teste başlamak daha iyidir (early testing)**
- Hatalar kümelenir (defect clustering)**
- Gübre paradoksuna dikkat et (pesticide paradox)**
- Test bağlam bağımlıdır (context dependent)**
- Hata bulmak ve düzeltmek kullanıcıların isteklerini karşılamaz (absence-of-error fallacy)**

#### **Test hatanın varlığını kanıtlar**

Test sadece bulunan hata ile ilgilidir yazılımın geri kalanı hakkında bilgi vermez yani uzun süre koşturulan testlerden sonra halen hata (bug) bulunamıysa bu yazılımın artık hatalardan arındı anlamına gelmez.

#### **Detaylı (exhaustive) test imkansızdır**

Bir yazılımı uçtan-uca herşeyini test etmek imkansızdır. Mutlak test edilemeyen ya da edilmesi imkansız olan bazı durumlar olabilir. Bu yüzden "%100 test ettik" lafi hatalıdır.

#### **Erken teste başlamak daha iyidir (early testing)**

Hataların çözümü daha kolay ve maliyeti düşüktür.

#### **Hatalar kümelenir (defect clustering)**

Eğer bir hata bulunmuş ise bu hatayı tetikleyen başka noktalarda olabilir. Genel anlamda hata noktalarında yazılımcının kötü günü gibi psikolojik sebeplerde vardır bu yuzden hatanın bulunduğu çevreye dikkat edilmelidir.

#### **Gübre paradoksuna dikkat et (pesticide paradox)**

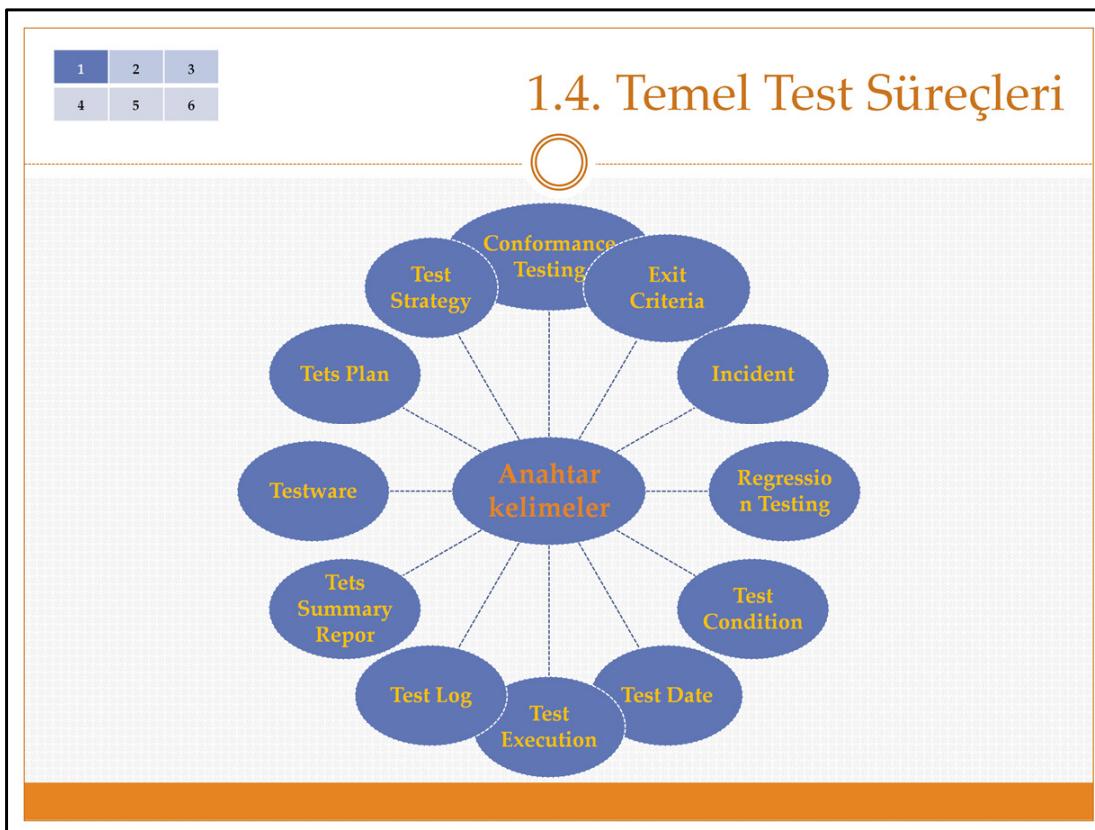
Bir yazılım üzerine sürekli aynı testleri koşturmak bir süre sonra yazılım üzerinde hata bulunamamasına sebep olur bu durum yazılımın hatadan ayılandığına değil uygulanan testlerin değiştirilmesi gerektiğini belirtisidir. Toprağı sürekli aynı gübre ile gübrelemek onun verimi sürekli arttırmaz, bir noktadan sonra zayıflatır.

#### **Test bağlam bağımlıdır (context dependent)**

Yazılımın konusuna göre testler farklılık göstermelidir. Örneğin e-business projesi ile bir askeri proje aynı test yöntemleri ile test edilmemelidir.

#### **Hata bulmak ve düzeltmek kullanıcıların isteklerini karşılamaz. (absence-of-error fallacy)**

Hatayı oluşturan durum kullanıcının istemediği durum olabilir yalnız bu hatanın düzeltilmesi kullanıcını isteklerini yerine getirmek demek değildir.



### Anahtar kelimeler:

**Conformance Testing:** Compliance testing, uyumluluk testleri

**Exit Criteria:** Testi sonlandırma kriterleri

**Incident:** Hadise, vaka, (hata)

**Regression Testing:** Regresyon testi; bulunan hata sonucu yazılımda yapılan değişikliğin sisteme etkisinin testi

**Test Condition:** Testi gerçekleştirmek için gerekli koşullar

**Test Date:** Testin gerçekleştirildiği tarih

**Test Execution:** Testin gerçekleştirilmesi

**Test Log:** Test yapılması sırasında tutulan hertürlü kayıt, döküman, vb.

**Test Plan:** Test sonrasında testin sonuçlarının özeti halinde sunulduğu test dökümanı

**Testware:** Test sonrasında ortaya çıkan her türlü test kayıtları, ürünleri, vb.,

**Test Strategy:** Test sırasında uygulması gereken kuralların bulunduğu test başlamadan önce hazırlanan plan

1	2	3
4	5	6

## 1.4 Temel Test Süreçleri?



- Temel test süreçleri aşağıdaki aşamalardan oluşur
  - Planlama ve Kontrol (Planning and control)
  - Analiz ve tasarım (Analysis and design)
  - Oluşturma ve yürütme (Implementation and execution)
  - Değerlendirme (Evaluating exit criteria and reporting)
  - Testi sonlandırma (Test closure activities)

1	2	3
4	5	6

## 1.4.1 Planlama ve Kontrol



- Tüm planlar bu aşamada yapılır
  - MTP (Master Test Plan) çıkartılır
- Test Stratejisi belirlenir
  - Test Yaklaşımı (Test Approach) tayin edilir
- Teste Başlama (Entry Criteria) ve Testi Sonlandırma (Exit Criteria) kriterleri belirlenir

### MTP

Master Test Planı içerisinde test yönetimine ait tüm bilgileri içeren dökümandır. Bu döküman planlama aşamasında hazırlanır ve tamamlandıktan sonra tüm ilgili paydaşların (yazılımcı, analist, testçi, proje yöneticisi, ...) her zaman ulaşabileceğii şekilde yayınlanır. MTP daha sonraki aşamalarda ikilik çıkması durumunda başvurulacak kaynak niteliğindedir bu yüzden özenle hazırlanmalı ilgili kişilerin onayları alınmalıdır.

### Teste Başlama Kriteri

Test başlama kriterleri, herhangi bir **yazılım çıktıının** testine başlamadan önce sahip olması gereklidir en temel özelliklerin listesidir. Bu liste SDLC göz önüne alındığında çeşitli kriterlerden söz edilebilir. Örneğin, geliştirme tamamlandı ve yazılım üzerinde fonksiyonel testlere başlanacak, yazılımın hazır olup olmadığını kontrol etmek gerekiyor. Bu durumda uzman ya da moderator tarafından şu kontroller yapılabilir:

- 30 dk boyunca kontrol edildi ve bir sayfada 3 büyük defect'ten başka birsey bulanımıyorsa,
- 5 değişik sayfada 10'dan fazla önemli defect bulunmamış ise,
- ...

### Testi Sonlandırma Kriterleri

Test için sonsuz zaman ve kaynak olmadığından bir süre sonra test sonlandırmak gereklidir. Genelde risk analizi yapılarak ne kadar test yapılacağına karar verilir. Test sonlandırmak için belirlenmiş kriterlere **exit criteria** denir.

1	2	3
4	5	6

## 1.4.1 Planlama ve Kontrol



- Test Stratejisi:
  - Projeye özgüdür
  - Ortak kanı verir
  - Süreçler belirlenir
- Test Yaklaşımı:
  - Daha küçütür, test tiplerini gruplar
- Değerlendirilmesi gereken bileşenler:
  - Yazılım geliştirme stratejileri
  - Müşteri gereksinimleri
  - Test amaçları
  - Proje konusu

### Test Stratejisi:

- Test yaklaşımının ana hatlarını çizen ve projeye özgü hazırlanan kurallar bütünüdür.
- Organizasyondaki herkesin bilgilenmesi ve bu doğrultuda hareket etmesini sağlar.
- Test caselerin tasarılanması, test süreçlerin belirlenmesi, teste başlama ve testi sonlandırma kriterlerin belirlenmesinde ana rol oynar.

### Test Yaklaşımı:

Projeyi oluşturulan bütün bileşenleri içerisine katarak daha başarılı test sonuçları yakalamak için kullanılan test tiplerinin gruplandırıldığı test anlayışıdır.

### Değerlendirilmesi gereken bileşenler:

Test stratejisini ve yaklaşımını belirlemek için bu kriterler göz önüne alınmalıdır.

1	2	3
4	5	6

## 1.4.1 Planlama ve Kontrol



- ISQTB Örnek Test Yaklaşımları:

- Analytical strategies, such as risk-based testing
- Model-based strategies, such as operational profiling
- Methodical strategies, such as quality-characteristic based
- Process- or standard-compliant strategies, such as IEEE 829-based
- Dynamic and heuristic strategies, such as using bug-based attacks
- Consultative strategies, such as user-directed testing
- Regression testing strategies, such as extensive automation.

Örnek bir değerlendirme yapılır ise;

Bir e-ticaret firmasını örnek olarak aldığımızda 2 Alanda inceleme yapılabilir: **Proje konusu ve Geliştirme Ortamı**

**Analitik Yaklaşım:** İnternet üzerinden satış yapıldığı için bazı riskler vardır. Örneğin; güvenlik problemleri, firma imajını zedeyecek görsel ve yazım hataları, ... Olmasından dolayı test caselerin risklerinin sınıflandırılmasında kullanılabilir.

**Dinamic Yaklaşım:** Gereksinimlerin olmadığı, planlamanın tam olarak yapılmadığı hızlı bir yazılım geliştirme ortamı olmasından ötürü de dinamik test yaklaşımı kullanılabilir.

Analitik ve Dinamic test stratejilerinin bileşiminden oluşan; **Risk Temelli - Dinamik test yaklaşımı** belirlenebilir.

- *Hata verme olasılığı ve daha yüksek alanlara daha fazla test yapmak*
- *Keşif tabanlı testler yaparak hata tahminlerinde bulunarak, ilgili alanlara yoğunlaşmak*

1	2	3
4	5	6

## 1.4.1 Planlama ve Kontrol



- Nekadar Test Etmek Gerekir?

- Sürekli, herzaman
- Planladığınız zaman bittiğinde
- Kullanıcı/müşteri yeterli hissettiğinde
- Yazılımın doğru çalıştığını ispat ettiğinizde
- Yazılımın doğru çalıştığından emin olduğunuzda
- Yazılımın risk değerlendirmesine bağlı olarak

### **Yazılımın doğru çalıştığından emin olduğunuzda**

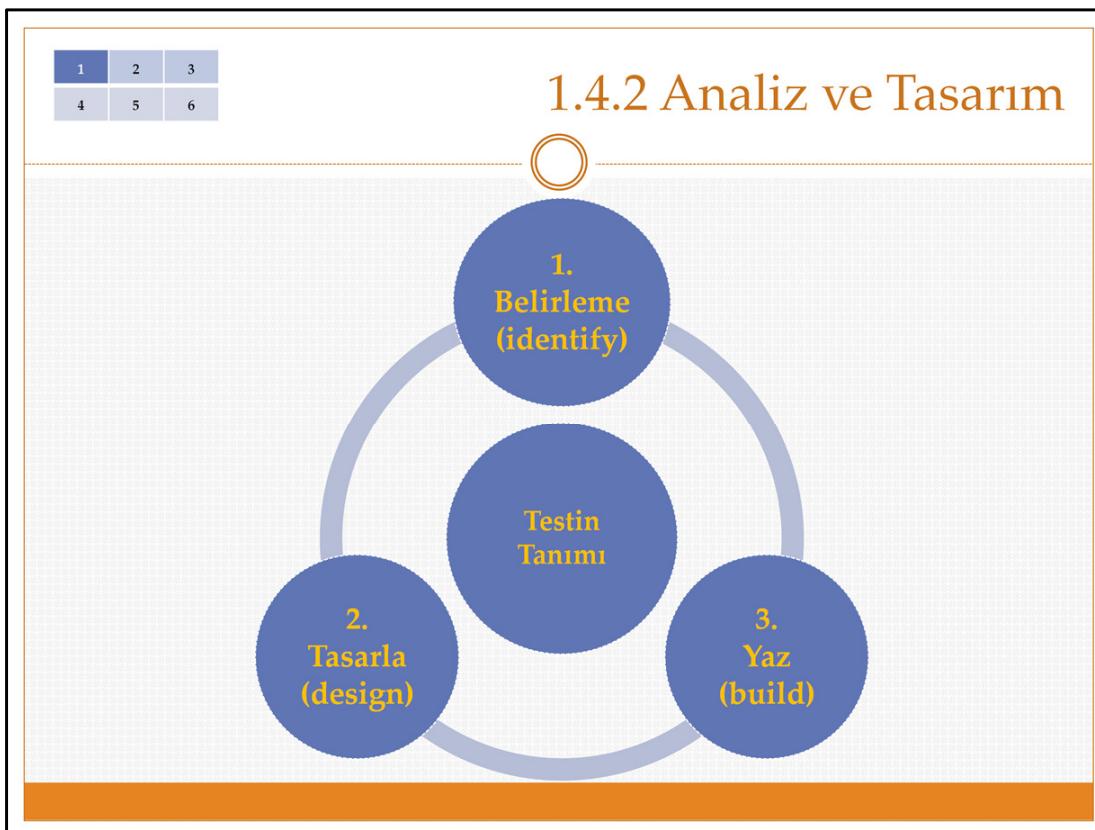
Yazılım gereksinimleri göz önüne alındığında çok fazla olmayabilir veya uzun bir zaman aralığında testlerinizi başarılı bir şekilde tamamlamış ve daha fazla hata bulamıyor olabiliyorsınız bu durumda yazılımın doğru çalıştığı konusunda kendinizi emin hissediyorsanız testleri durdurarak yayına almak (**go live**) gerekir. Fakat emin olma durumu kişiden kişiye göre ve deneyimlere göre değişiklik gösterdiği için bu madde herzaman geçerli değildir.

### **Yazılımın risk değerlendirmesine bağlı olarak**

Test için zaman herzaman kısıtlıdır. Herşeyi planlandığı gibi test etmek genelde mümkün olmaz bu durumda riskleri göz önüne alarak:

- Nelerin en çok,
- Nelerin ilk,
- Hangi bölümün kısmı veya tamamını,
- Test edilecek bölümlerin ne kadarlık zaman içinde,
- Nelerin test edilmeyeceğini

Belirleyerek test zamanını en etkin nasıl kullanılması gerekiğinin analizi yapıldığında ve bu zamanın sonlanması ile test bitirilebilir.



### Identify

Neyin test edilceğini belirlenmeli ve **test edilebilecek (testability)** konular bir liste haline getirilerek önem sırasına göre sıralanmalıdır. Testin başarılı sonuçlanması için gerekli yeter koşullar berirlenmeli ve oluşabilcek en kötü durumlar ve bu durumda yazılım kararlılık tablosu çıkartılmalı. Yazılımın yapısı hakkında bilgiler edinilir.

### Design

Üst düzey test caseler çıkarılır. Belirlenen konuların nasıl test edileceği belirlenmeli ve test caselerin yazımı için gerekli çalışmalar yapılmalıdır. Teste kullanılacak veriler belirlenmeli, veriler (**input**) sonucunda yazılımın verdiği cevap (**output**) belirlenmeli.

### Build

Test caseleri yazılmalı. Scriptler yazılarak test araçları üzerinden testlerde yapılabilir.

1	2	3
4	5	6

### 1.4.3 Oluşturma ve Çalıştırma



- Test Caseleri oluştur
  - Detay seviyesi anlaşılır olmalı
- Çalıştırma (**execution**)
  - Önemli test caseleri önce koşturulmalı
  - Manuel olarak ya da otomatik testler koşturulabilir
  - Bütün test caseleri işletilemeye bilir
    - Hata bulunması halinde düzeltilmesi beklenecek
    - Çok fazla hata bulunması durumunda testi durdurulabilir
    - Zaman yeterli olmayabilir

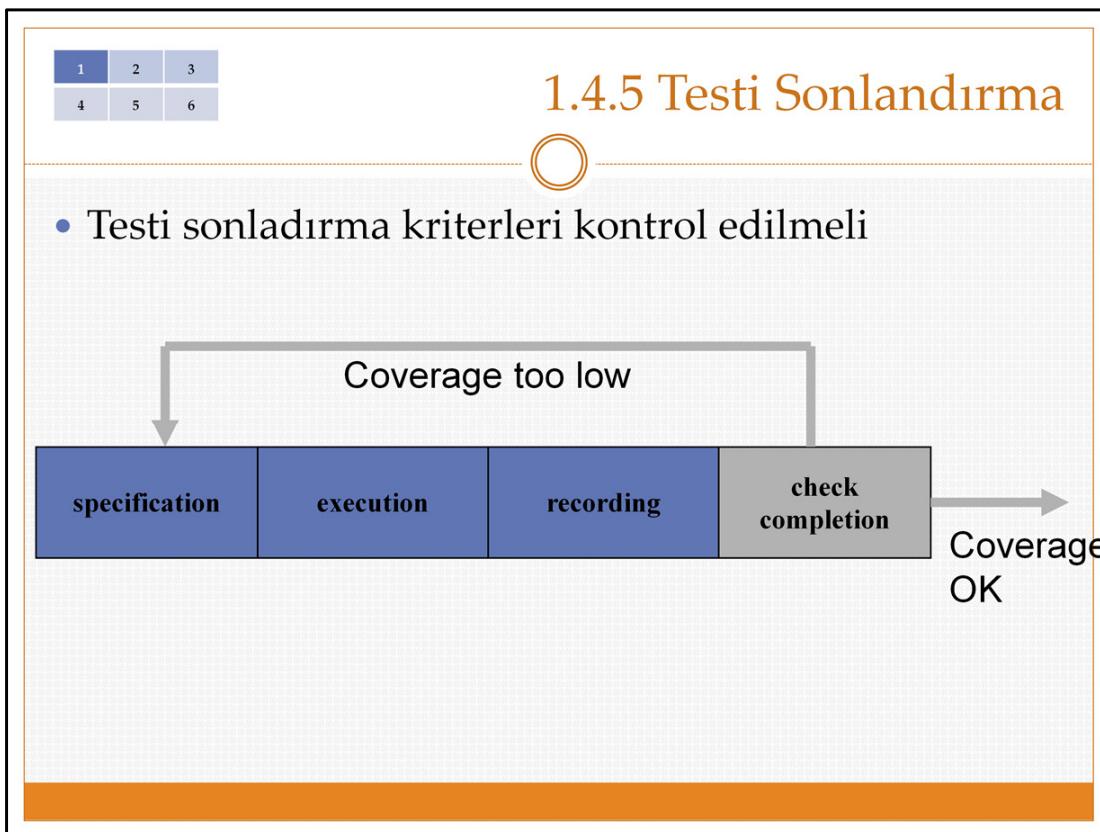
1	2	3
4	5	6

#### 1.4.4 Değerlendirme

- Testi sonlandırma kriterlerinin değerlendirilmesi ve raporlanması
  - Exit Criteria gerçekleştiğinde test sonuçları değerlendirilmelidir
  - Çıkan sonuçlar raporlandırılmalıdır
  - Yazılımın yayınlanmasından sonra oluşabilecek riskler belirlenmelidir

##### Değerlendirme ve Raporlama

MTP dökümümanına uyarak ve test sonlandırma kriterleri gerçekleştiğinde daha fazla test yapılmadan test sonuçları değerlendirilmelidir. Bu aşamada bulunan hatalar ve bu hataların sonucunda oluşturulan raporlar incelenmeli, halen düzeltilmeyi bekleyen hatalar var ise bunlar vurgulanmalıdır.



#### Exit Criteria

**Test Coverage** (işletilen testlerin toplam testlere oranı) kontrol edilmelidir. Coverage şu başlıklar altında incelenebilir:

- Branch coverage
- Statement coverage
- Condition coverage
- User requirements
- Saptanan hatalar
- Para ve zaman

Eğer süreç sonunda testi sonlandırma kriterleri yeterli değil ise tekrar test detaylarına bakılarak süreç yeniden işletilmelidir.

1	2	3
4	5	6

## 1.5. Test Psikolojisi



- Neden Test

- Güven inşaa etmek için ✓
- Yazılımın doğru çalıştığını ispat etmek için ✗
- Gereksinimlere uygunluğunu göstermek için ✓
- Hata var ise bulmak için ✓
- Masrafları azaltmak için ✓
- Sistemin kullanıcı ihtiyaçlarını karşıladığı göstermek için ✓
- Yazılımın kalitesini sürdürmek için ✓

1	2	3
4	5	6

## 1.5. Test Psikolojisi



- Geneleksel Test Yaklaşımları ile:
  - Yazılımın yapması gereken özellikleri
  - Yazılımın yapmadığı özellikleri saptanır

**Sonuç: çalışıyor mu?**

**Başarısı: çalışanlar bulunur**

**Kolay test caseleri hazırlanır**



**Sonuç:**  
**Hatalar bulunamaz**

1	2	3
4	5	6

## 1.5. Test Psikolojisi

- Daha iyi bir Test Yaklaşımı ile:
  - Yazılımın yapmaması gereken özelliklerini
  - Yazılımın neleri yapmadığı saptanır

**Sonuç: defect bulunur**

**Başarısı: sistem hataları saptanır**

**Zor test caseleri hazırlanır**

**Sonuç:**

**Çok az hata kalır**

### Test Yaklaşımı

Projeye uygulanacak test tiplerini belirleyen anlayışa test yaklaşımı denir. Başarılı test yaklaşımı ile detay analizden ortaya çıkan test caseleride içeresine katan test caseleri hazırlamak mümkün olur. Üst düzey analiz çıktıları daha kolay anlaşılır olduğundan, acemice uygulanan bir test yaklaşımı ile bunun ötesine gitmek zor olabilir. Daha iyi bir test yaklaşımı ile;

- Yazılımın neleri yaptığı ortaya çıkartılır ve bunların içerisinde de yapmaması gereken özellikler ayıklanır ve bunlar hata olarak raporlanır.
- Ayrıca yazılımın yapması gereken fakat yapmadığı fonksiyonlarında ortaya çıkartılmış olur.

Geleneksel test yaklaşımı ile kolay test caseleri hazırlanır ve ancak yüzeysel hatalar bulunabilir. Sonuç olarak bulunması gereken hatalar yazılımda kalır ve geresinimlerden yazılıma aktarılmamış olan fonksiyonlar saptanmamış olur. Bu da yazılımın kalitesini ve müşteri memnuniyetini düşürür.

1	2	3
4	5	6

## 1.5. Test Psikolojisi



- The Testing Paradox

Testin amacı: HATA BULMAK

Hata bulmak GÜVENİ zedeler?

Testin amacı: GÜVENİ zedelemek

Testin amacı: GÜVEN vermek

GÜVEN

vermenin en iyi yolu onu  
yıkmaya çalışmaktadır

1	2	3
4	5	6

## 1.5. Test Psikolojisi



### • Testçinin özellikleri

- Test mühendisi, lideri veya yöneticisi süreç hakkında bilgili ve iletişim kurmakta zorlanmayan, test sürecini uygulamak gibi hedefleri olan kişilller olmalıdır.
- Geliştirme ortamı kötü bile olsa test ortamı daima ideal yapıya daha yakın olmalı
- Test ortamı ile ideal test ortamı arasındaki farkı bilen ve bu farklılıktan dolayı ortaya çıkabilecek sorunları analiz edebilen özellikte olmalı

Test bir süreçtir! Bu yüzden uygulamanın bir özelliğinden öte uygulamanın hayat döngüsünün test edilmesi gerekmektedir. Uygulamanın tüm yönleri ele alınmalı; gereksinimlerden yola çıkararak uygulamanın istenilene ne kadar yakın?, İstenilenin de ideale ne kadar yakın? gibi sorular karşısında pratik cevap verebilen kişilerin test mühendisliğini tercih etmeleri gereklidir.

Ayrıca geliştirme süreçleri her zaman ideal yapıda olmayabiliyor bununla birlikte planan süreçlerde sapmalardan dolayı uygulamanın teste hazır olmasındaki gecikmeler test süresinin kısalmasına veya testin amacının değiştirilmesine neden olabiliyor. Testten sorumlu kişilerin bu durumu iyi yönetmeleri gerekmektedir. İletişim yetenekleri güçlü ve test süreçlerini uygulatmak gibi hedefleri olan kişiler olmalıdır. Bir yazılım ekibinde süreçler ne kadar zayıfsa oradan çıkacak ürünlerde okadar zayıf olur bu yüzden test ekinin daha kaliteli ve hedefleri daha net olması gereklidir.

Test ortamı çeşitli sebeplerden dolayı her zaman ideal yapıda olmayabilir bu durumda test mühendisi, ideal test ortamı ile mevcut test ortamı arasındaki farkı bilmeli ve test edilemeyen fonksiyon yayınlanan ortamda ne gibi hatalara sebep olacağını analiz edebilen özellikte olmalı.

1	2	3
4	5	6

## 1.5. Test Psikolojisi



- **Testçinin İhtiyaçları**

- Süreç hakkında tam bilgi
- Yazılımın içeriği hakkında bilgi
- Teste gelen kodun tamamlanmış olması
- Uzmanlık alanı olarak kabul görülmesi
- Hataları bulma hakkı
- Test planı ve detaylandırılmasında söz hakkı
- Ciddi hataları raporlama yetkisi (tekrar gözlenemeyen)
- Gelecekteki hata seviyesi hakkında öngörüde bulunma
- Test süreçlerini iyileştirme yetkisi

Test yapabilmek için bazı yetkilerin ve hakların organizasyonda test mühendislerine tanınması gerekmektedir. Bu yetkiler sayesinde testciler, daha iyi test yapma, saptanan hataları tarafsız bir şekilde raporlama ve çözüdürebilme kabiliyeti kazanır. Test bir süreç olmasından dolayı, test süreçlerinde saptanan hataları düzeltme yetkisiyle birlikte daha iyi test yapabilmenin önü açılabilir.

1	2	3
4	5	6

## 1.5. Test Psikolojisi



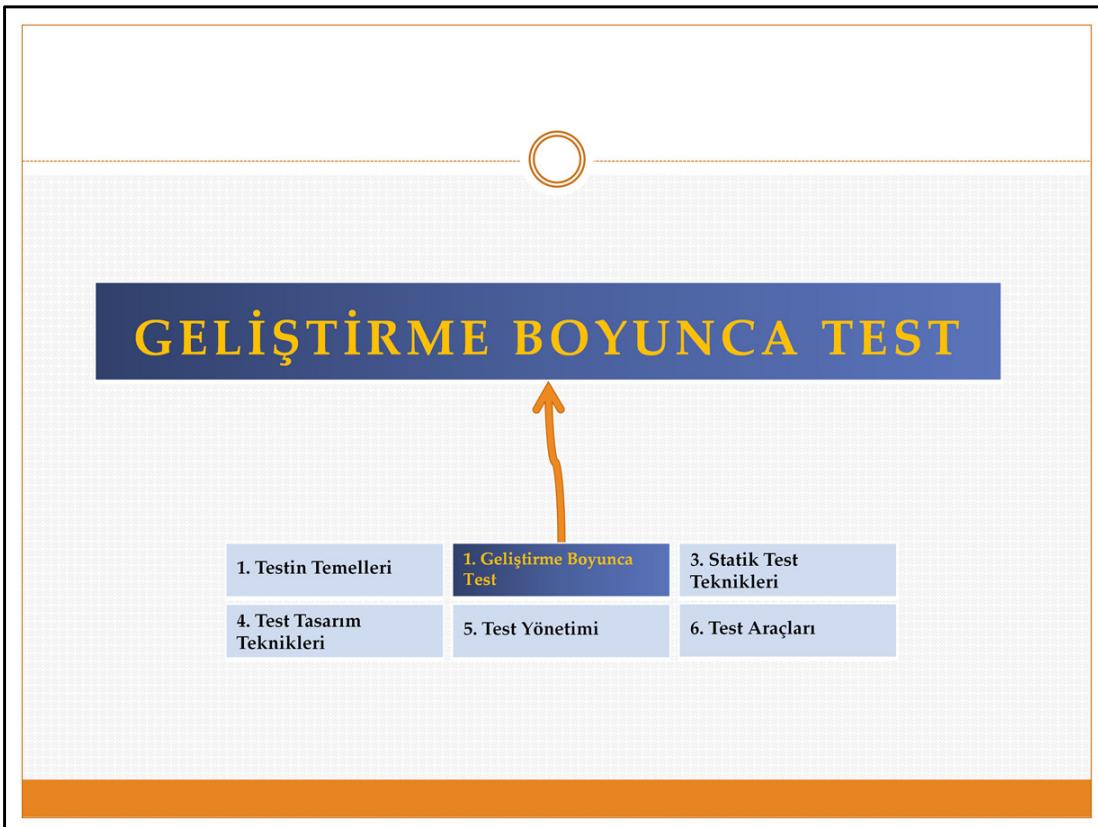
### • Testçinin Sorumlulukları

- Test planını takip etmek
- Hataları tarafsız ve gerçekçi raporlamak
- Hatayı raporlamadan önce kontrol etmek
- Yazılımcıyı değil yazılımı test ettiğinin bilincinde olmak
- Riskleri tarafsız değerlendirmek
- Hataları önceliklendirmek (prioritize)
- Gerçekleri paylaşmak

Testçinin görevi atanan yazılımı belirlenen test planına uyararak testlerini gerçekleştirmektir. Burada amaç ise yazılımın gereksinimlerinin onaylanması ve doğrulanmasıdır (verification and validation). Bu işi gerçekleştirirken bulunan hatalar tarafsız ve herkesin anlayacağı bir şekilde raporlanmalı ve unutulmamalıdır ki **yazılımcı değil yazılım test ediliyor** bu yüzden hatalar tekrar kontrol edilmeli ve kibar bir dil kullanılmalıdır.

# ISTQB Müfredatı

-  Testin Temelleri
-  **Geliştirme Boyunca Test**
-  Statik Test Teknikleri
-  Test Tasarım Teknikleri
-  Test Yönetimi
-  Test Araçları



1	2	3
4	5	6

## 2. Geliştirme Boyunca Test



 Yazılım Geliştirme Modelleri

 Test Seviyesi

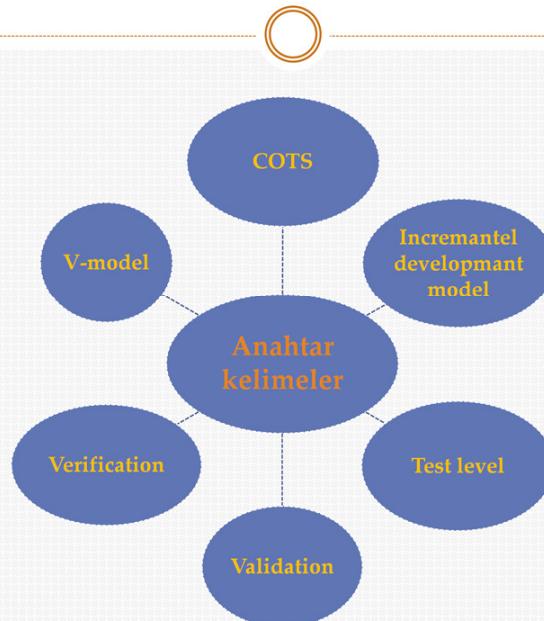
 Test Türleri

 Bakım Testi

SDLC (Software Development Life Cycle – Yazılım Geliştirme Yaşam Döngüsü) boyunca test etmek niçin gerekli bu ünitede tartışılabilecek. Yazılım geliştirme methodları neler olduğu ve farklı methodlarda nasıl daha iyi test edileceği yine bu ünitenin konusu içerisinde yer almaktadır.

1	2	3
4	5	6

## 2.1 Yazılım Geliştirme Modelleri?



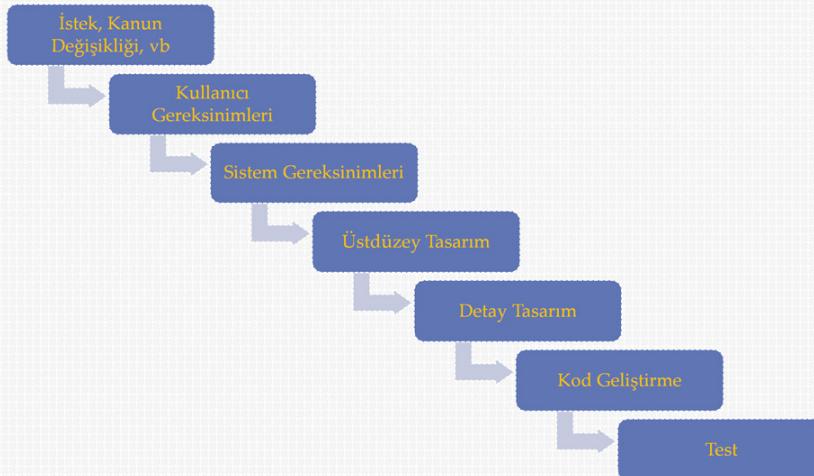
### Anahtar Kelimeler:

- **COTS (Commercial off-the-shelf):** paket programlar
- **Incremantel developmant model:** Arttırımsal (iteratif) yazılım geliştirme modeli
- **Test level:** Test seviyesi, birim-entegrasyon- ...
- **Validation:** Doğrulama
- **Verification:** Onaylama
- **V-model:** Şelale modele alternatif olarak geliştirilen, testing herbir aşamada olduğu yazılım geliştirme modeli

1	2	3
4	5	6

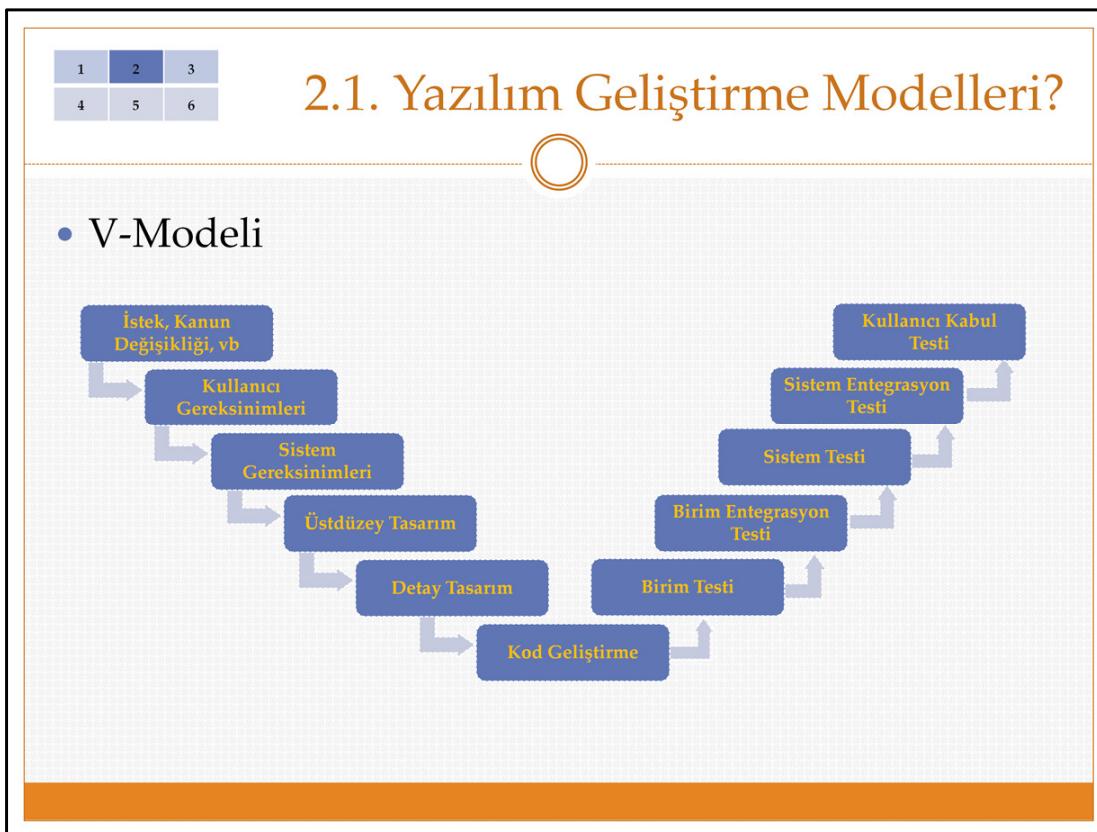
## 2.1. Yazılım Geliştirme Modelleri?

- Waterfall (Şelale) Modeli



Şelale modeline göre yazılım geliştirme süreçleri birbirini takip eden farklılaşmış yazılım uzmanlık alanlarından oluşmaktadır. Yazılımın ihtiyaç duyulması (İstek, Kanun Değişikliği, vb) aşamasından başlar en sondaki aşama olan test ile sonlanır ve yazılım devreye alınır. Yani her bir aşamada kendisine iş gelene kadar beklemektedir. Burada karşılaşılan en büyük sorun ise zamanlama sorunuudur. Yazılıma başlanırken uygulanan plana bağlı kalınmadığı durumlarda, örneğin test aşamasına gelene kadar her aşamadaki aşamadaki ufakda olsa geçikmeler kümülatif olarak toplandığında test için ayrılan sürenin kısalması veya olmaması durumuyla karşı karşıya kalınabilir.

Şelale modelinde özellikle analizde oluşan hatalar testte yakalandığında hatanın maliyeti çok yüksek olmaktadır. Yani hatalı durum için yeniden analiz-tasarım-kodlama-test süreçlerinin işletilmesi gerekmekte ve planlanan süreyi uzatmaktadır. Genel olarak büyük projelerde şelale modelinin uygulanması önerilmemektedir.

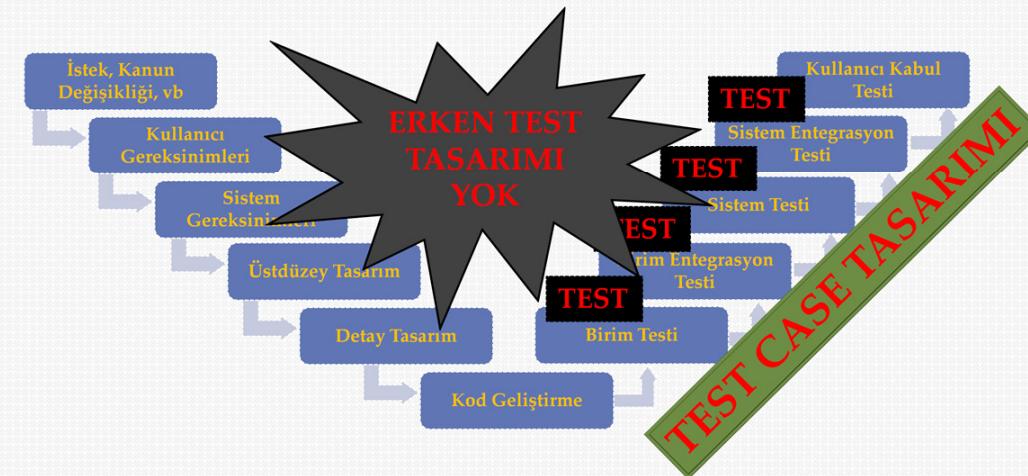


Şekilde görünen yazılım geliştirme modeli şeklinde dolayı V-Model ismini almıştır. Buradaki alında iki kısma ayrılabilir. Birinci kısım İstek Toplama – Kod Geliştirme ve ikinci kısım ise Kod Geliştirme – Kullanıcı Kabul Testleri’ne kadar olan kısımdır. V-Modeline göre herbir yazılım sürecine farklı test yöntemleri uygulanır böylece ortaya çıkabilecek hataları daha önce bulma yeteneği kazanır. Böyle test aktivitesi yazılım geliştir hayat döngüsü boyunca uygulanmış olur.

1	2	3
4	5	6

## 2.1. Yazılım Geliştirme Modelleri?

- V-Modeli: Geç Test (late testing)



Geç tasarım modelinde Kod Geliştirme süreci bittiğinde test süreci başlar. Daha öncesinde test için hazırlık yapılmamasından dolayı, testlerin koşturulmasından önce test caselerin hazırlanması yani tasarımı yapılması gerekmektedir. Bu durumda V-Modelin avantajı tam olarak kullanılmayıp yine test uzmanları test aktivitesine başlamak için kod geliştirme sürecinin tamamlanmasını beklerler. Bu durumda ise erken test tasarımları olmadığından dolayı yine test süreci başladığında test tasarımları başlamaktadır.

1	2	3
4	5	6

## 2.1. Yazılım Geliştirme Modelleri?

○

- V-Modeli: Geç Test (late testing)

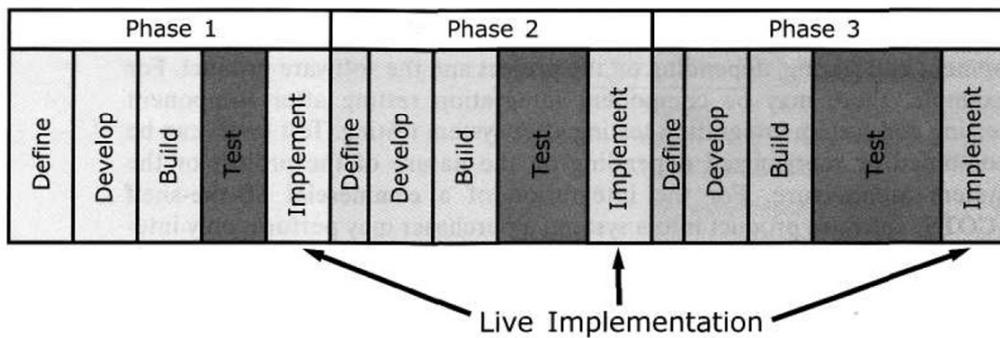
The diagram illustrates the V-Model for software development. It features two parallel vertical columns of boxes connected by arrows, forming a V-shape. The left column represents the 'Top Level' of requirements and design, while the right column represents the 'Bottom Level' of implementation and testing. The boxes are arranged in pairs along the V-shape, with arrows indicating a flow from top-left to bottom-right. A large green diagonal banner across the middle of the V-shape contains the text 'TEST CASE TASARIMI' (Test Case Design) on the left and 'TEST CASE CALISTIRMA' (Test Case Execution) on the right. The boxes in the left column are: 'İstek, Kanun Değişikliği, vb' (Requirements, Law Changes, etc.), 'Kullanıcı Gereksinimleri' (User Requirements), 'Sistem Gereksinimleri' (System Requirements), 'Üstdüzen Tasarım' (High-level Design), 'Detay Tasarım' (Detailed Design), and 'Kod Geliştirme' (Code Development). The boxes in the right column are: 'Kullanıcı Kabul Testi' (User Acceptance Test), 'Sistem Entegrasyon Testi' (System Integration Test), 'Sistem Testi' (System Test), 'Birim Entegrasyon Testi' (Module Integration Test), 'Birim Testi' (Module Test), and 'Kullanıcı Kabul Testi' (User Acceptance Test, repeated at the end). The word 'TEST' is printed in red capital letters above each of the six boxes in the left column.

Erken test modelinde test aktivitesi daha erken başlamaktadır. İsteklerin aldığı aşamadan itibaren test aktivitesi test tasarımları ve model base test şeklinde gerçekleştirilebilir. Yani gereksinimler toplanmasıyla birlikte gereksinimler test edilebilir veya analiz çıktıları ortaya çıktığında süreç içerisinde mantıksız, muallak veya işletilmesi anlamsız olan süreçler çıkartılarak daha kod geliştirme sürecinde yazılımcı için daha açık ve net analiz çıktıları sağlanabilir. Daha sonra hazırlanan test caseler, test case çalışma aşamasında hızla çalıştırılarak süreç daha hızlı işletilebilir. Bu model ile süreç toptan iyileştirilmiş ve test aşamasına daha olgun yazılımın gelmesine fayda sağlayabilir.

1	2	3
4	5	6

## 2.1. Yazılım Geliştirme Modelleri?

- İteratif Yazılım Geliştirme



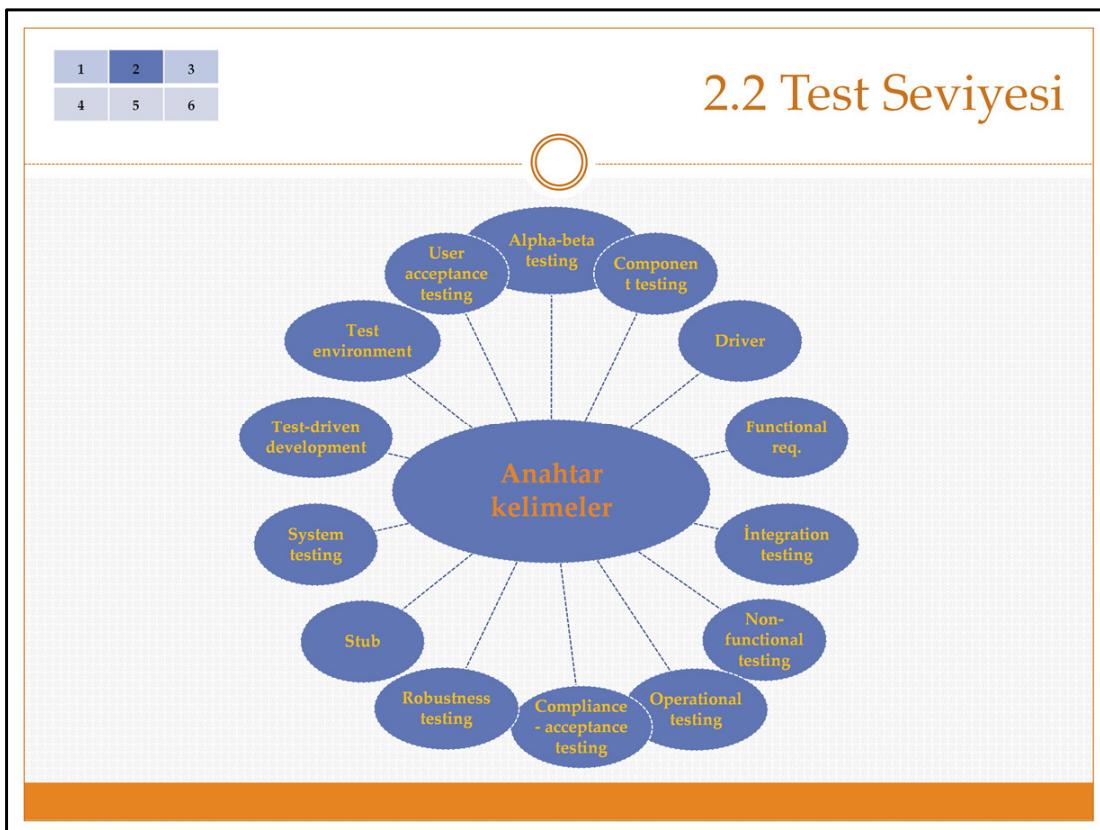
İteratif yazılım geliştirme yöntemi, şelale modelinin (geleneksel yöntemler) alternatifi olarak düşünülebilir. Çünkü şelale modelini ele aldığımda tüm yazılım süreci tek bir iterasyondan oluşmakta ve en son adımda test süreci bir patlama şeklinde herşeyi test etmeyi amaçlamaktadır. İteratif yazılım geliştirme modelinde ise tüm yazılımı tek bir iterasyondan öte süresi belli olan ve bu süre içerisinde yazılımı ilgili parçalara bölgerek belirlenen süre içerisinde bitirmeyi amaçlamaktadır. İteratif modelde her iterasyonun süresi bellidir yazılım içerisindeki tüm roldeki kişiler bu zamanlamanın farkındadır ve olası geçikmeler sadece ilgili iterasyonu etkiler. Bu sebepten yönetilebilirliği diğer modellere göre daha iyidir.

1	2	3
4	5	6

## 2.1. Yazılım Geliştirme Modelleri?

- Agile Yazılım Geliştirme Modeli
  - XP Modeli
    - Extreme Programming
  - Daha fazla insan odaklı
  - Daha fazla iterasyon içerir
  - Test mühendisi yoktur
    - Herkes yazılım mühendisidir
  - Test scripts first
    - Önce test scriptleri yazılır sonra koda başlanır

Agile metodu iteratif yazılım geliştirme içerisinde yer almaktadır. Daha fazla insan odaklıdır çünkü konuların (işlerin) günlük olarak yapılan kısa toplantılarla üzerinden geçer ve yazılımcının canını sıkan durumlar var ise onlar dile getirilir ve paylaşılır. Agile dahada özelleşmiş bir yöntemi olan XP (extreme programming) yöntemi ise öncelikle test caseler için scriptler yazılır ve bu scriptler zaman zaman çalıştırılarak tamamen hatasız geçene kadar yazılım geliştirilir. Yani önce test scripleri yazılır sonra ürün kodu. Bunun içindirki bu yöntemde herkes biraz test mühendisidir, test mühendisleride yazılım uzmanıdır; yani herkes yazılım uzmanı olarak değerlendirilir.



### Anahtar Kelimeler:

**Alpha-beta testing:** Alfa testi, yazılım geliştirme ortamında yapılan kullanıcı kabul testleridir; Beta testi ise kullanıcıların kendi ortamlarında bir fiil uygulamayı kullanarak yaptıkları test aktiviteleridir. Eğer belirli sayıda ve davet ile belli profilde kullanıcılara izin verilerek beta testi amaçlanıyor ise buna **closed-beta** testi denir; kullanıcıarda herhangi bir sınırlama yok ise **open-beta** testi denir.

**Component testing:** Birim testi (unit testing)

**Driver:** Sürücü

**Functional req. :** Fonksiyonel Gereksinimler

**Integration testing:** Entegrasyon testi

**Non-functional testing:** Fonksiyonel olmayan testler. Örn: performans, güvenlik, kullanılabilirlik, ...

**Operational testing:** Operasyonel testler

**Compliance - acceptance testing:** Uygunluk (kurallar, yasalar, vb) testi

**Robustness testing:** Sağlamlık testi, Birim veya sistemin uygun olmayan durumlarda çalışabilmesi

**Stub:** Bir birimi test etmek için hazırlanmış program parçaları

**System testing:** Sistem tesleri, entegre çalışan sistemlerin testi

**Test-driven development:** Test yönelik yazılım geliştirme. Test caseler önce hazırlanır ve tüm caseler testten geçene kadar yazılım devam eder.

**Test environment:** Test ortamı

**User acceptance testing:** Kullanıcı kabul testleri

1	2	3
4	5	6

## 2.1. Test Seviyesi

- **Test Seviyesi (Test Level) nedir?**

- Farklı seviyelerde yapılan test aktivitelerine göre isimlendirilir.
- Uygulanan test türleri farklı olabilir
- **Test seviyeleri:**
  - Birim Test
  - Entegrasyon Testi
  - Sistem Testi
  - Kabul Testleri

Test seviyeleri testin hangi derinlikte yapılağı ile ilgilidir. Örneğin birim test veya birim entegrasyon test seviyeleri genelde kod düzeyindedir ve burada yazılan kodların doğruluğu, gerekliliği tartışılarken daha üst seviyedeki testlerde örneğin kabul testlerinde genelde karakutu olarak adlandırılan testler uygulanarak gereksinim, kanun, vb. Kriterlere uygunlugu bakılır. Bu yüzden farklı test seviyelerinde farklı testler uygulanır.

1	2	3
4	5	6

## 2.1. Test Seviyesi



- Birim Testi

- Diğer isimleri

- Component, unit, module, program testing

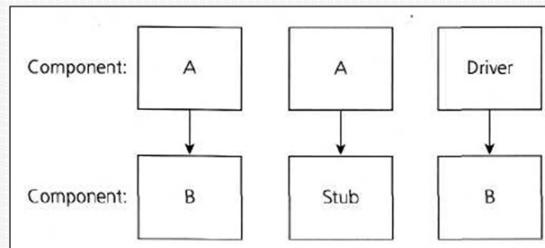
- Sadece bir birim test edilir

- Gerekli olan diğer birim yada sistemler için **stubs**, **driver** kullanılır

- Yapısal (structural) testlerdir

- Kod Seviyesine inilir

- Programcılar yapar



Birim testleri genelde yazılımcının sorumluluğunda olan testlerdir. Kod seviyesinde yapılır. Birim kelimesi tek bir fonksiyonun testide olabilir bir sınıf veya bir modülün testinde içerebilir. Kapsamı kodu yazan kişi tarafından belirlenir.

Tek bir iş test edildiği için bağlantılı diğer birimlerden izole edilmelidir yani giriş ve çıkışların her zaman doğru döndüğü düşünülerek test yapılır. Bu yeden diğer birimlerle olan alışverişler **stub** veya **driver** olarak adlandırılan objelerle gerçekleştirilir. Diğer birimlerde karşılaşılacak sorunların çözümü birim entegrasyon testinin içerisinde çözülebilir.

1	2	3
4	5	6

## 2.1. Test Seviyesi

- Entegrasyon Testi
  - Farklı sistemlerle bağlatılar
    - İşletim sistemleri
    - Dosyalama sistemleri
    - Arayüzler
  - İki seviyede entegrasyon olabilir
    - Birim Entegrasyon
    - Sistem Entegrasyon
  - Entegrasyon test stratejileri
    - Big-bang
    - Incremental (top-down, bottom-up, functional)

Farklı birimlerin ya da sistemlerin birlikte çalışırken ortaya çıkabilecek sorunların test edilmesine entegrasyon testi denir. Birden fazla birim test edilir ise birim entegrasyon; birden fazla sistem test edilirse sistem entegrasyon testi denir.

Neler test edilir:

- Farklı birimlerin iletişimi
- Bireysel olara set edilemeyen birimler
- Fonksiyonel olmayan özellikler (memory leak, ...)

1	2	3
4	5	6

## 2.1. Test Seviyesi



### • Sistem Testi

- Tüm sistem göz önüne alınır
  - ✖ Gereksinimler
  - ✖ İş süreçleri
  - ✖ Use-caseler
- Son aşamada yapılır
  - ✖ Genelde test uzmanları, 3rd party firmalar, ... Yapar
  - ✖ Herşey test edilir, yazılı olmayan kısımlar bulunabilir
    - Black Box (exploratory testing)
    - White Box (tabloların değerleri üzerine)
- Fonksiyonel + Fonksiyonel olmayan özellikler
  - ✖ Test ortamı kontrol edilmelidir.

Yazılım geliştirmede belli bir olgunluğa geldiğinde uygulanan test seviyedir. Bu seviyede herşey test edilir. Bu yüzden yazılı olan gereksinimler dışında kişilerin deneyimleride önemlidir ve bu testler genelde test uzmanları veya bu konuda uzmanlaşmış test danışmaları şeklinde yapılması önerilmektedir. Bu aşamada kara kutu testleri kadar beyaz kutu testleride yapmak mümkündür. Örneğin kullanıcıya sunulan bir tablodaki değerlerle üretilen bir liste de değerlerin doğru geldiğini ya da hatalı istek durumunda sistemin verdiği hata ve uyarı mesajlarının doğruluğu yine bu test seviyesinde test edilir.

Test edilen yazılımın versiyonuda yine test eden grup tarafından kontrol edilmeli bu yüzden test ortamının kontrolü test uzmanlarında olmalıdır.

1	2	3
4	5	6

## 2.1. Test Seviyesi

- **Kabul Testleri**

- Kullanıcı Kabul Testleri
- Operasyonel Kabul Testleri
- Uyumluluk Testleri
  - Kanunlara Uyumluluk
  - Tarayıcı Uyumluluk
- Alfa Testi
- Beta Testi

### **Kullanıcı Kabul Testleri**

Kullanıcılarından toplanan gereksinimlere uygunluk test edilir (verification). Kullanıcıların her istediği doğru olmayı bilir. Bu gibi durumlar analiz edildiyse analiz sonuçları göz önüne alınarak yapılan testler de (validation) bu grubta yer almaktadır.

### **Operasyonel Kabul Testleri**

Yazılımın operasyonel diğer grublarla olan çalışmaları test edilir. Herhangi bir backup/restore durumunda yazılımın kararlı sürümde dönme hızı ve bu süreç boyunca gösterdiği kararsız durumlar test edilir. Ayrıca bakım ve güvenlik kontrollerinde karşılaşılabilen zorluklar test edilir.

### **Uyumluluk Testleri**

İki grupta incelenebilir. Yazılımın kullanıldığı ülkenin kurallarına olan uyumluk ve diğer kişisel haklara zarar verebilecek özellikler test edilir. Tarayıcı uyumluluk ise, web uygulamalarında geçerli testlerdir. Kullanıcıların kullandığı tarayıcılardan desteklenenlerde oluşabilecek görsel, fonksiyonel, güvenlik ve performans gibi özelliklerin test edilmesi tarayıcı uyumluluk testleri içerisinde yer almaktadır.

### **Alfa Testi**

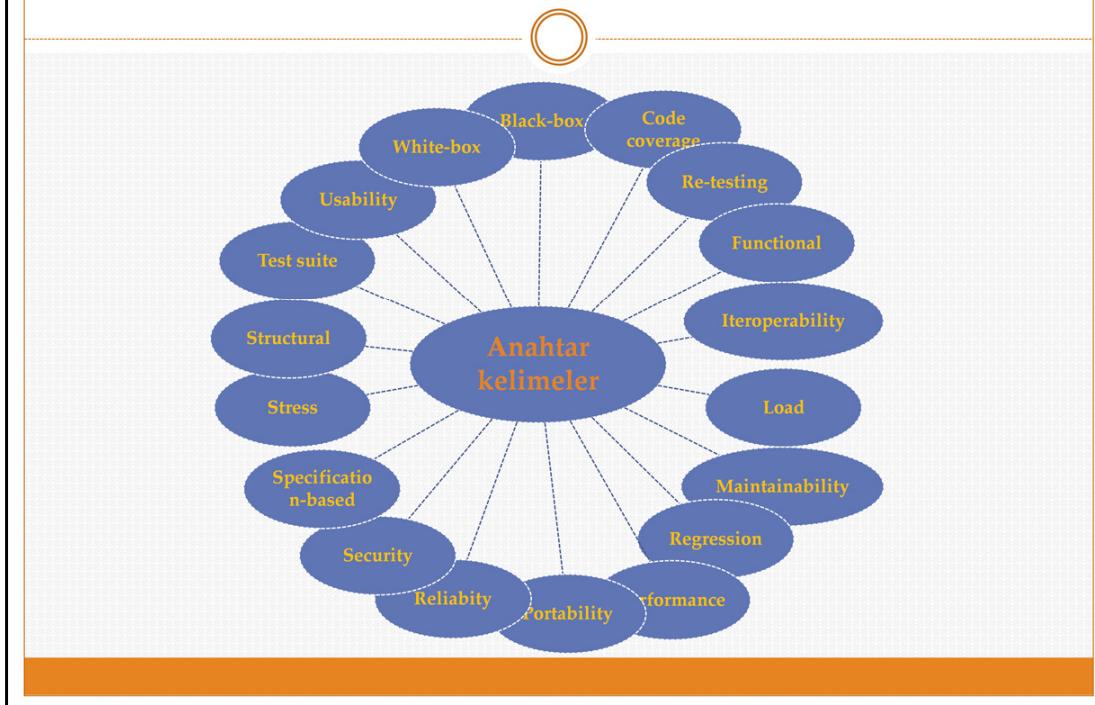
Geliştirmenin sonlanması yakın geliştirme ortamında yapılan testlerdir.

### **Beta Testi**

Yazılımın belli bir olgunluğa geldikten sonra sadece hedef kullanıcıları ile geliştirme ortamı dışında kullanıcının kendi ortamında yaptığı testlerdir.

1	2	3
4	5	6

## 2.3. Test Türleri



### Anahtar Kelimeler:

**Black-box:** Kara kutu testleri

**Code coverage:** Yazılan kodların ne kadarının çalıştırıldığını ve ne kadarının çalıştırılmadığını kontrol eden testler bütünü

**Re-testing:** Bulunan hata sonrasında yapılan değişikliğin test edilmesi

**Functional Testing:** Fonksiyonel testler

**Interoperability Testing:** Birim veya sistemlerin birlikte çalışabilmelerinin testi

**Load Testing:** Yük testi, belli bir yükte(yogunlukta) yazılımın verdiği cevaplar

**Maintainability Testing:** Bakım yapılmasına ne kadar uygun olduğu, güncellemenin maliyeti

**Regression Testing:** Regresyon testleri

**Performance Testing:** Performans testi

**Portability Testing:** Taşınabilirlik testleri

**Reliability Testing:** Tutarlılık testleri

**Security Testing:** Güvelik testleri

**Specification-based Testing:** Gereksinimlerden yola çıkılarak test etmek

**Stress Testing:** Yük testi, sınır değerleri bulmak için yapılan yıpratıcı testler

**Structural Testing:** Beyaz kutu testleri

**Test suite:** Test yapılması için gerekli ortam

**Usability Testing:** Kullanılabilirlik testi

**White-box Testing:** Beyaz kutu testleri

1	2	3
4	5	6

## 2.3. Test Türleri

- Fonksiyonel Testler
  - Black-box
    - Gereksinim temelli testler
    - İş-süreç temelli testler
- Fonksiyonel Olmayan Testler
  - Performance, stress, load
  - Usability
  - Maintainability
  - Reliability
  - Portability
- Yapısal Testler (Structural Testing)
  - White-box
- Regresyon Testleri

1	2	3
4	5	6

## 2.3. Test Türleri



- Fonksiyonel Testler

- Fonksiyonlar test edilir
- “(if...what) ne yapar?” sorusuna cevap verir
- Genelde Black-box testlerdir
- İki farklı boyutta incelenebilir
  - ✖ Gereksinim temelli testler
  - ✖ İş-süreç temelli testler

Yazılımın fonksiyonları test edilir. Yazılım kararlaştırıldığı gibi fonksiyonlarını yerine getirip getirmede bakılır. Fonksiyonların doğruluğu iki şekilde kontrol edilebilir. Birincisi kullanıcılarından toplanan gereksinimlere uygunluğu ve ikinci olarak da iş süreç modeline bakarak doğru bir süreç işletildiği kontrol edilebilir. Yani yazılımın "... Durumunda ne yapar?" sorusuna cevap veren testlerdir.

Fonksiyonel testler genelde kara-kutu testler olarak adlandırılır fakat bunla birlikte beyaz-kutu testlerde olabilir. Yazılımın iç yapısı genelde görülmez, yazılıma verilen girdilerin çıktıları kontrol edilir. Fakat bazı durumlarda örneğin çok fazla girdinin manuel olarak işletilemediği durumlarda (örneğin performans testlerinde) scriptler yardımıyla yapılan fonksiyonel testlerde olabilir, böyle durumlar ise beyaz-kutu testlere örnektir.

1	2	3
4	5	6

## 2.3. Test Türleri

- Fonksiyonel Olmayan Testler

- "(if...how) nasıl yapar?" sorusuna cevap verir
- Genelde White-box testlerdir
- En önemli türleri
  - Performance, stress, load
  - Usability
  - Maintainability
  - Reliability
  - Portability
  - Security
  - Compatibility

Yazılımın fonksiyonel olmayan diğer kalite unsurları test edilir. Bu kriterler; güvenlik, hız, tutarlılık, bakım yapılabılırliği, kullanılabilirlik, taşınabilirlik, ... şeklinde sıralamak mümkündür.

- **Performance, stress, load:** Performans ölçümlenir. **Performance** testinde istenilen performans değerleri içerisinde kullanılabilecek maksimum kullanıcı testpit edilir, **load** testinde ise belirli bir yükte yazılım verdiği cevap süresi; **stress** testinde ise yazılımın dayanabileceği en büyük yük tespit edilir.
- **Usability:** Yazılımın kullanıcı deneyimleri bakımından doğru olmayan yanları, kullanımı zorlaştıran özellikleri tespit edilir. Genelde kullanılabilirlik test uzmanları tarafından test edilir.
- **Maintainability:** Bakım sırasında yazılımın ne kadar sorun yaratacağı test edilir.
- **Reliability:** farklı ortamlarda ve koşullarda yazılımın verdiği cevaplar test edilir.
- **Portability:** Yazılımın farklı bir platforma taşınması durumunda ne gibi sorunlar oluşturacağı tespit edilir.
- **Security:** Güvenlik açıkları olup olmadığı, kullanıcının yaşamak istemeyeceği durumlar tespit edilir.
- **Compatibility:** Kullanıcının ortamları göz önüne alınarak farklı kullanıcı ortamlarında oluşabilecek hatalar tespit edilir.

Genelde beyaz-kutu testlerdir yani kod, script, test araçları ile yapılabilen testler olmakla birlikte performance, kullanılabilirlik tarafından yazılımın fonksiyonlarının fonksiyonel olmayan gereksinimlere uygunluğu test edilir. Yani kara-kutu testlerde bu bölümde olabilir.

1	2	3
4	5	6

## 2.3. Test Türleri



- Yapısal Testler (Structural Testing)
  - Yapı, mimari test edilir
  - White-box ağırlıklı testlerdir
    - Code coverage
    - Statement coverage
    - Decision coverage
    - Path coverage

1	2	3
4	5	6

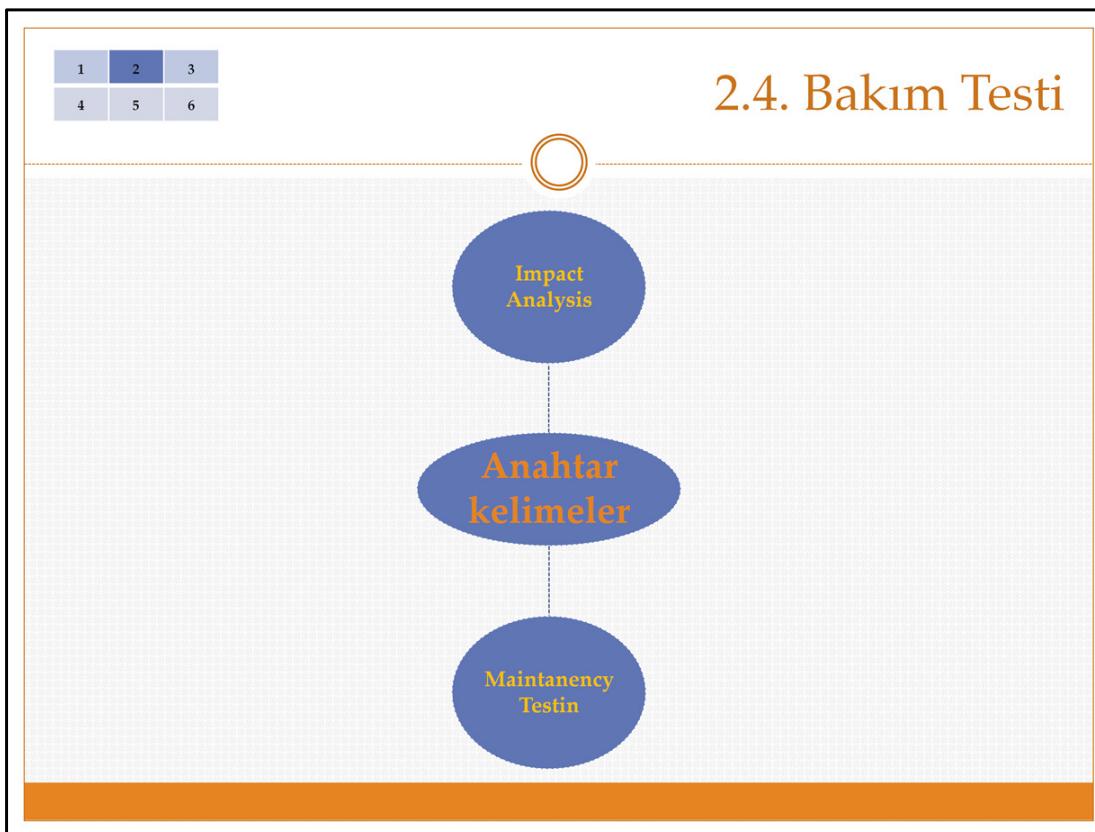
## 2.3. Test Türleri

- Regresyon ve Onaylama Testleri (regression testing, confirmation testing)

- Değişikliğe bağlı testlerdir
- Onaylama testi (re-test, confirmation test)
  - Saptanmış hatanın kontrolü
  - Yerel testtir
- Regresyon Testleri
  - İlgili bölümler test edilir
  - Genel testlerdir
  - Sistemde oluşabilecek yan etkiler saptanır
  - Otomasyona uygundur
  - Yazılımda her güncelleme sonrası yapılmalıdır

Regresyon ve onaylama testleri bir arada anlatılır çünkü her iki test türüde yazılımda yapılan değişiklik sonrası yapılması gereken test aktiviteleridir. Onaylama (confirmation) testleri, en son bulunan hatanın düzeltilmesi sonrasında hatanın var olup olmadığını kontrolüdür. Bu yüzden hatalı durum ne ise aynı durum tekrar edilir, yazılım yerel olarak test edilmiş olur.

Regresyon (regression) testleri ise yazılımda üzerinde herhangi bir değişiklik var ise bunun etkilileri göz önüne alınarak olumsuz etkilenebilecek tüm alt sistemler test edilir. Yani yerel testler ziyade daha genel testlerdir. Regresyona dahil edilecek modül veya fonksiyonlar tamamen değişiklik yapılan konuya özeldir. Zaman sınırlaması, risk değerlendirmesi ve etki analizi sonrasında kapsam belirlenebilir. Sık-sık yapıldığından otomasyona uygundur. Farklı kapsamı olan otomasyon test scriptleri ile regresyon testleri yapılabilir.



### Anahtar Kelimeler:

**Impact Analysis:** Etki analizi, yapılan güncellemenin sistem üzerine etkileri nasıl anlaşılır

**Maintanence Testing:** Bakım testleri, yazılım devreye alındıktan sonra bakım gerektirdiği durumlarda güncelleme sonrasında yapılan testlerdir.

1	2	3
4	5	6

## 2.4. Bakım Testi

- Neden Gerekli?
  - Kalitenin devamlılığı
  - Kullanıcı istekleri
  - Kullanıcı ortamındaki değişiklikler
  - Değişen yasalar
  - Performance, güvenlik, uyumluluk, ... eksikleri
- Bakıma uygunluk testi değildir
  - Maintainability testi değildir.

### Bakım testleri neden gereklidir?

Bir yazılım devreye alındıktan sonra bazen yıllarca değişiklik yapılmadan kullanılabilir.

Fakat günümüzde bir çok yazılım çeşitli sebeplerden dolayı sık sık güncelleme gerektirebilir. Bu sebeplerden bazıları:

- Değişen işletim sistemleri, kullanıcı ortamlarında olabilecek hertürlü değişiklikler
- Kalitenin devamlılığı için saptanan iyileştirmeler yapılabilir,
- Performans kayıpları, güvenlik açıkları, yeni teknolojilere uyumluluk problemleri,
- Kişilerin istekleri değişmiş olabilir,
- Yasalardaki değişiklikler yazılımın değişmesine neden olabilir. Yeni özellikler eklenebileceği gibi var olan özelliklerin kaldırılmasına veya değiştirilmesine sebep olabilir.

Bakım testi gerekli hallerde ve bilinen periodlarda olabilir. Test edilen işlevler her durumda aynı olmayabilir fakat genel olarak sistemin tamamı da test edilebilir. Bakım testi yazılımın bakıma uygun olup olmadığına testi değil yazılımın işlevlerini normal şekilde devam ettirip etirmedeğinin testidir.

1	2	3
4	5	6

## 2.4. Bakım Testi

- Planlı Yapılmalıdır
  - Herkes bilmeli
  - Hazırlık yapılmalı
- Etki Analizi Yapılmalıdır
  - Değişikliğin etkisi bilinmeli
  - Nereler ne kadar test dilecek
- Gereksinimler belirlenmeli
  - Test sonuçları karşılaştırılmalı
  - Hatalı durumlar raporlanmalı
  - Gereksinim listesi güncellenmeli

Bakım testleri planlı yapılmalıdır çünkü yazılımı kullanan tüm kullanıcıları ilgilendiren durumlar olabilir. Bakım sırasında yazılımın hizmetinde kısıtlamalar olabilir veya test için kullanılacak özel objeceler olabilir. Bu durumdan kullanıcıların haberdar olmali ve planlama yapmalıdır. Diğer yandan etki analizi yapılarak değişikliğin etkisi araştırılmalı ve bakım testinin içeriği belirlenmelidir. Plan dahilinde bakım testine hazırlık yapmak gerekebilir.

Bakım testi sırasında test sonuçlarını karşılaştırmak için mutlaka bazı dökümanlara gerek vardır. Gereksinim, uzman görüşü, mevcut doğru çalışan sistemler test girdisi olarak alınabilir. Diğer yandan bu gereksinimler olmaksızın yapılan testler sadece **doğrulama** olur **onaylama** olmaz.

“ Without a specification, you cannot really test, only explore. You can validate, but not verify.”

Kaynak: <http://www.slideshare.net/onsoftwaretest/istqb-iseb-lecture-notes-2-presentation>

# ISTQB Müfredatı

- Testin Temelleri
- Geliştirme Boyunca Test
- **Statik Test Teknikleri**
- Test Tasarım Teknikleri
- Test Yönetimi
- Test Araçları



1	2	3
4	5	6

### 3. Statik Test Teknikleri



-  Gözden Geçirme ve Test Süreçleri
-  Gözden Geçirme Süreçleri
-  Araçlarla Statik Analiz

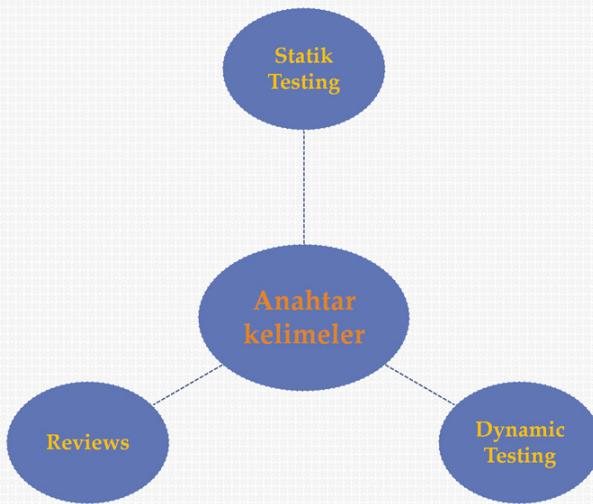
#### Statik Test Tekniği

Yazılım geliştirme boyunca kod seviyesine inen ve geliştirme yapan mühendislere destek vererek hataları önceden kestirmelerine yardımcı olabilen yazılım destek teknikleridir. Statik test teknikleri uygunlandığında herhangi bir kod çalıştırması olmaz ve yazılı olan her şey gözden geçirme sürecine dahil edilerek çıktıları üzerindeki hatalar önceden kestirilir.

Bazı statik analiz yapan araçlar ile yazılan kodun kalitesi ortaya konabilir. Bu araçlar genel anlamda yazılan kod satırı sayısını, kodun ne kadar karmaşık yapıda olduğunu, ne kadar sınıf, nesne, fonksiyonun var olduğu ve çalışmasına olanak olmayan (un-reachable code) kod parçalarını araştırmaya yarar. Statik test teknikleri ile oluşabilecek hataları önceden kestirebiliriz fakat hiçbir zaman dinamik test (**dynamic testing methods**) tekniklerinin yerine geçmez.

1	2	3
4	5	6

### 3.1. Gözden Geçirme ve Test Süreçleri



#### Anahtar kelimeler

- **Statik Testing (Statik Testler):** Yazılımın kodu çalıştırılmadan, yazılımın kalitesiyle ilgili detaylarının tartışıldığı genelde kod seviyesinde olan ve yazılımcının kendi hatalarını bulması şeklinde gerçekleşen yazılım test aktiviteleridir. Genelde düşünce hatası(error, mistake) bulunur.
- **Dynamic Testing (Dinamik Testler):** Yazılımın çalıştırılarak yapılan testlerdir. Genelde kod belli bir olgunluk seviyesine gelir ve versiyon (sürüm) numarası verilerek dinamik testlere başlanır. Giriş-Çıklılar kontrol edilir yanı yazılıma girilen her bir aktivitede yazılımın verdiği cevapların doğruluğu kontrol edilir. Genelde hataları (failure) bulunur.
- **Reviews (Gözden Geçirme):** Statik test teknikleri içerisinde yer almaktadır. Kod, döküman, gereksinimler daha uzman kişilerce gözden geçirilerek daha sonra ortaya çıkabilecek hatalar ayıklanmış olur.

1	2	3
4	5	6

## 3.1. Gözden Geçirme ve Test Süreçleri



- Bireysel Gözden Geçirmeler Yöntemleri
  - Desk checking
  - Proof reading
- Grup Gözden Geçirme Yöntemleri
  - Reviews
    - Formal
    - Informal
  - Walkthrough
  - Inspection

Bireysel olarak yapılan gözden geçirme yöntemleri yazılımcının kendisinin yaptığı test yöntemleridir. Bu yöntemlerle grup gözden geçirmeleri öncesinde kendi hatalarını azaltarak daha olgun bir bir kod ortaya çıkartır. Birey aktiviteler aşağıdaki isimlerle anılır:lar:

- **Desk Checking (masada kontrol):** Yazılan kodun çıktısı alınır ve yazılımcı masa başında bilgisayardan yardım almadan satır satır kod üzerinden geçerek olayı simule etmiş olur. Yazılımcı mantık sorguları kontrol eder ve hataları kendi bulur.
- **Proof Reading (uzman deneyimi):** Daha önceden yapılmış ve sorunsuz çalışan uzmanlar tarafından hazırlanmış kod, algoritma vb., dökümanları okuyarak kendi çıktılarını test edilir.

Grup olarak yapılan gözden geçirmler ise konuda uzmanlaşmış kişiler tarafından yapılır. Burada amaç sistematik bir şekilde ön yargılardan uzaklaşarak başka kişilerin yakayabileceği hata veya hataya sebep verebilecek zayıf yapıları ortadan kaldırmak amaçlanır. Grup gözden geçirmeleri aşağıdaki methodlardan oluşur:

- **Reviews (Gözden geçirme):** Formal ya da informal yapılabilir. Bireysel veya grup olarak yapılabilir fakat genelde grup yapılır. Formal yapılan gözden geçirmeler hazırlıklı ve zamanlaması önceden planlanmış olur. Uzmanlar konuyu değerlendirerek kendi fikirlerini ortaya koyar ve bir ortak karar çıkması durumunda uygulamasına geçilir. Informal olarak yapılan gözden geçirmeler genelde hazırlıksız ve kısa görüşmeler şeklinde gerçekleşir.
- **Walkthrough (üzerinden geçme):** Planlı bir şekilde ve öncesinde katılımcıların hazırlanarak katıldığı görüşmelerdir. Walkthrough'da dökümanın veya kodun sahibi toplantıda çıktılarının üzerinden satır satır geçer.
- **Inspection (inceleme):** En formal olan gözden geçirme teknigidir. Planlı, dökümante ve araçlar yardımı ile yapılır. Gözden geçirme sırasında saptanan hatalar raporlanır ve düzeltilmesi için yazılımcıya haber verilir. Bu sayede inspector (inceleyeciler) ile kodu yazan kişi ortak kaniya vardıklarında süreç tamamlanmış olur ve onaylanan kod, döküman bir sonraki süreçte geçer.

1	2	3
4	5	6

## 3.1. Gözden Geçirme ve Test Süreçleri



- Neler Gözden Geçirmeye Dahil Edilir

- Prensipler, strateji, iş planları, pazarlama veya reklam dökümanları, sözleşmeler
- Sistem gereksinimleri, fizibilite çalışması,
- Kabul test planı, test dizaynı, test caseler, test sonuçları
- Sistem dizaynı, kodlar, mantık yapıları
- Kullanıcı ve eğitim dökümanı
- ... Yazılım sürecindeki tüm çıktılar

Gözden geçirme işlemleri genel anlamda yazılımcının ürettiği kod parçalarının hedef almakla birlikte SDLC'de her süreçte üretilen çıktılar gözden geçirmeye dahil edilebilir. Böylelikle konudan ve detaydan bağımsız olarak süreçteki tüm çıktıların kalitesi artar ve test süreci boyunca yakalanan hata ve etkileri enaza indirgenmiş olur.

1	2	3
4	5	6

## 3.1. Gözden Geçirme ve Test Süreçleri



### • Gözden Geçirmenin Faydaları

- Daha az defect ve yazılım maliyeti
- Yazılım geliştirme faliyetlerinde verim artar
- Yazılım geliştirme süresini kısaltır
- Hata seviyesini düşürür, daha az ciddi hatalar
- Müşteri ilişkilerini geliştirir
- Deneyim paylaşımı artar
- Standartlara uyum artar

Gözden geçirme faliyetlerinin sağladığı faydalardır; özellikle gereksinimlerin gözden geçirilmesinde saptanan tutarsızlıklar yazılım maliyeti üzerine önemli etkileri vardır. Erken evrelerde yakalanan hataların düzeltilememesi daha sonraki süreçler ne kadar iyi olursa olsun ürettiği çıktılar hatalı olacaktır. Bu durumda yazılım ürünü ortaya çıktığında bu tür hataların düzeltilmesi ancak hatanın başladığı süreçteki tutarsızlığın bulunması ve ordan sonraki tüm süreçlerde bulunan hatanın düzeltilmesiyle giderilecektir.

Gözden geçirme ile daha fazla bilgi paylaşımı ve gruplar arasındaki ahenk artar. Müşterinin istekleri daha net anlaşılır ve tutarsızlıklar giderilebilir. Bu sayede müşteri memnuniyeti artar.

1	2	3
4	5	6

## 3.2. Gözden Geçirme Süreçleri



### Anahtar Kelimeler:

- **Entry criteria (Teste başlama kriteri):** Geliştirmenin belli bir olgunluga erişip (smoke testten geçebilen) test alınması için gerekli kriterlerdir.
- **Exit Criteria (Testi sonlandırma kriterleri):** Yeterli miktarda test yapılmışlığının anlaşılması için konulan kriterlere testi sonlandırma kriterleri denir. Risk yönetimi yapmak daha sağlıklı sonuçlar verir.
- **Formal Review (Formal gözden geçirme):** Planlı ve hazırlanarak yapılan ve süreçleri olan gözden geçirmelerdir.
- **Informal Review (Formal olmayan gözden geçirmeler):** Hızlıca ve plansız bir şekilde sonuca gitmek adına uzman kişilerle yapılan ve genelde dökümantasyon edilmeyen toplantılardır.
- **Inspection (İnceleme):** Kuralları olan ve bulguların raporlandığı ve bir lider tarafından yönetilen bireysel veya grubca yapılan gözden geçirme toplantılarıdır.
- **Moderator (Moderatör):** Toplantıları yöneten kişiler
- **Reviewer (İneleyici):** Kod, döküman, vb. inceleyen uzmanlaşmış kişiler
- **Scribe (Yönetici):** Planlamayı yapar, yönetir, iletişim kurar.
- **Technical Review (Teknik gözden geçirmeler):** Kod, sistem gereksinimleri, database tasarımları gibi teknik konuların tartışıldığı toplantılardır. Teknik gözden geçirmelerin sorunu çözmek, alternatif çözümler üzerine tartışmak ve en etkティブine çözüme karar vermek, kurallara uygunluğu denetlemek gibi amaçları olabilir.
- **Walkthrough:** Üstünden geçmek. Yazarın neler yaptığı ilgili kişilere aktarır, yazıcı (scribe) katılabilir, ve amacı öğretmek, hata bulmak, katılımcı yorumu almak şeklinde sıralanabilir.

1	2	3
4	5	6

## 3.2. Gözden Geçirme Süreçleri



- Gözden Geçirme Aşamaları (Formal Reviews)
  - Planning
  - Overview / Kick-off meeting
  - Preparation / individual checking
  - Review meeting
  - Rework
  - Follow-up

Bu süreçler formal gözden geçirmeler için geçerlidir. Formal olmayan gözden geçirmeler genelde planlı ve hazırlıklı olmadığı için herhangi süreçten bahsetmek zordur. Formal gözden geçirmeler ise:

**Plan Yapma → Bilgilendirme → Hazırlanma → Toplantılar → Düzeltme Çalışmaları → Takip Etme**

şeklinde sıralanabilir. Takip etme süreci saptanan hata ve eksiklerin giderilmesinin kontrol edilmesi şeklindeki ve her zaman olmayabilir.

1	2	3
4	5	6

## 3.2. Gözden Geçirme Süreçleri



- **Roller**

- Leader/ Moderator
- Author
- Reviewers/Inspectors
- Managers
- Scribe

### Roller

- **Leader/ Moderator (Lider/Moderatör):** Toplantıları yöneten kişiler
- **Author (Yazar):** Kod, döküman, vb. sahibi
- **Reviewers/Inspectors (İneleyici):** Kod, döküman, vb. inceleyen uzmanlaşmış kişiler
- **Managers (Yönetici):** Planlamayı yapar, yönetir, iletişim kurar.
- **Scribe (Yazıcı):** Toplatı sırasındaki tüm bulguları, gereklilikleri, vb. Kayıteder.

1	2	3
4	5	6

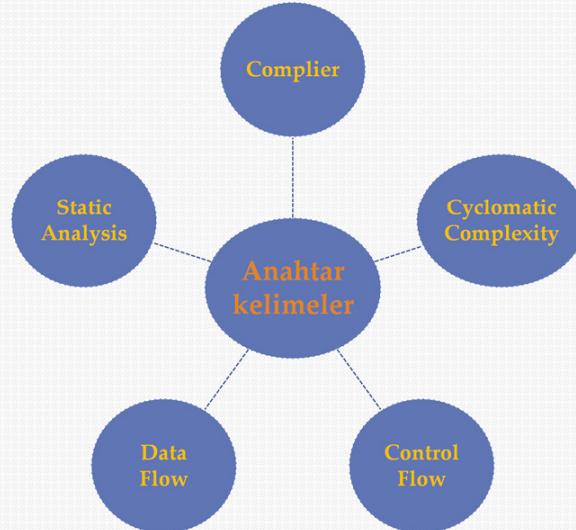
## 3.2. Gözden Geçirme Süreçleri



- Gözden Geçirmenin Başarı Kriterleri
  - Açık ve net hedefler olmalı
  - Doğru kişiler toplantılara katılmalı
  - Testcilerin erken test etmesi için ürünü öğrenmesine olanak sağlamak
  - Bulunan hatalar hoş karşılanmalı, önyargı olmamalı
  - Psikolojik sorunların üstesinden gelinmeli
  - Güvenli bir ortamda yapılmalı
  - Uygun teknikler kullanılmalı
  - Hata bulunması için etkili olabilecekler listesi olmalı
  - Eğitim verilmiş olmalı
  - Yönetim desteği olmalı

1	2	3
4	5	6

### 3.3. Araçlarla Statik Analiz



#### Anahtar Kelimeler:

- **Complier (Derleyici):** Yazılan kodların derlenmesi ve çalıştırılmasını sağlayan araçlardır.
- **Cyclomatic Complexity:** Yazılan kodun ne kadar karmaşık olduğunu gösteren bir parametre.
- **Control Flow (Akış şeması):** Yazılan kodun işletilmesi sırasında neleri yapacağını gösteren yol, akış şeması.
- **Data Flow (Veri Akışı):** Veri akışını (oluşturma – kullanma – yoketme) adımlarını soyut bir şekilde anlatan akış şemalarıdır.
- **Static Analysis (Statik analiz):** Yazılımı oluşturan parçaların (kod, analiz, gereksinim, vb.) çalıştırılmaksızın analiz edilmesidir.

1	2	3
4	5	6

### 3.3. Araçlarla Statik Analiz



- **Statik Analizin Önemi**

- Çalıştırmadan önce hataların bulunması
- Karmaşık kodların bulunması
- Dinamik teste bulunması zor hatalar
- Tutarlılıkların bulunması
- Tasarımın bakım açısından geliştirilmesi
- Hatalardan arınma

Statik analiz yapan araçlar programları çalıştırırlar, analiz yaparlar. Analiz yapılacak parçalar kod olabileceği gibi HTML veya XML gibi dosyalarda olabilir. Bu tür programlar akış şemasını ve data akışını çıkartabilir. Bu akışlar yardımıyla yazılan kodla ilgili (döngüler, satır sayısı, tekrar eden kod parçaları, vb.) bilgiler verir.

1	2	3
4	5	6

### 3.3. Araçlarla Statik Analiz



- Statik Analizle Saptanabilecek Hatalar
  - Değer atanmamış değişkenler
  - Modul ve birimler arasında tutarsız arayüzler
  - Kullanılmayan değişkenler
  - Tipi uygun olmayan değişkenler
  - Erişilmeyen ölü kod parçaları
  - Sonsuz döngü gibi hatalı yapılar
  - Standartların göz ardı edilmesi
  - Güvenlik açıkları
  - Syntax hataları

Statik analiz sonucunda bulunan hatalar yazılımın iç yapısıyla ilgilidir. Burada bulunan her bir bulgunun önemi, düzeltilemediği durumda yazılıma maliyeti çok büyük olur. Kullanılan dile göre değişiklikler gösterebilir. Bu tür araçları kullanmak beraberinde yazılım dilini iyi bilmeyi zorunlu hale getirir, bu yüzden genelde yazılım geliştiriciler tarafından kullanılır. Bulunan hataların anlamlandırılması gereklidir ve düzeltme yapıldıktan sonra aynı statik analiz teknikleri kullanılarak tekrar analiz yapılması gereklidir. Özellikle birim ve birim entegrasyon testi sırasındaki testlerde çok sayıda uyarı veya hata verebilir, bu uyarıların dikkatli bir şekilde yönetilmesi gereklidir.

# ISTQB Müfredatı

- Testin Temelleri
- Geliştirme Boyunca Test
- Statik Test Teknikleri
- **Test Tasarım Teknikleri**
- Test Yönetimi
- Test Araçları



1	2	3
4	5	6

## 4. Test Tasarım Teknikleri



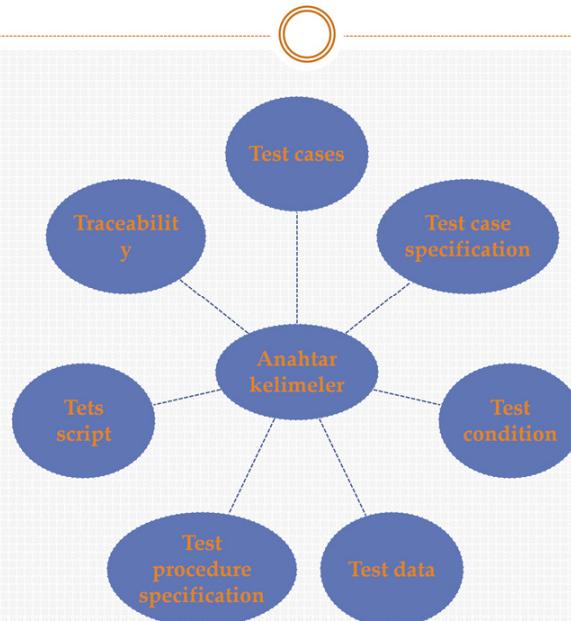
- Test Geliştirme Süreci
- Test Tasarım Kategorileri
- Kara Kutu Testler
- Beyaz Kutu Testler
- Deneyim Temelli Testler
- Uygun Test Tekniğinin Seçimi

### Test Tasarım Teknikleri

Statik test teknikleriyle kod veya döküman incelemesi yapılıyor ama bahis konusu olan kod çalıştırılmadan uygun tekniklerle hatalar saptanıyordu. Dinamik test teknikleriyle (dynamic test techniques) ise kod çalıştırılıyor ve test teknikleri çalışan koda uygulanıyor. Bu yüzden yazılımın belli bir olgunluğa erişmiş olması gereklidir. İteratif yazılım geliştirme süreci işletiliyor ise iterasyon içerisindeki işlerin tamamlanmış olması ideal dünyada istenilen durumdur.

1	2	3
4	5	6

## 4.1. Test Geliştirme Süreci



### Anahtar kelimeler

- **Test cases (Test Kase):** Bir bulguyu test etmek için gerekli tüm bilgileri (test datası, test adımları, test açıklaması, test başarılı sonucu, vb.) içerisinde barındıran test faliyetine genel olarak test case denir.
- **Test case specification (Test Kase Açıklaması):** Bir grub test case için (test set) hazırlanmış ve o test seti koşturmak için gerekli bilgileri içeren dökümandır.
- **Test condition (Test Koşulu):** Bir veya birden fazla test casenin koşturulması için gerekli koşullar.
- **Test data (Test Verisi):** Test caseleri koşturmak için kullanılan veriler (kullanıcı bilgileri, şifre, kupon, kod, numara, vb.)
- **Test procedure specification (Test Koşurma Adımları):** Test caseleri koşturmak için yapılması gerekenlerin (test steps) sıralı bir şekilde (adım-adım) verildiği döküman.
- **Tets script:** Test caseleri otomatik koşturmak için yazılmış kod parçaları.
- **Traceability:** Döküman veya program içerisinde ilgili parçaların izlenebilmesine (ilgili kod nerden çağrıldığı, nerelerde kullanıldığı, vb.) olanak veren birzelliktir.

1	2	3
4	5	6

## 4.1. Test Geliştirme Süreci



- Test dökümantasyonu
  - Değişkendir (formal – formal olmayan)
  - İdeal yapı nedir?
- Test analizi
  - ne, ne?
  - Nasıl?
  - Veriler
  - Şartlar
  - Sonuç
- Traceability (izlenebilirlik)
  - Hatanın bulunması
  - Yatay – Dikey

Testçilerin test yaparken kullandıkları döküman ve testleri sonucunda ortaya koydukları dökümantasyon formal ve formal olmayan firmalar arasında farklılıklar gösterebilir. Milyonlarca kişinin kullandığı bir web uygulaması ile belirli bir grub tarafından kullanılan bir grafik uygulamasının gereksinimleri veya kurumsal kültürü olan çok uluslu bir firma ile start-up firmasının iş kültürü arasındaki farklılıklardan dolayı her iki durumda farklı değerlendirmek gerekir. Genel manada işe yarayandan fazla döküman üretmek kimsenin faydasına olmadığı gibi dinamik yapıyada zarar verebilir.

Test yapabilmek için bazı bilgilerin hazır olması gereklidir. Gerekli tüm gereksinimlerin araştırılması işine **test analizi** denir. Gerekli bilgiler kısaca şunlardan oluşur:

- Neyin test edileceği
- Nasıl test edileceği
- Test etmek için gerekli veriler
- Testin başarılı olması için gerekli koşullar
- Testin sonuçlarını değerlendirmek için gerekli bilgiler

Herhangi bir özelliğin (fonksiyon, sınıf, vb.) değişmesine mükatip yazılım içerisinde etkilenen kısımların izlenmesi ve etkilenen test caselerin yeni duruma göre güncellenmesi gereklidir. Eğer bir yazılımda bu özellik kolay anlaşılır değil ise güncellemeler karşısında test caseleri düzenlemek zorlaşır.

1	2	3
4	5	6

## 4.1. Test Geliştirme Süreci



- **Test Tasarımı**

- Hatalı durumun belirlenmesi
- Koşullar
- Test “oracle”
- İstenilen (expected result) – sonuç (result)

- **Testin Tatbiki (implementation)**

- Gruplama
- Sıralama

Test edebilmek için test tasarımlının yapılması gereklidir (test case hazırlayabilmek için). Test tasarımları için öncelikle tasarım sırasında gerekli bilgilerin toplanması ve bunlara bağlı kalarak tasarıma geçilmelidir. Test tasarımlında en önemli nokta hatalı durumun varlığını ortaya koyacak gerekli şartlar ve bu şartlar altında yapılması gerekenler ve daha sonra istenilen durumun ne olduğunu belirtmektedir. Hatalı durumdan bağımsız olarak doğru durumun bulunması için gerekli bilgiler **“test oracle”** denir.

Test caseleri hazırlanıktan sonra tatbikine (test implementation) geçilmeden önce test caseleri amaçları ve fonksiyonları göz önüne alarak yönetilebilir gruplar (**test set**) haline dönüştürülmelidir. Gruplanan test caseleri ise kendi içerisinde sıralamak, önceliklendirmek gerekmektedir. Örneğin satınalma yapılmadan sipariş iptali yapılamaz veya yama (**patch**) yazılıma aktarılmadan sonuc görülemez gibi mantık sıralaması yapılmalıdır.

1	2	3
4	5	6

## 4.2. Test Tasarım Kategorileri



### Anahtar kelimeler

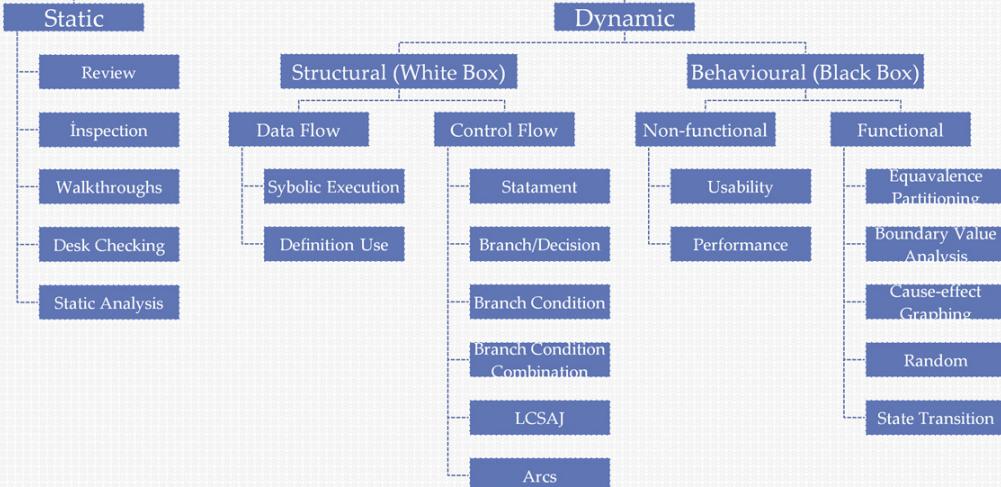
- **BB Test Design Tech. (Black Box – Kara Kutu Test Teknikleri):** Yazılımın fonksiyonları test edilir. Yazılımın iç yapısı (kod, database, aktarım, vb.) görülmemiş için kullanıcı gibi test test edilir, bu yüzden kara-kutu testler denir. Girişlere karşılık gelen çıkışlar kontrol edilir, hatalı çıkışlar saptanır.
- **Exp. Based Design Tech. (Experience Based – Deneyim Temelli Test Teknikleri):** Testcilerin deneyimlerinden yola çıkarak test edilir. Farklı alanlar için farklı test metodları kullanılır. Hataların öbekleşmesi (**defect clustering**) bu test teknliğinde önemli yeralır.
- **Spec. Based Design Tech. (Specification Based – Fonksiyon Temelli Test Teknikleri):** Kara kutu test teknikleridir.
- **Struc. Based Design Tech (Structural Based – Yapısal Test Teknikleri):** Beyaz kutu test teknikleridir.
- **WB Design Tech. (White Box – Beyaz Kutu Test Teknikleri):** Yazılımın iç dinamikleri (algoritmalar, mantık yapıları, database sorguları, mimari yapısı, vb.) göz önüne alınarak analiz çıktıları test edilir.

1	2	3
4	5	6

## 4.2. Test Tasarım Kategorileri



### Test Design Techniques



1	2	3
4	5	6

## 4.2. Test Tasarım Kategorileri

- Farklı Test Teknikleri Nerede Kullanılabilir
  - Kara Kutu Testleri
    - Her Seviyede
    - Fonksiyonlar Tanımlı Olmalı
  - Beyaz Kutu Testleri
    - Geliştirici
      - Birim
      - Birim Entegrasyon
    - Code Coverage
  - Deneyim Temelli Testler

### Test Tekniklerinin Kullanımı:

Kara Kutu testleri (black box) yazılımın her seviyesinde kullanılabilir fakat fonksiyonların ortaya çıkış olması gereklidir. Kullanıcı kabul testleri veya sistem testlerinde gereksinimler baz alınabilir, birim veya entegrasyon testlerinde ise alt seviye analiz dökümanları veya tasarım dökümanları baz alınabilir.

Beyaz Kutu testleri (yapısal testler) yazılımın iç yapısıyla ilgili olduğu için geliştirici birim testlerinde veya birimlerin entegrasyonu sırasında yapısal testleri kullanabilir. Araçlar yardımıyla birlikte kod kapsamının (code coverage) testi içinde kullanılabilir.

Deneyim temelli testler ise beyazkutu ve karakutu testleri tamamlayıcı olarak görev yapar. Tanımlı fonksiyon ve gereksinimlerin olmadığı (analiz eksikliğinde) kullanılabilir. Kişilerin deneyimlerinden yola çıkararak test edildiği için bazı önemli eksikler veya projeye özel maddeler göz ardı edilebilir. Genel olarak düşük riski veya zaman baskısının fazla olduğu durumlarda kullanılabilir.

1	2	3
4	5	6

### 4.3. Kara Kutu Test Teknikleri



#### Anahtar Kelimeler

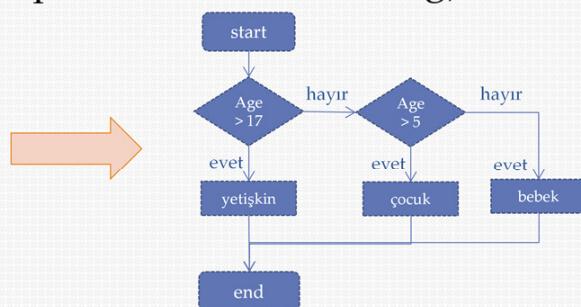
- **Boundary Value Analysis (Sınır Değer Analizi):** Sınır değerler kontrol edilir. Örneğin: gereksinimler içerisinde 18 yaş sınırı var ise yazılımın 17, 18 ve 19 girdilerine karşı ürettiği çıktılar test edilir.
- **Decision Table Testing (Karar Tablosu Testi):** Eğer kontrol edilecek çok sayıda durum var ise her bir durumu tablo içerisinde alınır ve herbir duruma karşı yazılımın verceği cevaplar yine tabloya yerleştirilerek karar tablosu oluşturulur.
- **Equivalence Partitioning (Eşdeğer Aralık Test):** Aynı sonuçlar üreten girdiler grüplanarak sadece farklı sonuçlar üreten giriş değerleri grublanmış sınırlandırılmış olur. Test için her bir gruptan bir giriş değerini kullanmak yeterli olacaktır. Örneğin: gereksinimler içerisinde 18 yaş sınırı var ise yazılımın 18 yaşına kadar olanlar (1, ...17) için **hayır**, 18den sonrası için (18, ...) **evet** üretecekse 5 ve 25 burada kullanılabilir.
- **State Transition Testing (Durum Geçiş Testi):** Yazılımın durum geçisi gösterdiği yerlerin testleridir.
- **Use Case Testing (Fayda Analizi Test):** Analiz çıktılarının testidir.

1	2	3
4	5	6

#### 4.3. Kara Kutu Test Teknikleri



```
if age > 17  
    print "yetişkin"  
  
if age > 5  
    print "çocuk"  
  
else  
    print "bebek"
```



## Equivalence Partitioning

Eşdeğer aralık yöntemi, aynı bölgedeki değerler girdi olarak kullanıldığında aynı sonucu verir ön koşulundan çalışmaktadır. Örnekte verilen kod parçasında 5 yaş ve daha küçükler **bebek**, 6-17 yaş aralığında bulunanlar **çocuk** ve 18 yaş üstündekiler **yetişkin** olarak değerlendiriliyor. Böyle bir durumda eş değer (aynı sonucu veren) bölgeleri sayı doğrusu üzerinde görebiliriz. Bu yöntemle test etmek istediğimizde kullanabilceğimiz giriş değerleri sırasıyla **(3, 12, 25)** veya kurala uygun başka değerler olabilir.

Burada dikkat edilmesi gereken bir başka konu ise giriş değerlerini 0 ile +sonsuz arasında sınırlandığını düşündük fakat gerçek hayatı kullanıcılardan her türlü giriş değerleri girebileceği unutulmamalıdır. Fakat bu yöntemde bu göz ardı edilir ve yazılımın bu kontrolleri yaptığı varsayılar.

1	2	3
4	5	6

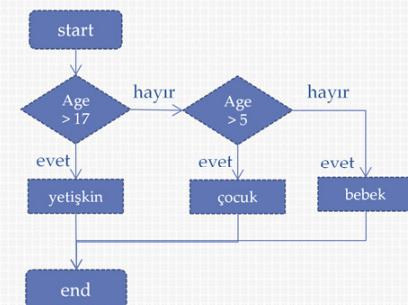
### 4.3. Kara Kutu Test Teknikleri

- Sınır Değer Testleri (Boundary Values)

```

if age > 17
    print "yetişkin"
if age > 5
    print "çocuk"
else
    print "bebek"

```



#### Sınır Değer Testleri

Sınır değer testleri değişim noktalarında yazılımın kararlılığını test etmek üzere yapılır. Bu yuzden eşdeğer aralıkların çıkartılması ve en büyük ve en küçük değişim noktalarının saptanması gereklidir. 2-değerli ve 3-değerli sınır değer testi yaklaşımı vardır. 2-değerli yaklaşımda değişim noktası ve bir sonraki değer alınırken 3-değerli yaklaşımda değişim noktası, bir önceki ve bir sonraki değerleri de içeresine alan üç farklı değer test edilir.

Buradaki örneginizde 17 yaş değişim noktasıdır yani 17 girdisiyle «çocuk» ve 18 girdisiyle «yetişkin» çıktıları alıcaktır; aynı şekilde 5 değeride değişim noktasıdır ve 5 ile «bebek», 6 ile «çocuk» çıktıları alıcaktır. Bu durumda 2-değer yaklaşımı ile **(5, 6; 17, 18)** sınır değerleri test için kullanmak gereklidir. 3-değer yaklaşımında ise değişim noktalarının bir önceki değerleri hesaba katılır yani sınır değer girdileri **(4, 5, 6; 16, 17, 18)** olarak kullanılması gereklidir.

1	2	3
4	5	6

### 4.3. Kara Kutu Test Teknikleri



- Karar Tablosu Testleri (Decision Table Tesing)

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
New customer (15%)	T	T	T	T	F	F	F	F
Loyalty card (10%)	T	T	F	F	T	T	F	F
Coupon (20%)	T	F	T	F	T	F	T	F
<b>Actions</b>								
Discount (%)	X	X	20	15	30	10	20	0

- Test edilmesi gereklili durum sayısı:

$$2^3 = 8$$

#### Karar Tablosu Testleri

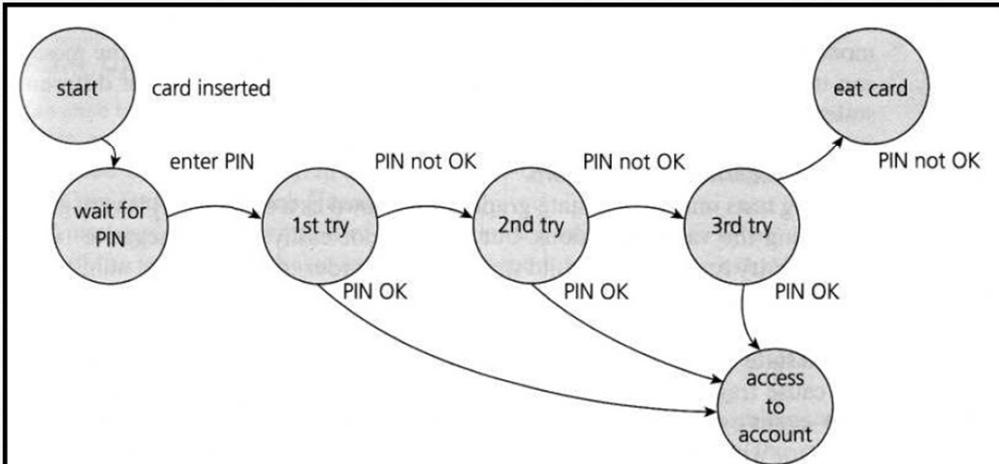
İş kuralları veya gereksinimlerden test edilecek koşulların sayısı fazla ise mevcut durum bir tablo üzerine aktarılara karar tablosu oluşturulabilir. Şekilde görüldüğü gibi her bir durum kural olarak tanımlanmıştır. Burada her koşul sadece doğru (**T**) veya yanlış (**F**) değerleri aldığı göz önüne alınmıştır dolayısıyla 3 şartın olası kombinasyonu ( $2^3 = 8$ ) ile test edilebilcek durum sayısı 8 adettir. Bu durum daha karmaşık sistemlerde daha büyük rakamlara ulaşabilir, böyle durumlarda olası durumları gruplamak ve her gruptan bir test set oluşturmak mantıklı olacaktır veya risk analizi yapmak test set sayısını azaltmak faydalı olabilir.

Bu tabloda yazılımın vermesi gereken sonuçlar Actions kısmında gösterilmiştir.

1	2	3
4	5	6

### 4.3. Kara Kutu Test Teknikleri

- Durum Geçiş Testi (State Transition Testing)



#### Durum Geçiş Testi

Geçiş durumlarda ortaya çıkabilecek sorunların kontrol edilmesidir. Şekilde görüldüğü gibi bir atmde olası durumlar ortaya konmuştur. Kullanıcının kartını atmeye yerlestiresiyle başlayan süreç sınırlı durumlar arasında geçiş yaparak kullanıcının isteklerini yerine getirmesini sağlamaktadır. Örneğin, kişinin hesap bilgilerine ulaşması için öncelikle şifresini doğru girmesi ve hesap bilgilerine tıklaması gerekmektedir. Eğer kişi hesap bilgilerini 3kez art arda doğru girmesse kullanıcının kartını atm el koymaktadır.

Başka bir durum ise hesabına erişerek hesabından para çekmesidir. Hesabında yeteince para ver ise parasını çekebilir yok ise bu sefer geçiş durumunda hata oluşacak ve «para öde» durumuna geçiş yapmadan «limit yetersiz» uyarısı durumuna geçiş yapması gerekecektir.

1	2	3
4	5	6

### 4.3. Kara Kutu Test Teknikleri



- Fayda Analizi Testi (Use Case Testing)

	Step	Description
<b>Main Success Scenario</b>  A: Actor S: System	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
<b>Extensions</b>	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

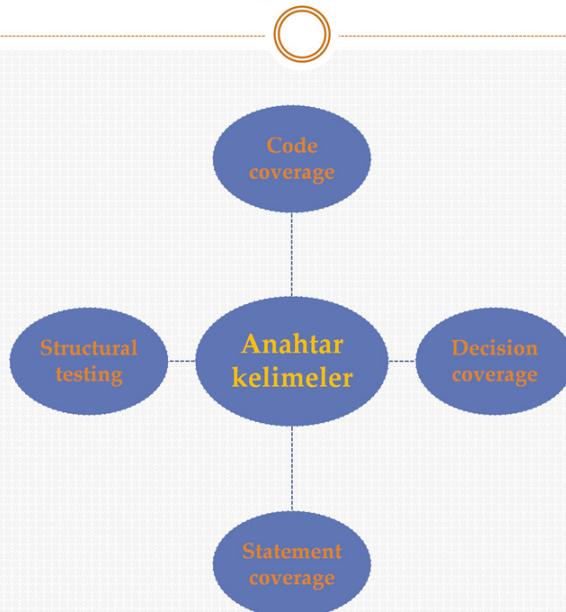
#### Fayda Analizi Testi

Use-case anlam olarak bakıldığından, bir sistemin harici bir kaynaktan gelen isteklerine verdiği cevap olarak tarif etmek mümkündür. Use-Case test tekniğinde bir kullanıcı aktör (A) ve yazılıma karşılık olarakta bir sistem (S) vardır. Aktörün her hareketi ve bu hareketine karşılık sistemin verdiği cevaplar geçerli bir durum olarak kayıt edilir.

Use case testi ile entegrasyondan kaynaklı hataları bulmak kolaydır çünkü aktör farklı arayüzler arasında sürekli iletişim kurarak gerçek dünyadaki işlerini yapar. Şekilde de görüldüğü gibi aktörün (A) istekleri sistem (S) tarafından verdiği cevaplar sırasıyla ilerlemektedir bu ilerleyişte herhangi bir sorun yoktur ve kullanıcı mutludur yani burası **mutlu yol (happy path)** olarak adlandırılır. Ek bilgilerde (extensions) ise sistem tarafında verilebilecek alternatif cevaplar yer almaktadır.

1	2	3
4	5	6

## 4.4. Beyaz Kutu Test Teknikleri



### Anahtar Kelimeler

- **Code Coverage (Kod Kapsamı):** Hangi kod parçalarının çalıştırılıp hangilerinin çalıştırılmadığını ölçmek için kullanılan bir birimdir.
- **Decision Coverage (Kontrol Noktalarının Kapsamı):** Kod içerisindeki ne kadar döngünün çalıştırıldığını göstermek için kullanılan bir birimdir. Çalıştırılan toplam döngülerin koddaki tüm döngülere oranı ile bulunur
- **Statement Coverage (Tüm Noktaların Kapsamı):** koddaki ifadelerin çalışma oranını bulmak için kullanılan bir birimdir.
- **Structural Testing (Yapısal Testler):** Beyaz kutu testleride denilen yazılımın iç yapısını test etmeye yönelik test teslerdir.

1	2	3
4	5	6

## 4.4. Beyaz Kutu Test Teknikleri



- Test Kapsamı (Test Coverage)

$$\text{Kapsam} = \frac{\text{Calıştırılan Test Case Sayısı}}{\text{Toplam Test Case Sayısı}} \times 100$$

- Cyclomatic Complexity

$n$  = node (şekil sayısı)

$l$  = line (bağlantı sayısı)

$$CC = l - n + 2$$

### Test Kapsamı

Yazılım üzerinde ne kadar test yapıldığını anlamak üzere geliştirilmiş metriklere **kapsama (coverage)** denir ve genel olarak toplam çalıştırılan test casenin çalıştırılması gereken toplam test caselere oranıdır. Farklı kapsamları ölçmek için aynı formül kullanılsa da her bir kapsam kendi içerisinde değerlendirilir. Bu bölüm içerisinde değerlendirilecek bazı kapsamlar şunlardır:

- Statement Coverage (İfadelerin Kapsamı)
- Decision / Branch Coverage (Karar Noktaların Kapsamı)
- Linear Code Sequence And Jump (LCSAJ) (Olabilecek Tüm Durumların Kapsanması)

### Cyclomatic Complexity

Kodun akış diagramı çıkartıldıktan sonra kodun ne kadar karmaşık olduğunu gösteren bir metriktir. Formülle hesaplanabileceği gibi en basit şekilde toplam karar ifadelerinin (**if**) sayısı artı bir şeklinde de hesaplanabilir.

1	2	3
4	5	6

## 4.4. Beyaz Kutu Test Teknikleri



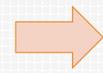
### • Statement Coverage

```
1 if age > 17  
2     print "yetişkin"  
3 if age > 5  
4     print "çocuk"  
5 else  
6     print "bebek"
```

Tanımı:

Tüm ifadelerin işletilmesi  
için gerekli en az test sayısı

Giriş Değeri	Çıkış Değeri
1	Bebek
6	Çocuk
18	yetişkin



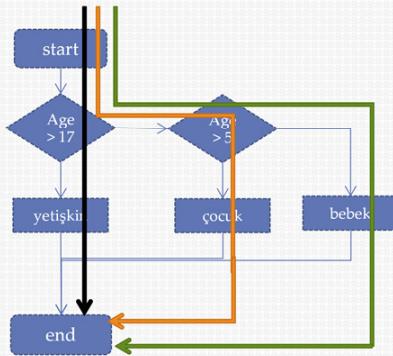
S. C. : 3

1	2	3
4	5	6

## 4.4. Beyaz Kutu Test Teknikleri



- Decision / Branch Coverage



Tanımı:

Tüm dalların işletilmesi için  
gerekli en az test sayısı

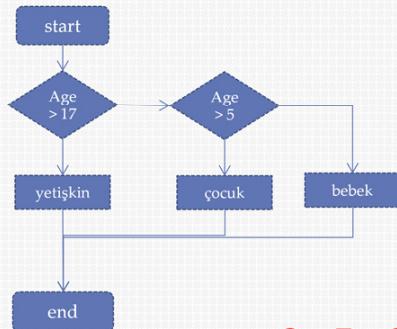
D. C. : 3

1	2	3
4	5	6

## 4.4. Beyaz Kutu Test Teknikleri



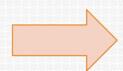
### • Cyclomatic Complexity



Tanımı:

- line – node + 2
- if + 1

- $8 - 7 + 2$
- $2 + 1$



C.C.: 3

1	2	3
4	5	6

## 4.5. Deneyim Temelli Test Teknikleri



### Anahtar Kelimeler

- **Error Guessing (Hata Tahmini):** Deneyimlerden yola çıkarak, uygulama ve konu bazlı hata oluşabilecek fonksiyon, durum, analiz noktalarının tahmin edilerek o amaca yönelik testleri planlamak. Analizin olmadığı ve zaman kısıtının olduğu durumlarda bu yola başvurulabilir.
- **Exploratory Testing (Keşif Temelli Test):** Yine deneyimlerin ön planda olduğu, analizin olmadığı durumlarda sistematic olarak uygulanmanın incelenmesi ve test edilmesine dayalı test tekniğidir.

1	2	3
4	5	6

## 4.5. Deneyim Temelli Test Teknikleri



- Error Guessing
  - Deneme yanılma
  - İlk test olarak kullanılmamalı
    - Tamamlayıcıdır
    - En uç noktalar bulunur
  - Kişisel deneyimler ön planda
    - Önceden benzerini test etmiş olmak
    - Hataları koklamak
  - Method yoktur
  - Eski defect haritası kullanılabilir
  - Test script yoktur

### Hata Tahmini

Hataları tahmin etme yöntemi tamamlayıcı bir yöntem olduğundan ilk method olarak kullanılamaz. Kural yoktur bu yüzden testçinin deneyimleri ön plandadır ve daha önceden benzer uygulamaları test etmiş olmak önemli bir avantajdır.

Analizin olmadığı ve sürenin kısıtlı olduğu durumlarda kullanılabilir. Deneme yanılma şeklinde yazılımın verdiği cevapların mantığa uygunluğu kontrol edilir. Tam olarak bir method olmasada, yazılım grubunun veya uygulama bazlı daha önceden kaydedilmiş hatalar (defect) yazılımın ekibinin güçlü ve zayıf yanlarını verebilir ve buradan çıkartılacak defect haritası hata tahmininde kullanılabilir.

1	2	3
4	5	6

## 4.5. Deneyim Temelli Test Teknikleri



- Keşif Temelli Testler
  - Testçi test yapar
  - Eşzamanlı tasarım ve çalışma
  - Döküman yok
  - Script yok
  - Diğer test teknikleride kullanılabilir
    - Sınır değer testi
    - Eşdeğer aralık
  - Amaç öğrenmek
    - Zayıf yanları
    - Güçlü yanları
    - Ne yapar
    - Ne yapmaz

### Keşif Temelli Test

Analizin zayıf ve olmadığı ve zaman sıkıntısı olan durumlarda yapılabilecek bir test teknigi veya diğer test tekniklerini tamamlayıcı olarak uygulamak doğru bir davranıştır. Analiz ve tasarım olmadığından, test yapılırken (test execution) sırasında bu işler paralel olarak yapılır. Test sırasında dökümantasyon yoktur sadece bulunan hatalar raporlanır. Testler manuel işletilir bu yüzden test scriptleri yoktur.

Bir metoda bağlı kalarak işletilmese de test sırasında testçinin karar verdiği yöntemler kullanılır. Keşif temelli testler isminden de anlaşılacağı üzere yazılımı keşfetmek amacı gütmektedir. Yazılıma testler yaparak yazılımin güçlü, zayıf yanlarını bulmak; yazılımin neler yapıp neler yapmadığını ve neleri yanlış yaptığı öğrenmek üzere yapılır.

1	2	3
4	5	6

## 4.6. Uygun Test Tekniğinin Seçimi



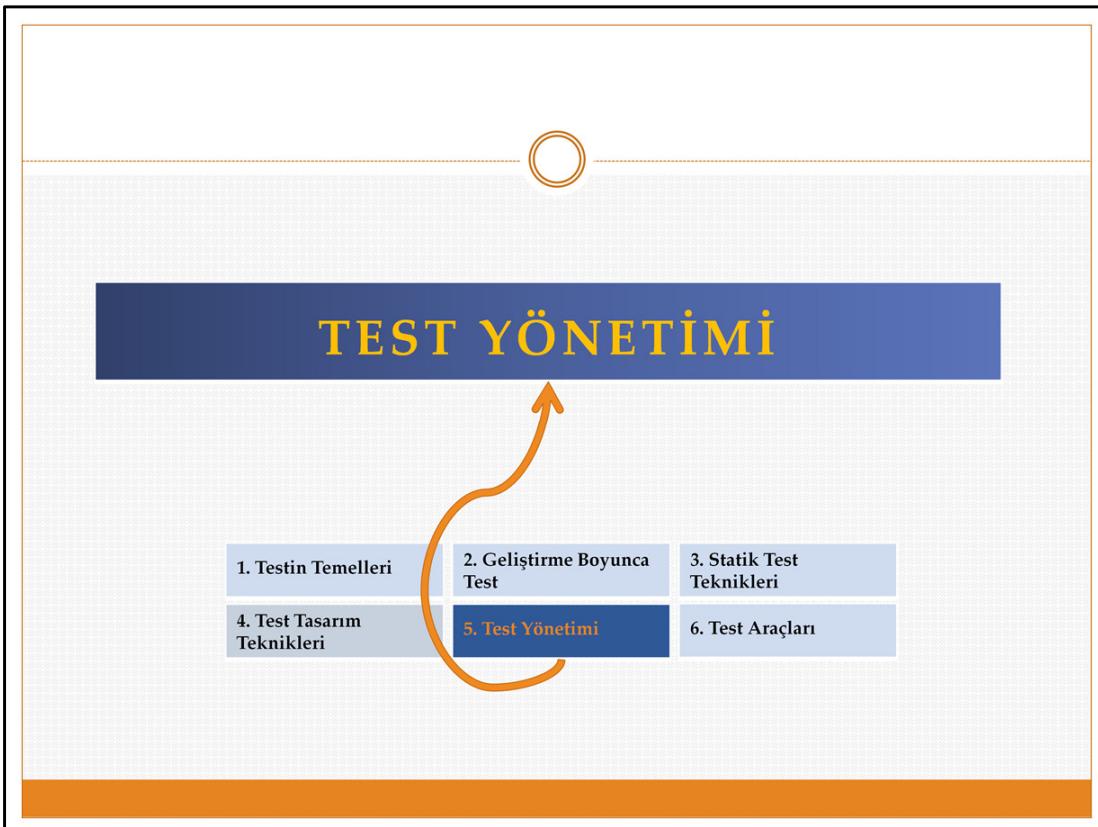
- **Doğru teknigi seçmek**
  - **En iyi teknik nedir?**
    - ✖ Her teknik iyidir
    - ✖ Doğrusunu seçmek önemli
  - **Önemli faktörler**
    - ✖ Sistemin türü
    - ✖ Standartlar
    - ✖ Gereksinimler
    - ✖ Risk seviyesi
    - ✖ Risk türleri
    - ✖ Dökümantasyon, zaman, testçi deneyimleri, ...

### Hangisi Doğru Teknik

Doğru teknığın bulunması testin başarısı ve yeterli kalitenin zaman kısıtı içerisinde yakalanması açısından önemlidir. En iyi teknigi kullanmak istenir fakat her durum için farklı teknikler farklı tür hataların bulunmasını sağlar. Bu yüzden en iyi teknigi aramak yerine, mevcut durum incelenmeli kısıtlar ve imkan göz önüne alınmalı ve verilere göre uygun yöntemler denenmelidir. Uygun yöntemi bulmak için sıralanmış faktörler göz önüne alınmalıdır, bu faktörler buradakilerle sınır olmamakla birlikte önem sıralaması yazılımın türüne ve risk türlerine göre farklılıklar gösterebilir.

# ISTQB Müfredatı

- Testin Temelleri
- Geliştirme Boyunca Test
- Statik Test Teknikleri
- Test Tasarım Teknikleri
- Test Yönetimi
- Test Araçları



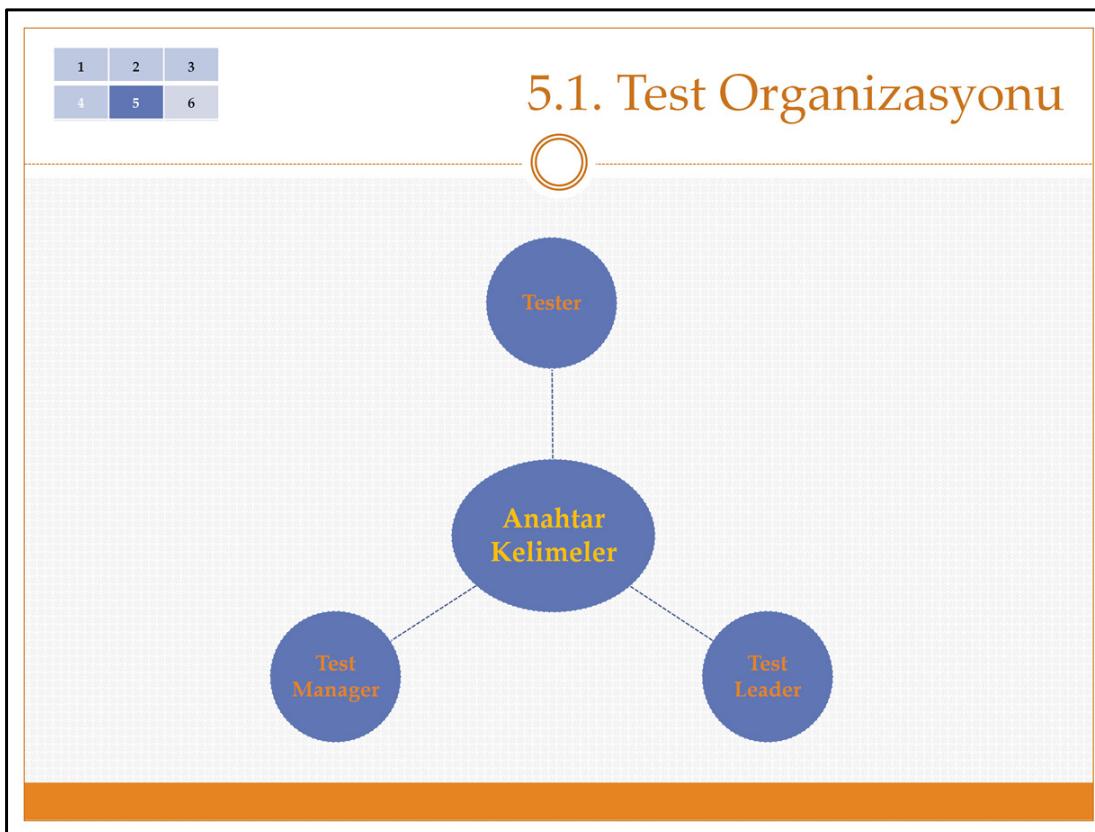
1	2	3
4	5	6

## 5. Test Yönetimi

- 
- Test Organizasyonu
  - Test Planı Ve Tahmini
  - Test İlerleme Gözetimi ve Kontrolü
  - Configürasyon Yönetimi
  - Risk ve Test
  - Hata Yönetimi

### Test Tasarım Teknikleri

Statik test teknikleriyle kod veya döküman incelemesi yapılıyor ama bahis konusu olan kod çalıştırılmadan uygun tekniklerle hatalar saptanıyordu. Dinamik test teknikleriyle (dynamic test techniques) ise kod çalıştırılıyor ve test teknikleri çalışan koda uygulanıyor. Bu yüzden yazılımın belli bir olgunluğa erişmiş olması gereklidir. İteratif yazılım geliştirme süreci işletiliyor ise iterasyon içerisindeki işlerin tamamlanmış olması ideal dünyada istenilen durumdur.



### Anahtar Kelimeler

- **Tester (Testçi):** Test yapan kişiler.
- **Test Leader (Test Lideri):** Geniş test grupları içerisinde testçilere görevlerini dağıtan, biraz daha üstten bakabilen kişilerdir. Sanılanın aksine birebir test yaparlar.
- **Test Manager (Test Grubu Yöneticisi):** Test grubunu yöneten kişilerdir.

1	2	3
4	5	6

## 5.1. Test Organizasyonu



- Farklı Test Organizasyonları
  - Geliştiriciler Testi
  - Geliştirici Takım Testi
  - Ayrık Test Grubu
  - Test Danışmanları
  - Outsource Test Hizmeti (Testing As A Service) (3rd Party)

Yazılım biriminin ürettiği kodları test etmek için farklı organizasyonlar oluşturulabilir. Bunların en basiti yazılımcı tarafından yazılan kod yine kendisi tarafından test edilerek kullanıcının kullanımına sunulması şeklindedir. Bu tür yapıtlarda test organizasyonundan bahsetmek güçtür çünkü aslında yazılımcının yaptığı test değil kendini sınamasıdır. Bu tür bir test işinde ciddi sorunlar veya riskler olmasından dolayı bu tip test aktivitesi daha çok sınırlı sayıda ve bilinçli kullanıcıların kullandığı programlar için geçerli olabilir.

Daha gelişmiş test organizasyonlarında ise gerçek bir formal yapıdan bahsedilebilir. Herbir organizasyonun kendi özel iyi ve kötü yanları bulunmaktadır. Bu yüzden en uygun yapıyı belirlerken yazılım grubunun ihtiyaçları, yeterliliklerini ve elde bulunan kaynakları iyi analiz etmek gereklidir. Test grubu genel anlamda diğer birimlere hizmet sunar yani üretime katkısı sınırlıdır fakat üretilen ürünün kalitesinde önemli rol oynar. Bu yüzden kalite gibi bir amaç var ise test grubu olmazsa olmazlardandır.

1	2	3
4	5	6

## 5.1. Test Organizasyonu



- Geliştiricilerin Testi

- **Artıları**

- Kodu İyi Bilir
    - Testçinin Kaçırdığı Hataları Görür
    - Hatanın Çözümü Kolaydır

- **Eksileri**

- Kendi İşini Bozmak İstemez
    - Oluşandan Çok Olması Gerekeni Görür
    - Öznel Değerlendirmeler Yapar

Yazılımcı kendi yazdığı kodu test eder. Yazılımcı daha çok kod yazan kişiler olduğundan dolayı bakış açısı her zaman kod düzeyinde olur. Yazılan kodun içerisinde mantık akışı, fonksiyon parametleri gibi en alt düzeyde incelemeler yapmaya alışkındır. Bu yüzden kendi yazdığı kodu çok iyi bilir, zayıf yanlarını tespit eder fakat üst düzeyde bakmak zorlaşır. Yani entegrasyon gibi diğer birimlerin testin içerisinde dahil olmasında görülebilecek hatalar üzerine düşünmez ve bu gibi etkileşimlerin sorunsuz çalıştığı üzerine yazar ve test eder.

Özellikle iyi bir analiz edilmeden yazılımcıdan kod yazmasını beklemek hatalı sonuçlar verebilir çünkü istenilenlerin analizi yazılımcıya kalır. Yazılımcı ise analizi kendi kod sınırları içerisinde değerlendirme tehlikesi taşır ve test sürecine geldiğinde ise hatalı sonuçları görmeme gibi durumlar veya kendi işini bozmaktan kaçınmak gibi tehlikeler olabilir. Fakat bununla birlikte bulduğu hataları çabucak çözebilir.

1	2	3
4	5	6

## 5.1. Test Organizasyonu



- Geliştirici Takım Testi

- **Artıları**

- Kod Bilgisi İyidir
    - Teknik Bilgisi İyidir
    - Yazılımcı İle Arası İyidir Daha Kolay Anlaşır

- **Eksileri**

- Yazılımcı Gibi Düşünür
    - Kendi İşlerinin Baskısı Vardır
    - Test Yeteneği Yoktur

Bir yazılımcının yazdığı kodu yine başka bir yazılımcı test eder. Test eden kişi yazılımcı olmasından dolayı teknik bilgisi ve kod yazma yeteneği gelişmiştir bu yüzden saptanan hataları daha iyi raporlama yapabilir ve kodu yazan yazılımcı ile daha kolay anlaşabilir. Yazılımcılarla birlikte iş yapmasından dolayı yardımlaşma daha üst seviyededir.

Fakat yazılımcıların yaptığı testler daha çok teknik detaylara odaklanma şeklinde olabilir ve iş birimlerinin istekleri göz ardı edilebilir. Ayrıca yazılımcı test işini ek iş olarak görür çünkü esas işi kendisine atanmış yazım gelistirme işlerine zaman harçar bu yüzden oluşan baskı tarafsız iş yapmasını engelleyebilir. Kısa zamanda daha iyi test yapmak ancak test yeteneklerinin gelişmesi ve doğru test tekniğinin doğru zamanda uygulanmasıyla olabilir. Yazılımcıların test üzerine fazla eğitim almadığını söylemek yanlış olmaz.

1	2	3
4	5	6

## 5.1. Test Organizasyonu



- Ayrık Test Grubu

- Artıları

- Test Bilgi Ve Deneyimi
    - Grup Olarak Kalite Odaklı
    - Yazılımı Tarafsız Değerlendirme
    - Test Yapmak Esas Görevi

- Eksileri

- Baska Grupların Baskısı
    - Kısıtlı Yardımlaşma
    - Kara Kutu Teste Odaklanması
    - Yalnızlaştırılma Tehlikesi

Yazılım grubunun yanında ayrıca esas işi test yapmak olan test grubunun bulunmasıyla oluşan bir organizasyondur. Test grubu kendi kalite standartlarını belirler ve buna uygun tamamlanmış işleri gerek ayrık olarak gerek entegrasyon bağlamında test ederek oluşan sürümleri, paketleri, programların kullanıcıya açılıp açılmaması (deployment) kararını ortaya koyar. Test grubunun sorumluluk alanı genelde yazılım grubunun ürettiği herşeyi test etmek ve firmanın kalite standartlarında olup olmadığını teyit etmek ve var ise riskleri raporlamaktır. Yani tarafsız bir şekilde yazılımı değerlendirebilir.

Test grubu bozulmaya müsait bir gruptur çünkü farklı grupların çıkarları bu grup tarafından kontrol edilmektedir. İş grubunun isteklerinin doğruluğu (validation), veya yazılım grubunun vaadettiğinin geçerliliği (verification) kontrol edilirken her bir grubun isteklerini tarafsız yerine getirmek ve bunu uzun süreler boyunca devam ettirmek zordur. Gerek ikili ilişkiler gerekse ast-üst kavramlarından dolayı veya patronun para odaklı mantıklı veya mantıksız isteklerinden dolayı tarafsızlığı koruma zordur. Yalnızlaşma ihtimali yani test işleri bir köşede başkalarının istediği yapma riski vardır.

1	2	3
4	5	6

## 5.1. Test Organizasyonu



### • Test Danışmanları

#### ◦ Artıları

- Test Bilgisi ve Deneyimi
- Yazılımı Tarafsız Değerlendirme
- Test Yapmak Esas Görevi

#### ◦ Eksileri

- Baska Grupların Baskısı
- Kısıtlı Yardımlaşma
- Kara Kutu Teste Odaklanma
- Yalnızlaştırılma Tehlikesi

Yazılım grubunun yanında ayrıca esas işi test yapmak olan test grubunun bulunmasıyla oluşan bir organizasyondur. Test grubu kendi kalite standartlarını belirler ve buna uygun tamamlanmış işleri gerek ayrık olarak gerek entegrasyon bağlamında test ederek oluşan sürümleri, paketleri, programların kullanıcıya açılıp açılmaması (deployment) kararını ortaya koyar. Test grubunun sorumluluk alanı genelde yazılım grubunun ürettiği herşeyi test etmek ve firmanın kalite standartlarında olup olmadığını teyit etmek ve var ise riskleri raporlamaktır. Yani tarafsız bir şekilde yazılımı değerlendirebilir.

Test grubu bozulmaya müsait bir gruptur çünkü farklı grupların çıkarları bu grup tarafından kontrol edilmektedir. İş grubunun isteklerinin doğruluğu (validation), veya yazılım grubunun vaadettiğinin geçerliliği (verification) kontrol edilirken her bir grubun isteklerini tarafsız yerine getirmek ve bunu uzun süreler boyunca devam ettirmek zordur. Gerek ikili ilişkiler gerekse ast-üst kavramlarından dolayı veya patronun para odaklı mantıklı veya mantıksız isteklerinden dolayı tarafsızlığı koruma zordur. Yalnızlaşma ihtimali yani test işleri bir köşede başkalarının istediği yapma riski vardır.

1	2	3
4	5	6

## 5.1. Test Organizasyonu



- Outsource Test Hizmeti (Testing As A Service) (3rd Party)
  - Artıları
    - Yüksek Test Bilgisi ve Deneyimi
    - Yazılımı Tarafsız Değerlendirme
    - Farklı Firmalarda Test Deneyimleri
  - Eksileri
    - Maliyeti Yüksek
    - Firmanın Ürünleri Hakkında Kısıtlı Bilgi
    - Edinilen Deneyimi Dışarı Paylaşma

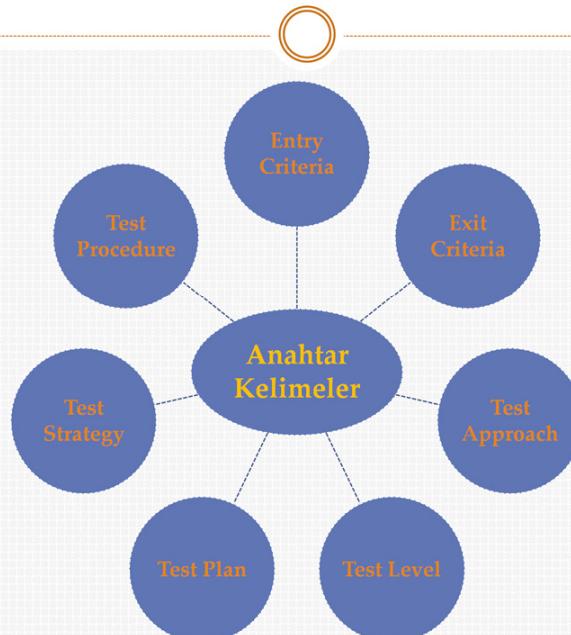
Yazılımın belli bir olgunluğa erişikten sonra firma dışında işi yazılım testi yapmak olan firmalara testisinin yaptırılması şeklidindedir. Yani bu tip organizasyonlar test işi bir hizmet olarak (testing as a service) harici kaynaklardan sağlanıyor ve sözleşme bittikten sonra hizmet tamamlanmış oluyor.

Bu tür organizasyonun en büyük faydaları farklı firmalarda bir çok test yapmış danışmanların geçerleme (validation) sırasında eksik ve hataları kolay görmeleri ürününün olgunluğunu arttırr. Ön yargılardan uzak yazılımı tarafsız değerlendirebilir.

Genel olarak **performance, güvenlik, kullanılabilirlik** gibi fonksiyonel olmayan uzmanlık gerektiren işlerin bu tür kaynaklar vasıtasiyla sağlanması daha mantıklıdır. Çünkü maliyetleri yüksek ve esasından firmanın içerisinde olmadığından fonksiyonel testler için ürün bilgisi fazla yoktur. Ayrıca yazılımın olgunluğu yeterli olmadığı durumlarda fonksiyonel testlerde çıkan hatalar bu tip testçiler tarafından fazlaca bulunabilir ve buda test için süreyi aşmaya neden olabilir. Geçerleme testleri için gerekli olan farklı firma ve ürün deneyimi eğer gizli veya güvenliği yüksek olan işler yapılıyorsa bu deneyimin, bilgi veya tekniğin test danışmanları tarafından dışarı çıkma ihtimali vardır.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



### Anahtar Kelimeler

- **Entry criteria (Teste başlama kriteri):** Geliştirmenin belli bir olgunluga erişip (smoke testten geçebilen) test alınması için gerekli kriterlerdir.
- **Exit Criteria (Testi sonlandırma kriterleri):** Yeterli miktarda test yapılmış yapılmadığının anlaşılması için konulan kriterlere testi sonlandırma kriterleri denir. Risk yönetimi yapmak daha sağlıklı sonuçlar verir.
- **Test Plan (Test Planı):** Test sırasında uyulması gereken kuralların bulunduğu test başlamadan önce hazırlanan plan
- **Test level (Test Seviyesi):** Test seviyesi, birim-integrasyon- ...
- **Test Approach (Test Yaklaşımı):** Hangi tip test grubunun hazırlanacağını tanımlayan firmanın benimsediği test methodlarının seçmmeye yarayan yöntemlerdir.
- **Test Strategy (Test Stratejisi):** Test yaklaşımında belirlenen test türlerinin işletilmesi, teste başlama, testi sonlandırma kriterleri, hata raporlaması, vb içersine alan test yönetimi.
- **Test Procedure (Test Adımları):** Testi işletmek için gerekli olan adımların anlatıldığı döküman.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- **Test Plan**

- Test işlerinin tamamlanması için gerekli proje planı
- Neden Gerekli?
  - Fikirleri toparlar
  - Değişimi yönetmeye yardımcı olur
  - İletişim kurmaya yarar

Test planı proje kapsamındaki test ile alakalı tüm işleri içersinde barındıran ve mantık sıralamasına sokan proje planıdır. Plana sadık kalarak test ilgili tüm işleri yapmak mümkün olacak şekilde tüm konuları içermelidir. Fakat her işin nasıl yapılacağı gibi detay bilgiler yerine herkesin anlayabilicegi ölçüde detayı içerir olmalıdır. Çünkü test planı farklı gruptan kişilerle iletişim kurmaya yardımcı olur.

Test sırasında oluşabilecek sapmalarda manevra kabiliyeti olabilmeli çünkü test sırasında ne kadar hatanın bulunacağı belli olmadığı için plandan sapmaları yönetebilmeli ve olabildiğince elemine etmelidir. Teste başlamadan önce ilgili kişilerle görüşürülerek düşünceleri toplanmalı ve ortak mutabaka varılmalıdır.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- **Test Plan**

- **İçeriği Nelerden Oluşur?**

- Nelerin test kapsamında olduğu ve olmadığı
    - Test amaçlarının ne olduğu
    - Proje ve ürün risklerinin neler olduğu
    - Proje ve ürün için en önemli olguyu
    - Kısıtları
    - Nelerin test etmeye daha müsait olduğu
    - Test etmek için gerekli programı

İyi bir test planı sayılı maddeleri içermelidir. Test planın amacı nesnel bir şekilde projeyi test ederken oluşabilecek sıkıntıları ortaya koymaktır. Kısıtlar bu yüzden önemlidir, test ortamında veya test edilemeyecek kısımların bulması teste harcanacak eforu daha sağlıklı planlamaya yardımcı olur ve daha verimli sonuçlar alınabilir.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- Teste Başlama Kriterleri (Entry Criteri)
  - Personel, araç-gereç, meterial ihtiyaçları
  - Test edilecek ürünün hazır olması
    - Smoke test
  - Test datası olmalı
  - Testi sonlandırma kriterlerinin hazır olmalıdır

Teste başlama kriterleri, geliştirilmekte olan üründen bir iterasyon alınıp bu iterasyonun test hazır olup olmadığını kontrol eden eksikleri belirlemek üzere hazırlanmış test kriterleridir. Iterasyonu alınan ürünün test için hazır olup olmadığını **smoke test** yaparak anlayabiliriz. Smoke test ile temel bazı fonksiyonları test eder ve çok fazla hata saptanmaz ise test ileri seviye testlere geçebiliriz. Eğer ürün smoke testten geçecek kadar olgunlaşmamış ise teste başlama kriterlerini sağlamadığı için reddedilebilir.

Teste başlama kriterleri ayrıca test sırasında ihtiyaç duyulacak olan araç-gereç, döküman, personel ve test datasınıda içerir. Test başlanıldığından testi sonlandırma kriterleri yerine geldiğinde test sonlandırmak gereklidir yani testi sonlandırma kriterleride teste başladıkten sonra ihtiyaç duyulacaktır.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- Testi Sonlandırma Kriterleri (Exit Criteria)

- Testin nezaman sonlandırılacağını belirler
  - Bir takım testlerin başarılı olduğunda
  - Belirli bir code coverage sağlandığında
  - Hata dağılımı kavranıldığında
  - Bütçe, zaman tamamlandığında
  - Hata bulunma sıklığı azaldığında
  - Kabul edilebilir riskler taşıdığında
  - Vb. kriterler

Testi tamamlandı artık ürünü kullanıcıya açılabilir denilebilmesi için gerekli kriterlere testi sonlandırma kriterleri (exit criteria) denir. Genel olarak ya test başarılı bir şekilde tamamlanış ya da test/yazılım için ayrılan süre/bütçe/personel/vb. Kriterlerin tükendiği zaman test sonlandırılır. İkinci durumda test kullanıcıya acılma kararını risk değerlendirmesi yapılarak sonuçlandırılabilir, karar vericilerin saptanan hataların doğurucağı zararları iiy analiz etmeleri gerekmektedir.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- Tahminler
  - Testci sayısı
    - Proje konusu
    - Proje riski
    - Ürün riski
  - Test süresi
    - Test case sayısı
    - Gereksinim sayısı
    - İşletim süresi
  - Karmaşıklığı/Zorluğu
    - Kullanılan methodlar
    - Yeni teknolojiler
    - Güvenlik açıkları

Teste başlamadan önce bazı tahminlerin yapılması test sürecinin planlanması açısından önemlidir. Tahminler içerisinde en önemli olanı kaliteli bir test süreci için gerekli en az süre ve test organizasyonunda bulanacak personel sayısı. Test sürenin tahmin edilmesi için göz önünde tutulması gereken parameterler; test case sayısı ve gereksinim sayılarıdır. Buradan yol çıkarak tahmini test süresi aşağıdaki gibi hesaplanabilir:

$$\text{test süresi} = (\text{test_case_sayısı} \times \text{test_isletim_süresi}) / \text{testci_sayısı}$$

Yazılımcı başına düşen testçi sayısı en çok merak edilen başka bir konudur. Böyle bir sıhırli sayı bulunmamakla birlikte bu sayı çok geniş bir aralıkta değişebilir. Bu sayı tamamen; projenin konusu, zorluk derecesi, yazılımcının olgunluk seviyesi, riskler, güvenlik kriterleri, firmanın imajı, kullanıcı profili gibi geniş kriterlerden etkilenir.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- Test Stratejisi:
  - Projeye özgüdür
  - Ortak kanı verir
  - Süreçler belirlenir
- Test Yaklaşımı:
  - Daha küçütür, test tiplerini gruplar
- Değerlendirilmesi gereken bileşenler:
  - Yazılım geliştirme stratejileri
  - Müşteri gereksinimleri
  - Test amaçları
  - Proje konusu

### Test Stratejisi:

- Test yaklaşımının ana hatlarını çizen ve projeye özgü hazırlanan kurallar bütünüdür.
- Organizasyondaki herkesin bilgilenmesi ve bu doğrultuda hareket etmesini sağlar.
- Test caselerin tasarılanması, test süreçlerin belirlenmesi, teste başlama ve testi sonlandırma kriterlerin belirlenmesinde ana rol oynar.

### Test Yaklaşımı:

Projeyi oluşturulan bütün bileşenleri içerisine katarak daha başarılı test sonuçları yakalamak için kullanılan test tiplerinin gruplandırıldığı test anlayışıdır.

### Değerlendirilmesi gereken bileşenler:

Test stratejisini ve yaklaşımını belirlemek için bu kriterler göz önüne alınmalıdır.

1	2	3
4	5	6

## 5.2. Test Planı ve Tahmini



- ISQTB Örnek Test Yaklaşımları:
  - Analytical strategies, such as risk-based testing
  - Model-based strategies, such as operational profiling
  - Methodical strategies, such as quality-characteristic based
  - Process- or standard-compliant strategies, such as IEEE 829-based
  - Dynamic and heuristic strategies, such as using bug-based attacks
  - Consultative strategies, such as user-directed testing
  - Regression testing strategies, such as extensive automation.

Örnek bir değerlendirme yapılır ise;

Bir e-ticaret firmasını örnek olarak aldığımızda 2 Alanda inceleme yapılabilir: **Proje konusu ve Geliştirme Ortamı**

**Analitik Yaklaşım:** İnternet üzerinden satış yapıldığı için bazı riskler vardır. Örneğin; güvenlik problemleri, firma imajını zedeyecek görsel ve yazım hataları, ... Olmasından dolayı test caselerin risklerinin sınıflandırılmasında kullanılabilir.

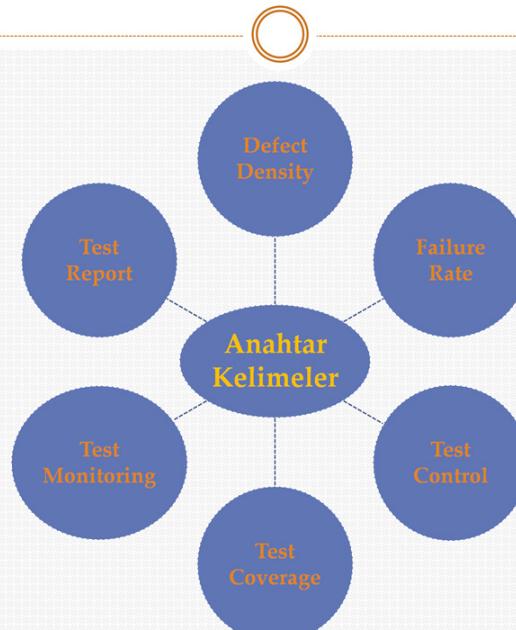
**Dinamic Yaklaşım:** Gereksinimlerin olmadığı, planlamanın tam olarak yapılmadığı hızlı bir yazılım geliştirme ortamı olmasından ötürü de dinamik test yaklaşımı kullanılabilir.

Analitik ve Dinamic test stratejilerinin bileşiminden oluşan; **Risk Temelli - Dinamik test yaklaşımı** belirlenebilir.

- *Hata verme olasılığı ve daha yüksek alanlara daha fazla test yapmak*
- *Keşif tabanlı testler yaparak hata tahminlerinde bulunarak, ilgili alanlara yoğunlaşmak*

1	2	3
4	5	6

### 5.3. Test İlerleme Gözetimi ve Kontrolü



#### Anahtar Kelimeler

- **Defect Density (Hata Yoğunluğu):** Hataların bulunma sıklığı, bir modül üzerinde saptanan hata oranı
- **Failure Rate (Hata Oranı):** Testte hata veren test case oranı
- **Test Control (Test Kontrolü):** Plandan sapma olmasının durumunda test caselerin sapmayı önlemek için kontrol edilmesi
- **Test Coverage (Test Kapsamı):** Mevcut test caselerin yazılımın ne kadarını test ettiğini gösteren bir birim
- **Test Monitoring (Test Görüntüleme):** Test caselerinin çalıştırılırken mevcut durum ile planlanan arasındaki farkları görüntülemek amacıyla kurulmuş test yönetimi faliyeti
- **Test Report (Test Raporu):** Testlerin tamamlanmasına mütekip çıkan sonuçların özet halinde sunulduğu rapor.

1	2	3
4	5	6

### 5.3. Test İlerleme Gözetimi ve Kontrolü



- Test Durum Görüntüleme
  - Plana nekadar uyuyor
  - Hatalar
    - Saptanan
    - Raporlanan
    - Çözülen
  - Nekadar daha emek gerekli
  - Test sonuçları takım içinde görülebilir

Teste başladıkten sonra testin gidişatını veya yazılımın kalitesini canlı olarak görüntüleme işine test durum görüntülemesi (test progress monitoring) denir. Genel olarak yönetici veya test takım liderleri anlık rapor vermek adına bu gibi faliyetleri kullanabilirler. Test durum görüntülemesi ayrıca test planına nekadar sadık kaldığını veya sapmalar var ise sapmaların nelerden kaynaklandığını gösterebilir. Planda tahmin edilen hata sayısındaki sapmalar şüphesiz test işletimi sırasında daha büyük sapmalara neden olacaktır.

Görüntüleme faliyetleri ayrıca takım içerisinde herkes tarafından görülebilir olduğundan herkesin kendi işin tarafsız değerlendirmesine ve diğerleri ile bilgi akışını sağlar. Ayrıca kalan test işi için nekadar daha emek (test efor) gerektiğinde buradan görüntülenebilir.

1	2	3
4	5	6

### 5.3. Test İlerleme Gözetimi ve Kontrolü



#### o Test Kontrolü

System Test Case Summary												
Cycle One												
Test	ID	Test Suite/Case	Status	System	Bug	Bug	Run	Plan	Act	Plan	Actual	Test
ID				Config	ID	RPN	By	Date	Date	Effort	Effort	Duration
<b>1.000 Functionality</b>												
1.001	File	Fail	A	701	1	LTW	1/8	1/8		4	6	6
1.002	Edit	Fail	A	709	1	LTW	1/9	1/10		4	8	8
				710	5							
				718	3							
				722	4							
1.003	Font	Pass	B			JHB	1/10	1/10		4	4	4
1.004	Tables	Warn	B	708	15	JHB	1/8	1/9		4	5	5
1.005	Printing	Skip						1/10		4		
<b>Suite Summary</b>							1/10	1/10	20	23		23
<b>2.000 Performance/Stress</b>												
2.001	Solaris Server	Warn	A,B,C	701	1	EM	1/10	1/13		4	8	24 Replan 1/11
2.002	NT Server	Fail	A,B,C	724	2	EM	1/11	1/14		4	4	24 Replan 1/12

ISTQB dökümanında verilen örnek raporda teste o anki mevcut durumu gösteren bir çok bilgi yer almaktadır. Örneğin buradaki raporda **Plan Effort** planlanan test için harçanacak emeği gösteriyor ve **Actual Effort** ise test işletilirken harçanan gerçek emeği göstermektedir. Bunlar ilk test için sırasıyla 4 ve 6 imiş yani planan dah küçük olduğu için planan sürede testin tamamlanması mümkün gözükmemektedir. Planan sürede testlerin tamamlanması için mutlaka bir müdehale gereklidir. Bu müdehaleler projeye göre farklılık göstermekle birlikte şunlar olabilir:

- Testci sayısını artırmak
- Test case sayısını azaltmak
  - Risk analizi yaparak
  - Belli riskleri kontrol altına alınabilir
- Süreyi uzatmak

1	2	3
4	5	6

## 5.4. Konfigürasyon Yönetimi

The diagram consists of three blue circles arranged horizontally. The first circle on the left contains the text 'Configuration Management'. The middle circle contains the text 'Anahtar Kelimeler'. The third circle on the right contains the text 'Version Control'. Dashed lines connect the center of the first circle to the center of the second, and the center of the second to the center of the third, forming a chain-like relationship between the three concepts.

### Anahtar Kelimeler

- **Configuration Management (Konfigürasyon Yönetimi):** Sistem içerisindeki tüm ekipman ve yazılımların (hardware and software) kontrol edildiği yönetim araçlarına konfigürasyon yönetimi denir.
- **Version Control (Versiyon Kontrolü):** Yazılımlar üzerinde yapılan değişimleri kontrol amacıyla geliştirilmiş ve gerektiğinde istenilen değişikliği (sürüm) geriye almaya yarayan yazılım araçlarıdır.

1	2	3
4	5	6

## 5.4. Konfigürasyon Yönetimi

- Konfigürasyon Yönetimi Nedir?

- Sistemdeki tüm araçların sürümlerini kontrol eder
- Birimlerin fonksiyonel ve fiziksel olarak tanımlarını dökümanete etmek
- Sistemdeki araçların durumunu günceller ve raporlar
- Hata oluşması durumunda hatayı bildirir
- Sistemdeki tüm araç ve yazılımların güncelliği denetler
- Sistemdeki araçların bir birleriyle olan çalışmalardaki sorunları kontrol eder

### Konfigürasyon Yönetimi Nedir?

Konfigürasyon yönetimi, yazılımın içinde bulunduğu tüm alt sistemleri bir bütün halinde yönetmeyi amaçlayan sistem yönetim araçlardır. Yazılım yaşayan bir canlı olarak düşünüldüğünde sürekli ve değişen ihtiyaçları olan bir hizmet aracıdır.

Kullanıcı her ne kadar çoğu zaman bu değişimleri hissetmesede gerek donanım gerekse yazılımda meydana gelen değişimler kimi zaman mecburi kimi zamanda isteklerden kaynaklanabilir. Konfigürasyon yönetimi ise bu değişimleri kontrol eden ve herhangi bir sorun çıkması durumunda en düzgün (stable)

konfigürasyonu yükleyebilen ve hizmet kesintisini en aza indirmeyi hedef edinmiş bir yazılım hizmetidir.

Test gurubu olarak bakıldığından konfigürasyonun daha farklı anlamları da vardır. Test ancak güvenli bileşenler mantığı ile işletilebilir. Yani sistem içerisinde **n** tane değişken olabilir fakat aynı anda sadece **bir** değişken test edilebilir bu durumda (**n** – 1) değişkenin normal (hatasız) çalıştığı varsayılmaktadır. Böyle bir sistemin kontrolünü ancak konfigürasyon yönetimi sağlayabilir. Yani test yapılrken hangi bileşenlerin hangi versiyonda oldukları, hangi donanımlar üzerinde koşturulduğu ve hangi bileşenlerin güncellendiğini bilerek test yapılabilir. Hata saptanması durumunda ise bu bilgiler eşliğinde hata çözüme ulaşılabilir.

1	2	3
4	5	6

## 5.4. Konfigürasyon Yönetimi



- Konfigürasyon Yönetimine Alınabilecek Bileşenler
  - Planlar
  - Süreç tanımları
  - Gereksinimler
  - Tasarım bilgileri
  - Çizimler
  - Ürün spesifikasyonları
  - Kod
  - Derleyiciler
  - Ürün bilgisi dosyaları
  - Ürün teknik yayınları

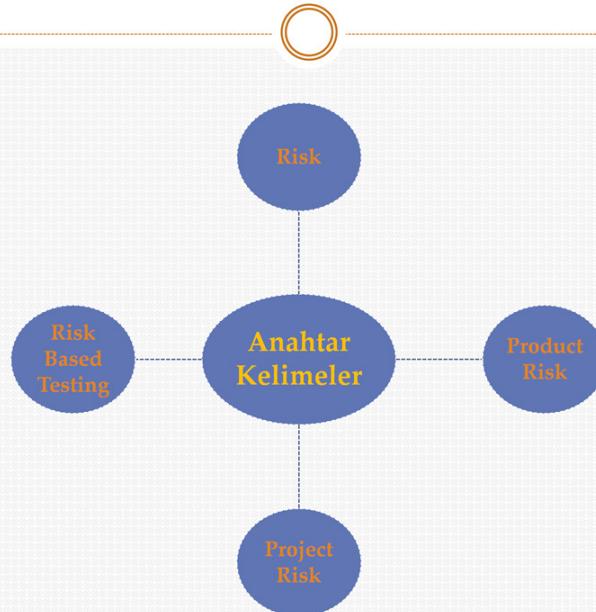
Konfigürasyon yönetimine bir çok sey dahil edilebilir. Sistemdeki tüm bileşemleri dahil etmek daha sağlıklı bir hizmet vermek açısından daha iyi olacaktır.

Kaynak:

[http://ieee.metu.edu/iffet/blog/Konfigurasyon\\_Yonetimi\\_CMII-V1.2.pdf](http://ieee.metu.edu/iffet/blog/Konfigurasyon_Yonetimi_CMII-V1.2.pdf)

1	2	3
4	5	6

## 5.5. Risk ve Test



### Anahtar Kelimeler

- **Risk:** Hatanın ortaya çıkma olasılığı.
- **Product Risk (Ürün Riski):** Ürünün istenildiği gibi tamamlanmasını engelleyebilecek etkenler.
- **Project Risk (Proje Riski):** Projenin istenildiği gibi tamamlanmasını engelleyebilecek etkenler.
- **Risk Based Testing (Risk Temelli Test):** Riskleri bilerek ve riskleri göz önünde tutarak test eforunu taksim etmek ve testleri yapmak riskin yüksek olduğu yerlerde yoğunlaştırmayı amaçyan test yaklaşımı (test approach).

1	2	3
4	5	6

## 5.5. Risk ve Test



- Ürün Riski (Product Risks)

- Ürünün kalitesine zarar verebilecek etkenler
  - Fonksiyonel hatalar
  - Güvenlik eksikleri
  - Güvenilirlik (reliability) eksikleri
  - Kullanılabilirlik hataları
  - Performans sorunları
- Müşteriyi memnuniyetini zarar verebilecek diğer etkenler

1	2	3
4	5	6

## 5.5. Risk ve Test



- **Proje Riski (Project Risks)**

- Test projesinin istenilen plana uymasını engelleyebilicek her türlü etken proje riski oluştur.

- **Proje riskleri**

- Süreçteki değişiklikler: lojistik, vb.
    - Test ortamında çalışmayan test araçları
    - Test caselerin yeniden yazılmasına sebep olabilecek değişiklikler
    - Test ortamındaki hatalar
    - Kaynak eksilmeleri: testçi, test analisti, vb.
    - Donanım eksiklikleri
    - Teknik problemler: tasarım, kod, vb.

1	2	3
4	5	6

## 5.5. Risk ve Test



- **Proje Riski (Project Risks)**

- Test projesinin istenilen plana uymasını engelleyebilicek her türlü etken proje riski oluştur.

- **Proje riskleri**

- Süreçteki değişiklikler: lojistik, vb.
    - Test ortamında çalışmayan test araçları
    - Test caselerin yeniden yazılmasına sebep olabilecek değişiklikler
    - Test ortamındaki hatalar
    - Kaynak eksilmeleri: testçi, test analisti, vb.
    - Donanım eksiklikleri
    - Teknik problemler: tasarım, kod, vb.

1	2	3
4	5	6

## 5.5. Risk ve Test

- Risk Temelli Test

- Risk analizi yapılmalı
  - Gereksinim tanım dökümanı
  - Paydaşlarla durum üzerine düşünülmeli
  - Konu uzmanlarından öneriler alınmalıdır
- Puanlandırılmalı
  - Farklı aralıklar kullanılabilir
    - 1-10
    - 1-5
    - 1-3
- Zaman planı ile riskler paylaştırılmalı

Risk temelli testin en önemli özelliklerinde birisi risk analizi yapılmasıdır. Risk analizi mevcut durum göz önüne alınarak hazırlanmış test caselere verilen puanlardır. Risk analizi yapmak demek konu üzerine derinlemesine incelemek demektir. Test caseler yazılrken kullanılan dökümanlar bu boyutta tekrar gözden geçirilebilir.

Test caselere risklerine göre puanlandırılırken farklı grupların çıkarları o gruptan kişiler tarafından temsil edilmesi önemlidir. Büyüzde tüm paydaşlar bir araya gelmeli ve puanlandırma üzerinde bağımsız düşüncelerini dile getirerek en çok vurgulanan puan ilgili test case için kullanılmalıdır. Puanlandırma kullanılacak puan aralığı önemlidir 1-10 gibi bir aralık kullanılması durumunda geniş bir secim sunulmaktadır ve 6 ile 7'nin farkını kavramak zor olabilir fakat 1-3 gibi bir aralıktı ise test caselerin alacağı puanların çoğunda eşitlik söz konusu olabilir. İdeal yapıda 1-5 aralığı daha çok kullanılan yöntemdir.

Risk temelli test tekniğinin son aşamasında ise puanlandırılan test caseler test için ayrılan zaman göz önüne alınarak ilgili sürenin paylaştırılmasıdır. Burada alınacak kararla en düşük risk seviyesindeki bazı test caselerin test edilmemesi şeklinde olabilir.

1	2	3
4	5	6

## 5.5. Risk ve Test

- Risk Analizi Örneği

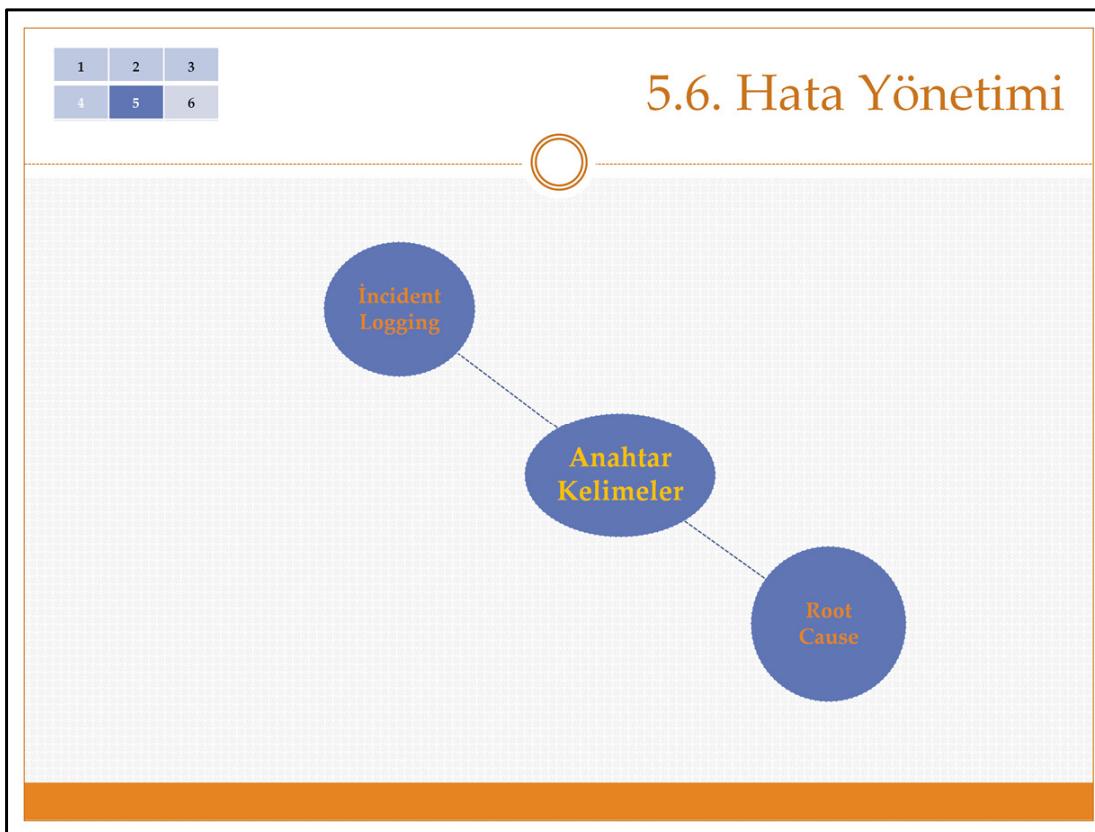
Gereksinim	Çıkma Olasılığı	Etkisi	Risk Değeri
Yeni Üyelik	3	4	$3 \times 4 = 12$
Satınalma	4	5	$4 \times 5 = 20$
Kullanıcı Bilgilerini Görme	1	2	$1 \times 2 = 2$

Not: 1 en küçük, 5 en büyük değere sahip

Tabloda görüleceği gibi 3 farklı durum söz konusu. Herhangi birisine puan verirken diğerlerini göz önüne almak gereklidir. Örneğin **Satınalma** eğer bir e-ticaret firmasıyla çok önemli bir gereksinim bu yüzden burda çababilcek bir sorunun etkisi de en büyük olacaktır. Fakat **Kullanıcı Bilgilerini Görme** üye için kısmen önemli olabilir fakat orada oluşabilecek hatanın etkisi çok fazla olmayabilir, bu yüzden en küçük puan verilebilir. **Yeni Üyelik** ise e-ticaret firması için önemlidir fakat kişi üye olmadan da alışveriş yapabiliyor ise etkisi biraz daha az olacaktır. Burada durum belki pazarlama grubu için çok önemli olabilir fakat iş geliştirme veya IT grubu için daha düşük öneme sahip olabilir. Yani ortaya çıkan ikilemi tarafsız puan vererek ve ortalaması alınarak konulabilir. Budurumda ekisi için 4 denebilir.

Gereksinimlerde hata çıkma olasılığı ise o fonksiyonun kullanma sıklığı ve koduyla ilgilidir. Eğer sık kullanılıyor ve çok fazla test edilmemiş olgun bir kod değil ise **Çıkma Olasığı** yüksek olacaktır. Aksi durumda ise daha düşük değerler alabilir.

Sonuçta elde eilen **Risk Değeri** bazı nekadar test edilmesi gerektiği söylemektedir. Bir başka deyişle ortak bir kararla risk değeri belli bir değerin altındakiler test edilmemeli gibi politik bir karar alınabilir. Örneğin burada 12'den düşük riski bulunanlar test edilmemelidir denildiğinde geriye kalan test caseler için test zamanı daha efektif kullanılabilir.



### Anahtar kelimeler

- **Incident Logging (Vaka Kaydı):** Test sırasında saptanan her türlü hata ve eksikliğin otomatik kayıt altına alındığı kayıt sistemi.
- **Root Cause (Problemin Özü):** Bir sorunu çözmek için yapılan neden sonuç araştırmalarıdır.

1	2	3
4	5	6

## 5.6. Hata Yönetimi



- Vaka Nedir?
  - Test sırasında beklenen sonuca uymayan herşey
- Sebepleri nelerdir?
  - Yazılımda hata var
  - Test doğru işletilmedi, testte hata var
  - Beklenen sonuç yanlış verilmiş
  - Test ortamında sorun vardır
- Hata yazılımda olabileceği gibi dökümantasyonda da olabilir!

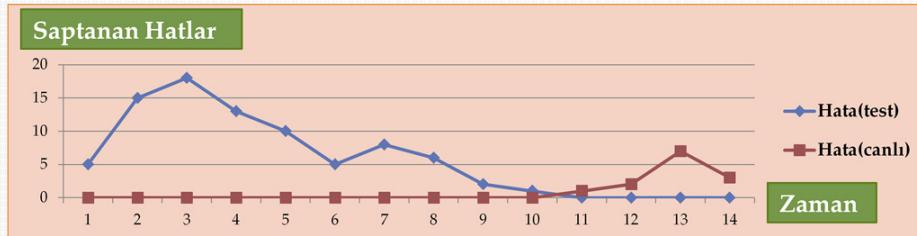
Test sırasında karşılaşılan sorunlar herzaman hata olmayabilir fakat beklenen sonuçlara uymayan sonuçlar elde ediliyor ise bu durum bir **vaka (incident)** olarak adlandırılır ve raporlanması gereklidir. Hatanın kaynakları incelenmeli ve sorunun çözümü için gerekli kişiler haberdar edilmelidir.

Saptanan hatalar genel olarak gerçekten yazılımdan kaynaklanan bir hatadır veya yazılımda sorun yoktur fakat diğer faktörler hatalı sonuç almaya yönlendiriyor. Bu durumda kararlı (stable) bir test ortamı herzaman gerekliliktedir ve konfigürasyon yönetimi de şarttır. Unutulmamalıdırki hata yazılımda olabileceği gibi test ortamında veya dökümantasyonda da olabilir. Bu yüzden bir vakanın hata (bug, defect) olarak adlandırılmadan önce emin olmak gereklidir. Bulunan hatalar anlaşılır bir şekilde raporlanmalıdır.

1	2	3
4	5	6

## 5.6. Hata Yönetimi

- **Hata Yakalama Oranı (Defect Detection Percentage)**
  - Test sırasında yakalanan hatanın tüm hatalara oranıdır
  - Test ortamından kaçan hataları gösterir



$$DDP = \frac{hata(test)}{hata(test) + hata(canlı)}$$

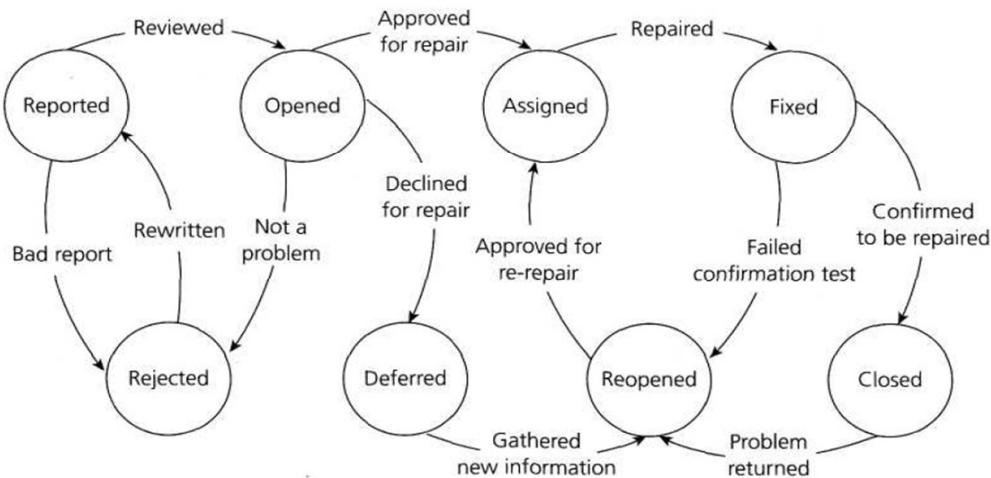
Saptanan hatalar grafiğine bakıldığında teste başlamakla birlikte hatalar bulunur genelde ilk hatanın bulunmasıyla hata oranında büyük bir artış olur ve bunların düzeltip kapatılmasıyla birlikte yeniden hata sayısı artar fakat bir süre sonra hata bulunmaz veya bulunma sıklığı çok azalmış olur. Yani testi sonlandırma aktivitesine işaret eder.

Fakat gerek test ortamının iyi olmaması gerek test caselerin iyi yazılmaması veya iyi bir test yaklaşımı uygulanmamasından veya testçilerin dikkatsizliğinden dolayı bazı hatalar canlı sisteme geçişten sonra kullanıcılar tarafından yakalanır. Testin verimliliğini ölçmek adına **Hata Yakalama Oranı** geliştirilmiştir. Formülde verildiği gibi teste saptanan hataların tüm hatalara oranı şeklindedir.

1	2	3
4	5	6

## 5.6. Hata Yönetimi

- **Hata Yaşam Döngüsü**

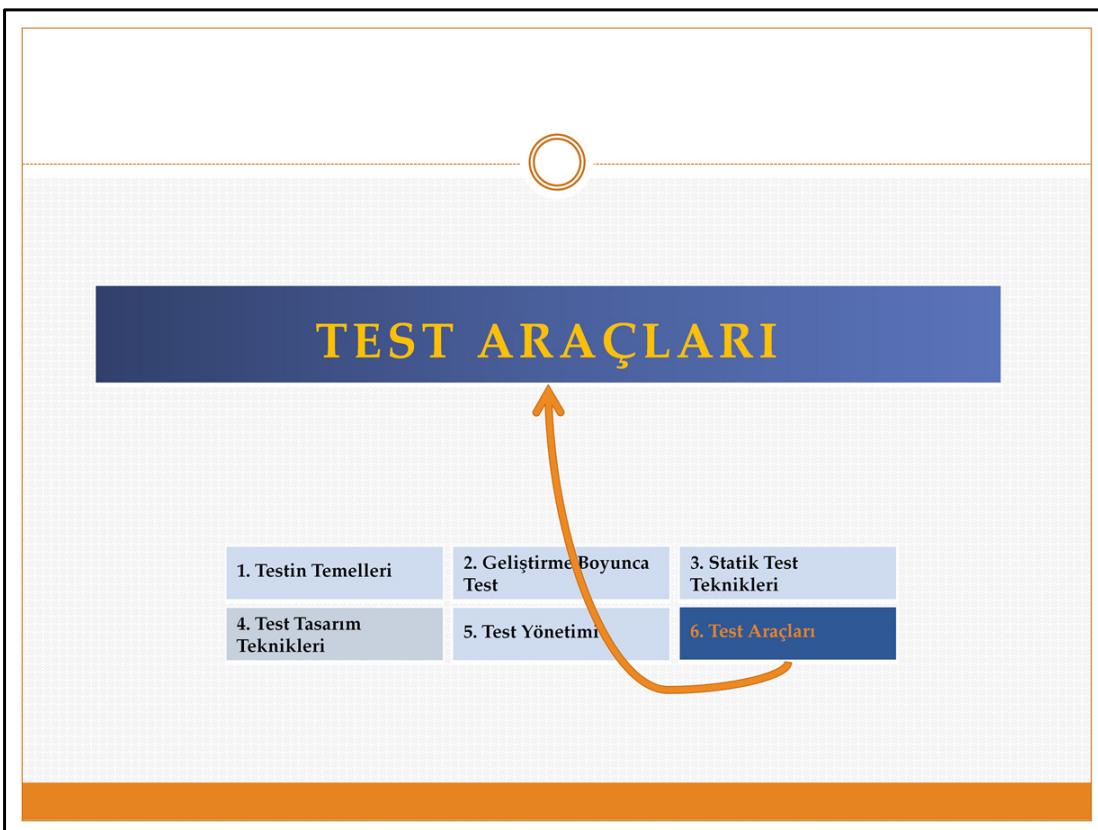


Herhangi bir vaka (incident) ile karşılaşıldığında raporlamadan önce hata olup olmadığından emin olmak gereklidir. Aksi durumda bulunan vakanın diğer kişiler üstüne boş bir uğraşı olacak ve hata değil denip kapatılacaktır. Bu durumda boş yere harcanmış emek ve testcinin itibar kaybı söz konusudur.

Saptanan gerçekten bir hata ise hatanın nasıl saptanıldığı anlaşılır bir şekilde raporlanmalı ve ilgili kişilere (yazılımcı veya organizasyona göre proje yönetcisi, yazılım lideri, vb) atanmalıdır (assign). Daha sonra hata çözüldüğünde hata **fixed** statüsüne çekilerek tekrar testciye atanıp test edilip soruna yol açmadığı anlaşıldığında hata testci tarafından **closed** statüsüne çekilebilir. Farklı durumlar şekilde anlatıldığı gibi olabilir.

## ISTQB Müfredatı

- Testin Temelleri
- Geliştirme Boyunca Test
- Statik Test Teknikleri
- Test Tasarım Teknikleri
- Test Yönetimi
- Test Araçları



1	2	3
4	5	6

## 5. Test Yönetimi

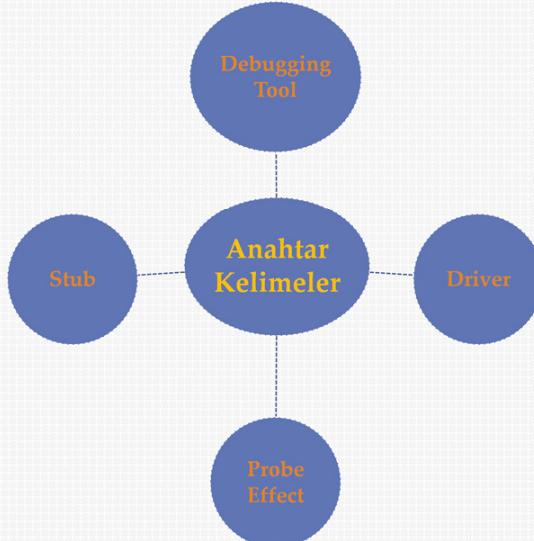
Çeşitli Test Araçları

Etkin Araç Kullanımı:  
Olası Risk ve Faydaları

Test Aracı İlk Kullanım

1	2	3
4	5	6

## 6.1. Çeşitli Test Araçları



### Anahtar Kelimeler

- **Debugging Tool (Hata Ayıklama Araçları – Debuging Araçları):** Programcıların hataları oluşturmak, araştırmak ve ayıklamak veya programları adım adım çalıştırırmak için kullandıkları araçlardır.
- **Driver:** Test sırasında olmayan bir birim veya bileşen yerine kullanarak testi gerçekleştirmeye yarayan test yazılım elemanları.
- **Probe Effect (Ölçücü Etkisi):** Test amacıyla kullanılan bir araç test edilen biriminin üzerine bir etki bırakır. Test sonuçlarına yansyan bu etkiye ölçücü (probe) etkisi denir. Performans ölçen araçların performansı ölçülen ürün üzerine olumsuz etkisi gibi. Kod coverage yapan bir aracın kodu incelemeye olması ve bazı kodları dahil edememesi gibi durumda başka bir örnektir. Daha uç bir örnek ise debugging yapan bir aracın varlığında hatanın bulunamaması yani hatanın oluşması için debuggerin çalışmıyor olması koşuludur. Bu **probe effect** Heisenberg'in uncertainty principle (belirsizlik kuralı) ile **Heizenbugs** adını almıştır.
- **Stub:** Yazılımın işletilemeyen kısmında tanımlı bir fonksiyon görevi gören ve o fonksiyon çalıştırılmadan programının istediği sonucu üreten kod parçalarıdır. Programı test etmek amacıyla doğru çalıştığından emin olunmayan fonksiyonları taklit ederek geriye kalan kısımlar test edilir.

1	2	3
4	5	6

## 6.1. Çeşitli Test Araçları



- Test Yönetim Araçlarının Özellikleri

- Test Yönetimi

- Test Verisi Takibi
    - Test Koşturulacağı Ortamlar
    - Planlanan, Yazılan, Koşturulan, Başarılı, Başarsız Test Case Sayısı

- Çalıştırılacak Testlerin Planlanması

- Manuel
    - Test Araçlarıyla

- Test Aktivitelerinin Yönetimi

- Harcanan Zaman
    - Plana Uyumluluk

Test yönetimi araçları genel olarak testleri işleyişi ile ilgili genel ve detay seviyede bilgiler veren araçlardır. Test execution (çalıştırılması) yoktur fakat başka amaçlar için kullanılan test araçlarına entegre açışarak gerekli yönetimsel verileri çekebilir.

Test yönetimi araçlarının esas amacı test sürecini yönetmek yani test planına sadık kalınarak testlerin işleyip işlemediğini takip etmek ve gerekli müdahaleleri yapmak için kullanılır.

1	2	3
4	5	6

## 6.1. Çeşitli Test Araçları



- Test Yönetim Araçlarının Özellikleri
  - Diğer Test Arçalarına Bağlantı
    - Test Execution Araçları
    - Vaka Yönetimi
    - Gereksinim Yönetimi
    - Konfigürasyon Yönetimi
  - İzlenebilirlik
    - Test Sonuçları
    - Hatalar
  - Test Sonuçlarına Erişim
  - Raporlama
    - Başarılı / Başarısız Test Run
    - Çözülmüş / Bekleyen / Kapatılan Hataların Oranı

Test yönetim araçları ayrıca yönetimsel bazı metrik raporlarının hazırlanmasına olanak sağlamalıdır. İlgili verileri diğer test araçlarından çekebilir ve buradan alınan verileri organizyonel ihtiyacı karşıယacak şekilde raporlar sunabilir.

En önemli raporlar yine test yönetimi ile ilgili olanlardır. Çalıştırılan / çalıştırılmayan test case sayısı, başarılı / başarısız test case sayısı, çözülmüş / bekleyen / reddedilen / kapatılan hata sayıları ve test için planlanan sürenin nekadar etkin kullanıldığına yönelik metrikler içerebilir.

1	2	3
4	5	6

## 6.1. Çeşitli Test Araçları



- Gereksinim Yönetimi Araçlarının Özellikler
  - Gereksinimleri depolar
  - Gereksinimlerin özelliklerini ile bilgiler
  - Gereksinimlerin tutarlığını kontrol eder
  - Önceliklendirme
  - Durumlarını gösterme
  - İzleneme
  - Test yönetim araçlarıyla entegrasyon
  - Gereksinimlerin kapsamını (coverage) gösterme

Gereksinim yönetim araçları yazılı gereksinimleri depolayarak gerekli hallederde durumlarına, özelliklerine , öncelik (priority) değerine göre çeşitli filtrelemeler yapan test aracıdır. Gereksinim yönetim toolarının genel amacı gereksinimleri toplamak ve test case hazırlamasına yardımcı olmaktadır. Yani test yönetim/test execution araçlarına entegre olmalıdır. Bu açıdan bakıldığından kimilerine göre gereksinim araçları test yönetim aracı değildir fakat önemli olan nokta testler hazırlanmış gereksinimlere göre yapılır (verification). Bu yüzden gereksinim yönetim araçları test için önemlidir. Bazı gelişmiş test yönetim araçları gereksinimler içerisindeki muallak ifadeleri kontrol ederek hatalı isteklerin çıkışmasını engelleyebilir.

Gereksinimlerde olabilecek değişimleri güncelleme imkanı sunmalıdır. Ek araçlar yardımıyla gereksinimlerin kapsamını gösterebilen gereksinim yönetim araçlarında vardır.

1	2	3
4	5	6

## 6.1. Çeşitli Test Araçları



- Vaka Yönetimi Araçlarının Özellikler
  - Vakanın özelliklerini kayıt eder
  - Dosya ekleme olanağı
  - Hataları önceliklendirme
  - Atama özelliği
  - Durum
    - Open
    - Rejected
    - Duplicate ...
  - Raporlama

Türkçesi **vaka yönetimi** olarak çevrilen **incident management** araçları ayrıca; defect-traking, defect-management, bug-traking veya bug management tool olarak da bilinir. Test süreçleri boyunca saptanan her vakanın hata, bug, defect olarak isimlendirmenin doğru olmayacağı için **incident management tool** demek daha mantıklıdır.

Bu araçların görevi saptanan her türlü vakanın kaydedilmesine, takip edilmesine ve raporlanması olanak sağlamaktır. Vaka yönetimi araçları vasıtasıyla saptanan hatalar çözülmek üzere ilgili kişilere atama yapmaya olanak sağlamalıdır bu yüzden test yönetimi veya mail serverleri ile entegre çalışabilmelidir. Ayrıca hata yaşam döngüsündeki tüm durumlarda gösterebilecek yetekte olmalıdır.

1	2	3
4	5	6

## 6.1. Çeşitli Test Araçları



- Konfigürasyon Yönetimi Araçlarının Özellikler
  - Testware ve software sürüm kontrolü
  - Farklı sürümler arasında izlenebilirlik
  - Sürüm ve konfigürasyon bağlantısını kurmak
  - Sürüm ve sürüm adayı (release candidate) yönetimi
  - Baselining (özel sürümler üretme)
  - Yazılıma müdahaleleri kontrol eder
    - Check in
    - Check out, merge, push istekleri

Konfigürasyon yönetimi direk olarak test yönetimini içeren bir araç değildir fakat iyi bir test organizasyonu için olmazsa olmazdır. Test açısından bakıldığından konfigürasyon yönetimi tüm test araçlarının (testware) ve test edilen yazılım ürünlerinin (software) sürümlerini yönetmeye yarayan sistem araçlarıdır. Doğru yazılım sürümünün doğru testware sürümünde test edildiğindenin kontrolü yine konfigürasyon yönetimin araçlarının özellikleridir.

1	2	3
4	5	6

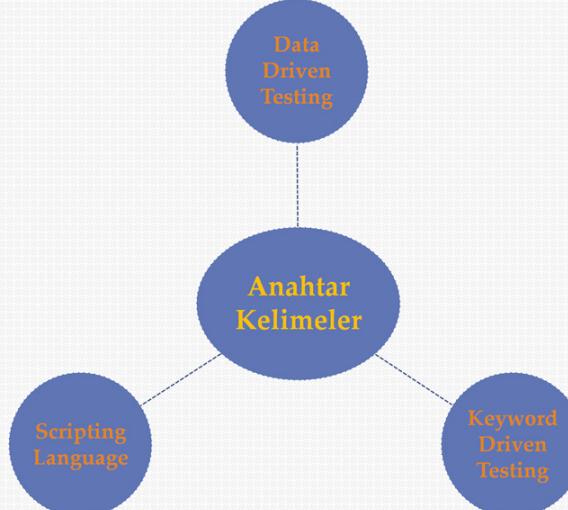
## 6.1. Çeşitli Test Araçları



- Diğer Test Araçları
  - Statik Test Araçları
    - Gözden Geçirme Araçları
    - Statik Analiz Araçları
    - Modelleme
  - Testcase Destek Araçları
    - Test Tasarım Araçları
    - Test Data Oluşturma Araçları
  - Test Çalıştırma Destek Araçları
    - Test Çalıştırma Araçları
    - Birim Test Yardımcı Araçları
    - Test Sonuçları Karşılaştırma Araçları
  - Performans Ölçüm Araçları
  - Güvenlik Test Destek Araçları

1	2	3
4	5	6

## 6.2. Etkin Araç Kullanımı



### Anahtar Kelimeler

- **Data Driven Testing:** Testler otomasyonla işletilir. Test için veriler giriş (input) ve çıkış (expected result) bir tabloya girilir ve otomasyon programı buradaki verileri kullanarak yazılım test edilir.
- **Keyword Driven Testing:** Fonksiyonel bazlı kelimeler üzerinden test case oluşturularak otomatik test yöntemidir. **Login** - **createUser** - **logout** gibi her bir kelimenin kendisi test case olacak şekilde test scriptleri hazırlanır ve senaryoya uygun kombinasyonlarla test oluşturulur.
- **Scripting Language (Scripting Dili):** Test otomasyonunda kullanılabilen programlama dili.

1	2	3
4	5	6

## 6.2. Etkin Araç Kullanımı



- Test Araçlarının Olası Faydaları
  - Yinelenen İşlerde Azalma
  - Daha Fazla Tutarlılık
  - Tekrar Çalışabilir İşler
  - Nesnel Değerlendirme
  - Testle İlgili Bilgilere Kolay Erişim

Test araçları sıkıcı olan ve manuel işletilmesi çok zaman alan, hata riski fazla olan veya manuel işletilmesine olanak olmayan işler için sağlayacağı avantajlar vardır. Bu tür işleri test araçları vasıtasiyla daha kısa zamanda daha güvenilir yapmak mümkün olabilir. Burada kazanılan zamanla zarfında testçiler farklı testler veya analizler yaparak ürün kalitesi üzerine katkı sağlayabilir.

Örneğin bir e-ticaret sitesi için fazla sürüm çıkmak (yazılım güncellemesi) olur ve firma için en büyük risk taşıyan **satınalma, yeni üyelik, siteye giriş** gibi test caselerin otomasyon yardımıyla yapmak hem zamandan kazanç sağlar hem de testin güvenirliliğini arttırr. Farklı bir açıdan sitenin performansı test edilecek yanı aynı anda kaç kişiye hizmet verilebiliyor bu kontrol edilecek. Böyle bir durumu araçsız test etmek mümkün değildir. Performans araçlarıyla istenilen sayıda sanal kullanıcı oluşturularak değişik senaryolar test edilebilir. Test araçları ayrıca uygulama üzerine yapılmış bu gibi bilgileri barındırır ve geçmiş-gelecek bağlantısıyla daha nesnel değerlendirmeler yapmaya yardımcı olur.

1	2	3
4	5	6

## 6.2.Etkin Araç Kullanımı



- Test Araçlarının Olası Riskleri

- Gerçekçi olmayan bekentiler
- Aracın ilk kullanımına fazla zaman harcamak
- Araç kullanımının devamlılığı için fazla zaman harcamak
- Araçlara fazla güvenmek
- Test araçlarının ürettiği hedefler için fazla zaman harcamak

Test aracı genel olarak işleri kolaylaştırmak ve daha kısa sürede daha kaliteli ürünler ortaya koymak amacıyla kullanılır. Fakat kullanılan test aracının amaç – araç ilişkisini iyi kurmak gereklidir yani amaç aracı kullanmak mı yoksa amaç kaliteyi yükseltmek mi. Her iki durumda araçtan bekentiler farklılık gösterir. Tabiki amaç kaliteyi yükseltmektir yani araç bir şekilde daha fazla emek harcamamızı ve aynı sonucu almamızı yolaçıyor ise aracı daha fazla kullanmak amaçımızın araç olduğu anlamına gelir.

Bazende kalite veya yazılım süreçlerindeki hatalar araç olmamısına dayandırılıarak herşeyi meşhur, pahalı bir araç alınarak çözülmeyeceği yönünde olur. Bu durumda araç süreci değiştiremeyeceğinin araçtan bekentiler gerçekçi olmayacağındır. Ayrıca araç iyi bilinmeli yapmayıcağı işleride aractan beklenilmemelidir. Nasılsa bu işi araç hızla yapıyor denilerek plan yapılmamalıdır.

Süreç düzgün işliyor ve ihtiyaca uygun bir araç seçilmiş olsa bile bazı sorunları irdelemek gereklidir. Örneğin yeni bir aracın firma içerisinde ilk kez kullanılmaya başlanması bir acemilik dönemi denektir ve bazı durumlarda ise eğitim gereklidir. Bununla birlikte araç kullanılmaya başlandıktan sonra devamlılık için bakım ve ilerletme çalışmaları içinde belli bir emek harcamak gereklidir. Harcama emeğin miktarı ve sağladığı yararlar birbirine kıyaslanmalıdır.

1	2	3
4	5	6

## 6.2. Test Aracı İlk Kullanım



- Test aracı secimi ve önemli noktalar
  - Organizasyon değişime hazırlığı
  - Aracın organizasyonda geliştireceği süreçler
  - Aracın tarafsız değerlendirilmesi
  - Aracın beklenilere cevabı
  - Tedarikçinin kalitesi
    - Eğitim
    - Destek
    - İtibar
    - Kullanıcı forum/yorumları (açık kaynaklar için)
  - İlk kullanım
    - Destek
    - Sıkıntılı noktalar
    - Mentor

Test aracı seçilmeden önce organizasyon incelenmeli ve gerçekten neye ihtiyaç duyulduğu tarafsız olarak değerlendirilmelidir. Ayrıca araçla birlikte süreçte ve bilgi seviyesinde yaşanacak değişimlere kullanıcıların ne kadar hazır olduğu ve kullanmak için gerekli donanıma (bilgi, deneyim, kariyer planı, eğitim isteği) sahip olup olmadığı gibi sorularda cevap bulmalı.

Piyasada test ürünlerinin sayısı çok fazladır bu yüzden tercih edilecek ürün tarafsız olarak değerlendirilmeli ve ihtiyaçlara cevap verip vermediği araştırılmalıdır. Tercih edilecek ürünün ilk kullanımın yaşanacak sıkıntılar çözülemiyorsa destek alınabilecek ürünler tercih edilmelidir. Ayrıca ürünlerin kullanıcı sayısı, eğitim dökümanları, sektördeki itibarı, forum siteleri, gelişmeye açılığı gibi faktörlerde göz önünde tutulmalıdır.