

# **SDLC**

## **Software Development Life Cycle**

### **(Yazılım Geliştirme Yaşam Döngüsü)**



**1. Ders**  
**28/10/2021**

**T E C H P R O E D**

# **SDLC**

## **Software Development Life Cycle**

### **(Yazılım Geliştirme Yaşam Döngüsü)**

SDLC, yazılım ürünlerini geliştirmek veya değiştirmek için planlanan yazılım geliştirme sürecidir. Yazılım ürünlerinin nasıl geliştirilmesi gerektiğini veya nasıl iyileştirilmesi gerektiğini anlatan ayrıntılı bir plan içerir.

SDLC, yüksek kaliteli (high quality) ve kullanıcı beklerini (user expectation) karşılayan yazılımları tasarlamak, geliştirmek ve test etmek için yazılım endüstrisi tarafından kullanılan bir süreçtir.

# HARDWARE, SOFTWARE (Donanım, Yazılım)

Donanım; kasa, merkezi işlem birimi, monitör, fare, klavye, bilgisayar veri depolama, grafik kartı, ses kartı, hoparlörler ve anakart gibi bir bilgisayarı oluşturan fiziksel genel adıdır. genellikle yazılım herhangi bir komut veya talimatı çalıştırma mak üzere yönlendirilir.



# HARDWARE, SOFTWARE

Yazılım, değişik ve çeşitli görevler yapma amaçlı tasarlanmış elektronik aygıtların birbirleriyle haberleşebilmesini ve uyumunu sağlayarak, görevlerini ya da kullanılabilirliklerini geliştirmeye yarayan makine komutlarıdır.

Software



# NEDEN SDLC ?

DİJİTAL 

## Güvenlik açığı ile dünya çapında ses getiren Twitter'ın piyasa değeri düştü

Twitter'ın hisseleri borsa kapanışının ardından yüzde 4 düşüş gösterdi.



Tuğçe İçözü  
16 Temmuz 2020



We are aware of a security incident impacting accounts on Twitter. We are investigating and taking steps to fix it. We will update everyone shortly.

ÖÖ 12:45 · 16 Tem 2020 · Twitter Web App

30 B Retweet 7.288 Alıntı Tweetler 111,7 B Beğeni

**Boeing'in önce uçakları sonra hisseleri düştü 25 milyar dolar kaybetti**



Paza günü Etiyopya'da 157 kişinin ölümüne neden olan Boeing 737 MAX uçak kazası sonrası ülkeler bir bir uçuşlarını durdururken şirketin hisseleri borsada yere çakıldı.

# NEDEN SDLC ?



[Fotoğraf: AA]

ABONE OL

Google News

Takip et: @trhaber

Sosyal medya platformları Whatsapp, Instagram ve Facebook'a dünya genelinde yaşanan erişim sorunu devam ediyor. Facebook'un hisseleri yüzde 5'in üzerinde düştü.

**Yemek Sepeti çöktü: Hem internet sitesine, hem de uygulamaya ulaşılamıyor**

18:37 10.01.2021 (güncellendi: 04:48 11.01.2021)



© Fotoğraf

Abone ol

Google News

Hem yemeksepeti.com hem Yemek Sepeti IOS ve Android uygulamalarına erişim sağlanamadı. Firmadan henüz bir açıklama yapılmadı.

# NEDEN SDLC ?

**Ünlü Alışveriş Sitesi Amazon Çöktü,  
Ürünler 3,6 Kuruştan Satıldı**



Haberler.Com - Haberler | Güncel  
16 Aralık 2014 Salı 13:02

**ABD'li online alışveriş devi Amazon.com'da yaşanan bir hacker skandalı nedeniyle 20 bine yakın ürün, 3,6 kuruştan satıldı.**

Ünlü alışveriş sitesi Amazon.com'un İngiltere operasyonunda yaşanan bir hacker skandalı, 20 bine yakın ürünün 1 penny'lik (yaklaşık 3.6 kuruş) fiyattan satılmasına yol açtı.

**Akbank'tan yeni açıklama!  
Akbank sistem arızası düzeldi mi, sorun devam ediyor mu?**

Akbank kullanıcıları salı gününden bu yana mobil ve internet bankacılığında sorun yaşıyordu. Öte yandan ATM ve pos cihazlarında da hata olduğu öne sürüldü. 'Oturum kapandı' sorunu ile karşılaşan vatandaşlar Akbank internet bankacılığı çöktü mü, siber saldırıya mı uğradı sorularını gündeme getirdi. Akbank'tan konu hakkında açıklama geldi.

• 09 Temmuz 2021 - 00:42 • Son Güncelleme: 09 Temmuz 2021 - 00:42



# Software Hatalarının Sonuçları

NASA, 4 Haziran 1996'da fırlatılması planlanan Ariane 5 uzay aracını kodlarken, Ariane 4 roketinin kodlarını kopyalayarak, bir hata yaptığıının farkında değildi. O gün fırlatma için geri sayım yapıldı ve roketin motorları ateşlenerek kalkış başladı. Hızlanarak yoluna 37 saniye boyunca devam eden Ariane 5 roketi; o saniyeden sonra yanlış yöne doğru 90 derece dönmeye başladı. Bu durum, roketin kendini imha etme mekanizmasını tetikledi. Bu kaza, NASA'ya 370 milyon dolara mal oldu.

T E C H P R O E D

# Software Hatalarının Sonuçları

Therac-25 makinesi, kanser hastalarının tedavisinde kullanılmak için tasarlanmıştı. Yapılan değişiklik Therac-20'de güvenlik önlemi olarak bulunan elektromekanik güvenlik kilitlerinin yazılımsal güvenlik önlemleriyle değiştirilmesiydi. Ne yazık ki gelişim olarak görülen bu hata, Therac-20'nin kodlarında bulunan ancak fark edilemeyen hatanın, Therac-25'te ortayamasına sebep oldu. Bu bug yüzünden yaklaşık 5 hastanın ağır dozda radyasyon sebebiyle hayatını kaybettiği raporlandı.

T E C H P R O E D

# Software Hatalarının Sonuçları

Dakikalar içinde kaybedilen 460 milyon dolar. New York borsasında piyasaları yönlendiren en büyük şirketlerden biri olan Knight Capital, 1 Ağustos 2012'de yeni bir yazılım güncellemesi yapma kararı aldı. Saat 09.00 sularında New York Borsası işlemler için açıldı ve Knight Capital'ın yatırımcıları, varlıklarını satmak veya almak için talimat verdi. Yalnızca 45 dakika sonra Knight Capital'ın sunucuları 4 milyon işlem gerçekleştirdi ve şirkete 460 milyon dolar kaybettirerek iflasın eşiğine getirdi.

Olay, bir teknisyenin yeni Perakende Likidite Programı (RLP) kodunu, Knight'in hisse senedi siparişleri için otomatik yönlendirme sistemi olan sekiz SMARS bilgisayar sunucusundan birine kopyalamayı unutması sonrasında

# Software Hatalarının Sonuçları

Mars Climate Orbiter Hatası (23 Eylül 1999): Gezegenler arası ilk iklim uydusu olarak 1997'de fırlatıldı. Mars Orbiter, 1999'da Mars'ın yörüngesinde kayboldu. Kazanın yazılımda kullanılan İngiliz ölçü birimlerinin metrik sisteme yanlış çevrilmesinden kaynaklandığı belirtildi. NASA'da bir ekip hesaplarında İngiliz ölçü birimini (inç) kullanırken, projeye katılan diğer ekipse metrik (cm) sistemi kullanmıştı. 125 milyon dolarlık uydunun yörüngeye sabitlenmeye çalışırken Mars'a olması gerekenden fazla yaklaşarak imha olduğu düşünülüyor.

TECHPRO EDO

# Software Hatalarının Sonuçları

1991 yılının şubat ayında gerçekleşen Körfez Savaşı sırasında ABD'nin Suudi Arabistan'ın Zahran şehrindeki üssünde bir patlama yaşandı. Patlamanın sebebi ise üste bulunan anti balistik füze sisteminin doğru çalışmamasıydı. Yapılan sorgulamaların ve araştırmaların sonucunda patlama sebebinin üste bulunan anti balistik füze sisteminin bir yazılım hatası yüzünden ateşlenmemesi olduğu anlaşıldı.

Bir insan için inanılmaz küçük olan 0,33 saniye, Al Hussein füzesini takip etmek için yapılan bir sistem için inanılmaz büyük bir hataydı. MIM-104 Patriot, havada bir cisim olduğunu algılamayı başardı ancak bug yüzünden cismi takip edemedi ve bunun bir füze olduğunu anlayamadı. Engellenemeyen füze yüzünden üste bulunan 28 asker hayatını kaybetti.

# NEDEN SDLC ?

- Internet browserlarının yıllara göre kullanımları

<https://www.visualcapitalist.com/internet-browser-market-share/>

- Dünyada en çok ziyaret edilen 50 site

<https://www.visualcapitalist.com/the-50-most-visited-websites-in-the-world/>

- Dünyanın en zengin kişileri

<https://www.visualcapitalist.com/top-10-richest-people-worldwide-2021/>

**User Expectation:** Kullanıcı Beklentisi

**High Quality:** Yüksek Kalite

T E C H P R O E D

# SDLC'NİN FAYDALARI

- 1) Projenin takibini ve kontrolunu saglar.
- 2) Tüm Planlama ve Process'in yatirimcilar tarafından gorulebilmesine imkan tanır.
- 3) Yapılan planlama ve toplantılarla projenin oluşturma ve geliştirme hızını artırır.
- 4) Tüm ekibin iletişimini güçlendirir.
- 5) Projenin risklerini azaltır.

Doğru yapılan SDLC , en yüksek düzeyde yönetim kontrolüne ve dokümantasyona izin verebilir . Geliştiriciler neyi neden inşa etmeleri gerektiğini anlarlar. Tüm taraflar hedef üzerinde önceden hemfikirdir ve bu hedefe ulaşmak için net bir plan görür. Herkes gerekli maliyetleri ve kaynakları anlar.

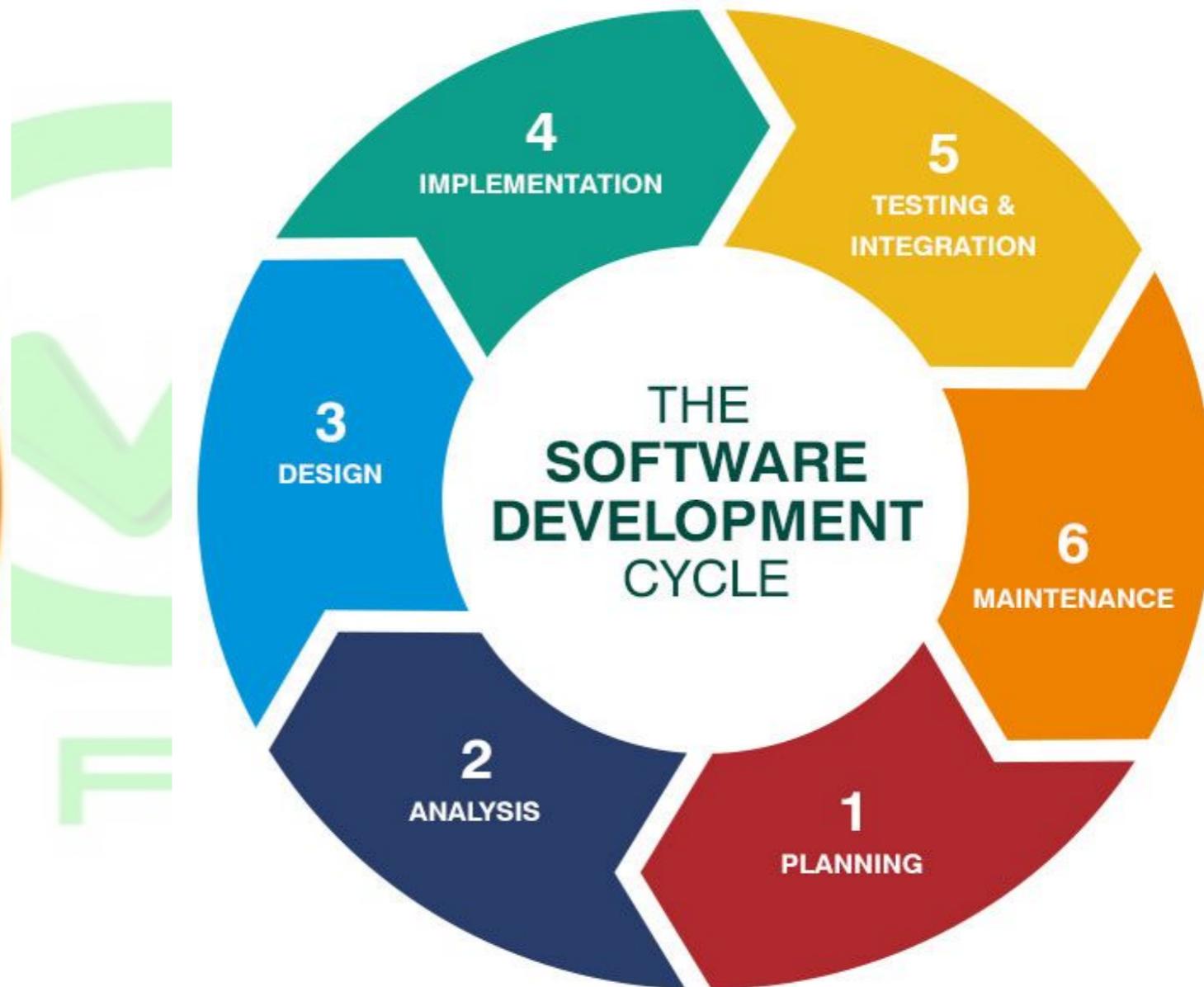
# SOFTWARE DEVELOPMENT PHASES

## (Yazılım Geliştirme Aşamaları)



# SOFTWARE DEVELOPMENT PHASES

## (Yazılım Geliştirme Aşamaları)



# SOFTWARE DEVELOPMENT PHASES (Yazılım Geliştirme Aşamaları)

**1) Planlama (Planning):** SDLC'nin ilk aşaması planlamadır. Fizibilite çalışmaları yapılır. Projenin ihtiyaçları belirlenir, maliyetler hesaplanır, projenin faydaları ve riskleri hesaplanır. Müşteri ile birlikte gereksinimler belirlenir.

**2) Analiz (Analysis):** Sistemin işlevlerinin ve gereksinimlerin ayrıntılı olarak incelenmesidir. Bu aşamada elde edilen veriler doküman haline getirilir. Ayrıca bu aşamada ekip çalışması çok önemlidir. Müşteri (steakholder), developer, sistem analisti(Tester, QA), iş analisti (Business Analyst, BA), Proje yöneticisi (Project Manager, BM) vb. rollere sahip kişiler bir araya gelerek çalışmayı sürdürür. Ekip arası iletişimin iyi olması, çalışma ortamında doğru kararlar verilmesine ve projenin iyi yönetilmesine yardımcı olur.



**3) Tasarım (Design):** Bu aşamada gereksinimlerinin analiz edilmesiyle yazılım sistemi tasarılanır. Tasarım aşamasında kodlama yapmak söz konusu değildir. Analiz aşamasında problemin ne olduğu belirlenirken tasarım aşamasında problemin nasıl çözüleceği belirlenir. Gereksinimleri karşılayan yazılım ürününün özellikleri , arayüzler ve yazılım ürününün faydaları , yetenekleri belirlenir.

Mimari tasarımda yazılım ürününün genel bir planı yapılır ve modüller belirlenir. Ayrıntılı tasarımda yazılımda kullanılacak algoritmalar, programlama dilleri , veritabanları ve bunun gibi detaylar belirlenir.



**4) Kodlama (Implementation):** Kodlama sürecinin başladığı aşamadır. Bu aşamanın önemli noktalardan birisi doğru kodlama yapılmasıdır. Doğru kodlama şekli bir başkasının da rahatça okuyabileceği ve bakım yapabileceği kodlar yazmaktır. Bu kodlama biçimini temiz kodlama (clean code) olarak adlandırılır.

**5) Test (Testing):** Bu aşamada gerçekleştirilmesi gereken önemli noktalardan diğerı test etmektir. Kodlama yapılırken ve kodlama sonrasında birçok test yapılır. Bu testlerden bazıları ; birim testleri, zorlanım-performans testi , yanlış değer testleri, tümleyim testi, kullanım senaryo testleri, yük testleri, kullanıcı kabul testi, yoldan geçen adam testi, test otomasyonu gibi testlerdir. Ayrıca analiz aşamasından itibaren testler yapılması yazılım ürününde hata oranını azaltacağı için kaliteyi arttırmış , maliyetleri(para ,zaman vb) azaltır. Bu yaklaşım erken test yaklaşımı (early testing) olarak adlandırılır.

**6) Teslim ve Bakım (Maintenance):** Tüm aşamaların tamamlanmasının ardından yazılım ürünü müştereye teslim edilir. Ürün teslim edilirken ürünle ilgili bilgiler , kullanım kılavuzu da müştereye teslim edilir. Ürünün teslim edilmesi ile birlikte bakım aşaması başlar ve yazılım ürününün ömrü süresince devam eder. Bu süreçte hata giderme, altyapıları iyileştirme , ürüne yeni özellikler ekleme gibi bakım faaliyetleri yapılır.

TECHPRO EOOD

# SDLC TEAM

- 1) Project Manager (PM)  
Proje Yöneticisi
- 2) Business Analyst (BA)  
İş Analisti
- 3) Developer (Dev)  
Yazılımcı
- 4) Quality Analyst (QA)  
Kalite Analisti (Tester)

TECH F



# PROJECT MANAGER (PM)

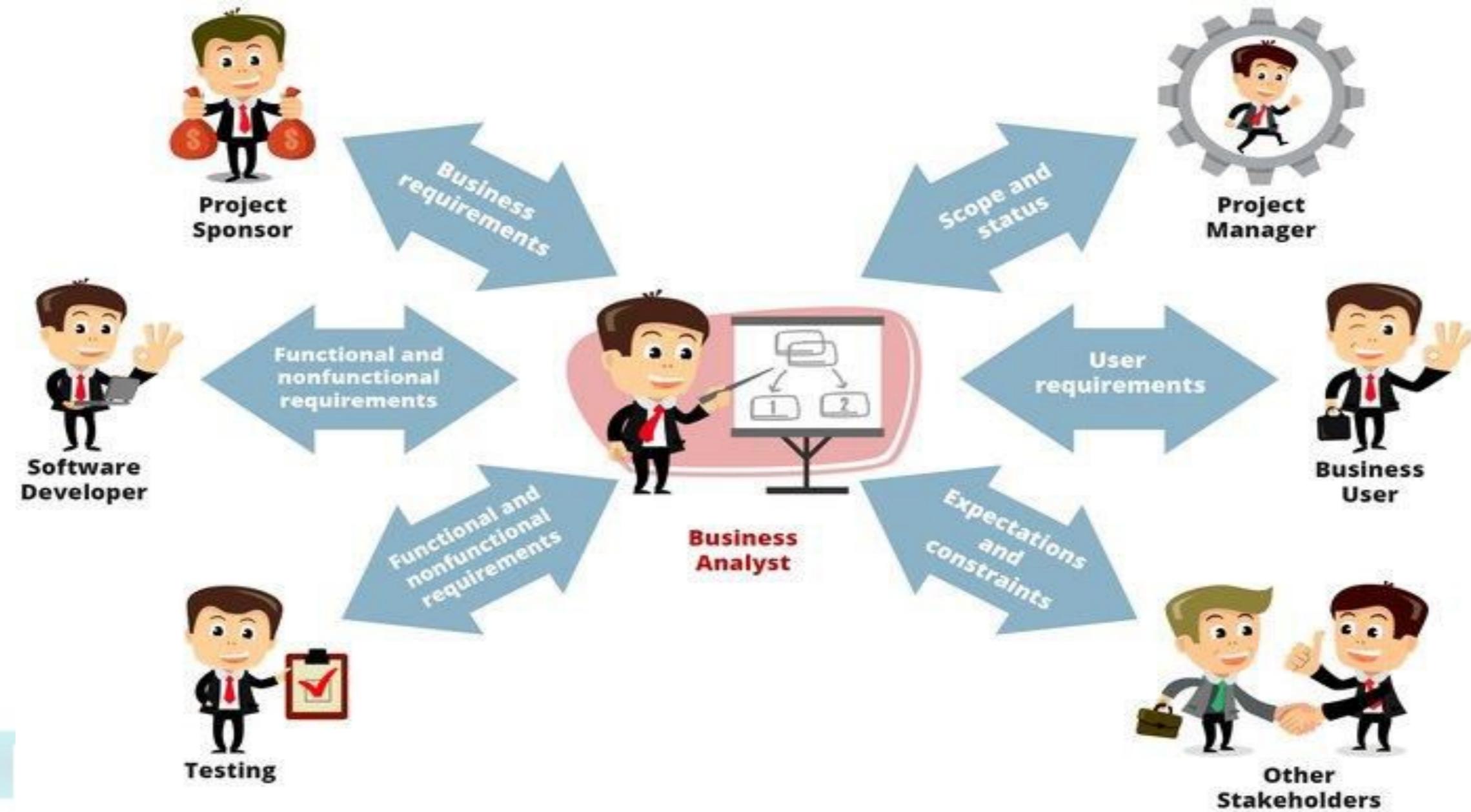
Project manager: Proje yöneticisi, takımındaki herkesin rolünü bilmesini ve yerine getirmesini ve bu rollerin gerçekleştirileceği inancına göre hareket etmesini sağlar.

- Proje planının geliştirilmesinden sorumludur
- Proje sahipleri (Stakeholder) ile yakın ilişki kurar
- Takım içerisindeki iletişimini sağlar
- Proje riskini yönetir
- Proje çizelgesini hazırlar
- Proje bütçesini yönetir
- Projede çıkabilecek karışıklıkları (conflicts) önler (Kriz yönetiminden sorumludur)
- Görev dağılımını yönetir.

# BUSINESS ANALYST (BA)

Şirketlerin, iş süreçlerini değerlendirme, gereklilikleri öngörme, iyileştirme alanlarını açığa çıkarma ve çözümler üretme faaliyetlerini yürütür. Bir proje veya programın ihtiyaçlarını belirleyerek, bunları yönetici ve ortaklara iletir. İş sorunlarına teknik çözümler geliştirmek için çalışır.

- Business sorunları ve teknoloji çözümleri arasında bir köprü vazifesi görür
- Requirements (Gereksinimler) yönetimini ve iletişimini sağlar.
- Alınan kararların anlaşılır bir dile dökülmesini sağlar.
- **Business Requirement Document (BRD)** oluşturur.  
(alınan tüm kararların ve gereksinimlerin dokümü)
- **Functional Requirement Document (FRD)** oluşturur.  
(yazılımı yapılacak olan bütün maddelerin dokümü)
- Yeteri kadar Functional Requirement toplandıktan sonra **use cases** oluşturur
- Akış şemasını oluşturur



# DEVELOPER (DEV, Yazılımcı)

Developer yazılım programlarının arkasındaki yaratıcı beyindir ve bu programları oluşturmak veya bir ekip tarafından oluşturulmalarını denetlemek için teknik becerilere sahip olan kişidir.

## Sorumluluklar:

- Kendilerine aktarılan **software requirement dokümanını** toparlar ve gereken application ve programın oluşumunu sağlar.
- Beklentileri ve gereksinimleri (costumer requirement) karşılayacak yüksek kalitede (**High Quality**) code yazarlar.
- Software dokümanını oluşturur ve önceki dökumanları günceller.



# QUALITY ANALYST (QA, Tester)

Müşteri ve kullanıcı memnuniyetini göz önünde bulundurarak analiz ve test aşamalarında gerekli düzenlemeleri yapan kişidir.

- Son kullanıcıya bırakılmadan önce analiz ve testlerdeki tüm hataların düzeltilmesini sağlayarak hatasız ürünler sunmak.
- Herhangi bir organizasyonun ürünlerini ve hizmetini beklenen kalite standartlarını karşılayacak şekilde oluşturulmasını sağlar.
- Oluşturulan application'ın istenilen plan çerçevesinde yapılmasını sağlar.
- Application'daki hatalar Quality Analyst tarafından bulunmalıdır ki Developer'lar bulunan hataların üzerinde çalışıp sorun teşkil etmeyecek ürün ortaya koyabilsevler (minimum seviyede bug).
- Testing yapılmasıının amacı herhangi bir application'da oluşabilecek hataların ortaya çıkarılmasıdır.



# Bugün Ne Öğrendik?

**SDLC:** Yüksek kaliteli (high quality) ve kullanıcı beküntilerini (user expectation) karşılayan yazılımları tasarlamak, geliştirmek ve test etmek için yazılım endüstrisi tarafından kullanılan bir süreçtir.

## SOFTWARE DEVELOPMENT PHASES (Yazılım Geliştirme Aşamaları)

Planlama (Planning)

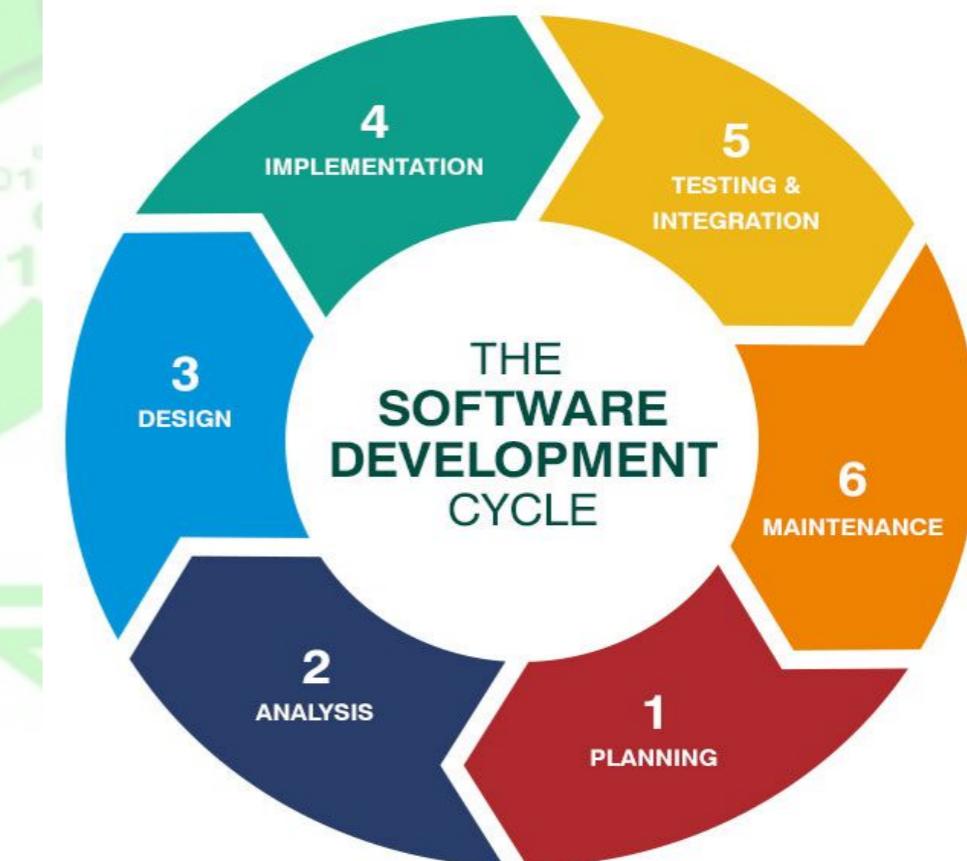
Analiz (Analysis)

Tasarım (Design)

Kodlama (Implementation)

Test (Testing)

Teslim ve Bakım (Maintenance)



# Bugün Ne Öğrendik?

## SDLC TEAM

- 1) Project Manager (PM) Proje Yöneticisi
- 2) Business Analyst (BA) İş Analisti
- 3) Developer (Dev) Yazılımcı
- 4) Quality Analyst (QA) Kalite Analisti (Tester)

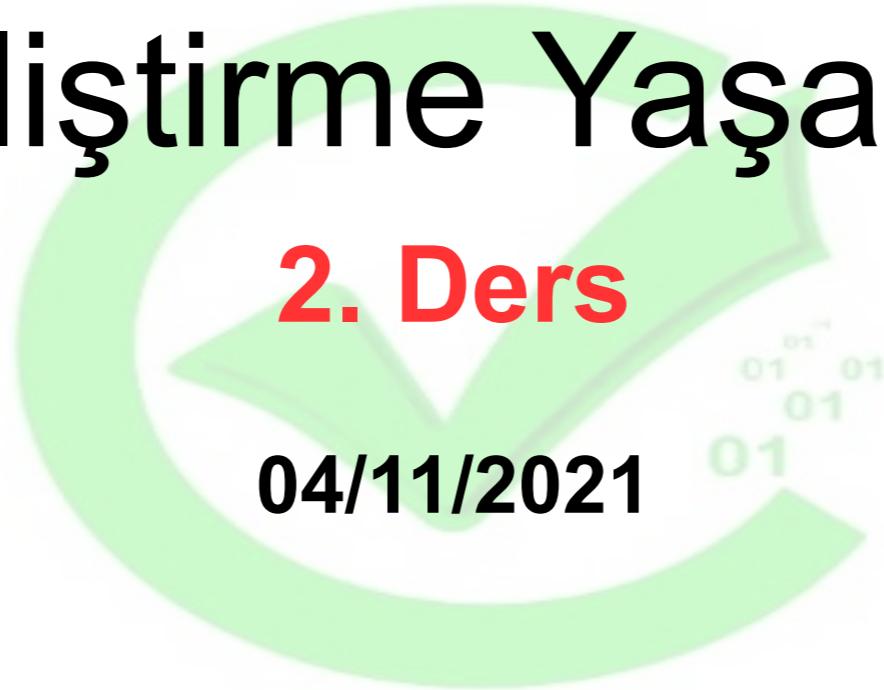
**Business Requirement Document (BRD):** Alınan tüm kararların ve gereksinimlerin dökümü

**Functional Requirement Document (FRD):** Yazılımı yapılacak olan bütün maddelerin dokümü

# **SDLC**

# **Software Development Life Cycle**

## **(Yazılım Geliştirme Yaşam Döngüsü)**



**2. Ders**

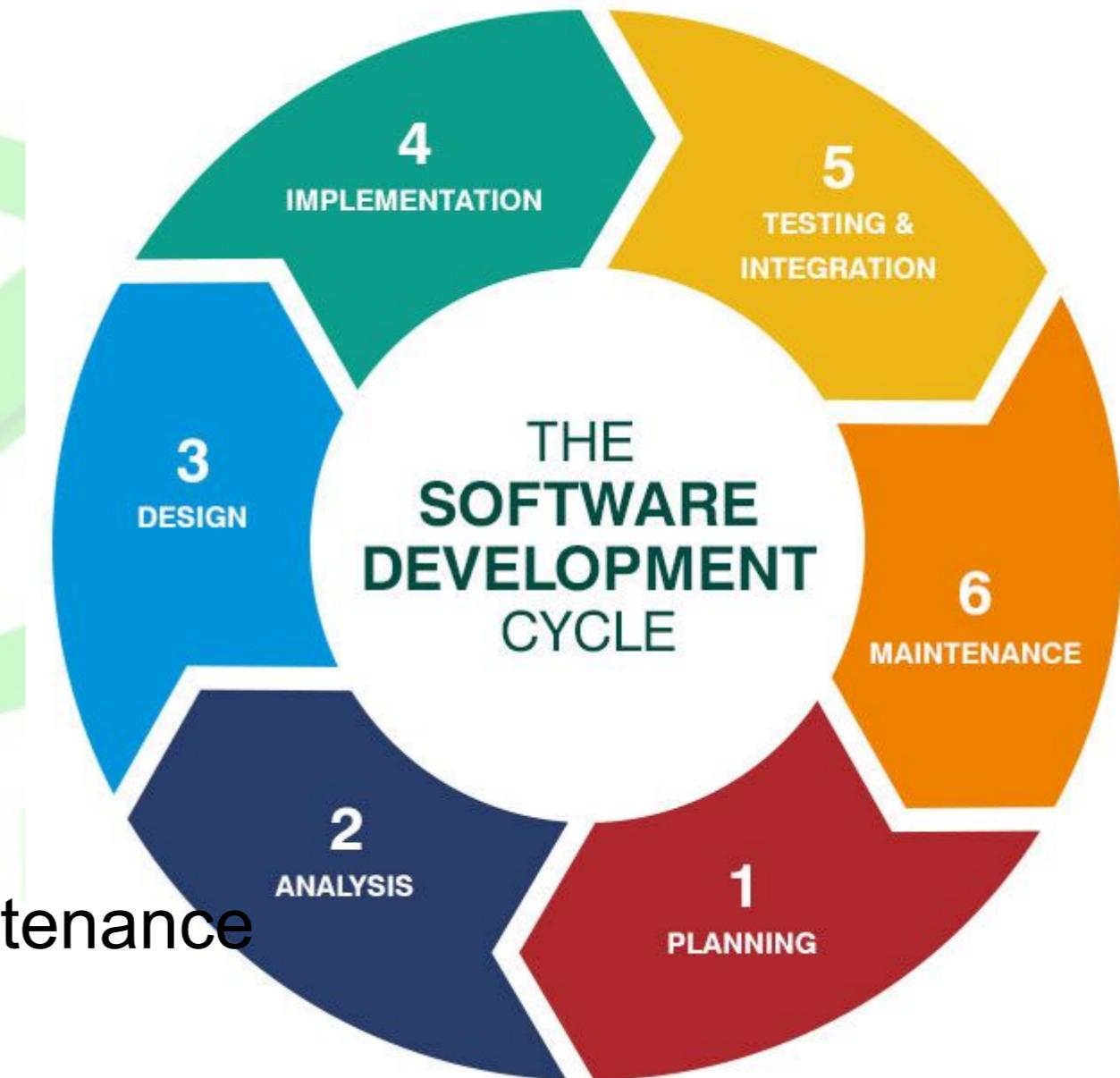
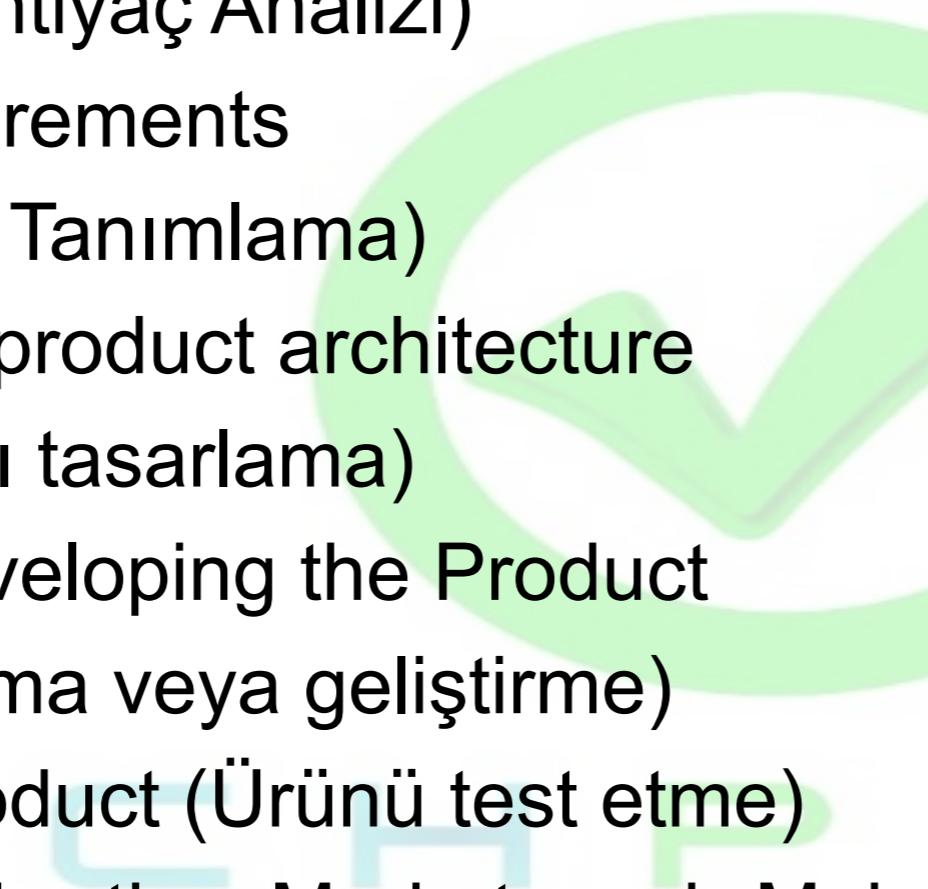
**04/11/2021**

**T E C H P R O E D**

# SOFTWARE DEVELOPMENT PHASES

## (Yazılım Geliştirme Aşamaları)

- 1) Planning and Requirement Analysis  
(Planlama ve İhtiyaç Analizi)
- 2) Defining Requirements  
(Gereksinimleri Tanımlama)
- 3) Designing the product architecture  
(Ürün dizaynını tasarlama)
- 4) Building or Developing the Product  
(Ürünü oluşturma veya geliştirme)
- 5) Testing the Product (Ürünü test etme)
- 6) Deployment in the Market and Maintenance  
(Ürünü pazarlama ve bakım)



## 6 PHASES OF THE SOFTWARE DEVELOPMENT LIFE CYCLE



**CTO => Chief Technology Officer**

**UX => User Experience Designer (Kullanıcı Deneyim Tasarımcısı)**

**UI => User Interface Designer (Kullanıcı Arayüz Tasarımcısı)**

<https://www.visualcapitalist.com/every-minute-internet-2020/>

<https://www.visualcapitalist.com/map-facebook-path-social-network-domination/>

# 1) PLANNING AND REQUIREMENT ANALYSIS

- İhtiyaç analizi SDLC'nin en önemli ve temel aşamasıdır.
- Müşteriden gelen fikirler de göz önünde bulundurularak ekibin kıdemli üyeleri (expert) tarafından gerçekleştirilir.
- Bu bilgiler daha sonra temel proje yaklaşımını planlamak için kullanılır.
- Kalite güvence gerekliliklerinin planlanması ve projeye ilişkili risklerin belirlenmesi de planlama aşamasında yapılır.
- Minimum risklerle projeyi başarıyla uygulamak için izlenebilecek teknik yaklaşımalar planlanır.



## 2) ANALYSIS, DEFINING REQUIREMENTS

İhtiyaç analizi yapıldıktan sonraki adım, ürün gereksinimlerini açıkça tanımlamak ve belgelendirmektir (dokumante etmek)  
Stakeholder / işletmeciden onay alınır.

Bu proje yaşam döngüsü boyunca tasarlanacak ve geliştirilecek tüm ürün gereksinimlerini içeren

**BRD** (Business Requirement Document): İş Gereksinimleri Dokümanı iş ihtiyaçlarını ve paydaş gereksinimleri açıklayan bir gereklilik paketidir.

**FRD** (Functional Requirement Document)

Teknik İşlev İhtiyacları Dokumani hazırlanır

FRD ve BRD en küçük User Case lere kadar hazırlanır



### 3) DESIGNING THE PRODUCT ARCHITECTURE:

BRD (Business Requirement Document)

Dizaynırların geliştirilecek ürün için en iyi dizaynla ortaya çıkacakları referanstır.

BRD'de belirtilen gereksinimlere dayanarak, ürün mimarisi için genellikle birden fazla tasarım yaklaşımı taslağı oluşturulur.

DDS(Design Document Specification):

Tasarım spesifikasyonu, bir ürün veya süreçle ilgili noktaların bir listesini sağlayan ayrıntılı bir belgedir.



## 4) BUILDING OR DEVELOPING THE PRODUCT

SDLC'ninbu aşamasında gerçek gelişme başlar ve ürün inşa edilir.

- Yazılımcılar (Developers), kuruluşları tarafından tanımlanan kodlama yönergelerine uymak zorundadır.
- Kodlama için FRD baz alınarak
- Developerlar gereken Funcionality'leri oluştururlar.
- Kodlama için C ++, Java, .Net vs. gibi farklı üst düzey programlama dilleri kullanılır.



## 5) TESTING THE PRODUCT

Bu aşama, ürün BRD'de tanımlanan kalite standartlarına ulaşıcaya kadar, ürün kusurlarının (bug) rapor edildiği, izlendiği, düzelttiği ve tekrar test edildiği aşamadır.

Ürün iş bekłentilerini de karşılamalıdır  
(requirement specifications)

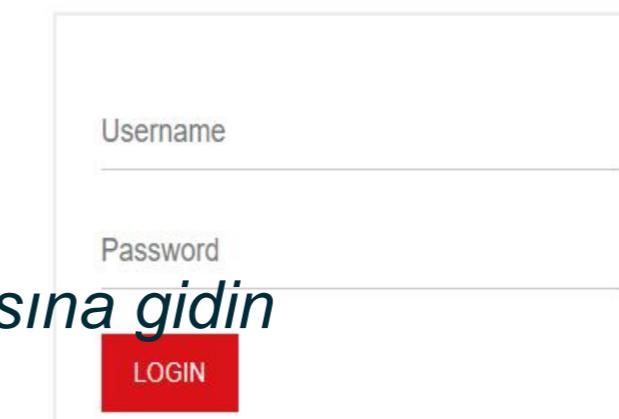
**STLC => Software Testing Life Cycle**

Test -> Takip -> Bulunan Hatanın Dev.  
gönderilmesi -> Raporlama-Duzeltme ->  
Yeniden Test etme -> Onaylama -> Raporlama

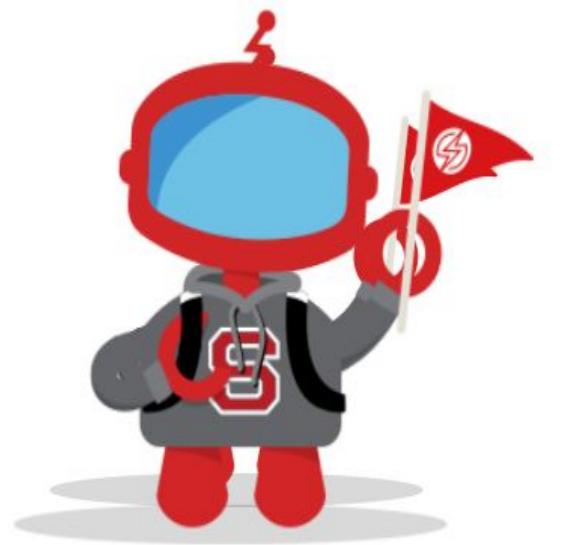


# ÖRNEK TEST

1. “<https://www.saucedemo.com>” adresine gidin.
2. Username kutusuna “standard\_user” yazdırın.
3. Password kutusuna “secret\_sauce” yazdırın.
4. Login tbuttonuna basın
5. İlk ürünün ismini kaydedin ve bu ürünün sayfasına gidin
6. Add to Cart butonuna basın
7. Alisveris sepetine tiklayın
8. Sektiginiz ürünün başarılı olarak sepete eklendiğini control edin
9. Sayfayı kapatın.



A screenshot of a login form. It has two input fields: "Username" and "Password", both currently empty. Below the fields is a red rectangular button labeled "LOGIN".



T E C H P R O E D

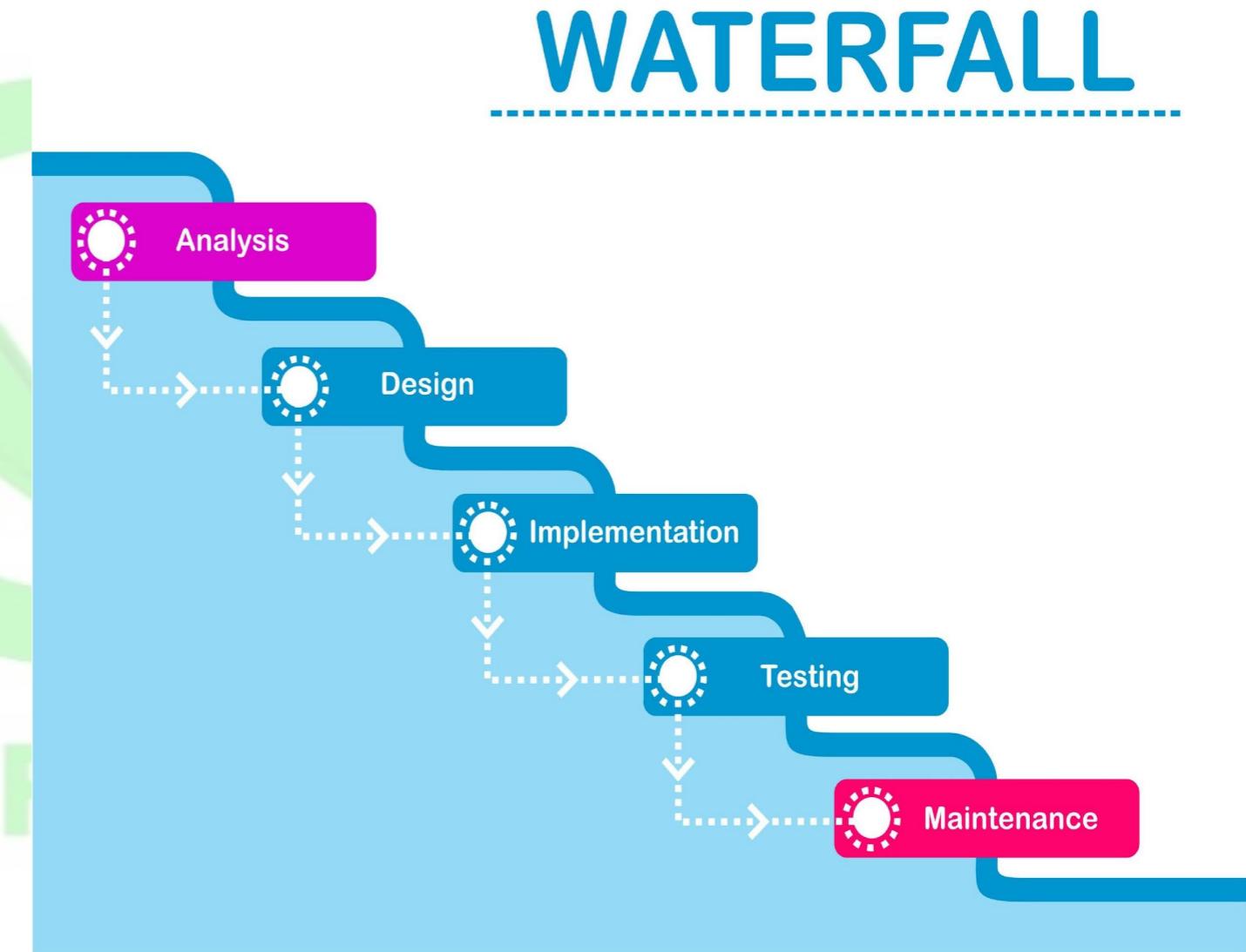
## 6) DEPLOYMENT AND MAINTENANCE

- Ürün test edildikten ve onaylandıktan sonra (hazır olduğunda), resmi olarak uygun görülen şekilde release edilir. (piyasaya sürürlür).
- Ürün piyasaya sunulduktan sonra mevcut müşteri tabanı için bakımı yapılır.
- Musteriden (End-User) gelen feedbackler ve Teknolojik Gelişmeler ile ihtiyaçlar yeniden belirlenir ve döngü yeniden başlatılır.

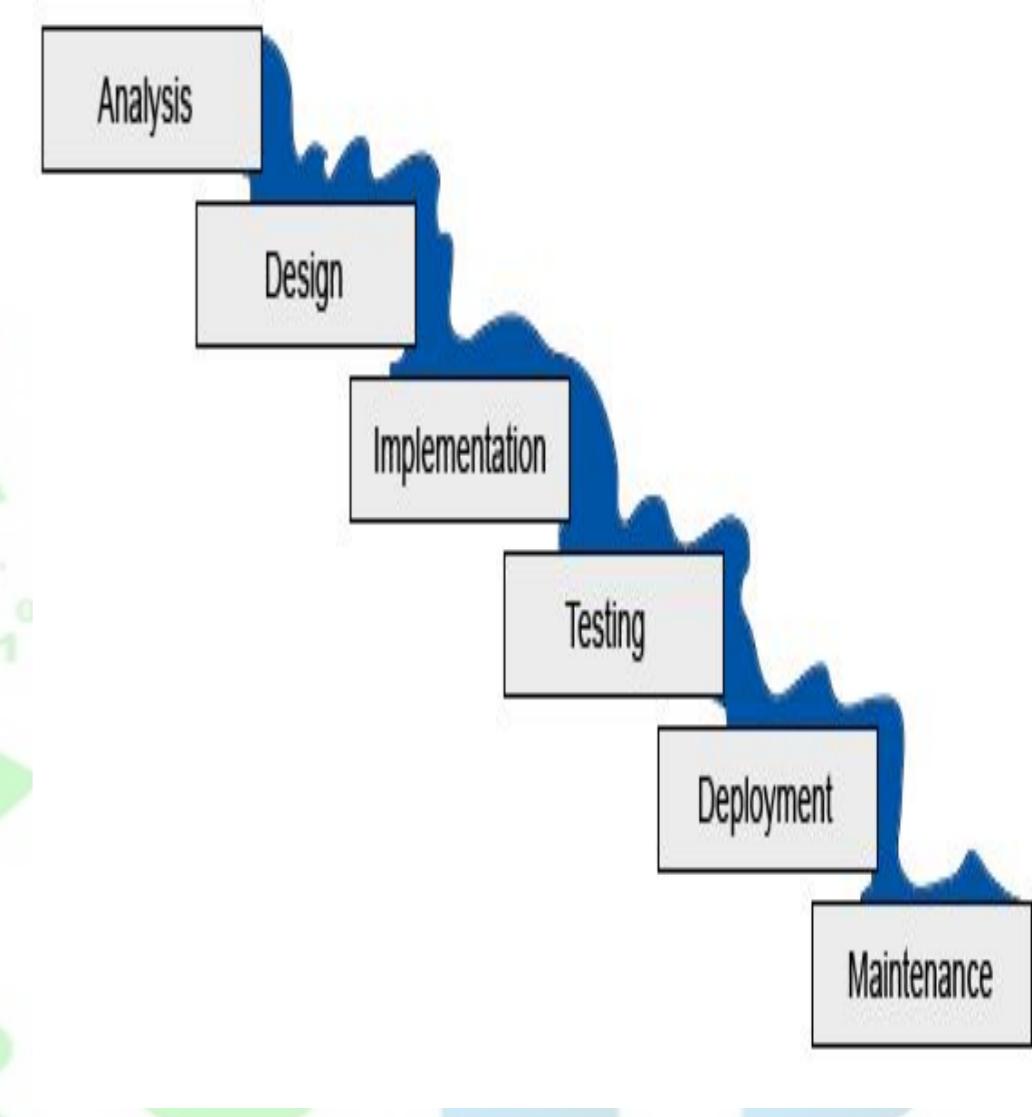


# WATERFALL METHOD (ŞELALE METODU)

- Şelale modeli (Waterfall) proje yönetim süreci; analiz, tasarım, yazılım, test, yayın gibi fazlardan oluşur.
- Geleneksel bir yöntemdir; süreçler tipki bir şelale gibi yukarıdan aşağıya doğrusal olarak işler.
- Bir faz tamamlanıp yenisine geçildiğinde, bir önceki faza geri dönülmez.
- Proje sahibi, proje tamamlandıktan sonra ürünü görebilir.



- Şelale modeli, analiz adımı ile başlar. Analiz adımında, tüm yazılım gereksinimleri net bir şekilde belirlenerek analiz dokümanı üretilir.
- Daha sonra, tasarım adımında; yazılımın arayüz, veritabanı, sınıf vb. tasarımları yapılarak tasarım dökümanı üretilir.
- Bir sonraki kodlama adımında yazılım; analiz ve tasarım dokümanlarında belirtilen şekilde kodlanır.
- Test adımında; analiz ve tasarım dokümanlarındaki tüm fonksiyonel ve fonksiyonel olmayan gereksinimler ve tasarımlar için test senaryoları yazılır ve bu test senaryoları icra edilerek yazılımın testleri yapılır.
- Test adımı sonunda, yazılımda herhangi bir hatası bulunamaz ise, entegrasyon adımına geçilir ve yazılım, canlı ortama entegre edilerek müşterinin kullanımına açılır.

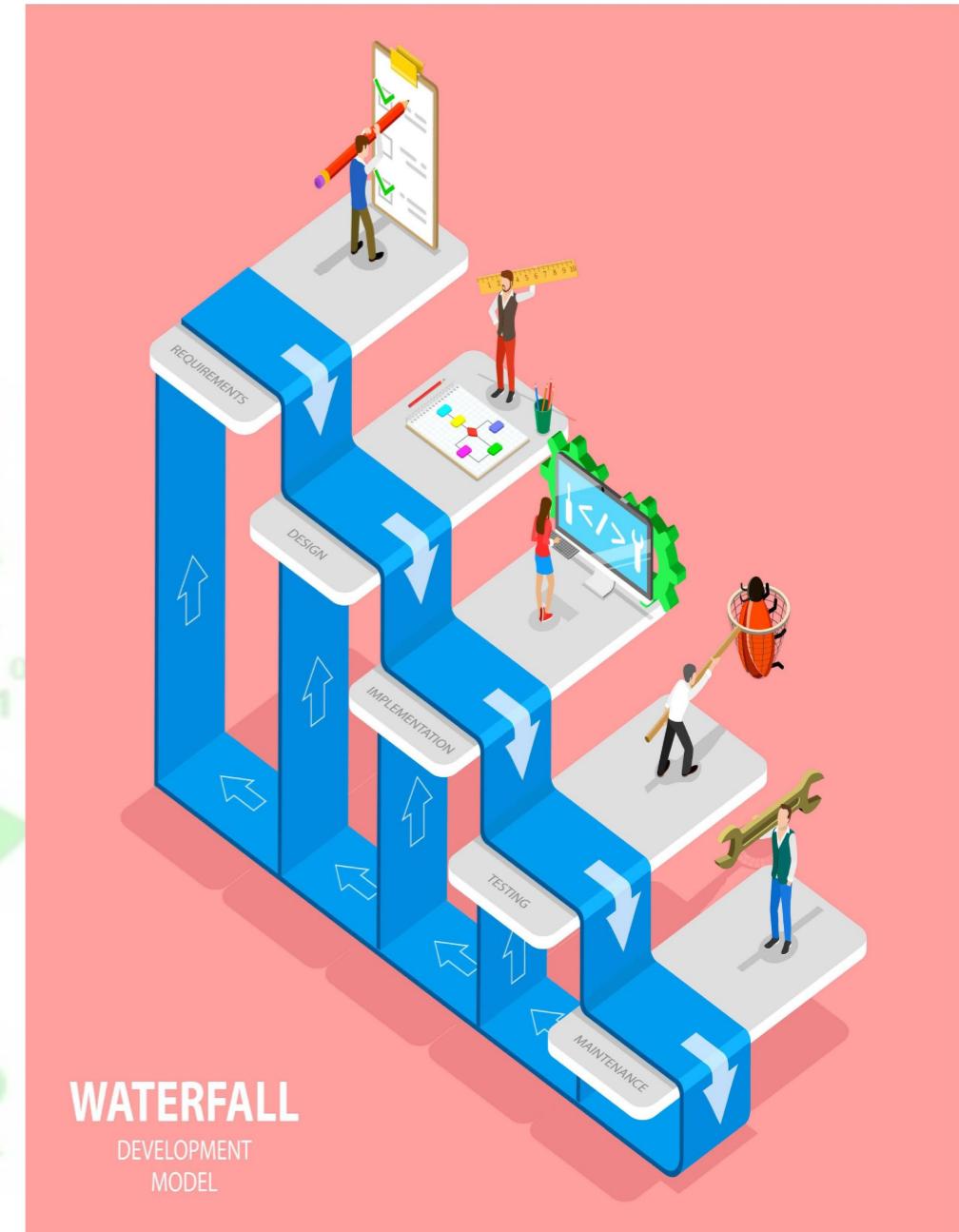


## AVANTAJLARI

- Kullanımı ve yönetimi kolaydır.
- Gereksinimler iyi anlaşılır.
- Proje bilgisini aktarmak daha kolaydır.
- Küçük projeler için daha iyidir
- Görevler mümkün olduğunca sabit kalır
- Kapsamlı dökümanlar oluşturulur.

## DEZAVANTAJLARI

- Değişim ve yenilik zordur.
- Müşteri öngörü ve önerileri önemsenmez.
- Projenin bitimine kadar çalışan ürün yok.
- Beklenmedik riskleri kolayca ele alamıyor.



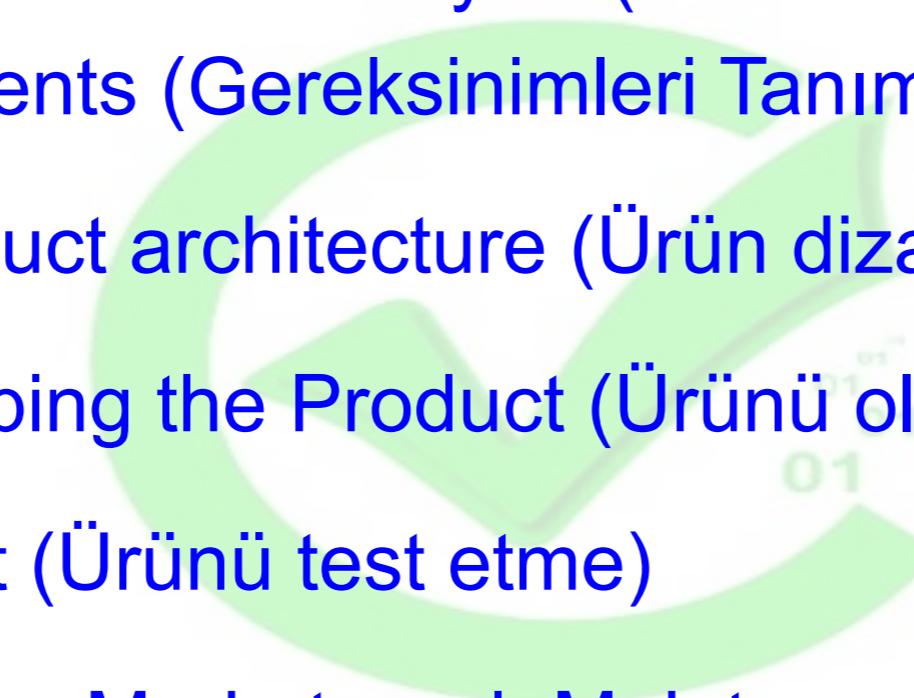
Şelale modelinde analiz ve tasarım aşamaları oldukça detaylı yapıldığından, bu adımlar uzun sürmektedir. Ancak, analiz ve tasarım aşamalarında gereksinimlerin ve tasarımın net bir şekilde ortaya konulmasından dolayı, kodlama ve test aşamaları çok kısa sürmektedir. Test aşamasında çıkan hata sayısı azdır.

TECH



- Şelale modelinde, üst adımlarda yapılan hataların yarattığı zaman kaybı oldukça fazladır. Örneğin test aşamasında karşılaşılan bir hatanın analizden kaynaklandığı tespit edilirse, analiz, tasarım ve test dokümanlarının güncellenmesi ve kodun düzeltilmesi gereklidir, ki bu oldukça ciddi zaman ve dolayısıyla para kaybına yol açar.
- Şelale modelinin dezavantajlarından bir tanesi de, ürünün ortaya çıkması için tüm aşamaların tamamlanmasını beklemek zorunda kalmaktır. Örneğin; proje 4 sene sürecek ise, müşterinin ilk prototipi görmesi için, analiz, tasarım ve kodlama aşamalarının bitmesini, yani en az 2-3 sene beklemesi gerekecektir. Bu durum; bazı sabırsız müşteriler için sorun teşkil edebilir. Bu gibi durumlarda Agile yöntemler daha faydalı olabilir.

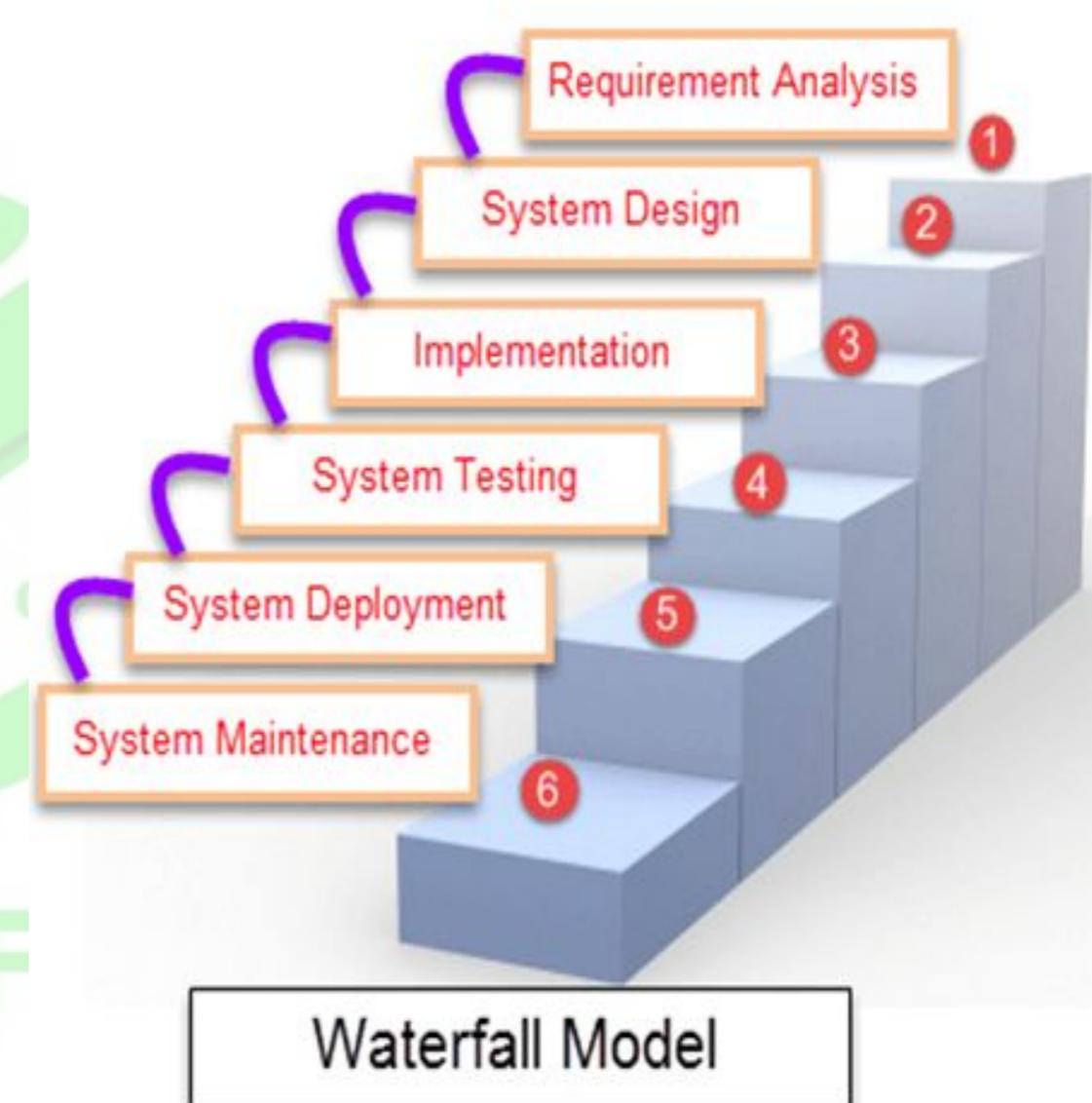
# Bugün Ne Öğrendik?

- 
- 1) **Planning** and Requirement Analysis (Planlama ve İhtiyaç Analizi)
  - 2) **Defining** Requirements (Gereksinimleri Tanımlama)
  - 3) **Designing** the product architecture (Ürün dizaynını tasarlama)
  - 4) **Building** or Developing the Product (Ürünü oluşturma veya geliştirme)
  - 5) **Testing** the Product (Ürünü test etme)
  - 6) **Deployment** in the Market and Maintenance (Ürünü pazarlama ve bakım)
- 

# Bugün Ne Öğrendik?

## Waterfall Method (Şelale Metodu):

Yazılım geliştirmeyi önceden tanımlanmış aşamalara bölen sıralı bir modeldir. Her aşama, aşamalar arasında çakışma olmadan bir sonraki aşamanın başlayabilmesi için tamamlanmalıdır. Her aşama, SDLC aşaması sırasında belirli aktiviteyi gerçekleştirmek için tasarlanmıştır. 1970 yılında Winston Royce tarafından tanıtıldı.

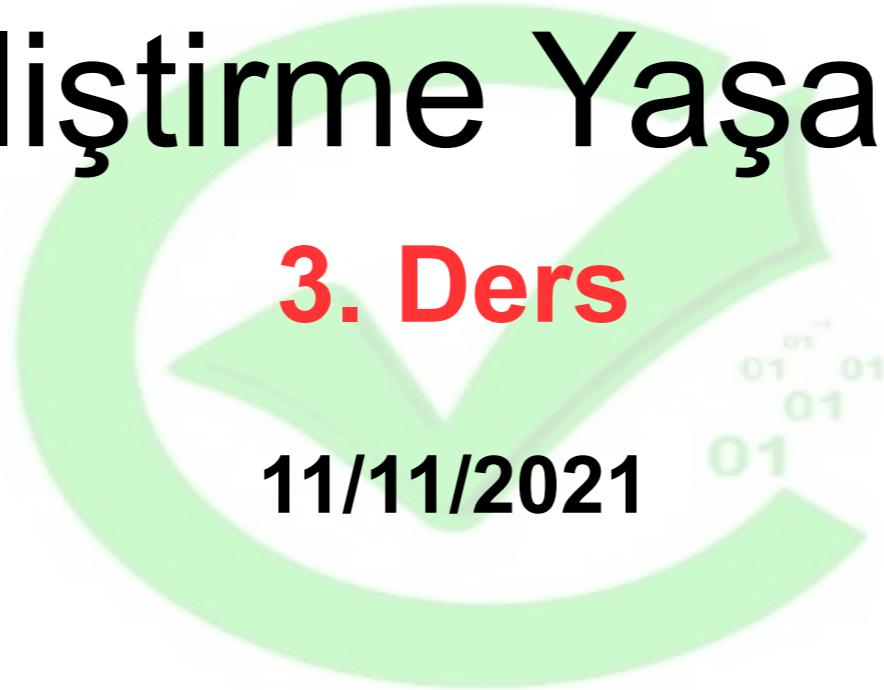


T E C H P F

# **SDLC**

# **Software Development Life Cycle**

## **(Yazılım Geliştirme Yaşam Döngüsü)**

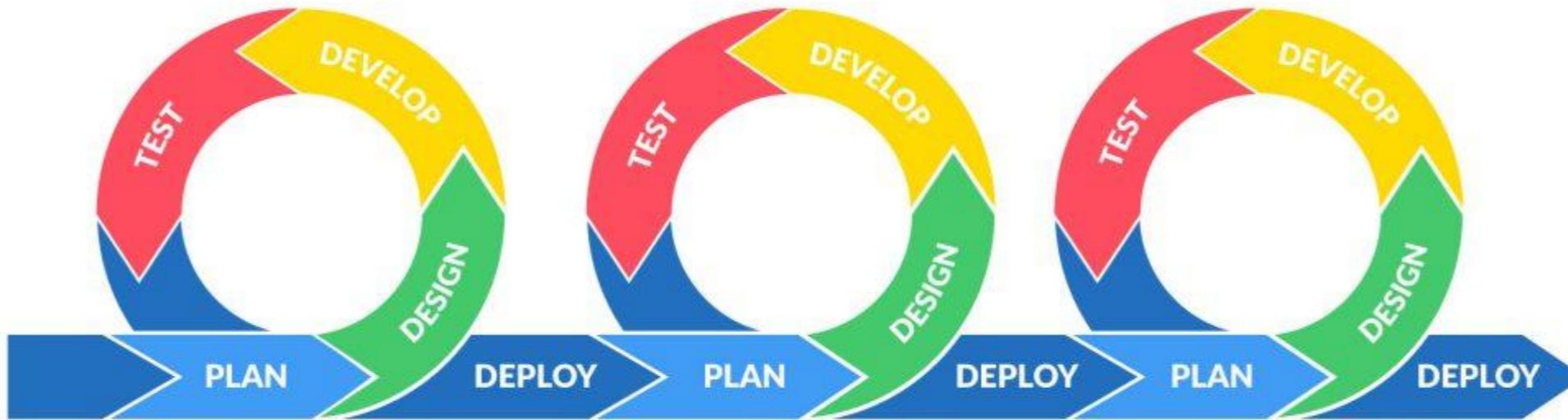


**3. Ders**

**11/11/2021**

**T E C H P R O E D**

# AGILE METHODOLOGY



Agile Metodoloji (Çevik Metodoloji) yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve dokümantasyonunu yapmaya yönelik, pratiğe dayalı bir yöntemdir.

TECHPRO E

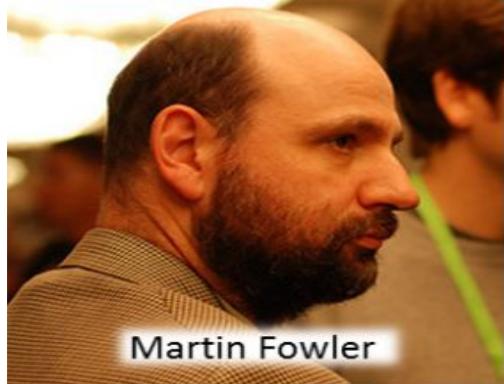
# AGILE METHODOLOGY

- Diğer sektörlerde ki yaklaşımların bir devamı olarak 1970'li yıllarda itibaren yazılım sektöründe uygulanmaya başlanmıştır. Aşırı kuralcı klasik yazılım süreç modellerine tepki olarak ortaya çıkan Agile Manifestosu öncesinde yazılımlar daha yüksek maliyetli ve daha yavaş geliştirilmekteydi.
- Yazılım geliştirme sürecini hızlandırmak, daha etkin kullanmak ve gerektiğinde dokümantete etmek amacıyla ortaya çıkan, Dünyada birçok yazılım firmasının farklı projelerinde benimsediği bu metodun kullanımı 1990'larda hız kazanmıştır.

# AGILE METHODOLOGY

- 2001 yılında yazılım dünyasının önde gelen isimlerinden 17 arkadaş; “Agile (Çevik) Yazılım Geliştirme Manifestosu” ve “Agile (Çevik) Yazılımın Prensipleri” ni yayımlamışlar, bu oluşumu ve gelişimini desteklemek için “Agile Alliance” adıyla, kar amacı gütmeyen bir organizasyon kurmuşlardır.
- Manifesto, nasıl daha iyi bir yazılım geliştirdiklerini ve bunu yapmak isteyenlere yol gösterecek 12 maddeden oluşmaktadır.

<https://www.agilealliance.org/>



# AGILE MANIFESTO

**1) İlk önceliğimiz kaliteli yazılımı müşteriye teslim edebilmektir.** Bu projenin ilk aşamalarından itibaren sürekli teslimlerle yapılır ve müşterinin yazılımı çok önceden kullanmaya başlayarak değer sağlamasına olanak sağlanır. Günümüzde çevik süreçlere artan ilginin başlıca nedenlerden biri , yapılan yatırımların hızlı geri dönüşünün olmasıdır.

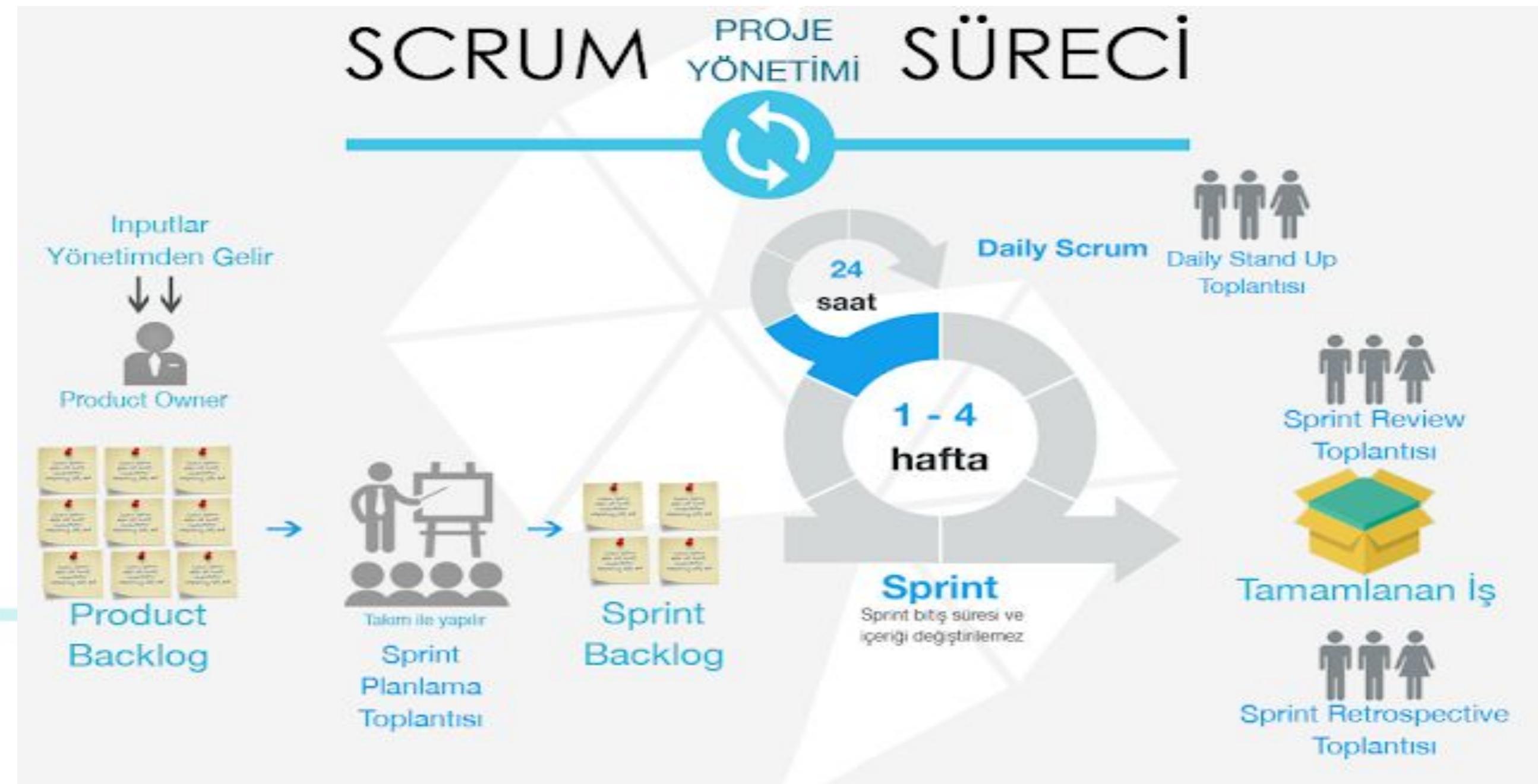
**2) Değişiklikler projenin ilerki aşamalarında dahi olsa kabul edilir.** Amaç müşterinin ihtiyaçlarını karşılayan, onlara yarar sağlayacak, gerçek değer katacak yazılım üretmektir ve ihtiyaçlarda meydana gelen değişiklikler projenin sonraki aşamalarında dahi yazılıma aksettirilmelidir. Test, güdümlü tasarım, kapsamlı otomatik testler, sürekli entegrasyon, basit tasarım gibi pratikler sayesinde değişikliklerin getireceği maliyetler minimuma indirilir ve süreç değişikliklere çabuk adapte hale getirilir.

- 3) Çok kısa aralıklarla yazılım teslimleri yapılır.** Bu aralıklar tipik olarak 2-4 hafta arasıdır. Bu sayede sürekli geri beslenim (feedback) sağlanır ve müşterinin tam istediği şekilde yazılım geliştirilerek ilerler.
- 4) Alan uzmanları , yazılımcılar, testçiler günlük olarak birlikte çalışırlar.** Farklı roller arasında duvarlar örülmez. Rol bazlı ekipler yerine yazılım özelliklerine(features) göre ekipler oluşturulur. Analist (BA), yazılım geliştirici(Dev), tester (QA) vb. aynı ekibin içinde çalışır ve sürekli iletişim halindedir.
- 5) Motive olmuş bireyler etrafında projeler oluşturun.** Ekip üyelerine kendileri ile ilgili alacakları kararlar konusunda güvenilir. Ekip kendi kendine organize olacak yetkiye sahiptir. Onlara ihtiyaç duydukları ortamı ve desteği verin ve işi tamamlamaları için onlara güvenin.

- 6) Bir geliştirme ekibine ve içinde bilgi aktarmanın en verimli ve etkili yöntemi yüz yüze görüşmedir.**
- 7) Çalışan yazılım, ilerlemenin birincil ölçüsüdür.** Projedeki gelişmenin tek ölçüsü o ana kadar geliştirilmiş özellikler ve çalışan yazılımdır.
- 8) Agile processes (sureçler) sürdürülebilir gelişmeyi teşvik eder.** Planlamaların sağlıklı olması için ekibin iş teslim hızının üzerinde çok oynanaması gereklidir. Örneğin fazla mesailer gibi yöntemlerle ekibin hızını geçici olarak artırmak tercih edilen yöntemlerden değildir.
- 9) Teknik mükemmelliğe ve iyi tasarıma verilen sürekli dikkat, çevikliği artırır.**

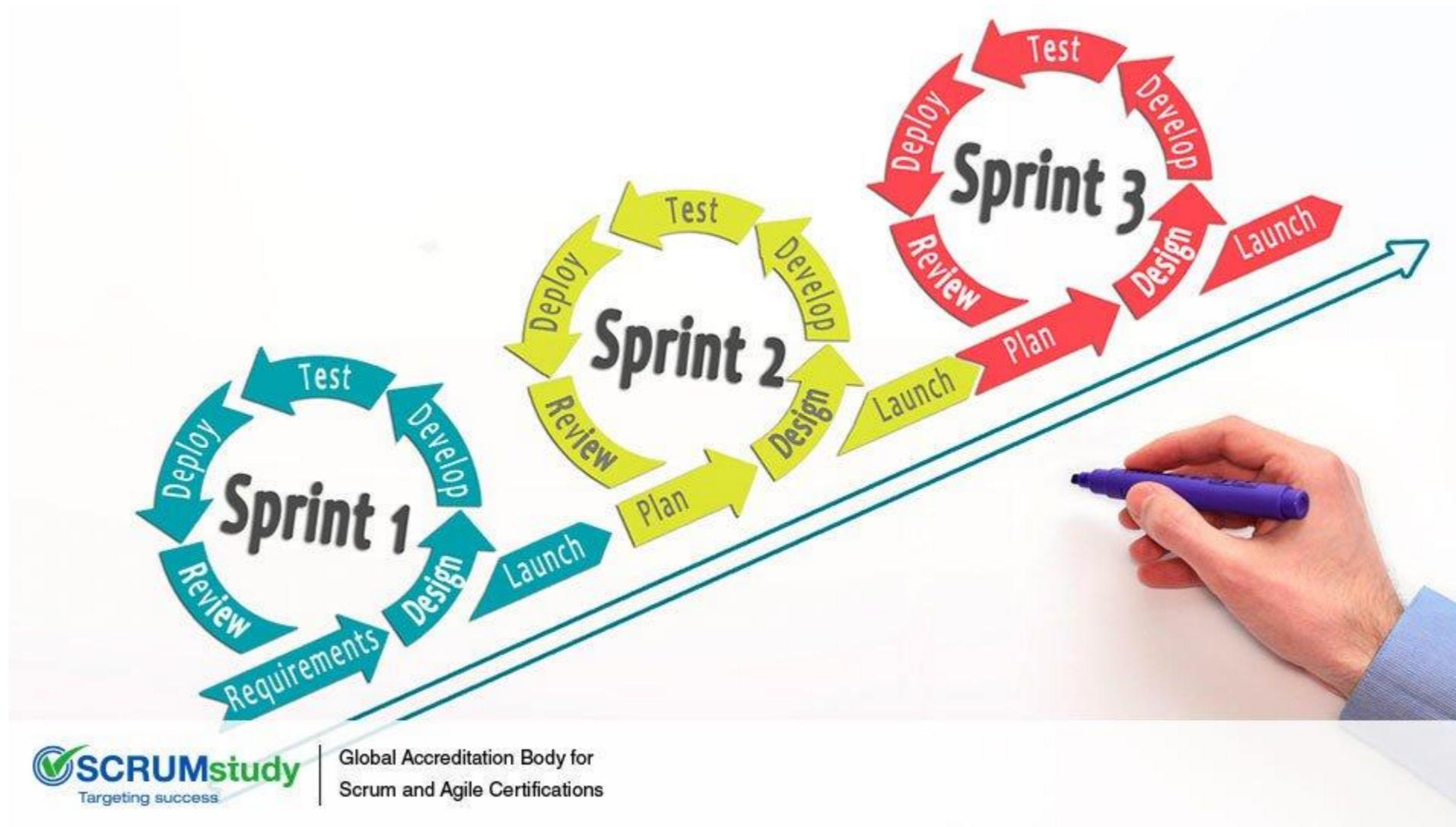
- 10) Sadelik esastır.** Sadelik anlayışı akla gelen baştan savma çözümü uygulamak yerine, anlaşılması ve sonradan değiştirilmesi kolay , maliyeti en düşük ve o an ki gereksinimleri karşılayan çözümü kullanmaktadır.
- 11) En iyi mimariler, gereksinimler ve tasarımlar kendi kendini organize eden ekiplerden ortaya çıkar.** En etkin çalışan ekipler kendilerini organize edebilen , bu konuda yetkin ekiplerdir. Ekip kendi çalışma yöntemlerini sorgulamakta ve gerekli değişiklikleri yapmakta özgürdür.
- 12) Ekip, düzenli aralıklarla nasıl daha etkili olunacağını düşünür, ardından davranışını buna göre ayarlar ve ayarlar.** Ekip kısa sürelerle toplanır, çalışma yöntemlerini gözden geçirir ve daha etkili çalışmak için geriye dönük (retrospective) toplantılar yaparak durumları gözden geçirir.

# SCRUM



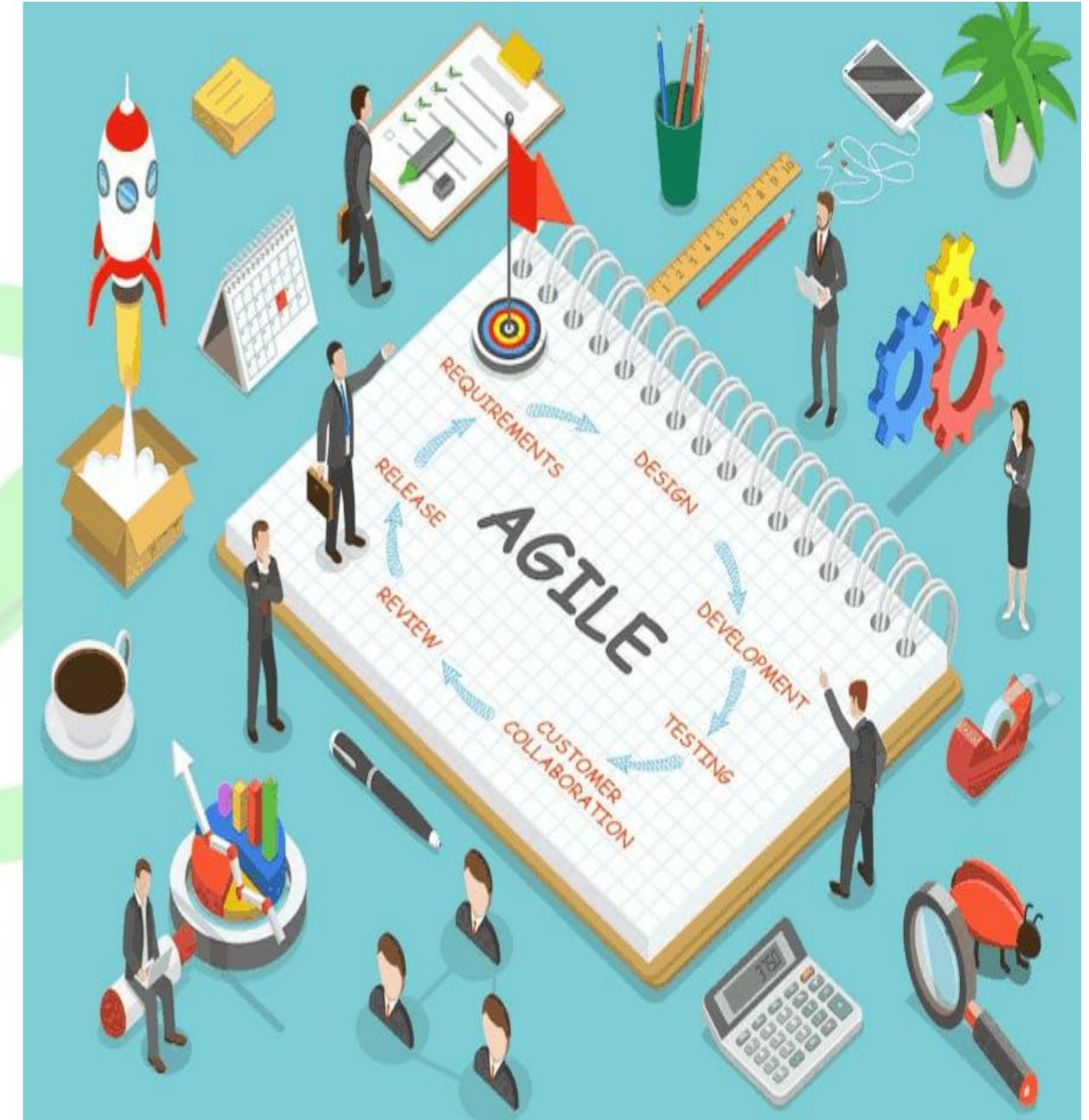
Zaman içerisinde projelerin daha büyük ve karmaşık bir hal alması, bununla beraber müşterinin büyük resmi göremeyip gereksinimlerini tam olarak ortaya koyamaması, teknolojinin çok hızlı değişmesi ile beraber gereksinimlerin çabuk değişmesi ve bunu projemize entegre edemeyişimiz gibi problemlerden dolayı çoğu proje başarısızlık ile sonuçlanmaya başladı. Böylece proje sürecinin yönetilmesi konusu önemli bir konu oldu ve “Çevik (Agile) Yazılım Geliştirme Manifestosu” ortaya çıktı.

**Scrum:** Agile proje yönetim metodolojilerinden biridir. Kompleks yazılım süreçlerinin yönetilmesi için kullanılır. Bunu yaparken bütünü parçalayan; tekrara dayalı bir yöntem izler. Düzenli geri bildirim ve planlamalarla hedefe ulaşmayı sağlar. Bu anlamda ihtiyaca yönelik ve esnek bir yapısı vardır. Müşteri ihtiyacına göre şekillendiği için müşterinin geri bildirimine göre yapılanmayı sağlar. İletişim ve takım çalışması çok önemlidir.



“Rugby” yaklaşımı olarak adlandırılan temel felsefe , “takımın, mesafenin tümünü hep beraber, bir birim halinde topu ileri geri atarak kat etmesidir.”

- Her sprint genellikle 2 ila 4 hafta veya en fazla bir takvim ayı sürer. Ürünleri her seferinde küçük bir parça oluşturmak, üretkenliği teşvik eder ve ekiplerin geri bildirime ve değişime yanıt vermesini ve tam olarak gerekli olanı oluşturmasını sağlar.
- Scrum'da ürün sprint'te tasarlanır, kodlanır ve test edilir.
- Yapılacak tüm işler, Product Backlog da biriktirilir, Product Owner (PO) nun belirlediği önceligi göre Sprint Backloguna alınır ve bir sprintte bitirilerek ürünün demosuna eklenir.



# AGILE TEAM

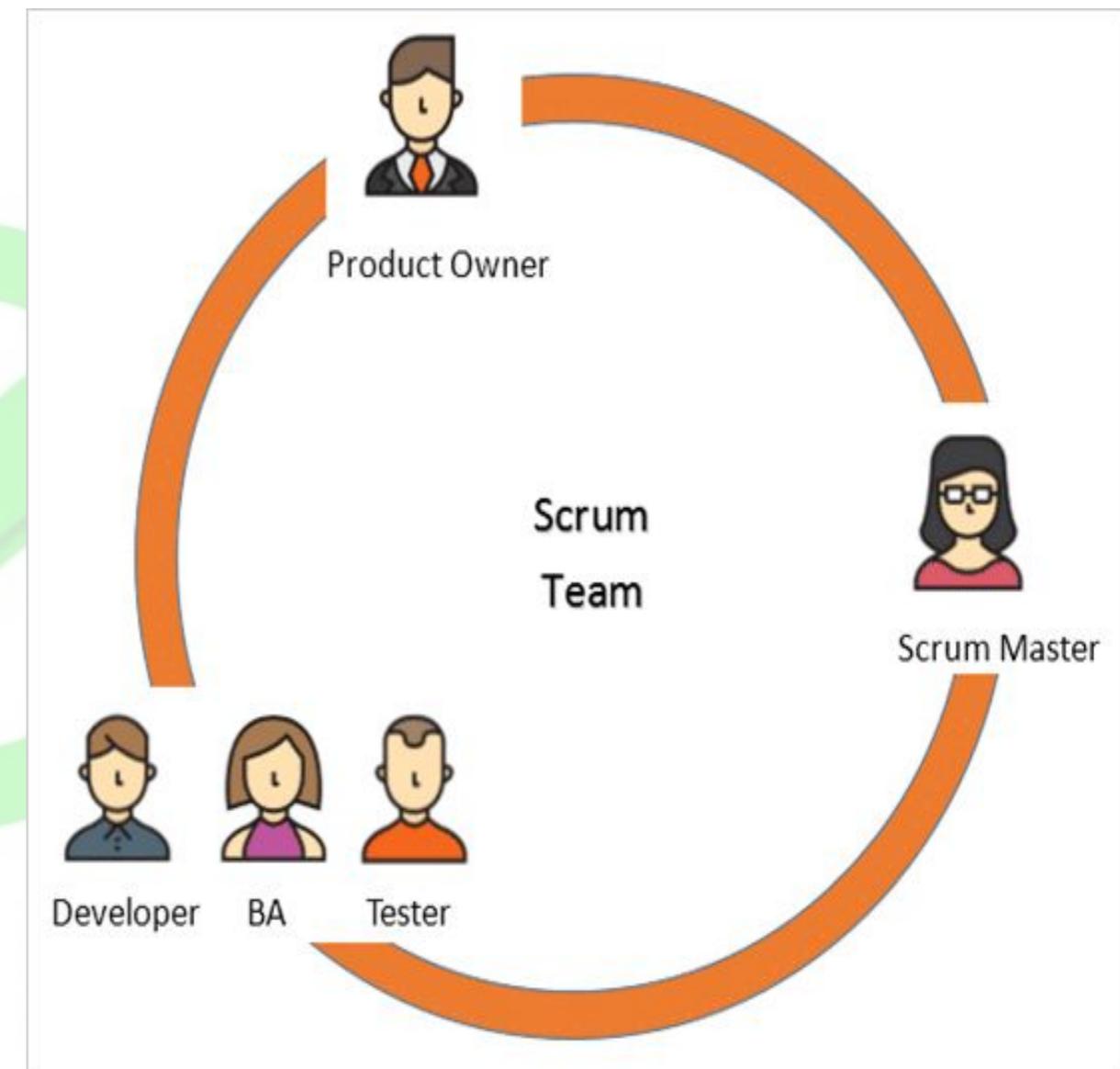
Pig Roller: Scrum sürecine dahil olanlar yani projede asıl işi yapan kişilerdir. Bunlar Scrum Master, Product Owner, Geliştirme Takımı'dır.

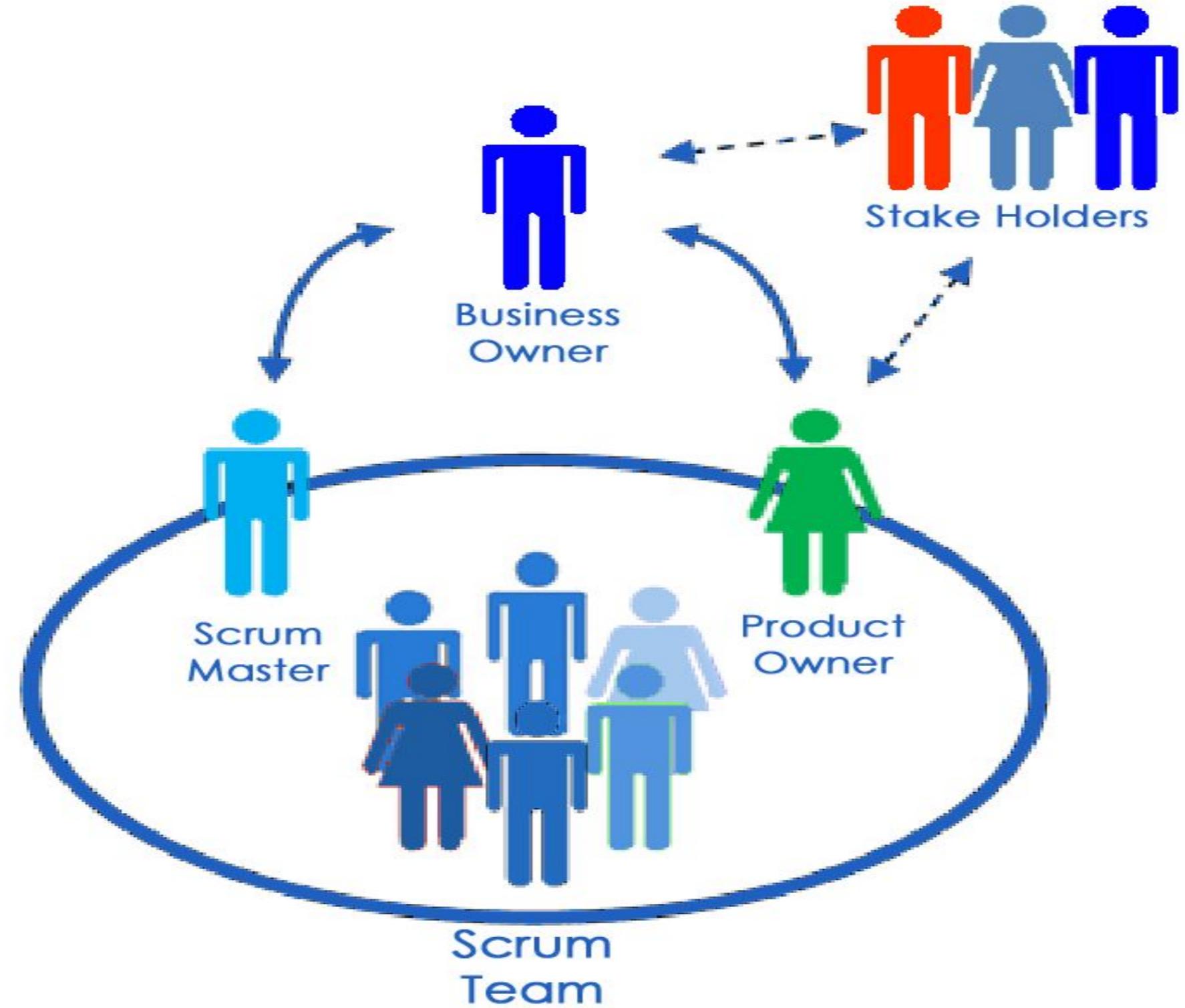
- 1) Product Owner (PO)
- 2) Scrum Master
- 3) Geliştirme Takımı

Chicken Roller: Scrum'ın işleyişinde aktif olarak yer almayan kişilerdir. Müşteriler, satıcılar gibi.

Takım kendi kendini örgütler. (**Self Organized**)

Böylece kendi içerisinde uyum içinde olan takımlar daha başarılı sonuçlar alırlar.

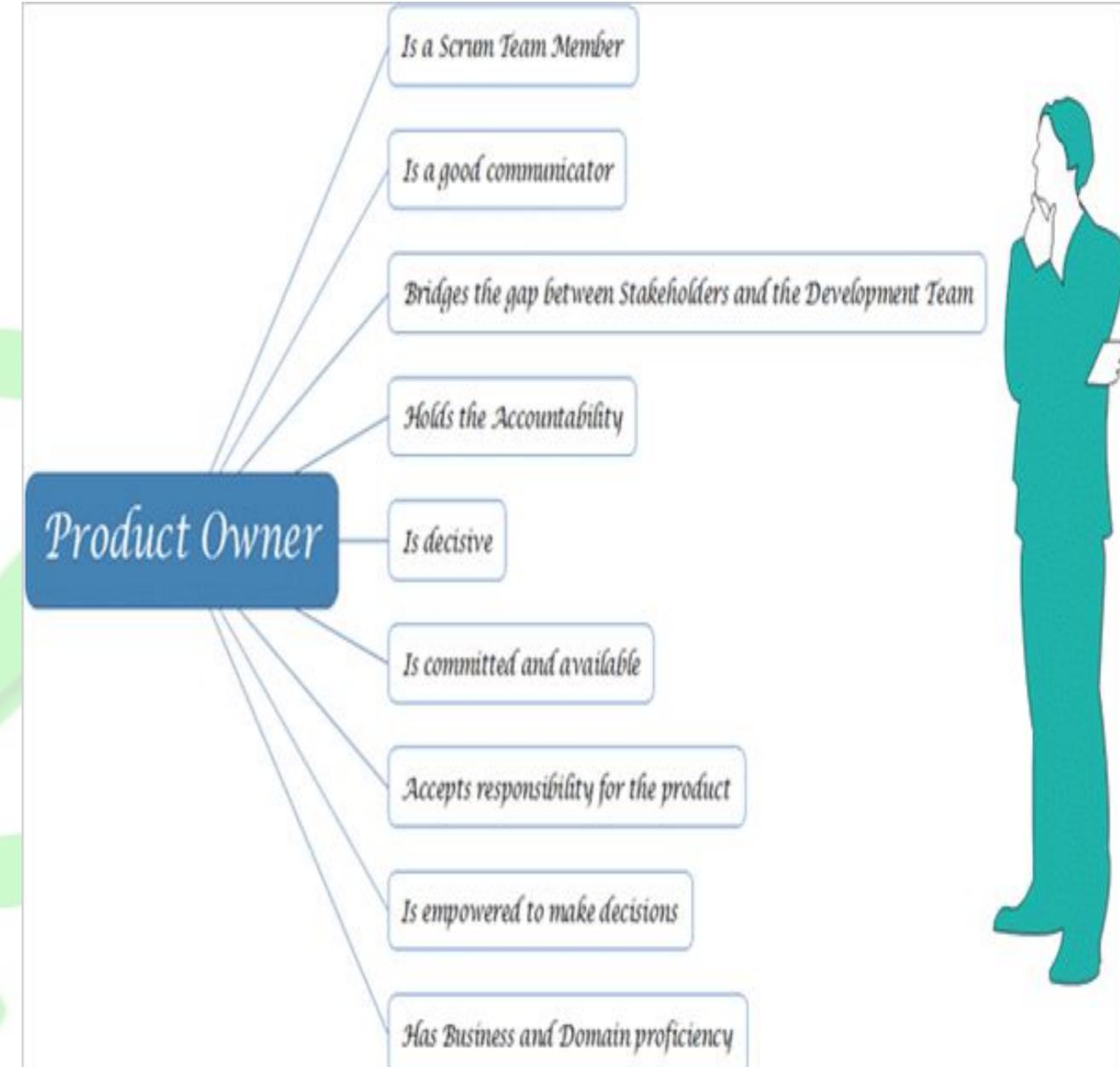
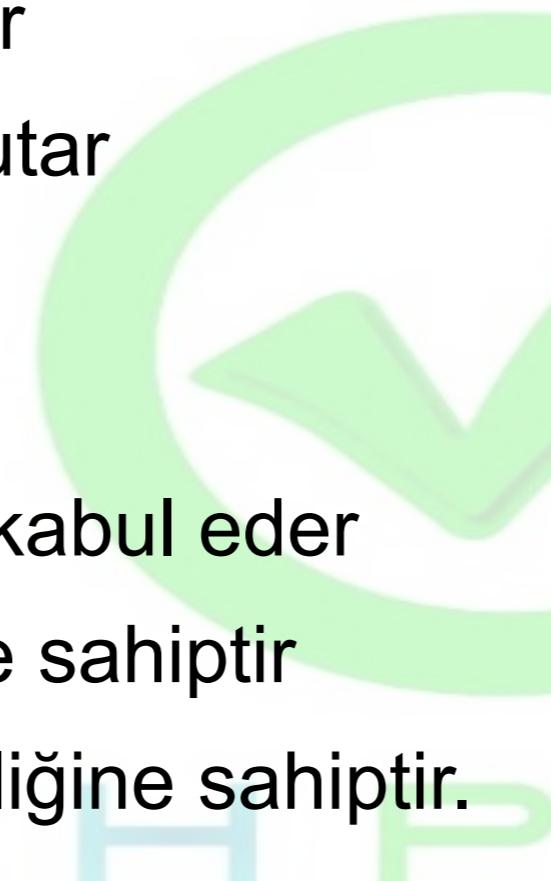




**1) Product Owner:** Geliştirme takımı ve müşteri arasındaki iletişimini sağlar. Projenin özelliklerini tanımlar. Projenin önceliklerine göre Product Backlog (iş listesi) oluşturur. Ekipte Stakeholders'ı temsil eder.

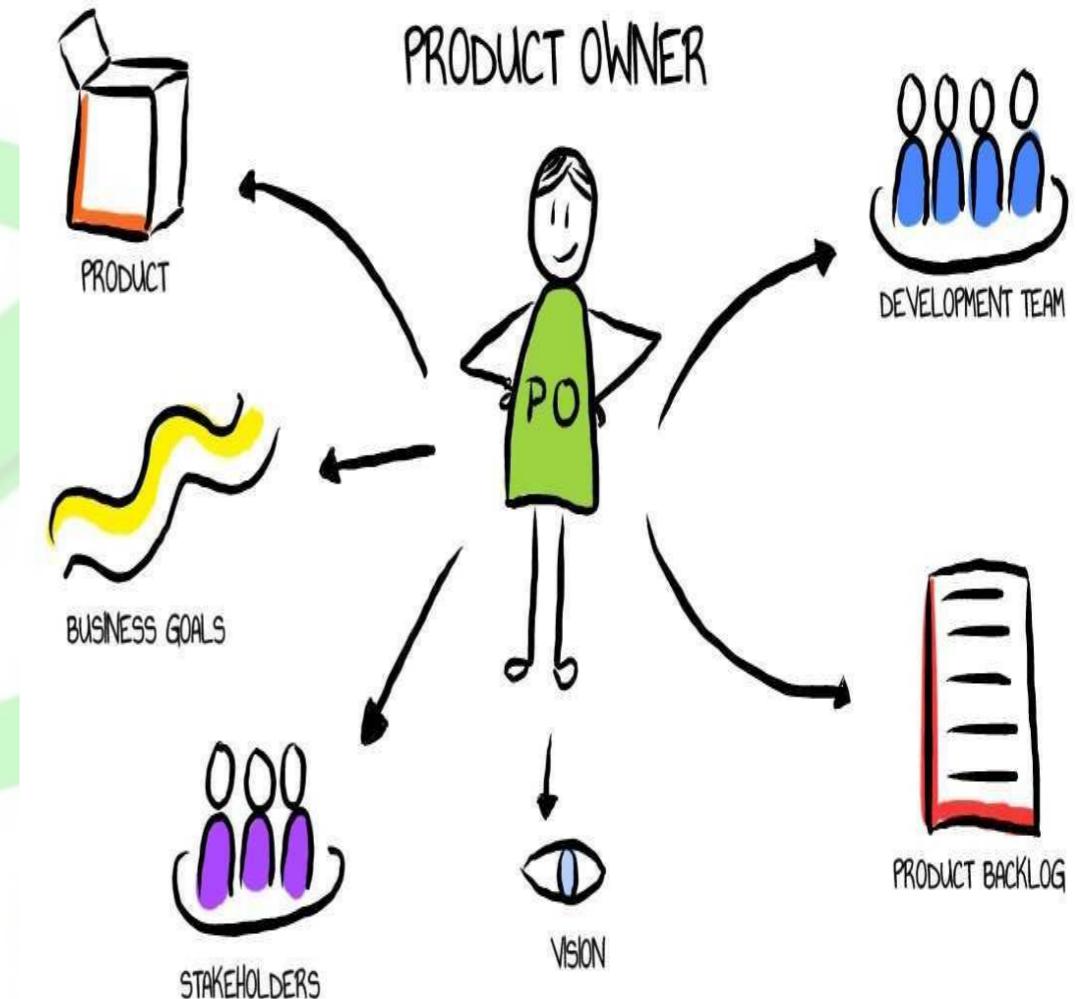
- İş Listesini (Product Backlog) yönetir. Birinci önceliği; iş listesi'ni yönetmektir. Product Backlog'i yönetmek demek, ürünü yönetmek demektir.
- Ürünle ilgili yapılacak bütün geliştirme Product Backlog'da bulunur.
- Product Backlog'un sahibi PO'dur.
- Product Backlog herkes tarafından erişilebilir ve anlaşılabilir olmalıdır.
- Product Owner, Geliştirme Takımı ve Stakeholders arasındaki iletişimini gerçekleştirir.
- Her Sprint'tehangi user story'lerin sprinte dahil edilecegine karar verir
- Sprint değerlendirme toplantılarının organizasyonunu yapar.
- Sprint değerlendirme toplantıları'nın sahibidir.

- Scrum takımının üyesidir
- İletişimi kuvvetlidir
- Müşteri ve Development Team arasında iletişim kurar
- Sorumluluğu elinde tutar
- Belirleyicidir
- Kararlı ve ulaşılabilir
- Urun sorumluluğunu kabul eder
- Karar verme yetkisine sahiptir
- İş ve etki alanı yeterliliğine sahiptir.



# PRODUCT OWNER KİM DEĞİLDİR?

- Development Team'in ve Scrum'in yöneticisi değildir.
- Geliştirme Takımı teknik konularda kendi kendini yönetir.
- PO, teknik bir geçmişe sahip olabilir.
- Bu, hangi işi kimin yapacağına karışabileceğinin anlamına gelmez.
- PO, iş, görev ataması yapamaz.
- İşin nasıl yapılacağına karışamaz!





**2) Sucrum Master:** Scrum kurallarını, teorilerini ve pratiklerini iyi bilir ve takımın bu kurallarını uygulamasından sorumlu kişidir. Takımın yöneticiği değildir. Takımı rahatsız eden, verimli çalışmalarını engelleyen durumları ortadan kaldırır.

“Scrum Master, takımın Scrum değerlerine, pratiklerine ve kurallarına bağlılığını garanti altına almakla sorumludur. Scrum Master, takımı ve organizasyonu Scrum'a adapte eder.”



- ScrumMaster, takımına rehberlik ve koçluk eder, karşılaşılan engelleri ortadan kaldırırmalarına yardımcı olur. Takım içi harmoniyi, ekip elemanları arasındaki uyumu, iletişimini artırmak için çabalar.
- Sprint Planlama, Sprint Retrospektif, Günlük Scrum ve Sprint Review gibi Scrum ritüellerini ve toplantılarını kolaylaştırır.
- ScrumMaster takımının güvenli ve sorunsuz bir ortamda çalışabildiğinden emin olmalı, gerekiğinde takım elemanlarına bireysel koçluk da dahil olmak üzere bir çok hizmetini sunmalıdır.



- Product Owner ile ilişkileri yönetir. Scrum Master, ürün sahibi tarafından belirlenmiş işlerin takımındaki herkes tarafından anlaşıldığından emin olur; ürün sahibinin iş listesini etkili bir şekilde organize edebilmesi için teknikler bularak ona yardımcı olur.
- Değişime Liderlik Eder. Tüm bu görevlerin yanı sıra Scrum Master'lar değişime liderlik ederler.
- Scrum Master, Scrum'ın ve organizasyondaki çevik değişimin vücut bulmuş halidir.



**Scrum Master Kim değildir?**

**3) Development Team:** Geliştirme Ekibi, Front-End Developer, Back-End Developer, Dev-Ops, QA (Tester), İş Analisti (BA), DBA vb. gibi özel becerilere sahip kişilerden oluşabilir.

- Product Owner'ınyapılacak işler listesi olan Product Backlog'tan, belli bir sürede (Sprint) yapabileceği kadar işi, Product Owner'ınbelirlediği önceliğe göre yapan geliştirme takımıdır.
- Teknik Konularda Sorumluluk Development Team'e aittir.
- PO ve SM development team'in yapacağı işlerin önceliğini belirleyebilir ama neyi nasıl yapacaklarına karışmazlar.
- Takım içerisindeki kişilerin rolleri ve yetenekleri ne olursa olsun, dışarıya karşı bir işin tamamlanmasından tüm takım sorumludur.



- Bir Sprint'e alınan bütün işleri tamamlayacak özelliklere sahip kişilerdir. sprint backlogu oluştururlar. Kendi kendini yönetir. İşin verilmesini beklemezler, işi kendileri alır ve geliştirirler.
- Development Team, Product Backlog'tançektiği bir Product Backlog Item'ı, Product Owner'ın önüne çalışan bir kod parçası olarak koymakla yükümlüdür.
- “Self Organize” olmalıdır. Development Team, sorumluluğunu aldığı işlerin yapılması için bir iş tanımlamasına veya bir iş takipçisine ihtiyaç duymaz.

TECHPROED



- Scrum takımının ütesidir
- Çapraz Fonksiyonel: Dışarıdan herhangi bir yardıma ihtiyaç duymadan çalışmalarını tamamlamak için gerekli tüm beceri setlerine sahiptir.
- Kendi kendine yeterli
- Kendi kendine organize
- Yetenekli
- Kararlı ve ulaşılabilir
- Sorumlu

# SCRUM SÜRECi

PROJE  
YÖNETİMİ



Inputlar  
Yönetimden Gelir



Product Owner



Product  
Backlog



Takım ile yapılır  
Sprint  
Planlama  
Toplantısı

Sprint  
Backlog

24  
saat

1 - 4  
hafta

Sprint

Sprint bitiş süresi ve  
iceriği değiştirilemez

Daily Scrum



Daily Stand Up  
Toplantısı



Sprint Review  
Toplantısı



Tamamlanan İş

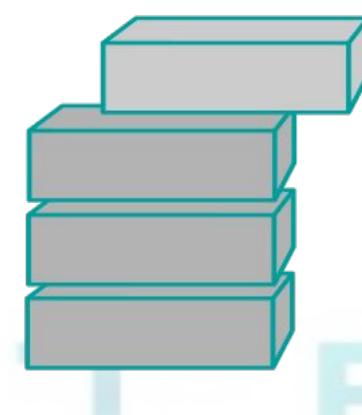


Sprint Retrospective  
Toplantısı

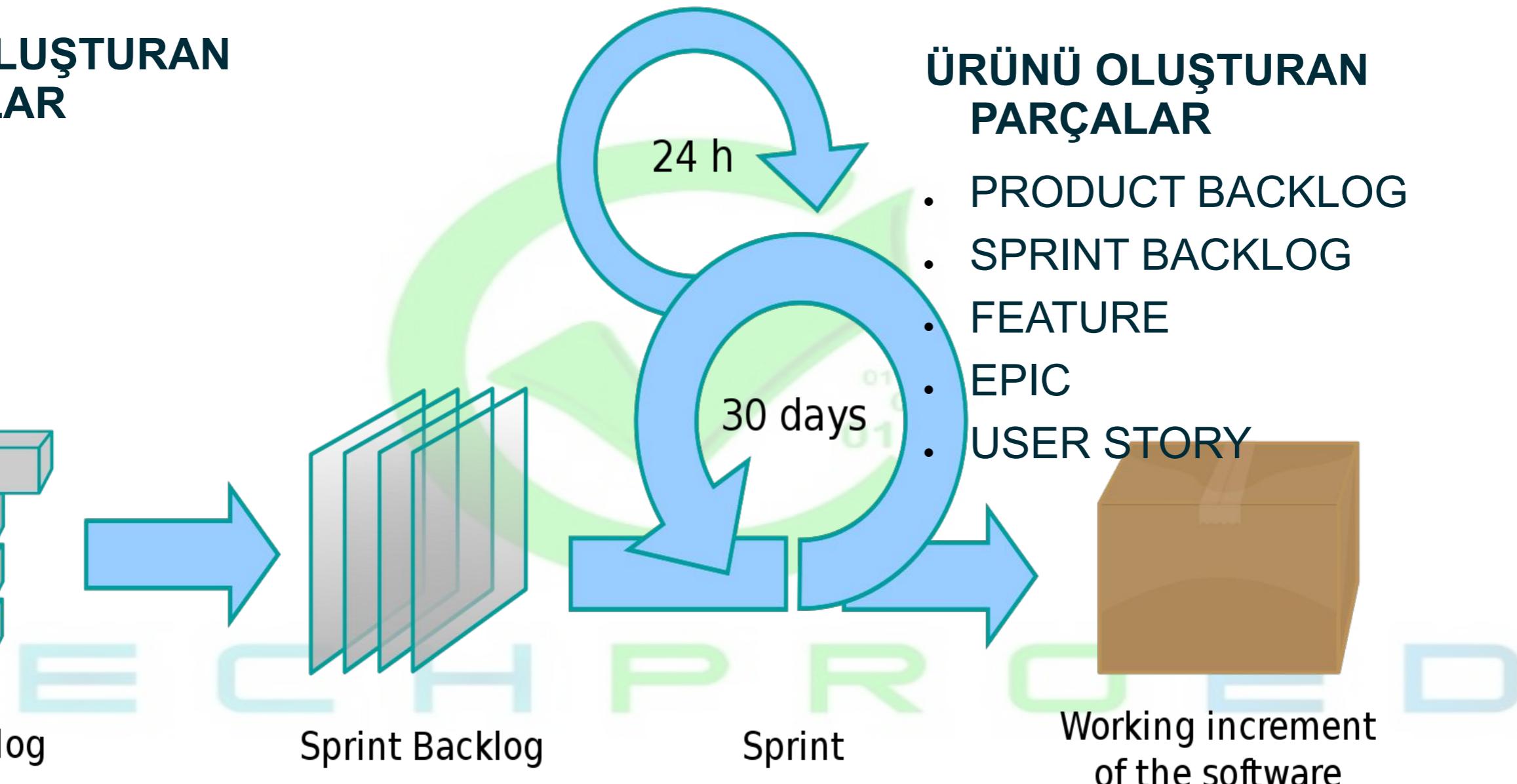
# BÜTÜNÜ OLUŞTURAN PARÇALAR

## SÜRECI OLUŞTURAN PARÇALAR

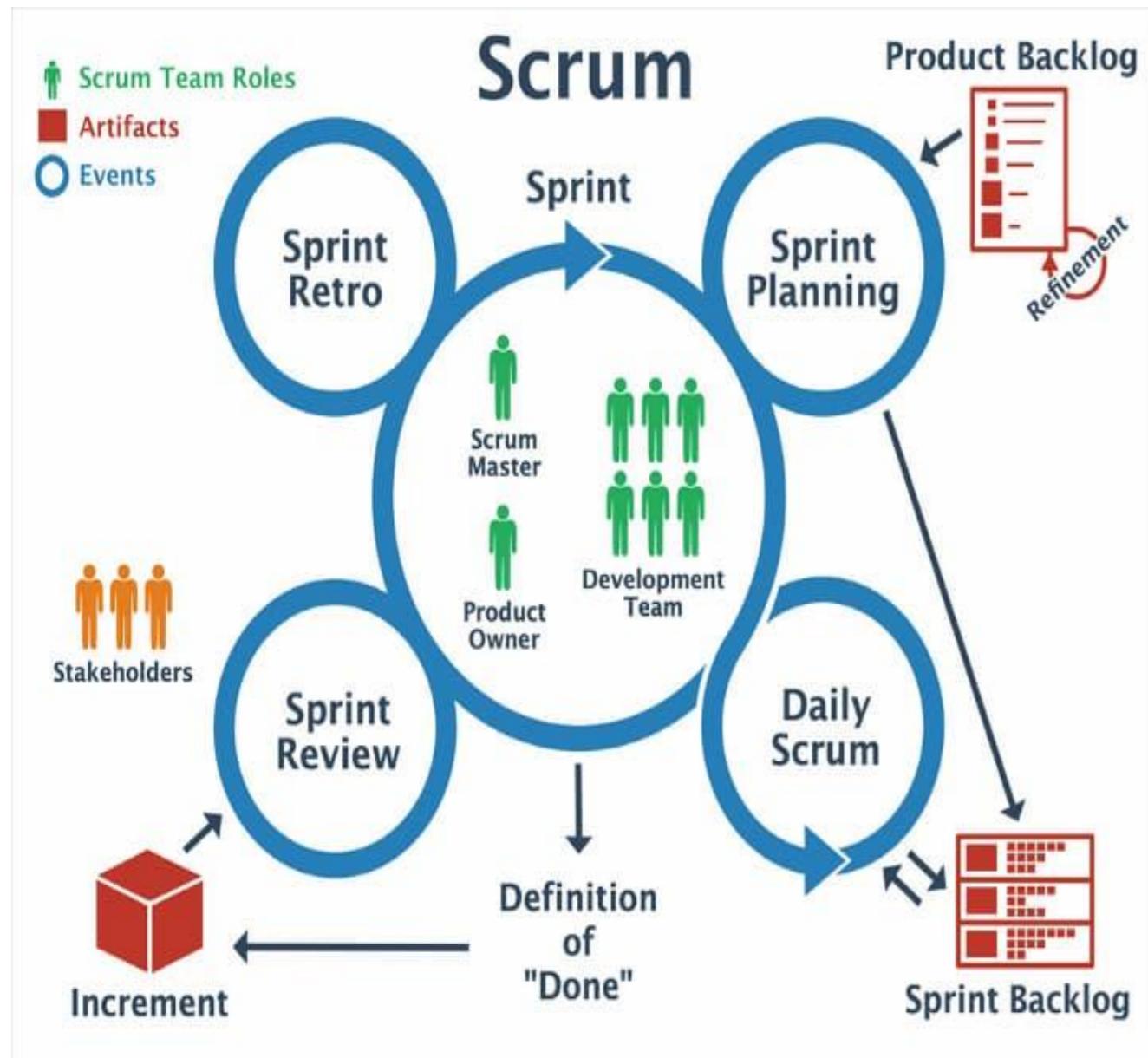
- SCRUM
- SPRINT



Product Backlog



# SCRUM'IN FAYDALARI



- Projenin açık ve net olması hem zaman kazandırır hem de projenin başarılı sonuçlanması sağlar.
- Projenin anlaşılabilir ve yönetilebilir parçalara ayrılması olası sorunları hızlı bir şekilde tespit etmekte ve düzeltmekte zaman kazandırır.
- Bütün ekibin, projenin tüm akışından haberdar olması takım içindeki iletişimini artırır.
- Müşteri ile geliştiriciler arasında güven oluşur ve böylelikle projenin başarılı sonuçlanması beklenir.
- Scrumyönteminin doğru ve anlaşılır bir şekilde uygulanmasını sağlayan kişi **Scrum Master'dır**.

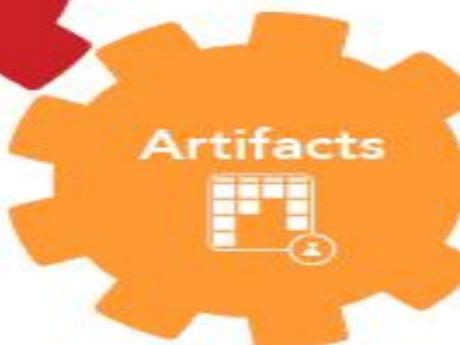
## Scrum's Simple Rules

3 Roles • 5 Events • 3 Artifacts



Without the 3-5-3 you  
are not doing Scrum.

scruminc.



# SPRINT

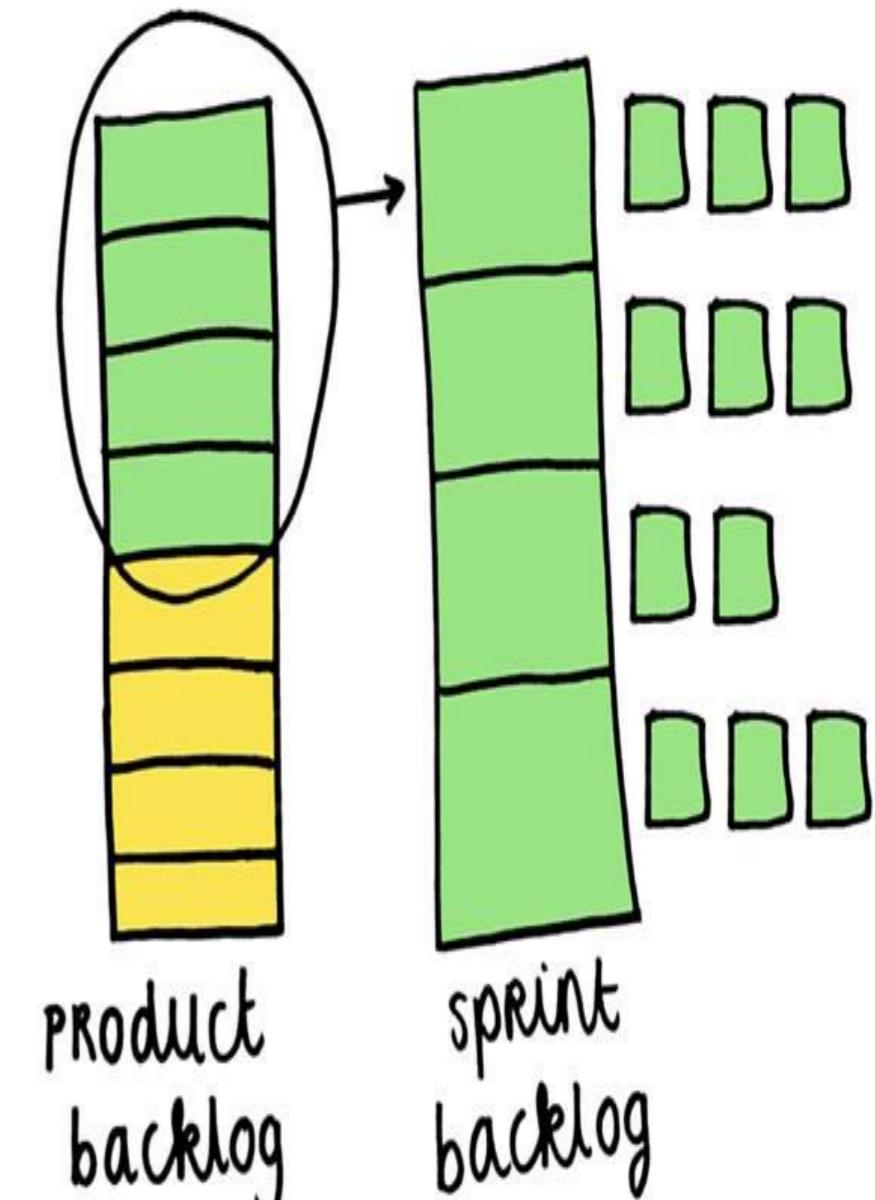
- Scrum süreci 2-4 hafta uzunlukta, ara vermekszin birbirini takip eden Sprint'lerden (koşu) oluşmaktadır.
- Sprint Scrum takımının ritmi, yani kalp atışıdır. Scrum içerisindeki tüm aktiviteler Sprint içerisinde gerçekleşir.
- Bu kısa süreler, takımlara esneklik ve çeviklik avantajı sağlamaktadır. Scrum takımları ürün geliştirmeye başlarken Sprint sürelerini belirler, başlangıç ve bitiş gün ve saatlerine karar verirler, sonrasında da alınan bu karar doğrultusunda ara vermekszin Sprint'leri koşmaya başlarlar.



# PRODUCT BACKLOG

**Product Backlog:** Proje için gerekli olan gereksinimler listesidir. Proje sonunda “Ne üretilmek isteniyor?” sorusuna cevap aranır. Product owner tarafından müşteriden gereksinimler alınır, öncelik sırasına göre sıralanır. Product owner, değişen ihtiyaçlara göre product backlog'a ekleme veya çıkarma yapabilir. Böylece değişim, projenin her aşamasında projeye kolayca entegre edilebilir olur.

**Product Backlog Item:** Product backlog içindeki her bir gereksinime verilen isimdir.

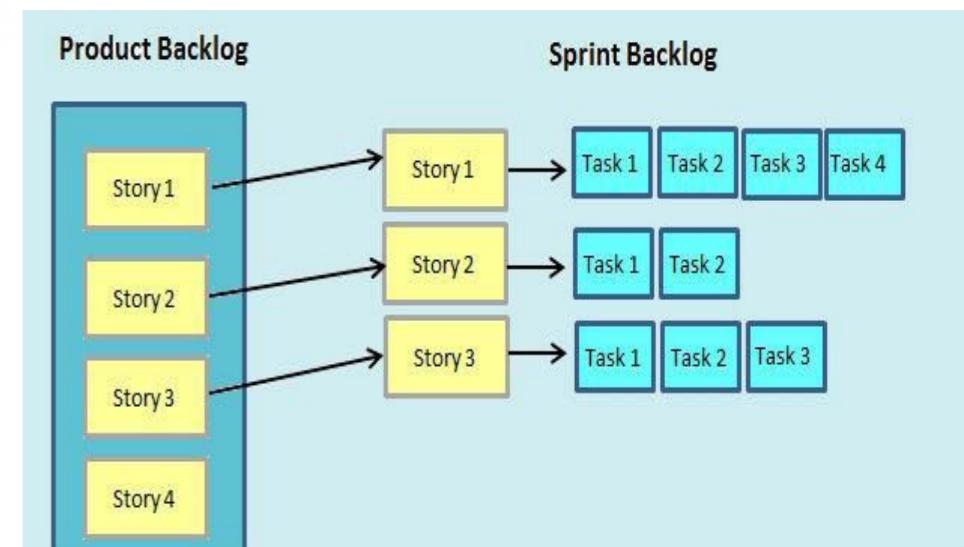
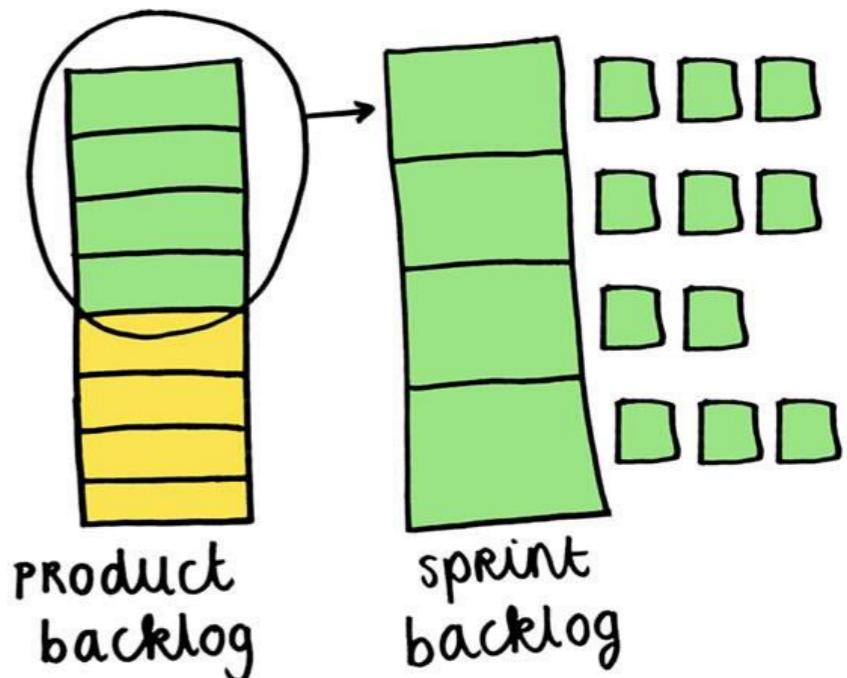


- Product Backlog yaşayan bir dokümandır. Çevresel koşullar, müşteri talepleri, teknolojik gelişmeler ve geliştirme yetkinliğinde meydana gelebilecek değişimlerden en yüksek faydayı sağlayacak şekilde sürekli değişir.
- İyi bir Product Backlog takımı öndeği 2-3 sprint için net anlaşılmış ve yeterince küçük PBI'ları barındırmalıdır.
- Product Owner (Ürün sahibi), Product Backlog'u yöneten yegane kişidir. Bu işi yaparken paydaşlardan, geliştirme takımından ve Scrum Master'dan destek ve yardım alabilir.

TECHPROED

# SPRINT BACKLOG

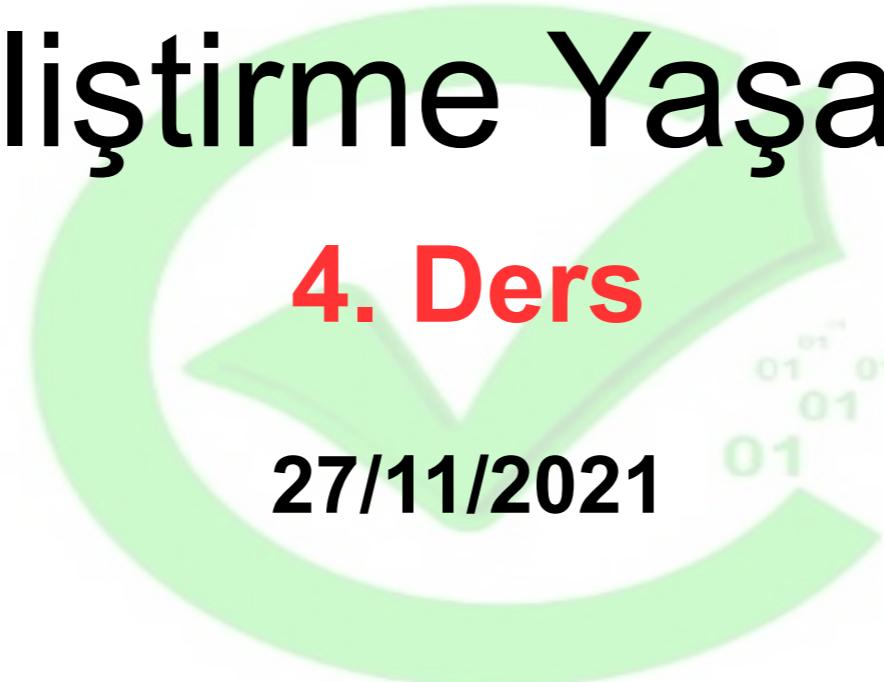
- Sprint Backlog, takımın Sprint boyunca yapacağı işlerin adımlandırılmış, detaylandırılmış, saatlendirilmiş içeriğini teşkil etmektedir.
- Sprint Backlog Development Team'in oluşturduğu, Development Team'in Sprint'te almış olduğu işleri ve bu işleri nasıl tamamlayacaklarına ilişkin stratejilerini içeren bir çıktıdır.
- Sprint Backlog'a alınacak user story'ler belirlenirkende product backlog'daki öncelik sırasını gözetilir. Düşük öncelikli olanları ilk sprint'lere koymazlar.
- Sprint Backlog'a alınacak user story miktarı takımın verdiği point'lere göre değişir.



# **SDLC**

# **Software Development Life Cycle**

## **(Yazılım Geliştirme Yaşam Döngüsü)**



**4. Ders**

**27/11/2021**

**T E C H P R O E D**

# Proje ve Süreç Yönetimi Nasıl Yapılır?

JIRA, Atlassian şirketi tarafından geliştirilen Proje ve Süreç Yönetim aracıdır. Bu araç şirketler tarafından hem iş akış süreçlerini takip etmek hemde müşterilerin veya kullanıcıların isteklerine, sorun ve hatalara çevik(agile) metodolojiler ile yanıt vermek amacıyla kullanılır.

JIRA'nın en temel amaçları:

- Sorun ve Hata Kayıtları,
- Sorun ve Hata Takibi,
- Gelişmiş Proje Yönetimi,
- Derinlemesine Çevik(Agile) Raporlama,
- Çevik(Agile) Yol Haritası Çıkarma,
- Koşu(Sprint) Planlaması.

## Artıları

- +3000'den fazla uygulamaya entegre edilebilir.(Freshdesk, GitHub, GitLab, Zendesk, vb.)
- +1000'den fazla eklenti.
- Özelleştirilebilir İş Akışları
- Gelişmiş takım iç görüleri
- Raporlama Araçları



<https://www.atlassian.com/software/jira>

## Eksileri

- Yeni başlayanlar için biraz karışık
- Rakiplerine göre çok daha pahalı

# Burndown Chart

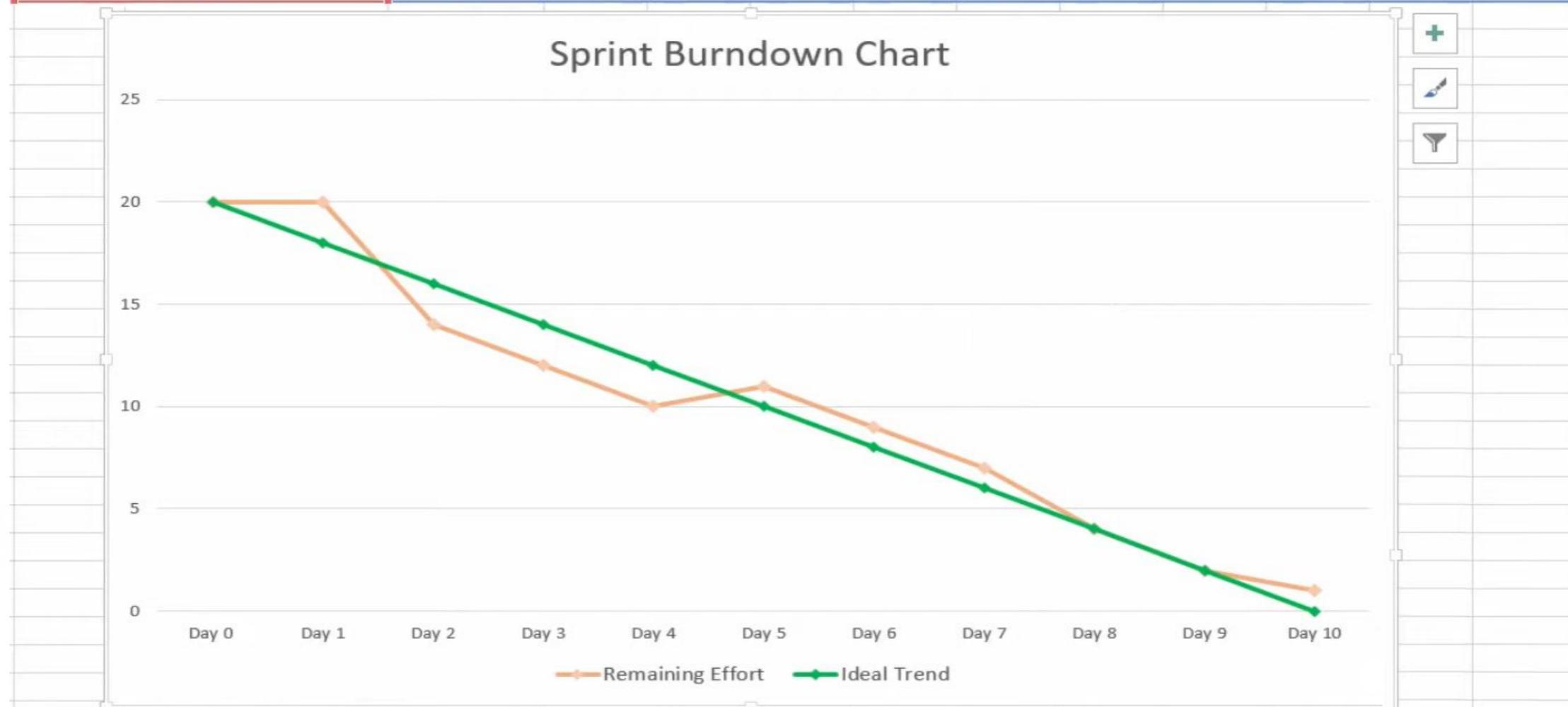
Burndown Grafiği aslında “kalan iş” ve “zaman” arasında çizilen bir çizgi grafiğidir. Bu grafikler sprint görevleri yapılrken ilerlemenin izlenmesi için scrum sprintleri çalıştırın takımlar tarafından kullanılır.

Bu, ekibin işlerinin plana göre gidip gitmediğini veya hedefe zamanında ulaşabilmeleri için değişiklik yapması gerekip gerekmeyeinin takibini kolaylaştırır.



Backlog ID	User Stories	Initial Estimate	Aug-10	Aug-11	Aug-12	Aug-13	Aug-14	Aug-17	Aug-18	Aug-19	Aug-20	Aug-21
		Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
123	Feature 1	2		1		1						
124	Feature 2	3		2								
125	Feature 3	4		1	1		-2	1	1	2		
126	Feature 4	5		1		1				1	1	1
127	Feature 5	6		1	1		1	1	1		1	
<b>Remaining Effort</b>		20	20	14	12	10	11	9	7	4	2	1
<b>Ideal Trend</b>		20	18	16	14	12	10	8	6	4	2	0

Sprint Burndown Chart



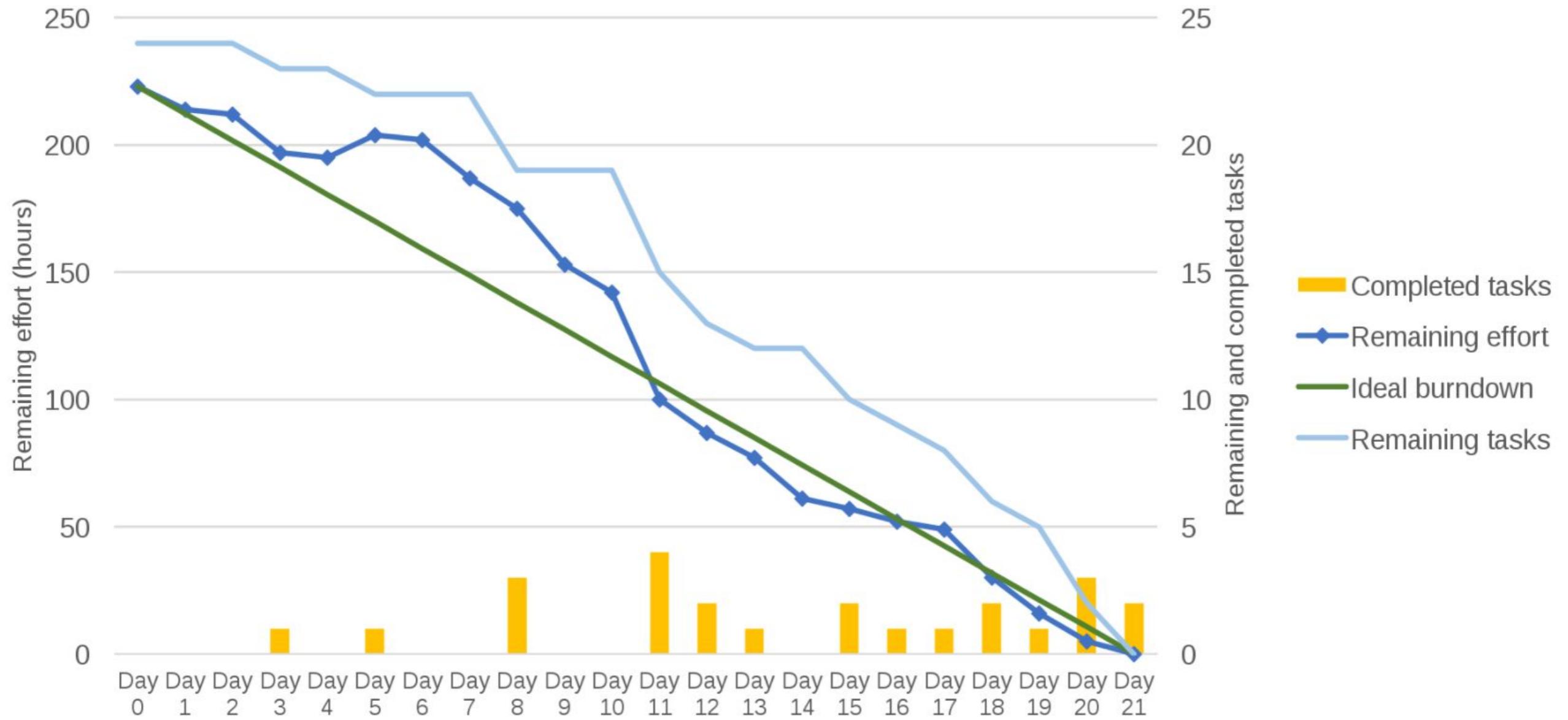


**Programın Arkasında(Behind Schedule):** Progress Line ,Guidline üzerindeyse, ekibinizin programın gerisinde kaldığı anlamına gelir.

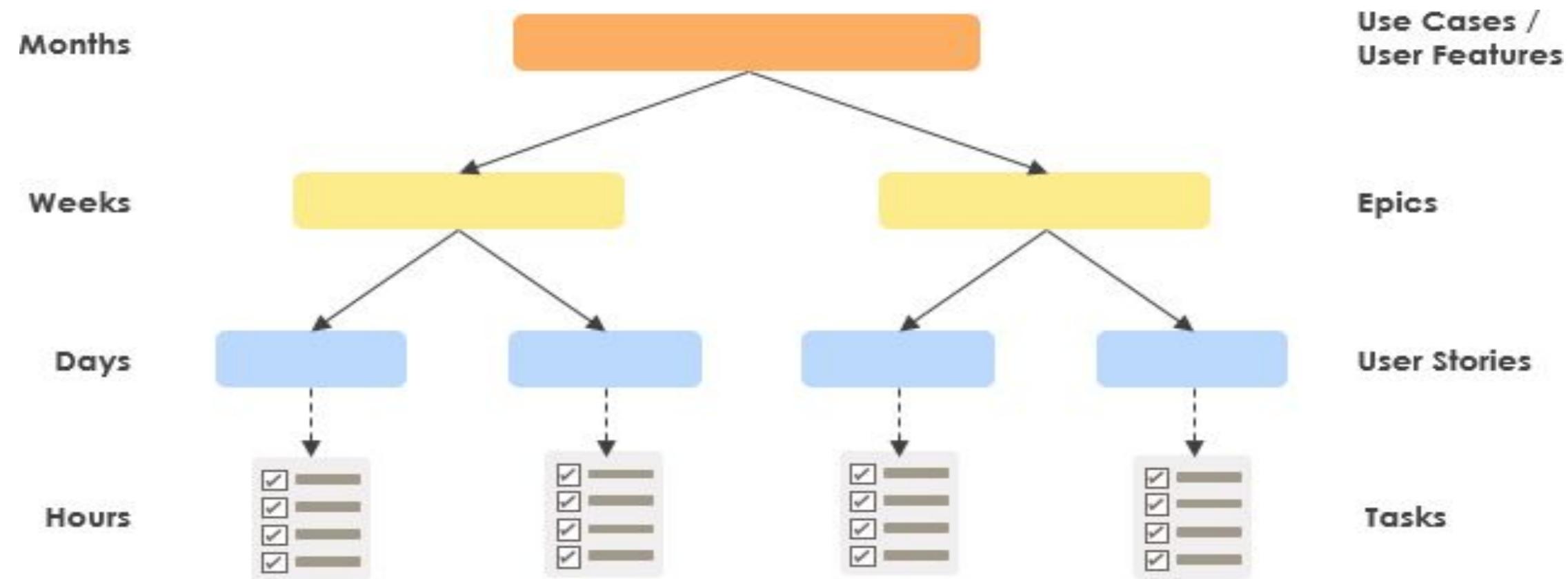
**Yolda(On-Track):** Progress Line ve Guidline birbirine yakınsa, ekibiniz bu hızda ilerlemeye devam ederse hedefi tam on ikiden vuracaktır.

**Programın İllerisinde(Ahead of Shedule):** Progress Line ,Guidline altında ise takım bu şekilde ilerlemeye devam ederse sprint bitiş tarihinden önce elindeki işleri bitirebilir. Bunun sebebi sprint planlanırken Story Point'lerin fazla verilmiş olmasıdır. Sprint boyunca herkesin meşgul olmasını sağlamak için sprinte daha fazla görev ataması yapılabilir.

## Sample Burndown Chart



# User Features, Epics, User Stories, Task



T E C H P R O E D

# USER STORY

User storyler 3 bölümden oluşur. Bunlar;

- Kullanıcının hikaye tanımı (Value statement)
- Kabul kriterleri (Acceptance criterias)
- Bitti tanımı (Definition of done)

User Stories, Agile metodolojisinin bir parçasıdır. Son kullanıcıya sunulacak herhangi bir özelliği veya ürünü açıklama yollarından biridir.

USER STORY #1	
Hikaye Tanımı (Value Statement)	Ürün ekibi yetkilisi olarak, paket yenilemelerini arttırmabilmek için; üyelerimize paketlerinin bitmesine 1 ay kala "yenileme yapmak ister misiniz" mail gönderimi yapılmasını, mail şablonu içerisinde X, Y, Z paket detayları ve ilgili müşteri temsilcilerinin ad, soyad, iletişim bilgilerinin yer olmasını istiyorum.
Kabul kriterleri (Acceptance criterias)	<ul style="list-style-type: none"><li>• Paket bitimine 1 ay kalan kullanıcılar mail gönderimlerinin başarıyla yapılması</li><li>• Üye – müşteri temsilcisi eşleşmesinin doğru yapılmış olması</li><li>• Gönderimlerin sadece mail gönderimine izin verilen kullanıcılar yapılması</li><li>• Mail opt-out kurgularının sağlıklı bir şekilde çalışması</li></ul>
Bitti Tanımı (Definition of Done)	<ul style="list-style-type: none"><li>• Kod kontrol edildi (Code review)</li><li>• Birim testleri yapıldı (Unit tests)</li><li>• Kabul kriterleri karşılandı (Acceptance criterias)</li><li>• Product Owner(PO) story kabulü (PO accepts user story)</li></ul>

TECHPRO E

User Story ID	Description	Acceptance Criteria
US 0001	ATM kullanıcısı olarak, nakit çekebilmem için PIN kodumu girmek istiyorum	PIN dört basamaklı olmalıdır PIN özel karakterlere (sembollere) izin vermemelidir PIN'in 30 saniye içinde girilmesi gereklidir, aksi takdirde işlem iptal edilir
US 0002	Facebook login (Giriş sayfası) geçersiz kimlik bilgileriyle erişilmemelidir	Gecersiz kullanıcı adı ile erişim sağlanamaz Geçersiz şifre ile erişim sağlanamaz Geçersiz kullanıcı adı ve şifre ile erişim sağlanamaz

User Stories üç ortak özelliği barındırır:

- Kısa ve açıklayıcı olma:** Ürün ya da özelliğin kullanarak ne yapılmak istendiğini kısaca anlatabilmenizi sağlar.
- Kullanıcı tipini belirleme:** Kullanıcı hakkında detaylı bilgi verilmese de, kullanıcının karakteristik özelliklerinin ürün ya da yazılımla ilişkisi vurgulanır.
- Problemi tanımlama:** Kullanıcı hikayeleri, ürünün ya da yazılımın çözümü gereken problemleri ortaya çıkarır.

	A	B	C	D	E	F	G	H
1	Feature / Drop	Epic	Testing level			Main Tools	Framework	Tools
2	Registration Page	Admin / Manager / employee / user	UI testing	Backend Testing	Database Testing	Java + Selenium + Api + JDBC	Cucumber BDD	Jira-Xray /Git GitHub
3	Sign in Page	Admin / Manager / employee / user	UI testing	Backend Testing	Database Testing	Java + Selenium + Api + JDBC	Cucumber BDD	Jira-Xray /Git GitHub
4	Operations for Employee	Manage Customers / Manage Accounts	UI testing	Backend Testing	Database Testing	Java + Selenium + Api + JDBC	Cucumber BDD	Jira-Xray /Git GitHub
5	Operations for Customer	My Accounts / Transfer Money	UI testing	Backend Testing	Database Testing	Java + Selenium + Api + JDBC	Cucumber BDD	Jira-Xray /Git GitHub
6	Operations for Admin	User management / Metrics / Health /Configurations Audits / Logs	UI testing	Backend Testing	Database Testing	Java + Selenium + Api + JDBC	Cucumber BDD	Jira-Xray /Git GitHub
7	Operations for Manager	User management / Metrics / Health /Configurations Audits / Logs	UI testing	Backend Testing	Database Testing	Java + Selenium + Api + JDBC	Cucumber BDD	Jira-Xray /Git GitHub

	A	B	C	D	E
1	User_Story_ID	Description	Acceptance Criteria	Priority	Validation
2	US_001	System should allow any user to register with valid credentials validating the success message	There should be a valid SSN respecting the "-" where necessary, it should be 9 digits long	High	UI and Backend Testing
3			There should be a valid name that contains chars and cannot be blank		
4			There should be a valid lastname that contains chars and it is a required field		
5			You can provide chars and digits to describe a valid address along with zip code		
6			User should provide 10 digit-long mobilephone number as a required field respecting the "-"		
7			User cannot use just digits, but can use any chars and digits along with them and of any length		
8			You should provide a valid email format that contains "@"sign and "." extensions		
9					
10	US_002	System should not allow anyone to register with invalid credentials seeing the message "If you want to sign in, you can try the default accounts:- Administrator (login="admin" and password="admin") - User (login="user" and password="user").	Any field on the registration page should not be left blank	High	UI and Backend Testing
11			SSN number cannot be of any chars nor spec chars except "-"		
12			Mobilephone number cannot be of any chars nor spec chars except "-"		
13			email id cannot be created without "@" sign and "." extensions		
14					
15	US_003	Registration page should restrict password usage to a secure and high level passcode	There should be at least 1 lowercase char for stronger password and see the level chart change accordingly	High	UI and Backend Testing
16			There should be at least 1 uppercase char and see the level chart change accordingly		
17			There should be at least 1 digit and see the level chart change accordingly		
18			There should be at least 1 special char and see the level bar change accordingly		
19			There should be at least 7 chars for a stronger password		
20					
21	US_004	Login page should accessible with valid credentials	There should be a valid username and password validating the success message to login	High	UI and Backend Testing
22			There should be an option to cancel login		
23					
24	US_005	Login page should not be accessible with invalid credentials	User cannot login with invalid username validating the error message	High	UI and Backend Testing
25			User cannot login with invalid password validating the error message		
26			User cannot login with invalid username and password validating the error message		
27			User with invalid credentials should be given an option to reset their password		
28			User should be given the option to navigate to registration page if they did not register yet		
29					

USER STORY	EPIC
Çalışabilir en küçük iş parçasıdır, bölünemez.	Kapsamlı bir iş parçasıdır, alt parçacıklara bölünebilir (User story'lere)
Bir sprint içerisinde geliştirilebilir, teslim edilebilir olmalıdır	Birden fazla sprint içerisinde geliştirmelere devam edilebilir.
Eforlanması daha kolaydır	Kapsam daha geniş olduğu için eforlanması daha zordur.
Geliştirme sonunda değerli ve eksiksiz olan dikey bir işlevsellik dilimi sunmalıdır.	Release fazlarından bir tanesi diyebiliriz.

# Takım Kapasitesi ve Puanlama

User storyler sprint backlog'a alınırken takım tarafından puanlanır (**POINT**)

- Sprint baslangic toplantisi veya refinement toplantisinda User Story okunur ve development team'in tum üyeleri user story'e point verir
- Verilen pointler kartlara写ざるに示される。
- Pointler Fibonacci Serisindeki sayilara gore verilir.  
1 1 2 3 5 8 13 21 34 .....
- Her üye niçin bu point'i verdigini açıklar ve team'in ortak karariyla user story'nin point'i belirlenir
- Team'deki developer ve QA sayisina gore takım (team) kapasitesi (**capacity**) belirlenir.

1 point = 1 günlük çalışma yani 8 saat



**Örnek:** 5 developer'in takım kapasitesi nedir?

$5 \times 1 = 5$  (5 developer'in günlük kapasitesi 5 point)

1 haftalık kapasitesi  $\Rightarrow 5 \times 5 = 25$  point

2 haftalık kapasitesi = 50 point

**Örnek:** 2 QA'in takım kapasitesi nedir?

$2 \times 1 = 2$  (2 developer'in günlük kapasitesi 2 point)

1 haftalık kapasitesi  $\Rightarrow 5 \times 2 = 10$  point

2 haftalık kapasitesi = 20 point

Development team'de 1 kişinin 2 haftalık sprintti yapacağı iş miktarı  $5+5=10$  point'tır.



# SCRUM SÜRECi

PROJE  
YÖNETİMİ



Inputlar  
Yönetimden Gelir



Product Owner



Product  
Backlog



Takım ile yapılır  
Sprint  
Planlama  
Toplantısı

Sprint  
Backlog



Daily Stand Up  
Toplantısı



Sprint Review  
Toplantısı



Tamamlanan İş



Sprint Retrospective  
Toplantısı

# TOPLANTILAR

- Daily Scrum (Daily Stand Up)  
(Günlük Scrum Toplantısı)
- Sprint Planning (Sprint Planlama)
- Sprint Review (Sprint İncelemesi)
- Sprint Retrospective Meeting  
(Geriye Dönük Sprint Değerlendirilmesi)



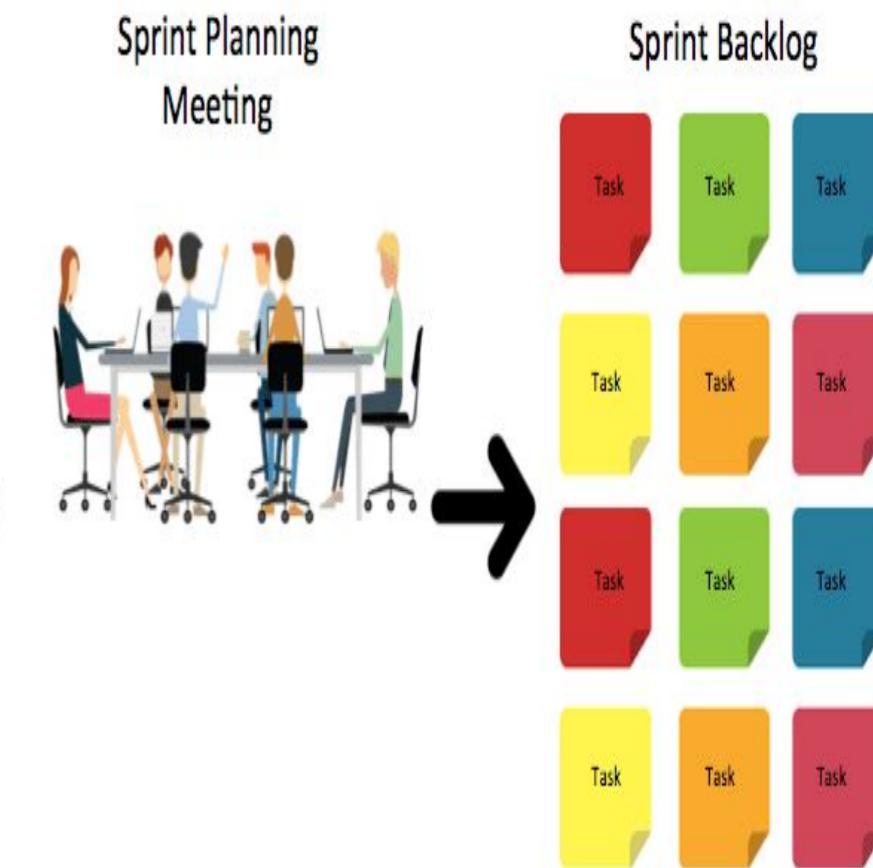
# Sprint Planning Meeting

Bu, her Sprint'i başlatan etkinliktir

- Product Owner ve Development ekibinin Sprint'e hangi Ürün İş Listesi Öğelerinin (PBI'ler) dahil edileceğini tartıştığı yerdir.
- Product Owner Sprint'e potansiyel olarak dahil edilmek üzere her PBI'ya öncelik verme hakkına sahip olsa da, Geliştirme ekibi yanıt vermeye, sorunları gündeme getirmeye ve gerektiğinde geri itmeye teşvik edilir.
- Development team daha sonra hız, kaynaklar ve mevcut zaman ve kaynakları etkileyebilecek faktörler hakkında bilgi sahibi olduklarıında Sprint'te kaç PBI sunabileceklerini tahmin eder.
- **Sprint Planlama Toplantısının sonucu, herkesin kabul ettiği gerçekçi ve ulaşılabilir bir Sprint Hedefi ve Sprint İş Listesi elde etmektir.**

## Sprint Planning Meeting

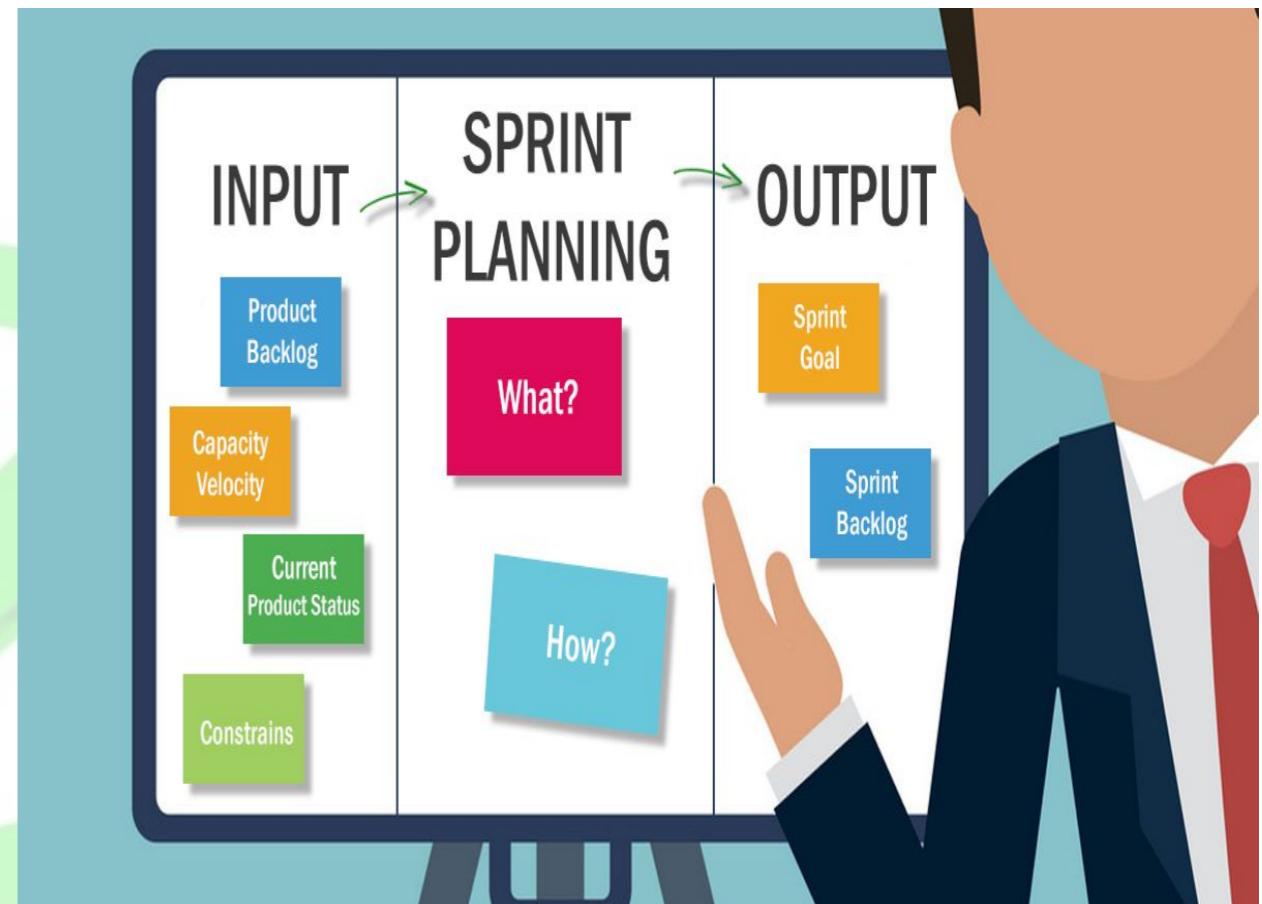
Sprint Planning  
Meeting



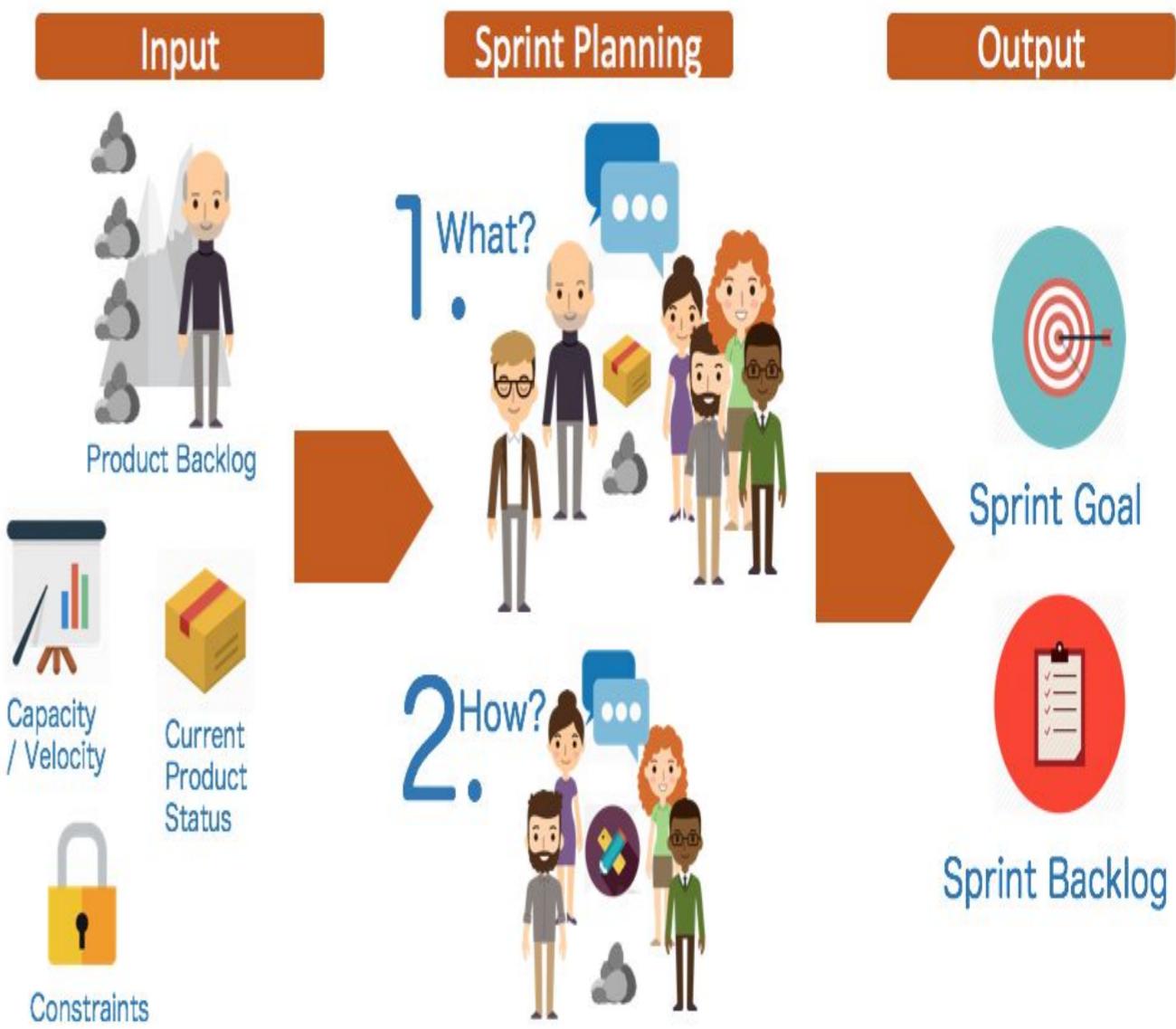
Sprint Planlamada şu sorulara cevap verir;

- 1) Başlayan Sprint'te task olarak ne teslim edilebilir? (Ne Yapacağınız)
- 2) Uygulamayı teslim etmek için gerekli olan iş nasıl başarılacak? (Nasıl Yapacağınız)
2. madde için gerekirse Sprint süresi içerisinde eski adıyla grooming yeni adıyla refinement (detaylandırma) aktivitesi gerçekleştirilebilir.

Scrum Master, toplantılarının yapılmasını ve Development Team'in etkinliğin amacını anlamasını sağlar.



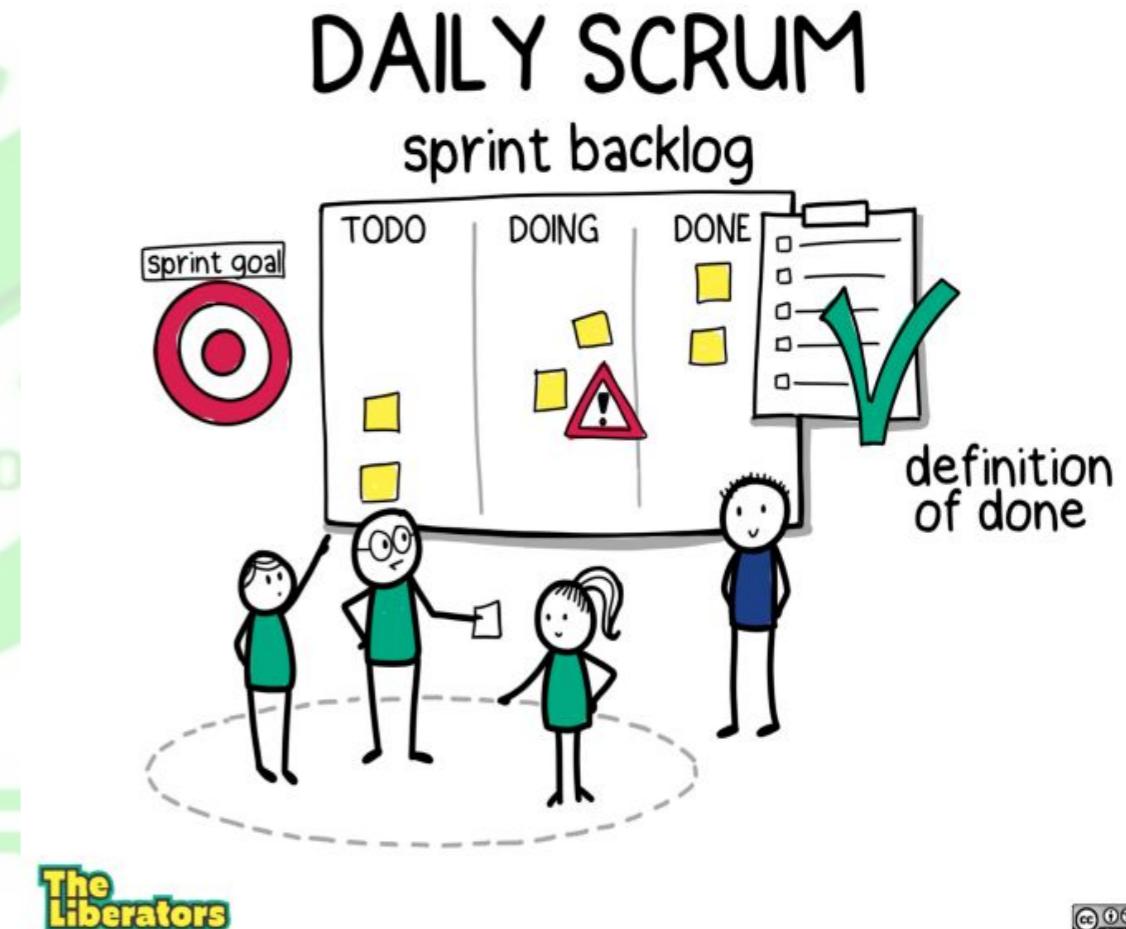
- Bir önceki Sprint Retrospektif Toplantısı'nda alınan kararlar içinde Sprint Planlama Toplantısını etkileyen aksiyonlar varsa bunları yerine getirildiği kontroledilir.
- Herşeyin başlangıcı iyi bir planlamadır. Sprint'e iyi başlarsanız başarılı bir şekilde bitirme şansınız yüksek olur. İyi Scrum Master, Scrum Takımı'nın iyi planlama yapmasını sağlar.
- İş Listesi Maddeleri'nde kabul kriterleri belirtilmiş mi?



# Daily Scrum (Daily StandUp) (Günlük Scrum)

Scrum zamanınızı ve kaynaklarınızı verimli bir şekilde kullanmayı amaçlar.

- Yaklaşık 15 dakika kadar sürer. Ayağa kalkmak zorunlu değildir. Ancak, birçok takım bu toplantıyı kısa ve öz tutmak için yararlı bir teknik olarak bulmaktadır.
- Geliştirme Ekibinin oto-kontrol yapması, Sprint Hedefi'ne ulaşma yolundaki ilerlemeyi değerlendirmesi ve önümüzdeki 24 saat boyunca faaliyetlerini gözden geçirmesi ve planlaması için bir fırsattır.
- Genelde dün neler yaptı? Bugün ne üzerinde çalışacağınız? Herhangi bir sıkıntı veya engel ile karşılaşmadık mı?



Sorulara verilecek cevaplar açık ve net olmalıdır.

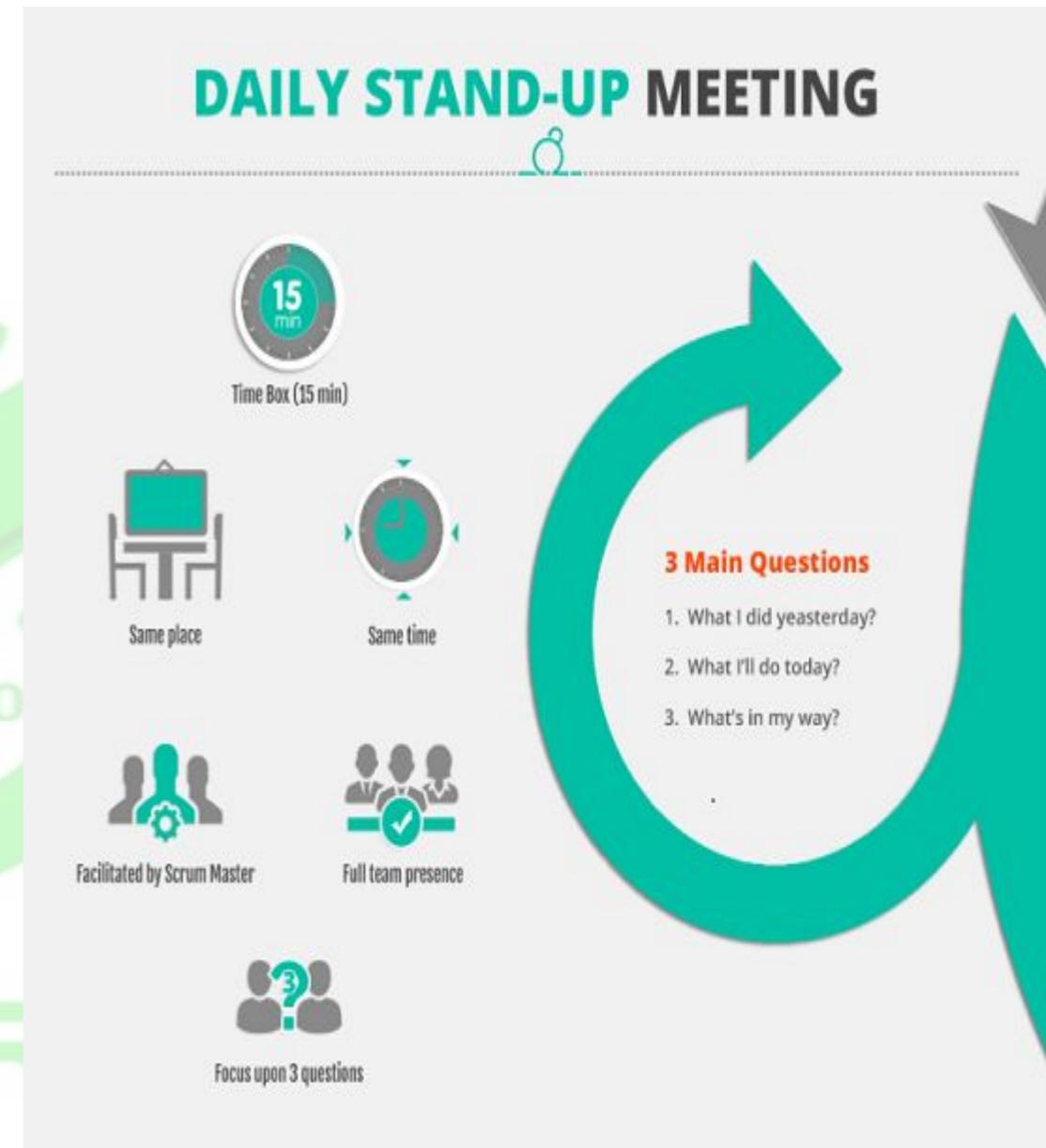
7 kişiden oluşan bir Development Team'i düşünürsek bir üyenin konuşabileceği 128 saniyesi olacaktır. Her soruya cevap verilecek süre ise 43 saniyedir.

### İyi örnek:

Dün müşteri tanımlama ekranını test ettim, herhangi bir sorunla karşılaşmadım.

Bugünse müşteri tanımlama ekranının, bireysel kredi planı çıkarma ekranı arkasındaki bağlantıyı test edeceğim eğer vaktim kalırsa da vadeli müşteri hesap ekranını test edeceğim. Sonra da Zeynep ile code review yapmayı planlıyorum.

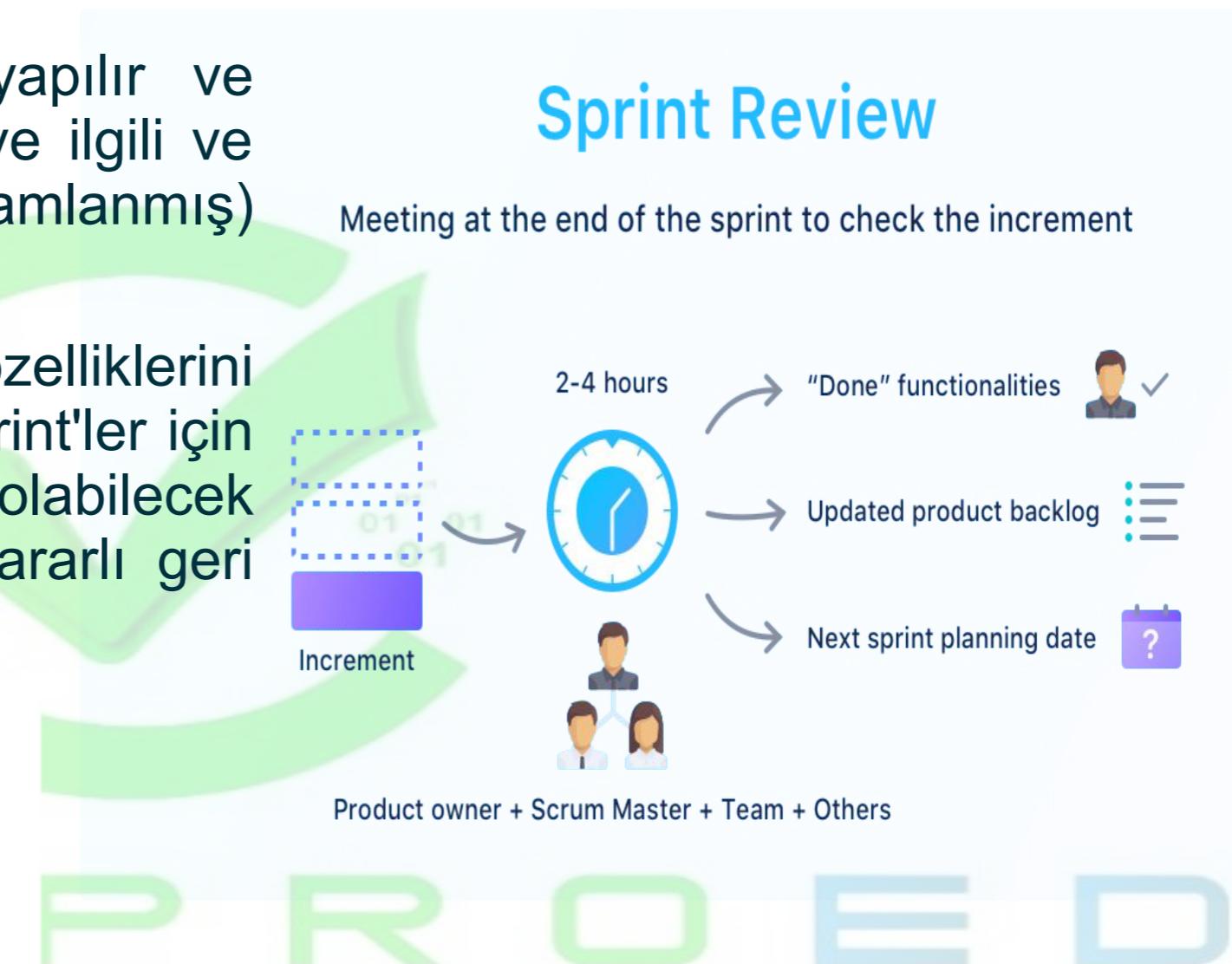
Önümde veritabanı bağlantı hatası var. Sürekli aynı hatayı alıyorum. Sizlerden biri karşılaştıysa beraber bakabilir miyiz?



# Sprint Review (Sprint İncelemesi)

- Genellikle Sprint'in son gününde yapılır ve Stakeholder'lara(müşteriler, yönetim ve ilgili ve ilgilenilen diğer herkes) "done" (tamamlanmış) yapılan işi gösterme fırsatı verir.
- Sprint sırasında üretilen çalışma özelliklerini göstermenin yanı sıra, gelecekteki sprint'ler için çalışmayı yönlendirmeye yardımcı olabilecek Ürün İş Listesi'ni ekleyebileceğiniz yararlı geri bildirimler de alıyorsunuz.

T

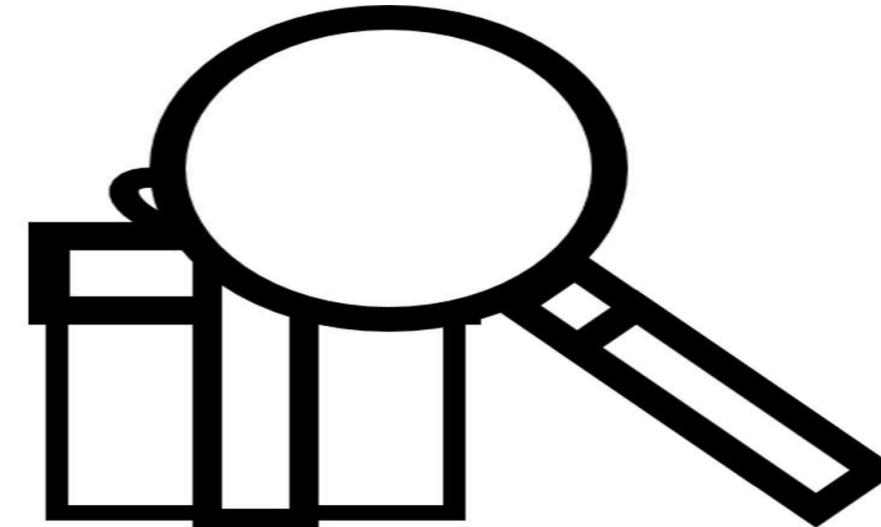


# Sprint Review

**Inspect:** The Increment

**Adapt:** The Product Backlog

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.



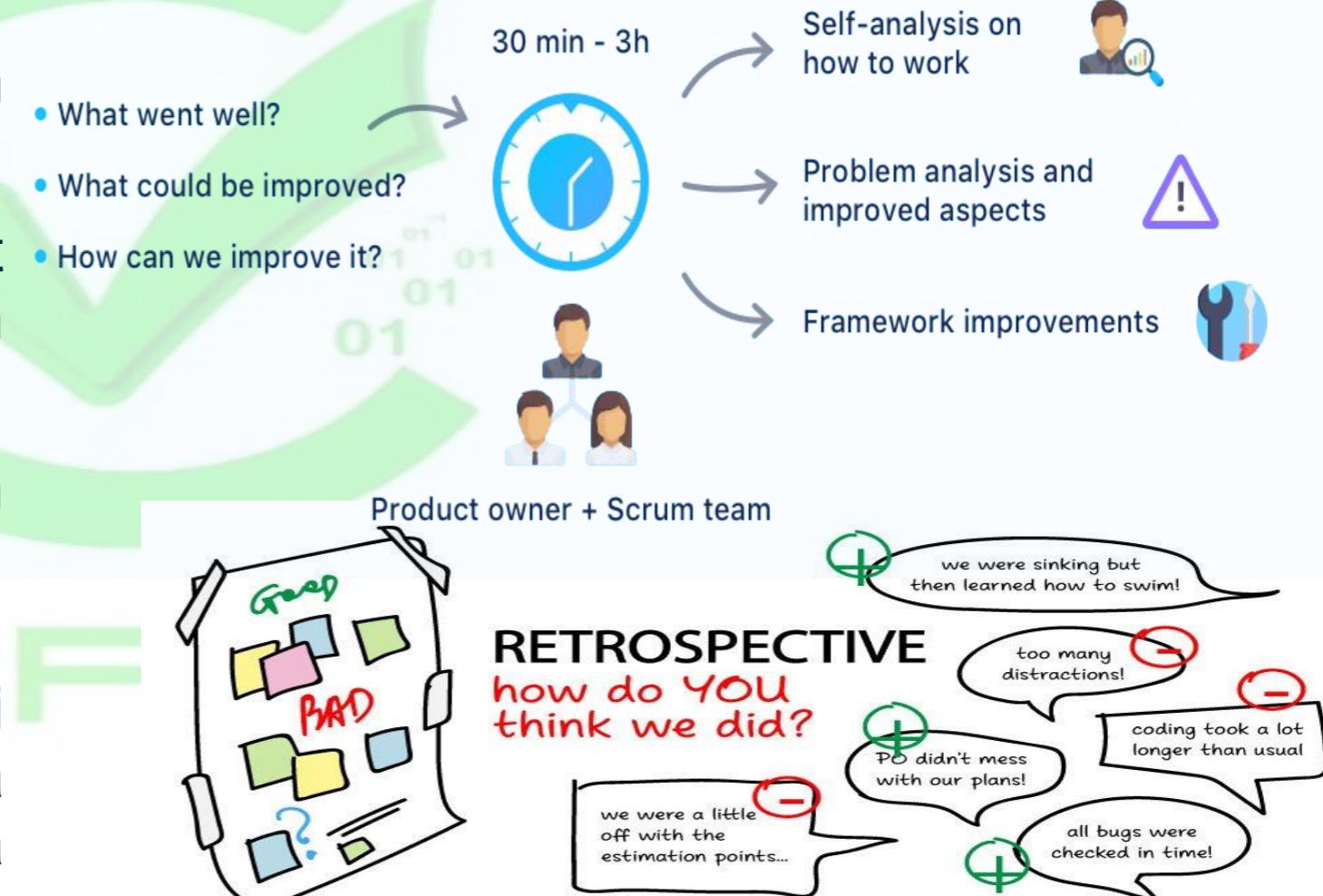
- SprintReview Toplantısı'nın sahibi Product Owner'dır.
- Burada yapılması gereken şey stakeholders'a “ürünün onların ürünü” olduğunu vurgulamaktır. Onlara anlatacaklarınıza, göstereceklerinizle sizden istediklerinin örtüşüp örtüşmediğinin bu toplantıda anlaşılacağını anlatmaktadır.
- Stakeholders çalışan ürünü görerek bu ürüne eklenmesi istedikleri yeni özellikler ya da değiştirilmesini istedikleri özelliklerin hangi niteliğe, görselliğe, yetkinliğe sahip olması gerektiğini yüz yüze, birinci ağızdan söyleyebilirler.

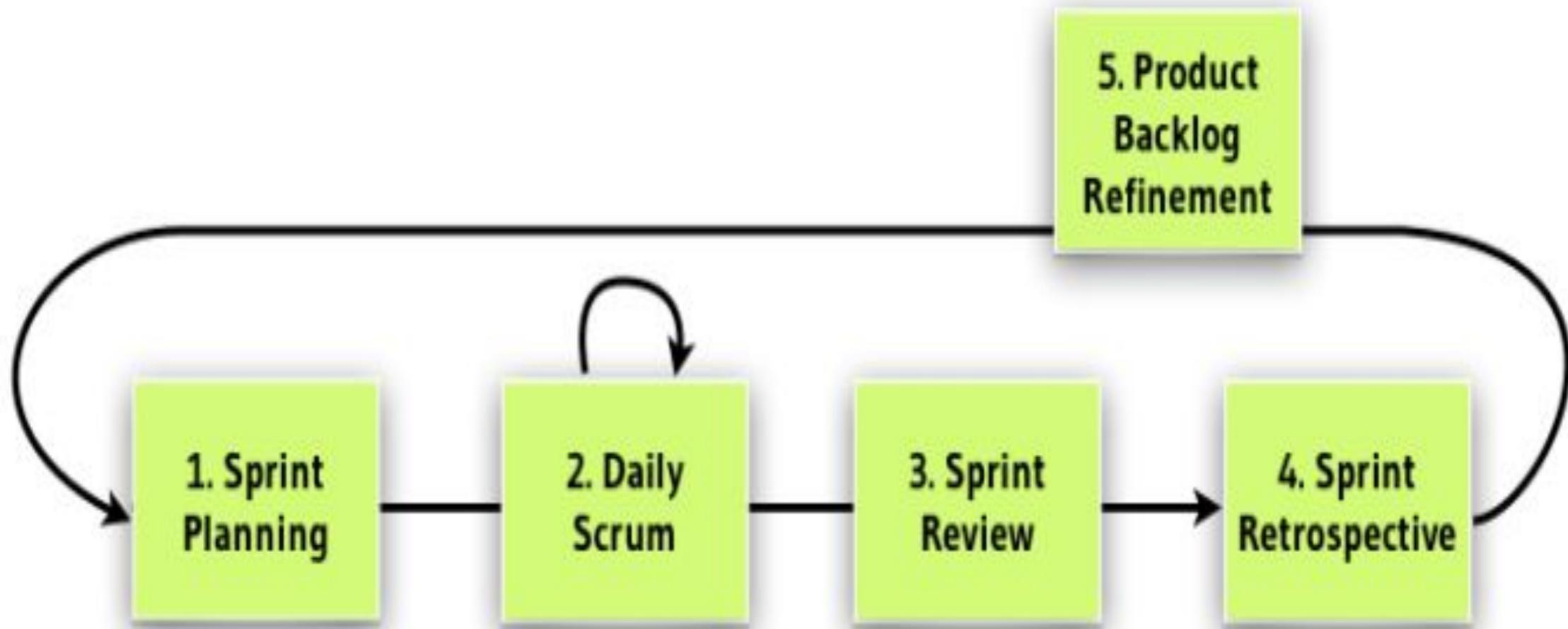
# Sprint Retrospective (Sprint Retrospektif)

- Retrospective toplantıları bir sonraki Sprint'te iş süreçlerini iyileştirmek için geçmiş Sprint'in incelendiği ve "nasıl daha iyi performans gösterebiliriz?" sorusuna cevap aranan toplantılardır. Bu toplantıya geliştirme ekibi, scrum master ve product owner katılır.
- Sprint'teki son toplantıdır ve "sprint review" toplantısının hemen ardından yapılır.
- Scrum ekibi gelecekteki Sprint'ler için nelerin geliştirilebileceğini ve nasıl yapmaları gerektiğini gözden geçirir.
- Ne tür engellerle karşılaşlıklar ve hangi fikirlerin ve güncellemelerində fazla gelişim sağlamalarına katkıda bulunduğuunu değerlendirirler.

## Sprint Retrospective

Meeting after Sprint Review to review processes





Sprint Execution

T E C H P R O E D

- Yazılım testi olmadan yazılım istenen kaliteye ulaşamaz.
- Testerler application'in kodlarına dahil olmasalar bile, kodun kalitesini artırmak için Developerlarla yakın çalışmalıdır.
- En iyi sonuçlar için yazılım testi ve kodlamanın iç içe gitmesi önemlidir.
- Test, yazılımın kalitesini değerlendirmeye yardımcı olur.
- Tüm test faaliyetleri planlama gerektirir.
- Test, yazılımın tam olarak gereksinimlerde belirtildiği gibi davranışını ve uygulanmasını sağlar.
- İhtiyaca(requirement) uygun olmayan her şey bir hatadır(defect,BUG).
- İyi test edilmiş yazılım kaliteli ve daha iyi geribildirim ve yorumlar anlamına gelir (iyi feedbackler).

# AGILE 12 PRENSİP

- 1) İlk önceliğimiz kaliteli yazılımı müşteriye teslim edebilmektir.
- 2) Değişiklikler projenin ilerki aşamalarında dahi olsa kabul edilir.
- 3) Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır.
- 4) İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdır.
- 5) Projelerin temelinde motive olmuş bireyler yer almmalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır.
- 6) Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüzyüze iletişimdir.

- 7) Çalışan yazılım ilerlemenin birincil ölçüsüdür.
- 8) Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir.
- 9) Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır.
- 10) Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca artırılması sanatı, olmazsa olmazlardandır.
- 11) En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar.
- 12) Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

# **SDLC**

# **Software Development Life Cycle**

## **(Yazılım Geliştirme Yaşam Döngüsü)**

**5. Ders**

**04/12/2021**

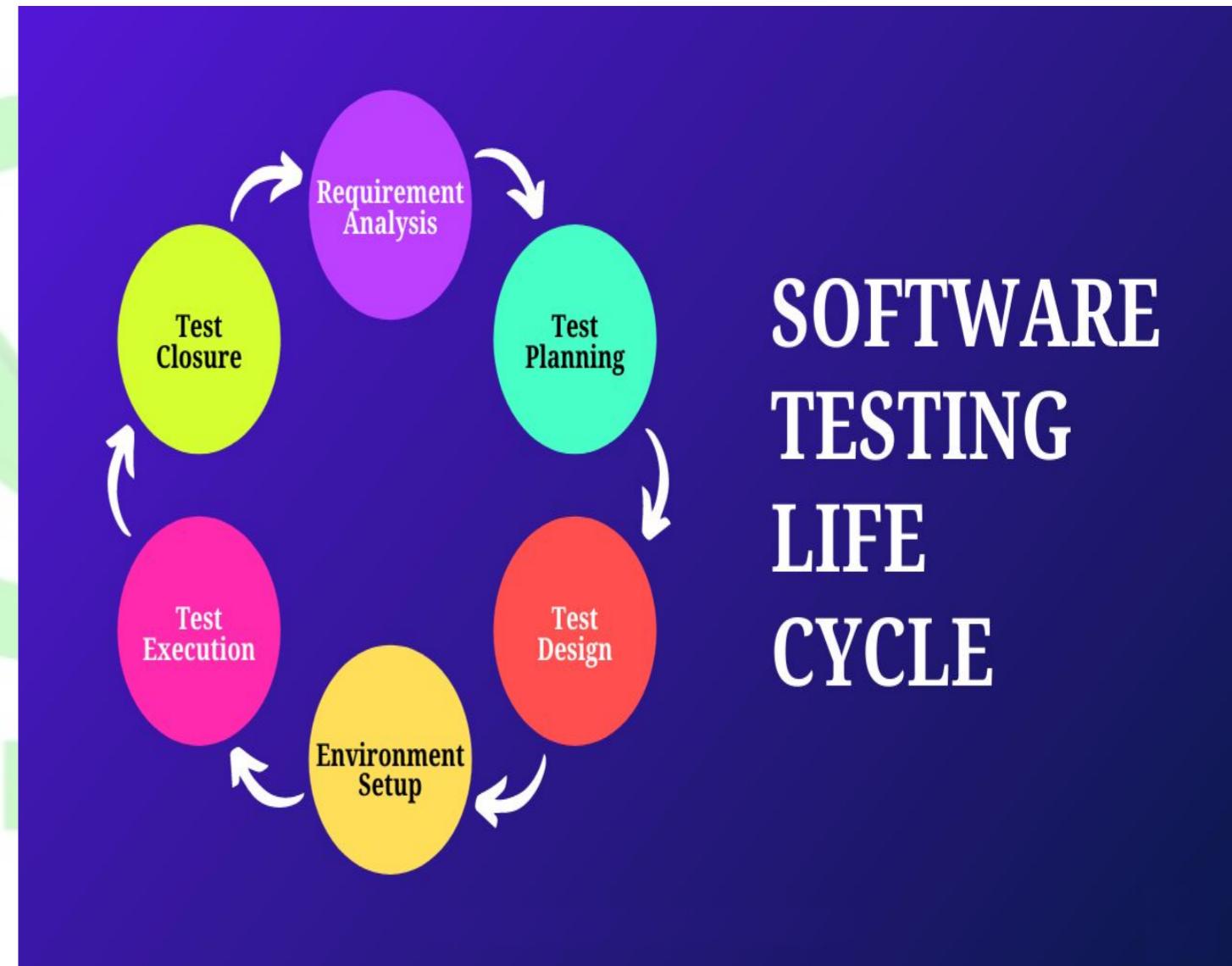
**T E C H P R O E D**

# STLC

## Software Testing Life Cycle

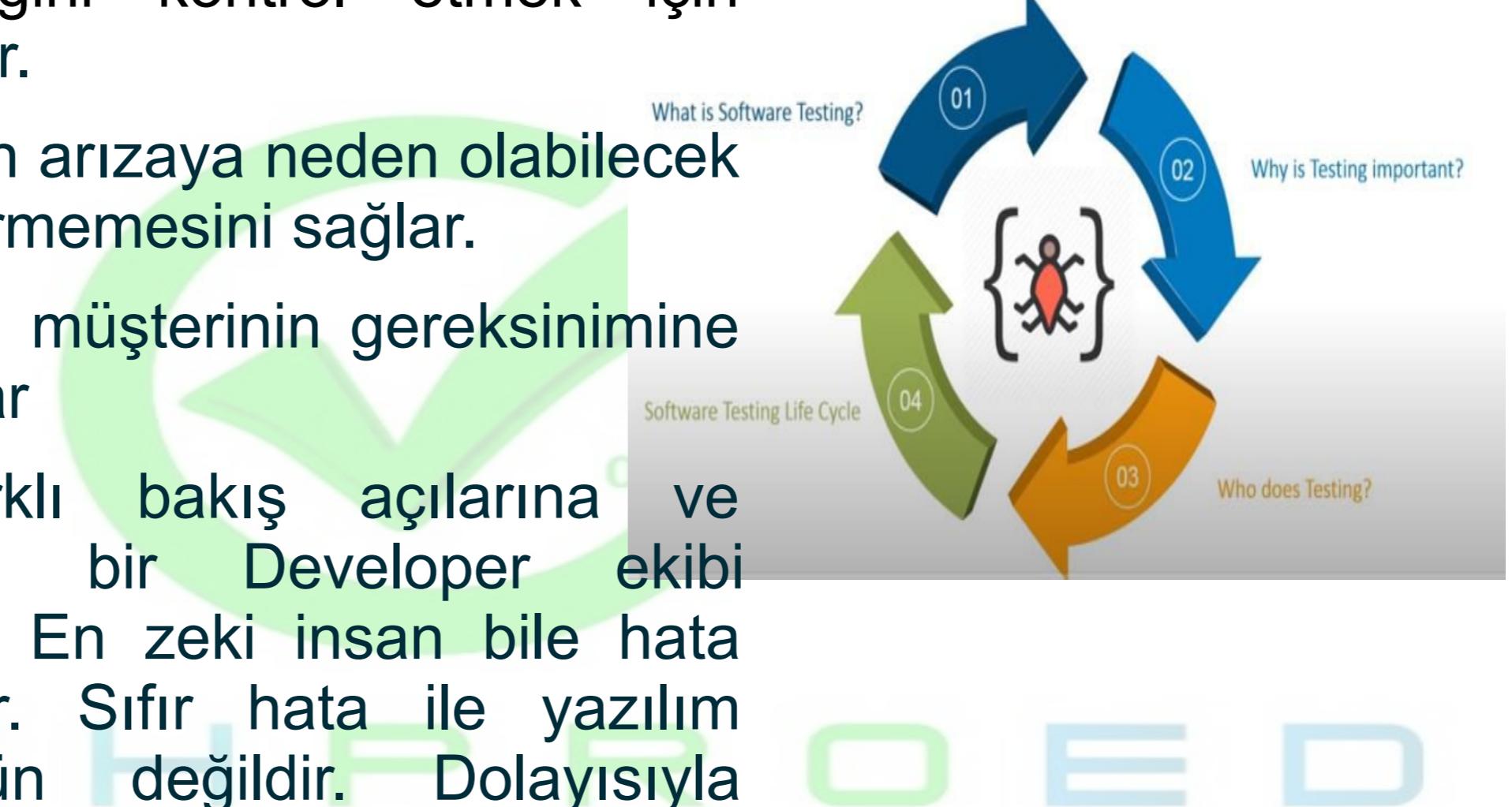
### Yazılım Test Yaşam Döngüsü

(STLC), yazılımı test etmek ve kalite standartlarının karşılandığından emin olmak için kullanılan bir süreçtir. Testler sistematik olarak birkaç aşamada gerçekleştirilir. Ürün geliştirme sırasında, bir ürün piyasaya sürülmeye uygun görülene kadar STLC'nin aşamaları birden çok kez gerçekleştirilebilir



# Software Testing Nedir ve Niçin Önemlidir?

- Yazılımın güvenilirliğini kontrol etmek için yazılım testi gereklidir.
- Yazılım testi, sistemin arızaya neden olabilecek herhangi bir hata içermemesini sağlar.
- Yazılım testi, ürünün müşterinin gereksinimine uygun olmasını sağlar
- Yazılım, hepsi farklı bakış açıllarına ve yaklaşımlara sahip bir Developer ekibi tarafından geliştirilir. En zeki insan bile hata yapma eğilimindedir. Sıfır hata ile yazılım oluşturmak mümkün değildir. Dolayısıyla Testing yaşam döngüsü yazılım geliştirme döngüsüne muhakkak dahil edilir.



- Yazılım testi olmadan yazılım istenen kaliteye ulaşamaz.
- Testerler uygulamanın kodlarına dahil olmasalar bile, kodun kalitesini artırmak için Developerlarla yakın çalışmalıdır.
- En iyi sonuçlar için yazılım testi ve kodlamanın iç içe gitmesi önemlidir.
- Test, yazılımın kalitesini değerlendirmeye yardımcı olur.
- Tüm test faaliyetleri planlama gerektirir.
- Test, yazılımın tam olarak gereksinimlerde belirtildiği gibi davranışını ve uygulanmasını sağlar.
- İhtiyaca (requirement) uygun olmayan her şey bir hatadır (defect, BUG).
- İyi test edilmiş yazılım kaliteli ve daha iyi geribildirim ve yorumlar anlamına gelir (iyi feedbackler).

- Yazılım testi, geliştirilen bilgisayar yazılımının; doğruluğunu, bütünlüğünü ve kalitesini belirlemek için kullanılan bir süreçtir.
- Ürünün son kullanıcılarla sunulmadan önce düzeltilebilmesi için yazılımda hata bulmak amacıyla yapılan bir dizi etkinliği içerir.
- Basit bir ifadeyle, yazılım testi, gerçek sonuçların beklenen sonuçlarla eşleşip eşleşmediğini kontrol etmek ve yazılım sisteminin hatasız olmasını sağlamak için yapılan bir etkinliktir.

## Software Testing Best Practices



- Test Early, Test Often
- Test Coverage and Code Coverage
- Automate Testing
- Testable Requirements
- End to End Testing
- Bug Prevention



**T E C H**

- Gereksinim Analizi
  - Test Planı
  - Test Senaryosu Geliştirme
  - Test ortamı Kurulumu
  - Test Gerçekleşmesi
  - Test Döngü Kapanışı
- 
- The diagram illustrates the Software Testing Life Cycle (STLC) with six sequential phases:
- Requirement Analysis** (Blue bar): Represented by a blue circle with a document icon.
  - Test Planning** (Yellow bar): Represented by a yellow circle with a checklist icon.
  - Test Case Development** (Red bar): Represented by a red circle with a laptop and gear icon.
  - Environment Setup** (Purple bar): Represented by a purple circle with a wrench and gear icon.
  - Test Execution** (Green bar): Represented by a green circle with a server icon.
  - Test Cycle Closure** (Blue bar): Represented by a blue circle with a document and checkmark icon.
- Arrows connect each phase to its corresponding step in the list above.

# 1) Requirements Analysis (Gereksinimlerin Analizi)

Sadece user story de açıklanan yazılım gereksinimlerini gözden geçirirsiniz.

- Susamış bir kişi olarak susuzluğumu gidermek için su istiyorum.
- Modaya ilgi duyan bir insan olarak içeceğimin güzel görünmesi için şemsiye konulmasını istiyorum.
- Sıcaktan bunaldığım için suyumla birlikte beni ferahlatacak 1 dilim limon istiyorum.
- Çok susadığım için büyük bir bardak su istiyorum.
- Hava sıcak olduğu için suyumun soğuk olmasını istiyorum.
- Cool gorunmek icin suyu içeceğim bir pipet istiyorum.



# User Story (Kullanıcı Hikayesi)

Kullanıcı Hikayesi (User Story), Agile ekiplerde kullanılan en iyi pratiklerdendir. Genellikle son kullanıcı veya bir sistem kullanıcısı bakış açısından yazılmaktadır. Başka bir deyişle, bir kullanıcı hikayesi kullanıcının türünü, ne istediğini ve nedenini açıklar. Bir gereksinimin basitleştirilmiş bir tanımını oluşturmaya yardımcı olur. Projeye bağlı olarak, kullanıcı hikayeleri müşteriler, kullanıcılar, yöneticiler veya geliştirme ekibi üyeleri dahil olmak üzere çeşitli paydaşlar tarafından yazılabilir.

TECHPROED



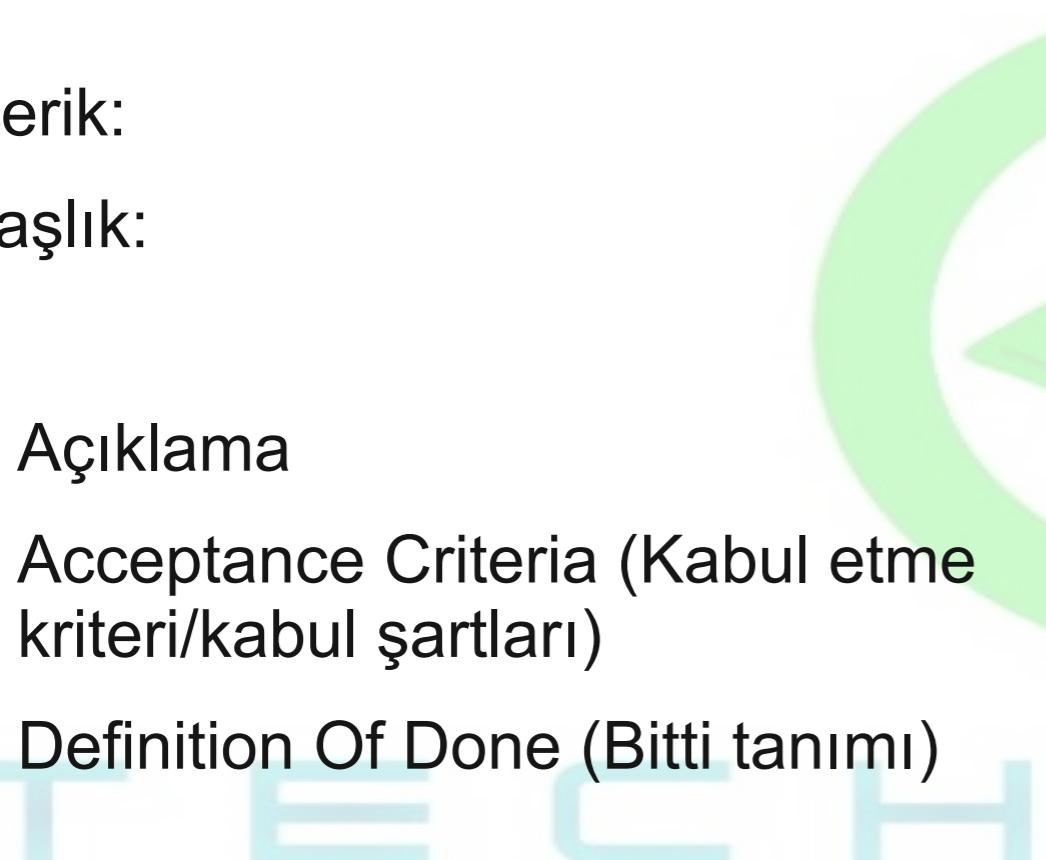
- Bir kullanıcı olarak şifre oluştururken, güvenlik amacıyla, güçlü bir şifre oluşturmak istiyorum.
- Bir yönetici olarak, çalışanlarının bilgilerini, filtreleyerek daha çabuk ulaşmak istiyorum.
- Bir kullanıcı olarak birden fazla kişiye, daha hızlı olacağı için aynı anda mesaj göndermek istiyorum
- “Sıkça sinemaya giden bir izleyici olarak, yakınımdaki sinema salonlarında vizyona yeni girecek filmler hakkında bütün bilgilere telefonumdan ulaşabilmek, böylece her filmin web sitesini ayrı ayrı kontrol etmeye ihtiyaç duymamak istiyorum.”

**User Story:** Bir kullanıcıya değer sağlayacak işlevselligin kısa açıklamasıdır.

**İçerik:**

**Başlık:**

- Açıklama
- Acceptance Criteria (Kabul etme kriteri/kabul şartları)
- Definition Of Done (Bitti tanımı)



Fonksiyon: Login Fonksiyonu

Ön Koşul:

- Kullanıcının login olabilmesi için, daha önceden başarılı bir üye kaydı olmalıdır.

Gereksinimler:

- Kullanıcının eposta adresini girmesi için bir text box olmalıdır.
- Kullanıcının şifresini girmesi için bir text box olmalıdır.
- Kullanıcının bu bilgileri server a göndermesi için bir buton olmalıdır.

Test Senaryoları:

Test Senaryoları	Test Durumları
Eposta alanına fazla karakter girilmesi	Negatif
Şifre alanına fazla karakter girilmesi	Negatif
Eposta alanına eposta formatında bir değer girilmemesi	Negatif
Eposta ve şifre alanlarına doğru bir değer girilmesi	Pozitif
Epostası doğru Şifresi yanlış bir değer girilmesi	Negatif
Alanlara hiç bir değer girilmeden formun gönderilmesi	Negatif
Gönderme butonuna iki kere tıklanması	Negatif
Daha önce üye olmayan bir eposta ile login denemesi	Negatif

**User Story:** İstenilen ürün veya özelliğin kısa ve anlaşılır tanımıdır.

Ürün veya özelliği isteyen kişinin perspektifinden yazılır,

Ürünün eski bir kullanıcısı veya muhtemel kullanıcısı olarak yazılır

Üç bölümden oluşur.

Bir (kullanıcı) olarak,(kullanıcıların memnun olduğu bir telefon araması) yapabilmek için (puanlara göre arama yapmak) istiyorum.

T E C H P R C

As a <user> I can <feature> so that <benefit>.



I ndependent

N egotiable

V aluable

E stimable

S mall (Sized appropriately)

T estable

TECHNIQUE

**Independent** : Bağımsız bir şekilde geliştirilebilmeli ve teslim edilebilmeli.

**Negotiable** : Ekip ve iş birimleri üzerinde tartışabilmeli.

**Valueable** : Kullanıcıya değer üretebilmeli.

**Estimable** : Efor tahmini yapılabilirmeli.

**Small** : Tek sprint içerisinde tamamlanabilecek parçalara ayrılmış olmalı.

**Testable** : Test edilebilir nitelikte olmalıdır.

# Characteristics of User Story

I

Independent

N

Negotiable

V

Valuable  
Verticals

E

Estimable

S

Small

T

Testable

The Story Should be shelf contained, in a Way so that there is no inherent dependencies on another story.



User Stories are not explicit contracts, and should leave space for discussion

User Stories must deliver values to the stakeholders or business

You must able to estimate the user story, once its is ready and groomed.

User Story Should not be so big as to it become impossible to plan, task out, prioritize with certainty

The User Story or its related description must provide the necessary information to make test development possible

# Acceptance Criteria (Kabul Kriteri)



**Spesifik:** Hedef kesin ve net tanımlanmalıdır.

**Ölçülebilir:** Hedef ölçülebilir olmalıdır.

**Başarılabilir:** Hedef Alıcılar tarafından kabul edilir olmalıdır.

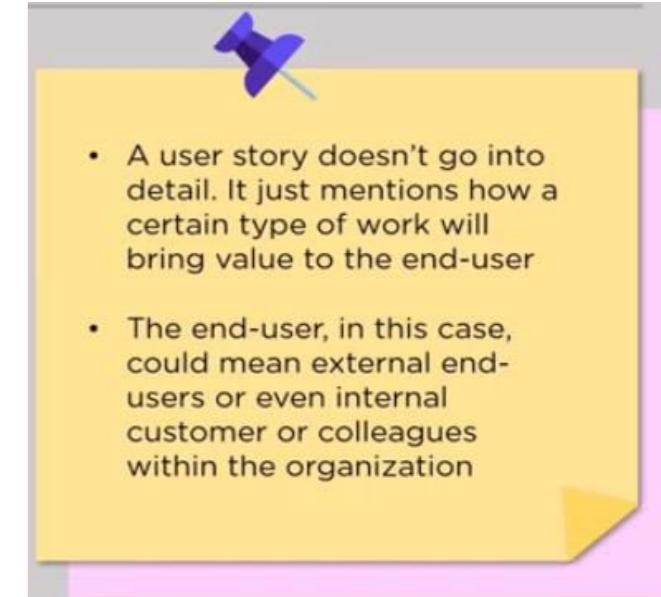
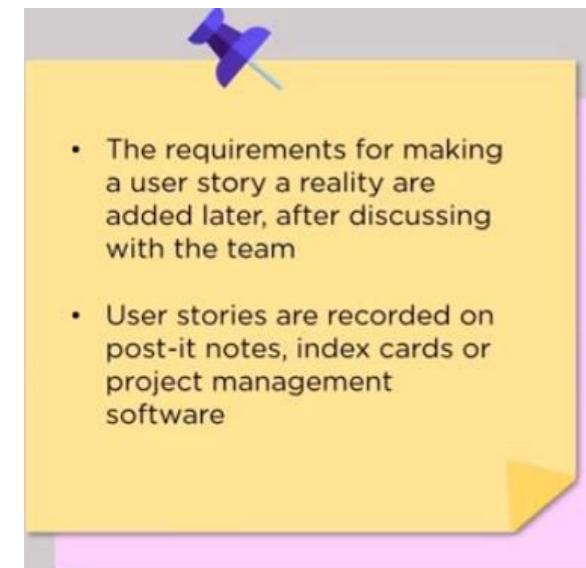
**Makul / Gerçekçi:** Hedef mümkün olmalıdır.

**Zamana bağlı:** Hedef net bir zaman takviminde erişilir olmalıdır.

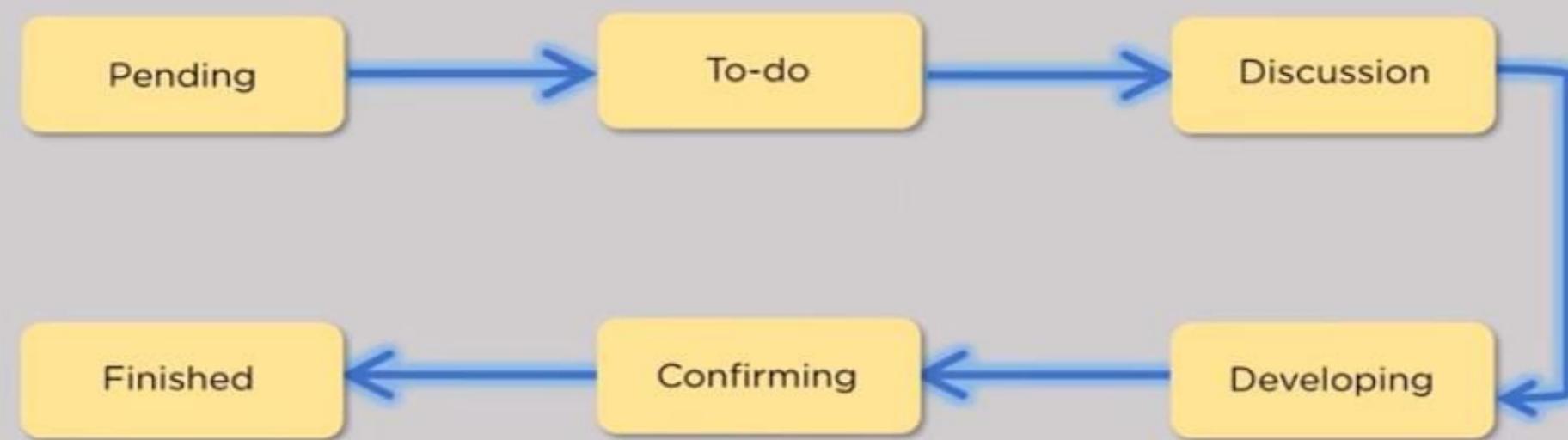
## Characteristics of Acceptance Criteria

S	M	A	R	T
Specific	Measurable	Achievable	Relevant	Time Bound
Target a Specific Area of improvement or increment	Quantify, or at least a possible indicator of progress	Make it realistic, that can be achieved.	Expectation should be related to the requirement, or the description of the story	Specify when the result can be achieved. Should have a specific duration or event.

- User story kullanıcılara (end-user, patron veya şirket içi personel) gereksinimlerini kendi perspektiflerinden, kendi ifadeleri ile ifade edebilme imkanı tanır.
- User Story teknik detaylara veya yazılım gereksinimlerine girmeden işin net tanımını ve Applicationa katacagi net degeri ortaya koyar
- User Story'nin hikayesi kullanıcının yazımı ile baslar, Scrum Team'in Product Backlog'a eklemesive sprinte dahil edip üzerinde çalışması ile gerçek hayata geçmis olur.



## LIFECYCLE OF A USER STORY



T E C H P R O E D

# Definition of Done (Bitti Tanımı)

Development Team, kendilerinden tamamlanması beklenen işleri Product Owner'a sunmadan önce bir kalite-kontrol süreci gibi kontrol listesi hazırlar. Bu liste tüm User Story'ler için ortaktır ve biten tüm işler bu prosedürden geçmesi gereklidir. Buna kısaca "Done" veya "DoD — Definition of Done (Bitti Tanımı)" denir. Bu liste genellikle şu kontrolleri içerir:

- Unit testleri
- QA testleri
- User Story'deki tüm dökümanların güncellenmesi
- Kodun review edilmesi
- User Story kabul kriterlerinin karşılanması
- Product Owner onayı

DoD checklisti, Sprint veya Release sonunda belirli tamamlama kriterlerinin karşılanıp karşılanmadığını kontrol etmek için de kullanılmak üzere hazırlanabilir.

## 2) Test Planning (Test Planlaması)

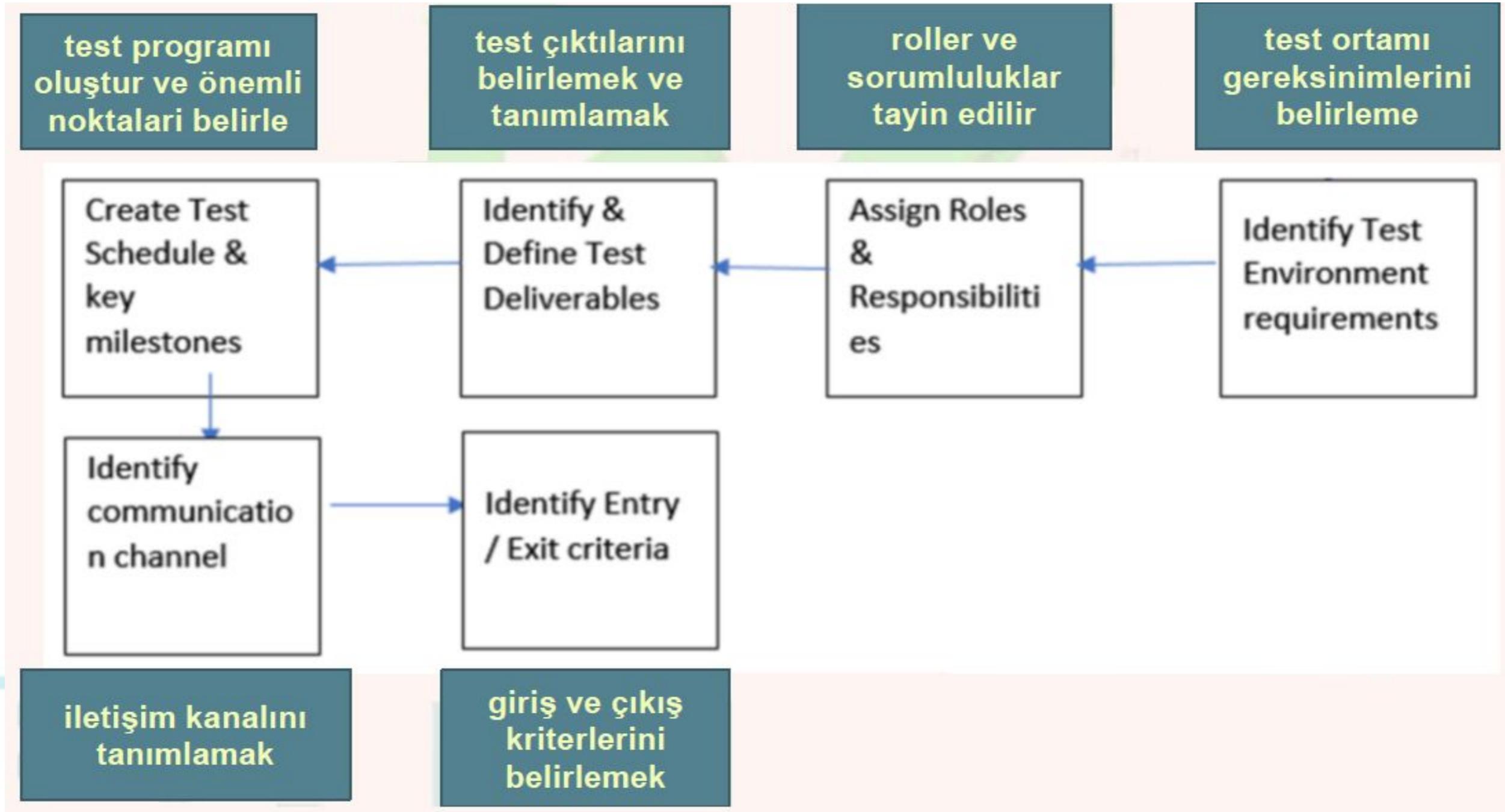
Neyin test edilmesi gerektiğine dair genel bir fikir topladıktan sonra, testler için 'plan yapılır'.

Test Planı belgesi, Ürün Açıklaması, Yazılım Gereksinimi Spesifikasiyonu (Software Requirement Specification SRS) veya Kullanım Senaryosu Belgelerinden (Use Case Documents) türetilmiştir.

Amaçlanan test faaliyetlerinin kapsamını, yaklaşımını, kaynaklarını ve programını açıklayan bir belgedir.

Test Planı belgesi genellikle Test lead veya Test Manager tarafından hazırlanır ve belgenin odak noktası neyin test edileceğini, nasıl test edileceğini, ne zaman test edileceğini ve hangi testi kimin yapacağını açıklamaktır.





### 3) TEST CASE OLUŞTURMA

Test case'ler gereksinimlere göre hazırlanan input'lar, olaylar ya da aksiyonlar ve bunlar sonucu oluşması beklenen sonuçların belirtildiği dökümanlardır. Test case'ler yazılımın temellerini oluşturan gereksinimler ve dizayndaki problemlerin, eksikliklerin de ortaya çıkarılmasını sağlar.

- En basit biçimde, bir test case, bir test yazılımının gereksinimleri karşılayıp karşılamadığını ve işlevlerini doğru bir şekilde yerine getirip getirmedğini belirlediği bir dizi koşul veya değişkendir.
- Test case, bir tester'ın gerçekleştirdiği tek bir, yürütülebilir testtir. Tek tek aşamalar (step) takip edilerek yapılır.
- Bir test case, bir şeyin davranışması gereği gibi davranışlığını doğrulamak için bir dizi adım talimat olarak düşünebilirsiniz.

# Bir test senaryosu genellikle şunları içerir

- **Test Case No:** Test Numarası (Kesin Olmalıdır)
- **Priority (Öncelik)**
- **Test Name:** Test Senaryosu Adı (Kesin Olmalıdır)
- **Test Step:** (Test Adımı, Adımları (Kesin Olmalıdır))
- **Result:** Beklenen Sonuç (Kesin Olmalıdır)
- **Status:** Durumu, Done/Undone - Pass/Fail (Kesin Olmalıdır)
- **Test Datası**
- **BUG (Hata) Sayısı** yazılır
- **Notes (Detay ve Notlar)**
- **Sprint No** (Agile & Scrum Çalışmasında Kullanılır)

**Test Senaryosu: Örnek**

- Test No : MO-LG-01
- Senaryo : Geçerli kullanıcı ad ve şifre ile giriş yap
- Özellik No : 1.1.1
- Önem : Yüksek (Yüksek, Orta, Düşük)
- Kategori : Müşteri kabul (*Teste kabul, Müşteri kabul, Baglanım, Tümllestirme, ...*)
- Tahmini Süre : 5 Dakika (*Tahmini testi kosturma süresi*)
- Bağımlılık : Yok (*Bu testten evvel kosturulması gereken testleri sırala*)
- Kurulum : Netscape Web Browser'ını çalıştır (*Prosedürden evvel yapılması gerekenler*)
- Prosedür : 1. <http://enstitu.hacettepe.edu.tr/akademik/> web sitesine git
- 2. Geçerli kullanıcı ad ve şifreyi gir
- 3. "Tamam" düğmesine bas
- 4. "HACETTEPE ÜNVERSTES - Lisansüstü Öğrenci Sistemi" ana sayfasının geldigini doğrula
- Temizlik : Çıkış linkine bas (*Ortamı bulduğum gibi bırakmak için yapılacaklar*)

Test Case ID	Test Objective	Pre-Condition	Steps	Test Data	Expected Result	Actual Result	Status
TC_001	I should not login to Facebook	Login should not be available at www.facebook.com	<p>1_Go to https://www.facebook.com/</p> <p>2_Click on username checkbox</p> <p>3_enter username</p> <p>4_Click on password checkbox</p> <p>5_enter the password</p> <p>6_click on login button</p>	username ="Never_Giveup" password ="ICanDolt"	user should not be able to login		



A	B		C	D	E	F
1	TEST NO	TEST NAME	TEST STEP	RESULT	STATUS	BUG
2	1	TCKN Label Kontrolü	TCKN alanına maksimum 11 nümerik ve zorunlu karakter girilmelidir.	Eksik veya hatalı karakter girince Tooltip'de uyarı vermelii, Hata vermediye Grid'e kayıt etmelidir.	DONE	
3	2	Çalışan(EMPLOYEES) Bilgileri Kontrolü	GSM(Telefon) Combosuna 0 hariç, 10 nümerik ve zorunlu karakter girilmelidir.	Eksik veya hatalı karakter girince Tooltip'de uyarı vermelii, Hata vermediye Grid'e kayıt etmelidir.	DONE	
4	3	Çalışan(EMPLOYEES) Bilgileri Kontrolü	Maaş(SALARY) bilgisinin servisten gelmesi ve Maaş Label'ın da görüntülenmesi.	Maaş bilgisi Disable olarak gözükmeli ve Boş olmamalıdır.	UNDONE	2
5						
6						

◀ ▶
FRONT-END
DATABASE
SERVICE
+
: 1

A	B		C	D	E
1	DATABASE ÖRNEK SENARYO				
2	TEST NO	TEST NAME	TEST STEP	RESULT	STATUS
3	1	EMPLOYEES Tablosu Kontrolü	Maaş Bilgisi EMPLOYEES.SALARY kolonunda tutulup JOBS.MIN_SALARY AND JOBS.MMAX_SALARY arasında değer almalıdır.	HR.EMPLOYEES.SALARY Nümerik olup Data Type NUMBER (8,2) ve NULL olmamalıdır.	DONE
4					
5					
6	SERVICE ÖRNEK SENARYO				
7	TEST NO	TEST NAME	TEST STEP	RESULT	STATUS
8	1	GetEMPLOYEES Servis Kontrolü	Çalışanların Adres Bilgisi GetEMPLOYEES servisinden beslenmektedir.	<STREETADDRESS></STREETADDRESS> Output Tagı içinde adres bilgileri servisten alınıp Adress Label'da Disable göstermelidir.	DONE

# TEST CASE OLUŞTURMA

1. *"http://webdriveruniversity.com/" adresine gidin*
2. *"Login Portal" a kadar asagi inin*
3. *"Login Portal" a tiklayin*
4. *Diger window'a gecin*
5. *"username" ve "password" kutularina rastgele deger yazdirin*
6. *"login" butonuna basin*
7. *Popup'ta cikan yazinin "validation failed" oldugunu test edin*
8. *Ok diyerek Popup'i kapatin*
9. *Ilk sayfaya geri donun*
10. *Ilk sayfaya donuldugunu test edin*

# TEST CASE'LERİ KİM YAZAR?

- Genellikle QA ekibinden tecrübeli biri test case'leri yazar.
- Test case hazırlamak için her ekip kendi standart şablonunu kullanır.
- Genelde manual testerlar yapar ve aynı dokumani kullanarak Automation testerler scriptlerini (kodlarını) yazarlar
- Developerlar normalde bir test data oluştururlar veya hazır olan bir datayı BA den veya PO alırlar sonra onu günceller ve geliştirirler.
- Biz de kendimiz developerlardan alıp aynı test datayı geliştirir ve test ederiz. Veya Test lead Acceptence Criteria'lara göre kendisi bir test data hazırlayabilir.
- SDET : Software Development Engineer In Test : Bir Framework'u sıfırdan oluşturabilecek kişidir.

Automation Tester : Hazır yazılmış test case'leri kullanarak test yapar, oluşturulmuş framework'leri kullanır.

# Örnek Test Case

A	B	C
User Story ID	Description	Acceptance Criteria
		Gecersiz kullanıcı adı ile erişim sağlanamaz
US 0002	Facebook login (Giriş sayfası) gecersiz kimlik bilgileriyle erişilmemelidir	Geçersiz şifre ile erişim sağlanamaz
		Geçersiz kullanıcı adı ve şifre ile erişim sağlanamaz

T E C H P R O E D

# Örnek Test Case

A	B	C	D	E	F	G	H	I
User Story ID	Test Case ID	Test Objective	Pre-Condition	Steps	Test Data	Expected Result	Actual Result	Status
US 0002	TC_001	Gecersiz kullanıcı adı ile erişim sağlanamaz	Login erisilebilir olmalıdır //www.facebook.com/ da	1_https://www.facebook.com/ gidi  2_kullanici adi textbox a tiklayiniz  3_yanlis bir kullanici adini giriniz  4_sifre textbox ina tiklayiniz  5_dogrular bir kullanici sifresi giriniz  6_login butonuna tiklayiniz  7  8	URL 3:  https://www.facebook.com kullanici adi ="pes_etmek_yok" sifre ="yapabiliirim"	kullanici erisim elde edememelidir	kullanici erisim elde edemedi	Pass
US 0002	TC_002	Geçersiz şifre ile erişim sağlanamaz	Login erisilebilir olmalıdır //www.facebook.com/ da	1_https://www.facebook.com/ gidi  2_kullanici adi textbox a tiklayiniz  3_dogrular kullanici adini giriniz  4_sifre textbox ina tiklayiniz  5_yanlis bir kullanici sifresi giriniz  6_login butonuna tiklayiniz  7	URL 3:  https://www.facebook.com kullanici adi = techproedusa@gmail.com sifresi ="yanlis_sifre"	kullanici sifre hatali elde etmemelidir ve giris izni verilmemelidir	Email adresi yanlis veerisim elde edilemedi	Fail
US 0003	TC_003	Geçersiz kullanıcı adı ve şifre ile erişim sağlanamaz	Login erisilebilir olmalıdır //www.facebook.com/ da	1_https://www.facebook.com/ gidi  2_kullanici adi textbox a tiklayiniz  3_yanlis kullanici adi giriniz  4_sifre textbox ina tiklayiniz  5_yanlis sifre giriniz  6_login butonuna tiklayiniz	URL 3:  https://www.facebook.com username ="yanlis_username" password ="yanlis_sifre"	kullanici erisim elde edememelidir	kullanici erisim elde edemedi	Pass

## 4) Test Environment Setup (Test Ortamı Kurulumu / Oluşturulması)

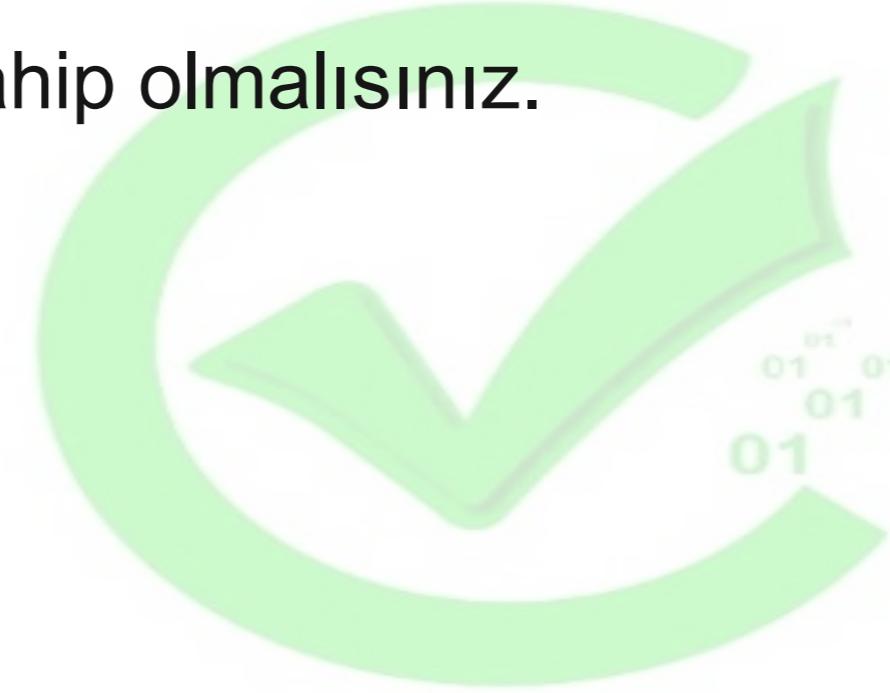
Testinizi nerede ve hangi ortamda gerçekleştireceğinizi bilmelisiniz ve test verilerinize de sahip olmalısınız.

Dev (Development)

Test (Test ortamı)

Stage (sahne)

Prod (Ürün)



T E C H P R O E D

## 5) Test Execution (Testin Uygulanması)

Her şey hazır olduğunda, planlanan tüm functionality (ler) test edebilir ve çalıştırılır.

Manuel Testerlar testlerini manuel olarak yapar.  
Automation (Cross-functional) Tester scriptlerini yazarak test ederler.

T E C H P R O E D

**Test case =>** adım adım neyi test edeceğiniizi açıklayan senaryolarınızdır

**Test plan =>** Test kapsamı, yaklaşım kaynakları ve amaçlanan test faaliyetlerinin zamanlamasını tanımlayan giriş ve çıkış kriterleri.

**Test execution =>** Scriptler hazır olduğunda, çalıştırır veya manuel olarak testinizi gerçekleştirirsiniz.

**Test data =>** functionality (leri) test etmek için kullanmanız gereken veriler

**Acceptance criteria =>** functionalityi test etmenin ayrıntılı tanımı

**Expected result =>** yapılan testin beklenen sonucu

**Actual result =>** testi yaptıktan sonra ortaya çıkan netice.

**Environments (test yapılan ortamlar (URL)=>** Dev,Test,Stage,Prod

T E C H P R O E D

# **SDLC**

**Software Development Life Cycle**  
**(Yazılım Geliştirme Yaşam Döngüsü)**

**Test Çeşitleri**

**6. Ders**

**11/12/2021**

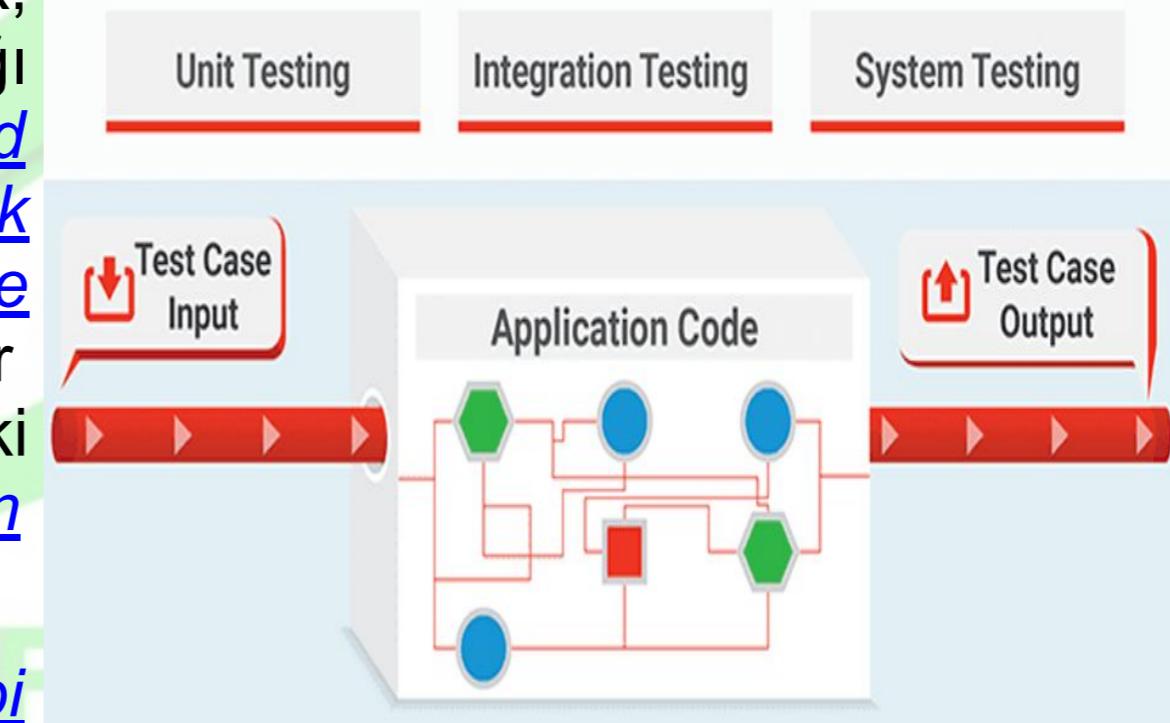
**T E C H P R O E D**

# Test Çeşitleri

1) **White Box Test (Beyaz Kutu Test):** Test edilen ögenin iç yapısının/tasarımının/uygulanmasının testçi tarafından bilindiği bir yazılım test yöntemidir. Tıpkı ismi gibi beyaz bir kutu içerisinde bakılarak yazılımın kodunun iç yapısının bilinerek, ölçümlenerek test senaryolarının tasarlandığı tekniktir. Bu yöntemdeki asıl amac kod parçacıklarının tek tek test edilerek aslında en küçük parçacık halinde bile sağlıklı bir şekilde çalıştırılabilirliğinin görülmESİdir. Gereksinimler sonucu belirlenen girdilerin kodun bir parçasındaki çıktıları karşılayıp karşılamadığı test edilir. Yazılımın işlevselligi test edilmez.

Yazılım test uzmanları tarafından da kullanıldığı gibi çoğunlukla yazılım geliştiriciler tarafından kullanılır. Bu yöntem birim, entegrasyon ve sistem testi seviyelerinde kullanılır.

## White Box Testing



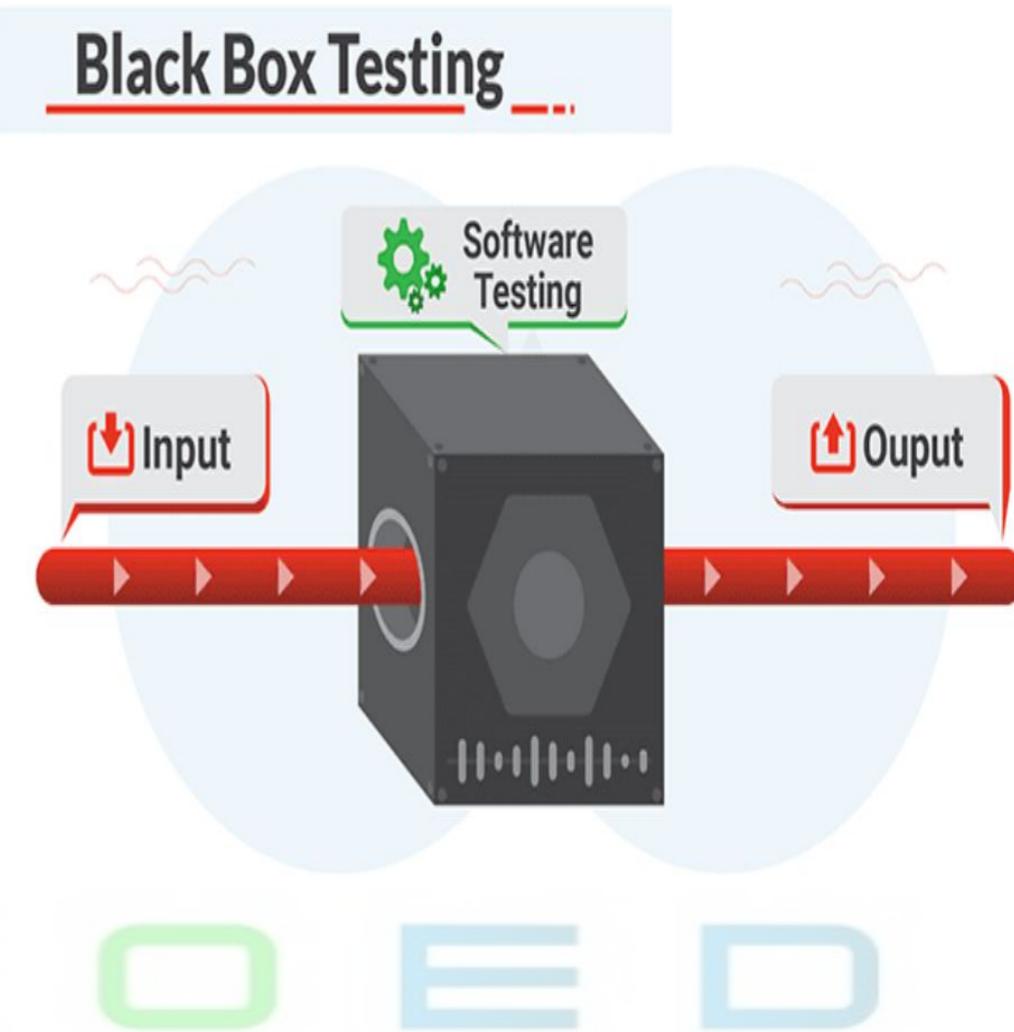
## Beyaz Kutu Testlerinin Avantajları

- Testler, yazılım geliştirilirken tasarlanıp yapıldığı için erken sürelerde müdahale edilebilmektedir.
- Geliştiriciler kendi kod parçacıklarını defalarca gözden geçirdikleri için kodun güvenilirliği yüksektir.
- Kod optimizasyonu fazladır.

## Beyaz Kutu Testlerinin Dezavantajları

- Kod okumayı bilen nitelikli bir yazılım test uzmanı bulmak zor ve maliyetlidir.
- Geliştirici kendi testini kendi yaptığı için gözden kaçabilen noktalar olabilir.
- Testler çok kapsamlı ve ayrıntılı olduğu için uzun zaman alabilmektedir.

**2) Black Box Test (Siyah Kutu Testi):** Test edilen ögenin iç yapısının/tasarımının/uygulanmasının test eden kişi tarafından bilinmediği bir yazılım test yöntemidir. Sistemin, yazılımın iç yapısı hakkında bilgi sahibi olunmadan yani koda bakılmadan sistemin işlevsellliğini ölçmeye yarayan test tekniğidir. İlgili sistemi bir kara kutu gibi görüp aslında içerisindeki mimari yapı veya kaynak kod hakkında bilgi sahibi olmadan uygulanabilir. Kara kutu testlerindeki amaç, gereksinimleri karşılayan çıktıların alınıp alınmadığını ölçümlemektir. Sistemden almayı beklediğimiz çıktılar kadar beklemediğimiz çıktılar da test edilmelidir.



Bu test tasarım tekniği, test uzmanlarının sıkılıkla kullandığı bir yöntemdir. Kara kutu testi; birim, entegrasyon, sistem

## Kara Kutu Testlerinin Avantajları:

- Test uzmanları tarafından yapıldığı için kod bilgisine ihtiyaç duyulmaz.
- Kodu geliştiren kişi ve test eden kişi farklı olduğu için farklı bakış açılarıyla test edilebilir ve görünmeyen hatalar bile kolaylıkla bulunabilir.
- Testler hızlı ve etkin bir biçimde uygulanabilir.
- Gereksinimlerin belirlenmesinin hemen ardından test senaryoları oluşturulabilir.
- Büyük ölçekli sistemlerde bu yöntem kullanıldığında oldukça yüksek verim alınır.

## Kara Kutu Testlerinin Dezavantajları

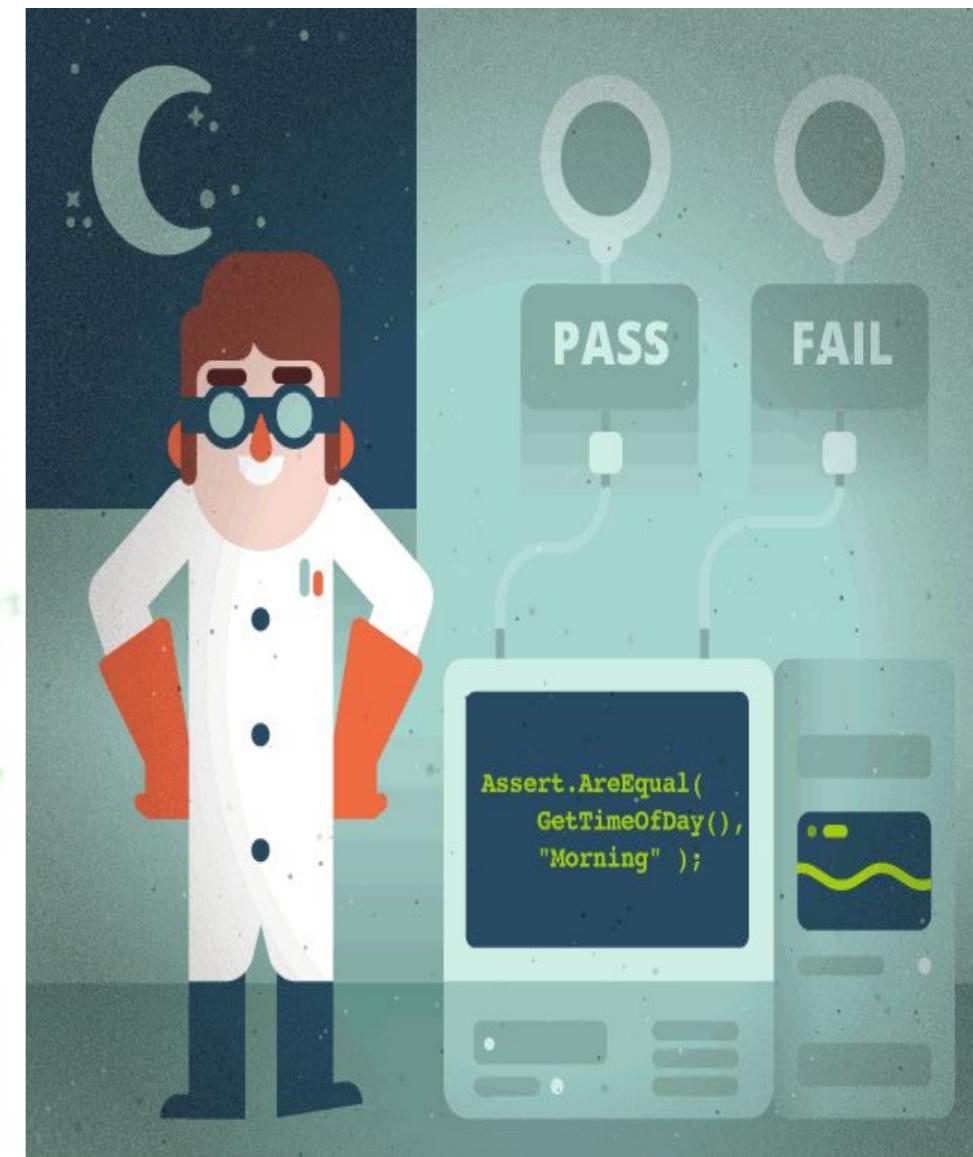
- Sistemin iç yapısı bilinmediği için kaynak kod içerisinde kümelenmiş hataların bulunması zorlaşır.
- Halihazırda geliştiricinin kendi yaptığı testler ile tekrara düşülebilir.
- Karmaşık kod blokları içeren yazılımlarda kullanılamaz.
- Bütün olasılıkları kontrol etmek mümkün değildir bu sebeple test edilmemiş fonksiyonlar olabilir.

<b>Black Box (Kara Kutu) Test</b>	<b>White Box (Beyaz Kutu) Test</b>
İç yapının veya programın veya kodun gizlendiği ve hakkında hiçbir şey bilinmediği bir yazılım testi yöntemidir.	Test eden kişinin yazılımın iç yapısı veya kodu veya programı hakkında bilgi sahibi olduğu yazılımı test etmenin bir yoludur.
Çoğunlukla yazılım test uzmanları tarafından yapılır.	Çoğunlukla yazılım geliştiriciler tarafından yapılır.
Uygulama bilgisi gereklidir.	Uygulama bilgisi gereklidir.
Dış veya dış yazılım testi olarak adlandırılabilir.	Dahili veya dahili yazılım testidir.
Yazılımın işlevsel testidir.	Yazılımın yapısal testidir.
Bu test, gereksinim özelliklerini belgesi temelinde başlatılabilir.	Bu tür yazılım testleri, detay tasarım belgesinden sonra başlatılır.
Programlama bilgisi gereklidir.	Programlama bilgisine sahip olmak zorunludur.

<b>Black Box (Kara Kutu) Test</b>	<b>White Box (Beyaz Kutu) Test</b>
Yazılımın davranış testidir.	Yazılımın mantık testidir.
Yazılımın daha yüksek test seviyeleri için geçerlidir.	Genellikle yazılım testinin daha düşük seviyelerine uygulanabilir.
Kapalı test olarak da adlandırılır.	Şeffaf kutu testi olarak da adlandırılır.
En az zaman alıcıdır.	Çoğu zaman alıcıdır.
Algoritma testi için uygun veya tercih edilmez.	Algoritma testi için uygundur.
Deneme yanılma yol ve yöntemlerle yapılabilir.	İç veya iç sınırlarla birlikte veri alanları daha iyi test edilebilir.
<b>Örnek:</b> anahtar kelimeleri kullanarak google'da bir şey arayın	<b>Örnek:</b> döngüleri kontrol etm

# Unit Test

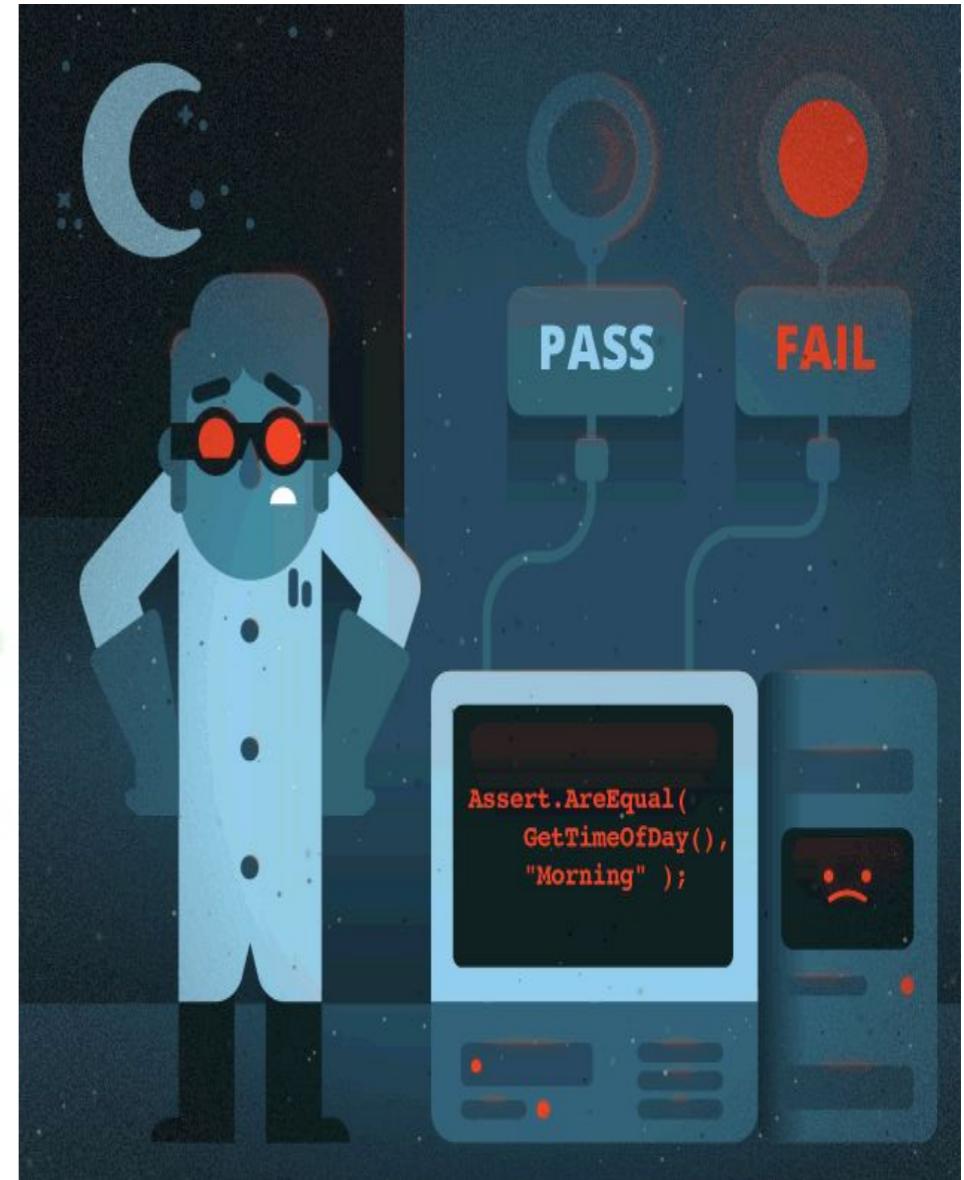
- Unit Test, bir yazılımın en küçük test edilebilir bölümlerinin, tek tek ve bağımsız olarak doğru çalışması için incelendiği bir yazılım geliştirme sürecidir. Unit Test yazılım testinin ilk seviyesidir ve entegrasyon testinden önce gelir. Unit Testleri geliştiriciler kendileri yazar ve yürütürler. Unit Testleri bir kod bölümünü izole eder ve doğruluğunu onaylar.
- Bazı Unit Test Framework'leri:  
JUnit, Spock, Nunit, TestNG, Jasmin, Mocha



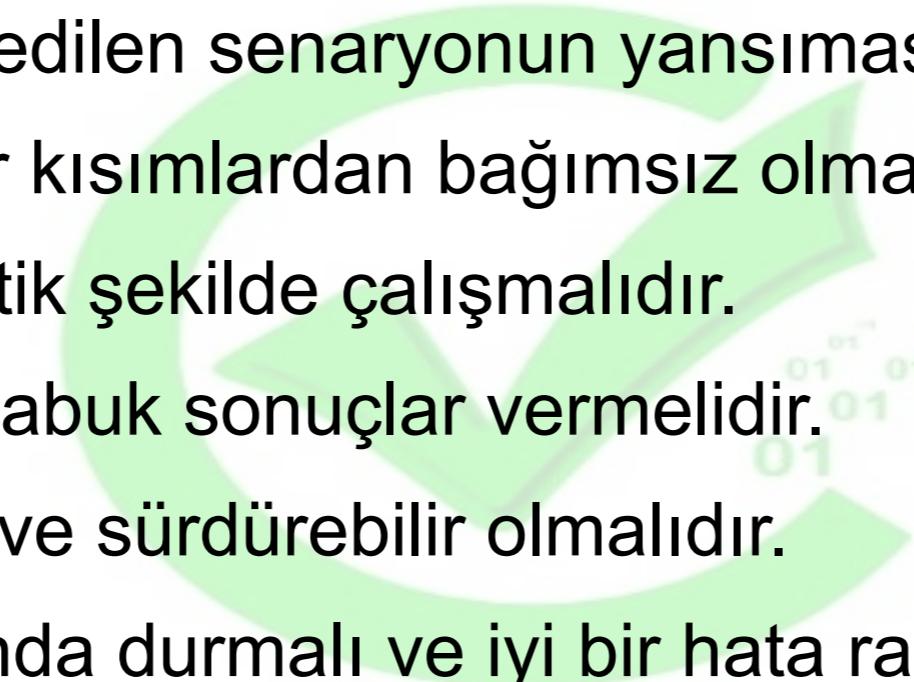
T E C H P R

- Unit Test yapmaktaki amacımız yazılımın her biriminin tasarlandığı şekilde gerçekleştiğini doğrulamaktır.
- Unit Test yazmak kodda yeniden düzenleme (Refactor) işlemini yapmayı kolaylaştırır. Kodda değişiklik yaptığımızda, Unit Testi çalıştırıp oluşturduğumuz algoritmaya uygun bir şekilde çalışıp çalışmadığını kolaylıkla test edebiliriz.
- Unit Test'ler tüm hataları ortaya çıkarmaz, çünkü her parça izole şekilde test edilmekte ve entegrasyon yapıldığında herşeyin düzenli çalışacağı anlamına gelmez.

TECHPR



# Unit Test Nasıl Yazılır?

- En küçük parçası test edilmeli
- Sadece bir senaryo test edilir, kullanılan adımlar belirlenir.
- Test method ismi test edilen senaryonun yansımıası olmalıdır.
- Test edilen kısım diğer kısımlardan bağımsız olmalıdır.
- Testlerimiz tam otomatik şekilde çalışmalıdır.
- Hızlı çalışılabilir ve çabuk sonuçlar vermelidir.
- Okunaklı, anlaşılabilir ve sürdürilebilir olmalıdır.
- Test başarısız olduğunda durmalı ve iyi bir hata raporu döndürmelidir. Bu hata raporunda neyi test etti? ne yapmalı? beklenen çıktı neydi ve gerçekte ne yaptı? 
- Birim testi sırasında tespit edilen hatalar, SDLC'de bir sonraki aşamaya geçmeden önce düzeltilmelidir.

```
using System;
using NUnit.Framework;
using TDD_with_VS2012;

namespace NUnitTests
{
    [TestFixture]
    public class UnitTests
    {
        [Test]
        public void Calculator_AdditionTest()
        {
            var calculator = new Calculator();
            Assert.AreEqual(calculator.Addition(2, 3), 5);
        }

        [Test]
        public void Calculator_MultiplicationTest()
        {
            var calculator = new Calculator();
            Assert.AreEqual(calculator.Multiplication(2, 3), 6);
        }
    }
}
```

**Örnek:** Bir developer, bir hesap makinesi uygulaması oluşturur. Birim testi, kullanıcının iki sayı girip doğru bir toplam alıp alamayacağını kontrol eder. Ayrı birim testleri, çıkarma, çarpma ve bölme gibi diğer hesap makinesi işlevlerini doğrular.

**Arrange:** Test edilecek koda verilecek olan input parametrelerinin belirlendiği ve test edilecek olan kodun bağımlı olduğu diğer bileşenlerin test anındaki bulunacakları durumlarının tanımlandığı kısımdır.

**Act:** Test edilecek olan kodun çalıştırıldığı aşamadır. Bu aşamada test edilecek olan fonksiyonu/metodu tetikleriz.

**Assert:** Test sonuçlarının doğrulanması aşamasıdır. Tetiklenen fonksiyon doğru sonucu üretiyor mu veya bağımlı olduğu bileşenler üzerinde beklenen aksiyonları tetikliyor mu kontrolünü bu aşamada yaparız.

```
@Test  
void findByExactMatchWithTwoPersons_returnsBothOfThemInInsertionOrder() {  
    // Arrange  
    final var phoneBook = new PhoneBook();  
    phoneBook.addPerson(new Person("James", "Malkovic", "+905554443321"));  
    phoneBook.addPerson(new Person("John", "Doe", "+905554443322"));  
    phoneBook.addPerson(new Person("Foo", "Bar", "+905554443323"));  
    phoneBook.addPerson(new Person("John", "Baz", "+905554443324"));  
  
    // Act  
    final var result = phoneBook.findByName("John");  
  
    // Assert  
    assertEquals(2, result.size());  
    assertEquals(new Person("John", "Doe", "+905554443322"), result.get(0));  
    assertEquals(new Person("John", "Baz", "+905554443324"), result.get(1));  
}
```

PRO E D

## Birim Testi Avantajı Nedir?

- Bir birim tarafından hangi işlevselligin sağlandığını ve nasıl kullanılacağını öğrenmek isteyen geliştiriciler, birim API'si hakkında temel bir anlayış kazanmak için birim testlerine bakabilirler.
- Birim testi, programcının kodu daha sonraki bir tarihte yeniden düzenlemesine ve modülün hala doğru şekilde çalıştığından emin olmasına izin verir (yani, Regresyon testi). Prosedür, tüm işlevler ve yöntemler için test senaryoları yazmaktadır, böylece bir değişiklik bir hataya neden olduğunda, hızlı bir şekilde tanımlanıp düzeltilebilir.
- Birim testinin modüler yapısı nedeniyle, projenin bazı kısımlarını diğerlerinin tamamlanmasını beklemeden test edebiliriz.

## Birim Testi Dezavantajları Nedir?

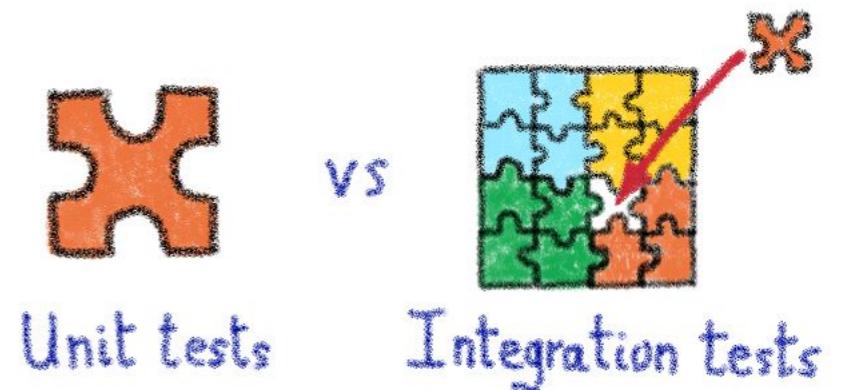
- Birim testinin bir programdaki her hatayı yakalaması beklenemez. En önemsiz programlarda bile tüm yürütme yollarını değerlendirmek mümkün değildir
- Doğası gereği birim testi, bir kod birimine odaklanır. Bu nedenle entegrasyon hatalarını veya geniş sistem düzeyinde hataları yakalayamaz.

# Integration Test (Entegrasyon Testi)

- Entegrasyon testi genellikle birim testiyle birlikte yapılır.
- Integration Test bize bir şeyin çalışıp çalışmadığını söylerken, Unit Test neden çalışmadığını söyler.
- Unit Test yazılımcı perspektifinden bakarken, Integration Test kullanıcı perspektifinden yazılır.
- Integration Test ile, sistemin kullanıcıların beklediği gibi çalıştığından emin oluruz.



- Entegrasyon testinde amaç oluşturulan yazılım modüllerinin, bir araya getirerek doğruluğunu sağlamaktır.
- Yazılım ürünü için oluşturulan tüm modüller bir araya getirilir ve bu şekilde test edilir. Metotlar birim başına testten geçerken, modüller halinde bir araya geldiğinde bazı hatalara sebep oluyor olabilirler.
- Entegrasyon testleri ile ise bu tarz yazılım ürünü problemlerinin henüz canlı ortama çıkmadan veya geliştirdiğimiz yeni bir modülünde sorunsuz çalışabileceğinden hızlı bir şekilde emin olabilmemizi sağlamaktadır.



```
using (var context = new EFContext())
{
    // Arrange
    // Yeni bir entity üretelim
    var testValue = new Value() { Content= "Blabla" };

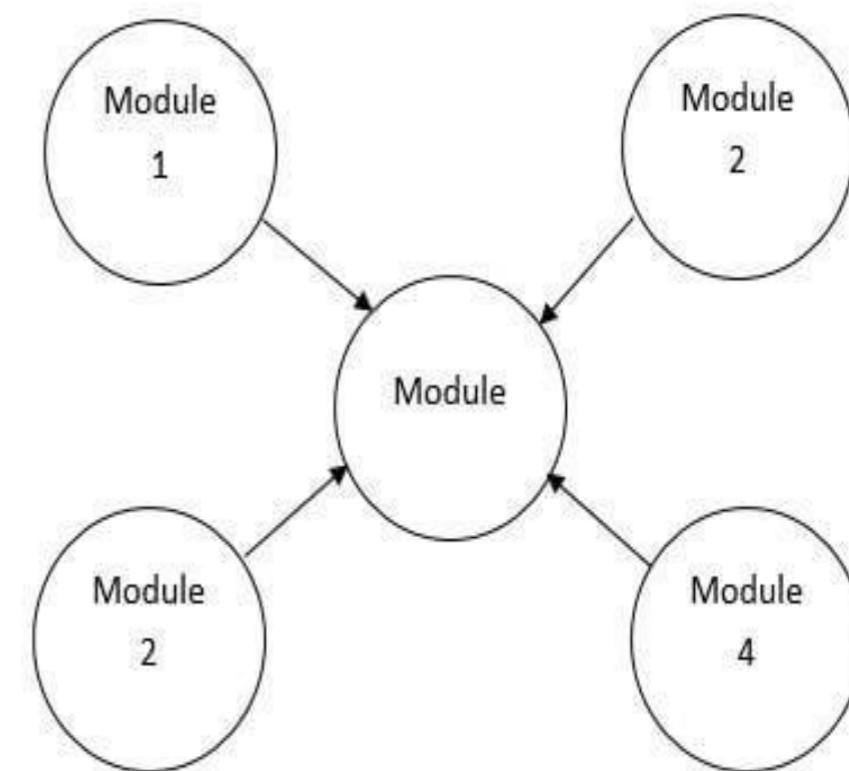
    // Act
    // DB'ye insert yapalım
    context.Values.Add(testValue);
    context.SaveChanges();
    // Insert yapmış olduğumuz entity'i geri okuyalım.
    var testValueFromDb = context.Values.Single(x => x.ID == testValue.ID);

    // Assert
    // Şimdi ise gelen entity'deki content'in eşitliğini kontrol edelim
    Assert.AreEqual("Blabla", testValueFromDb.Content);
}
```

Bu örnek test case’inde, “testValue” nun veritabanına doğru bir şekilde kaydedilip kaydedilemediğini doğruluyoruz.

# Big Bang Integration Test

- Big Bang testinde, birleştirilecek tüm modüllerin testleri bir arada, bütün olarak yapılmaktadır.
- Böylelikle hızlıca test yapılacağından dolayı büyük zaman kazancı sağlanır.
- Ancak tümden yapılan testlerde herhangi bir hata çıkması durumunda hatanın tespit edilmesindeki zorluk olumsuzluk yaratmaktadır.
- Hatanın nereden kaynaklandığının araştırılması uzun sürmektedir.

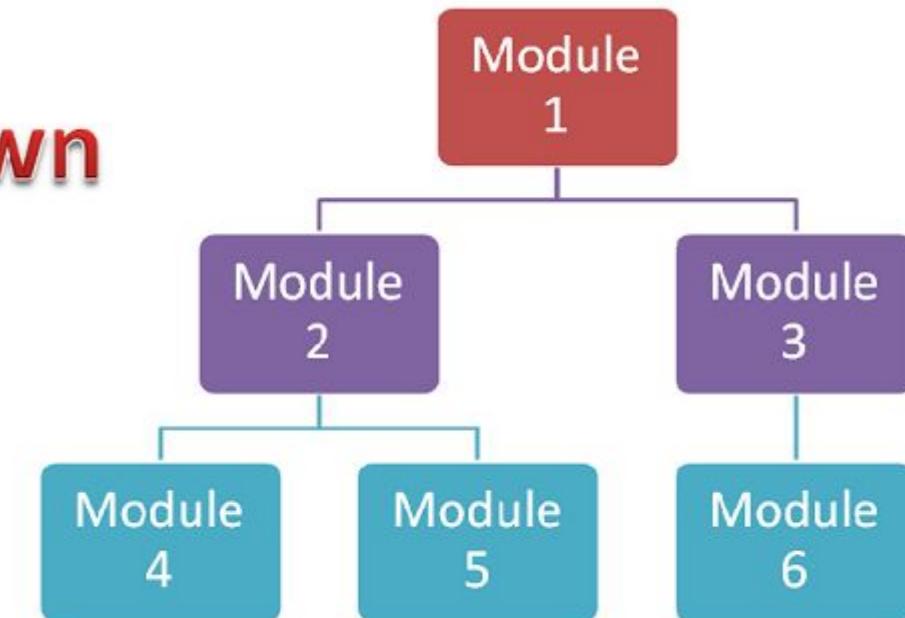


# Top Down Integration Test

Top Down testinde, birleştirilen modüllerin bir bütün olarak yukarıdan aşağıya doğru ele alınırken aynı zamanda modüllerin ayıklanarak test işlemlerinin uygulandığı yöntemdir.

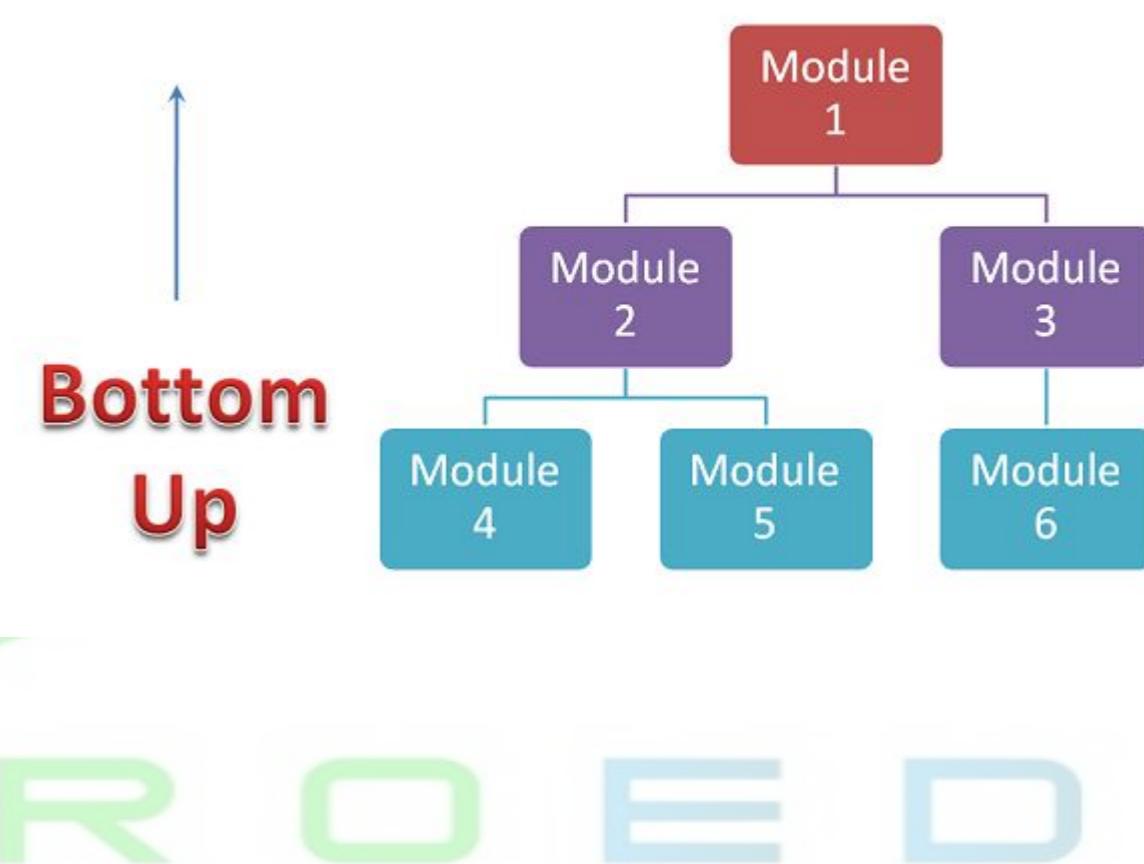
- Bu yöntemde her bir aşama kök olarak adlandırılır ve testin son aşamasında her bir kök kendi içerisinde test edilmiş olur.
- Buradaki amaç her bir modül kendi içerisinde test edilirken kesinlikle hatasız olmalıdır ki, iki hatasız modül birleştirildiğinde herhangi bir hataya olanak sağlasın.
- Bu test ile modüllerde kırılım yaratılarak hatanın kolay saptanması sağlanır.

**Top Down**



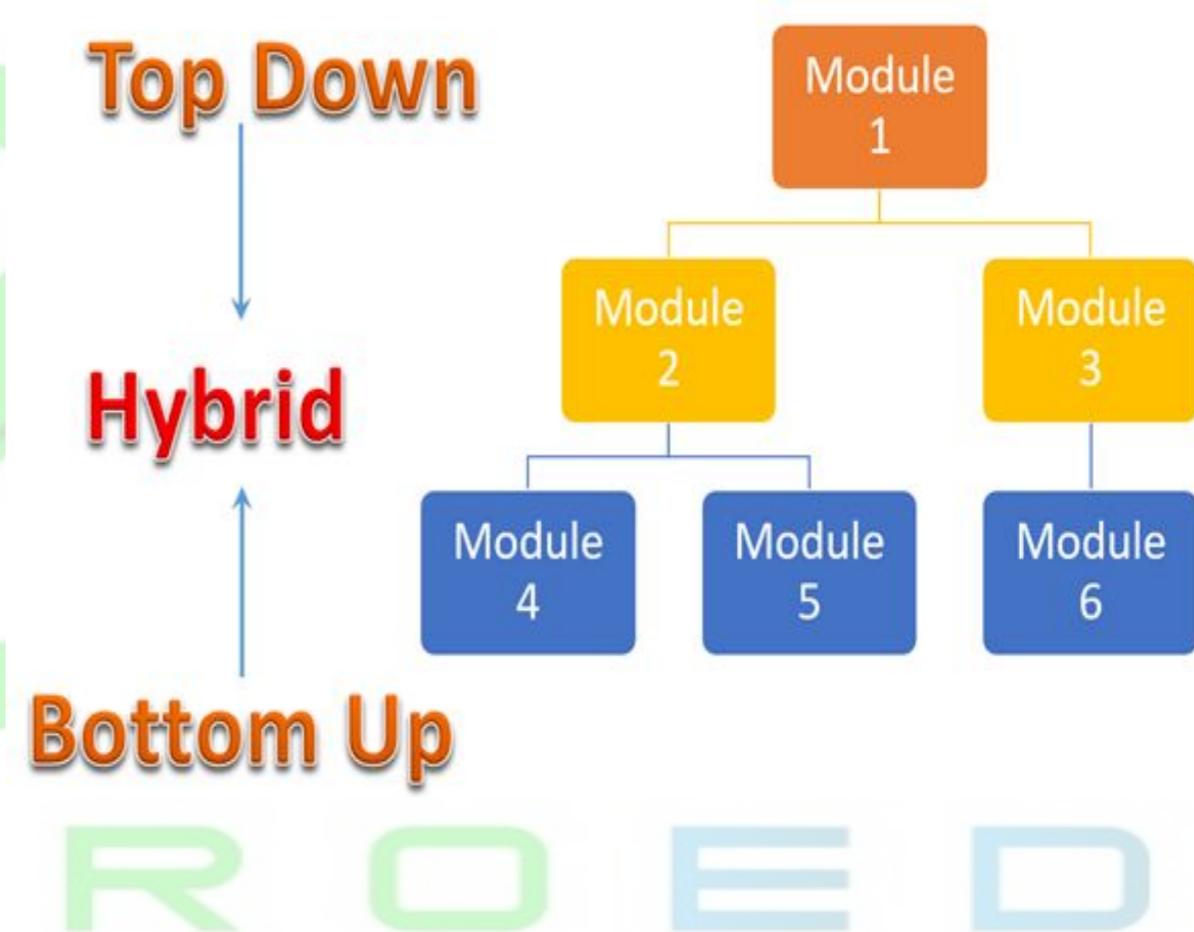
# Bottom Up Integration Test

- Bottom Up testinde, birim test adı altında tüm modüller ayrı ayrı test edilmektedir.
- Her bir modül önce kendi içerisinde test edilir ve daha sonra bir üst seviyesi ile birlikte ele alınarak test işlemlerine devam edilir.
- Bitiş noktasına kadar tüm modüllerin bu şekilde test edilmesi ile işlem tamamlanır.
- Burada da aşağıdan yukarı doğru test edilme mantığı bulunmaktadır ve aynı şekilde kendi içerisinde test edilen modülün test sonucu hatasız olması durumunda bir üst seviyeye ilerlemektedir.
- Bu test ile hatalar kolayca bulunabilir.



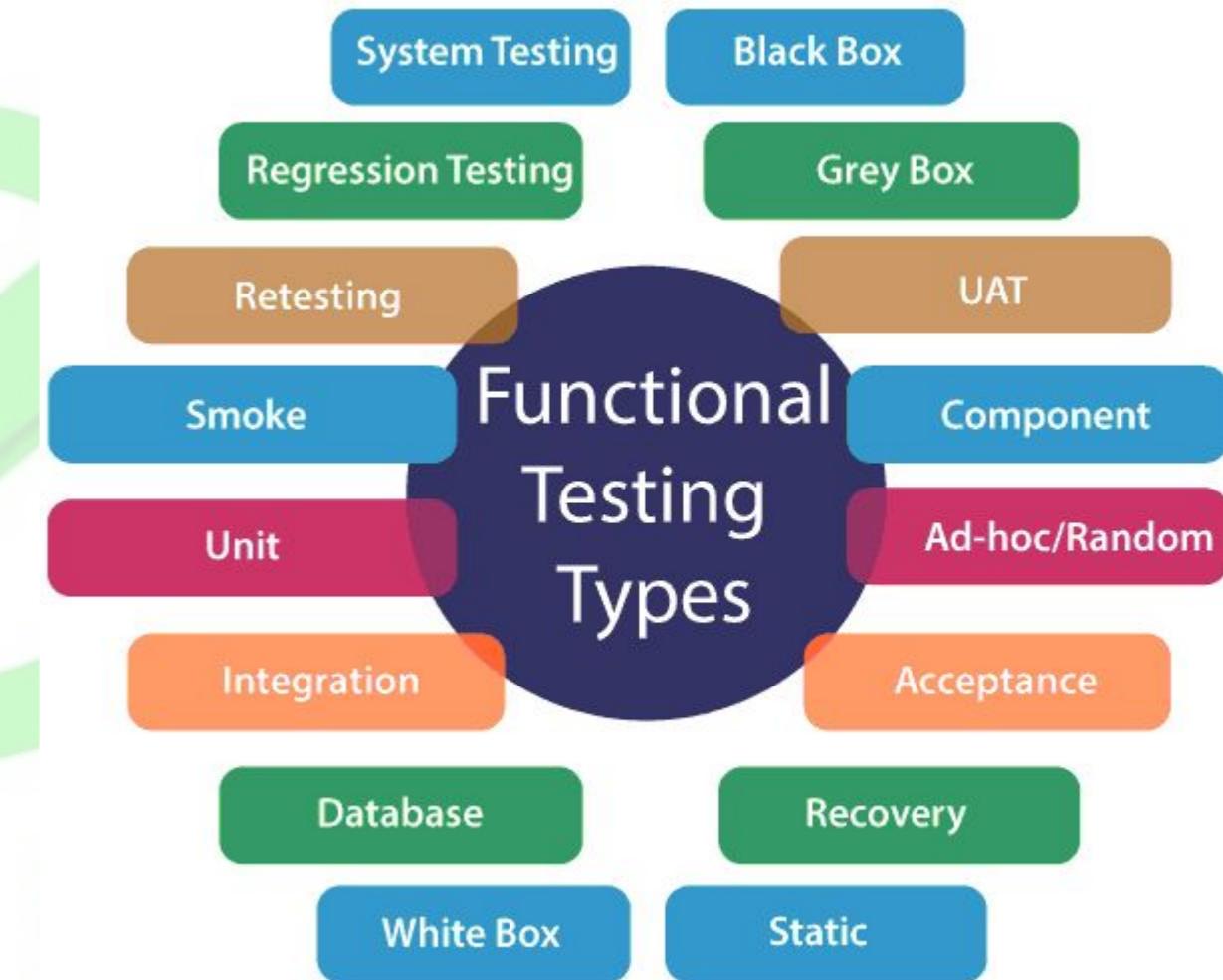
# Hybrid Integration Test

- Üst seviye modüllerin, alt seviye modüllerle test edildiği, aynı zamanda alt modüllerin de üst modüllerle entegre edilerek bir sistem olarak test edildiği bir stratejidir.
- Yukarıdan aşağıya ve aşağıdan yukarıya yaklaşımının bir kombinasyonudur, bu nedenle Hibrit Entegrasyon Testi olarak adlandırılır.



# Functional Test (Fonksiyonel Test)

- Fonksiyonel Test, testini gerçekleştirdiğimiz uygulamanın her bir fonksiyonunun verilen gereksinimlere uygun olarak çalışıp çalışmadığını doğrulayan test türüdür.
- Fonksiyonel test Black Box test altında kullanıldığı için uygulamanın kaynak kodu ile ilgili değildir.
- Bu testi gerçekleştirirken odak noktası daima uygulamanın ana işlevlerinin kullanıcı dostu olmasıdır.



# Non-Functional Test

Non-Functional test, Fonksiyonel testler kullanılarak test edilmeyen bir yazılım uygulamasının performansı, kullanılabilirliği, güvenilirliği vb. gibi işlevsel olmayan yönlerini kontrol etmek için kullanılan başka bir yazılım testi türüdür.

**Documentation testing (Dökümantasyon Test)**

**Installation testing (Kurulum Testi)**

**Performance testing (Performance Testi)**

**Load Testing (Yük Testi):** artan bir iş yükünde bir sistemin davranışını değerlendirmek için yürütülen bir tür performans testidir.

**Stress Testing (Stres Testi):** bir sistemin davranışını beklenen iş yükünün sınırlarında veya ötesinde değerlendirmek için yürütülen bir tür performans testidir.

**Endurance Testing (Dayanıklık Testi):** sürekli olarak önemli bir iş yükü verildiğinde bir sistemin davranışını değerlendirmek için yapılan bir performans testi türüdür.

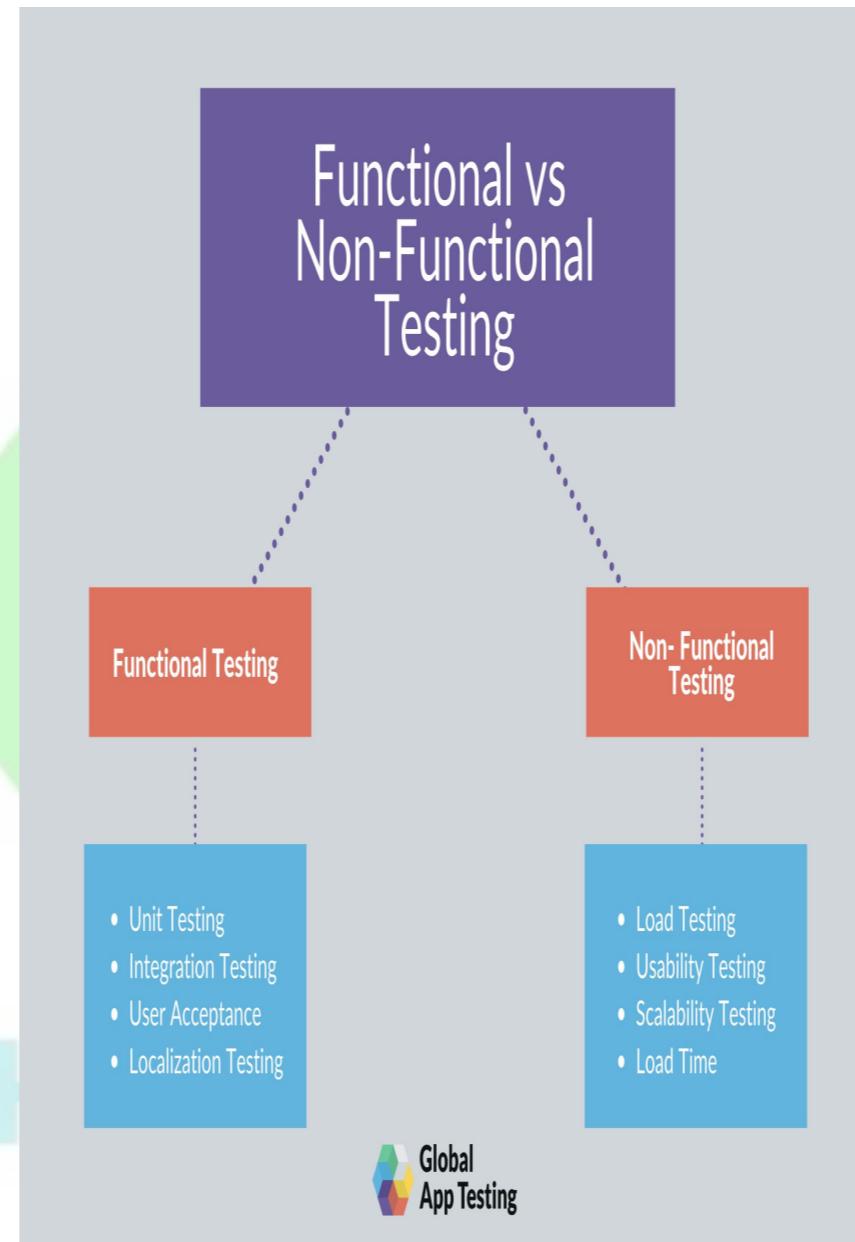
**Spike Testing (Spike Testi):** yük aniden ve önemli ölçüde arttığında bir sistemin davranışını değerlendirmek için yapılan bir tür performans testidir.

**Reliability testing (Güvenirlilik Testi)**

**Security testing (Güvenlik Testi)**

- Non-Functional (İşlevsel olmayan) testler, bir sistemin hazır olup olmadığını test etmeye yardımcı olur.
- Bir sistemin belirli davranışlarından ziyade bir sistemin çalışma şeklini tanımlar. Bu, bir sistemin fonksiyonlarını tanımlayan fonksiyonel gerekliliklere karşı test eden fonksiyonel testin tam tersidir.
- Temel olarak, işlevsel olmayan testler, işlevsel test kapsamında olmayan tüm işlevsel olmayan parametreleri kontrol etmek ve değerlendirmek için yapılır.
- İşlevsel Olmayan Testler, İşlevsel testler kadar önemlidir.
- Bir uygulamanın hız, ölçülebilirlik, güvenlik, güvenilirlik ve verimlilik gibi işlevsel olmayan tüm parametreleri, işlevsel olmayan testler kapsamında test edilir.
- Bir uygulamayı sağlamlaştırır ve belirli güvenlik açıklarına karşı hazırlar.

- Bir uygulamanın işlevlerinin spesifikasyona olarak çalıştığından emin olmak için yapılır.
- Fonksiyonel testi gerçekleştirmek için önce test girdisini tanımlamamız ve beklenen sonuçları seçilen test girdi değerleri ile hesaplamamız gereklidir.
- Sonra test senaryolarını yürütür ve gerçek verileri beklenen sonuçla karşılaştırırız.



- Uygulamanın beklenen iş yükü altında sorunsuz çalışmasını sağlamak için performans testi yapılır.
- Amaç, güvenilirlik, kaynak kullanımı vb. gibi performans sorunlarını bulmaktadır.
- Performans testi yaparken aklimızda tutmamız gereken üç ana nokta hızlı yanıt, maksimum kullanıcı yükü ve çeşitli ortamlarda kararlılıktır.

# Regression Test

- Regresyon testi, yapılan değişiklikler sonucunda sistemde değişiklik yapılan ve değişiklik yapılmayan tüm alanlarda oluşan yeni hataları bulmak için yapılan kapsamlı testtir.
- Regresyon testi, mevcut functionality'lerin sorunsuz çalıştığını emin makenin daha önce oluşturulan ve kullanılmış test case'lerin tümünün yeniden çalıştırılması demektir. (Bazen tüm Test Case'ler yerine belirlenen bazı Test Case'ler de çalıştırılabilir)
- Bu test, yeni kod değişiklerinin mevcut işlevler üzerinde yan etkileri olmadığını görmek için yapılır.
- Yazılıma geliştirme sürecinde eklenen her bir yeni işlevin var olan işlevleri bozmadığından emin olmak için, her bir döngüde Regresyon Testini uygulamak kaçınılmazdır.

## Retesting

VS

## Regression Testing

Tekrar testi, yazılım test edildikten sonra raporlanan hataların yazılım geliştiricile tarafından çözüldüğü bilgisi alındıktan sonra hatanı düzeltip düzelmediğini kontrol etmek için yapılan bir testtir.

Yeni kod değişikliklerinin mevcut işlevler üzerinde yan etkileri olmadığını görmek için yapılır.



## 1) Kısmi Regresyon

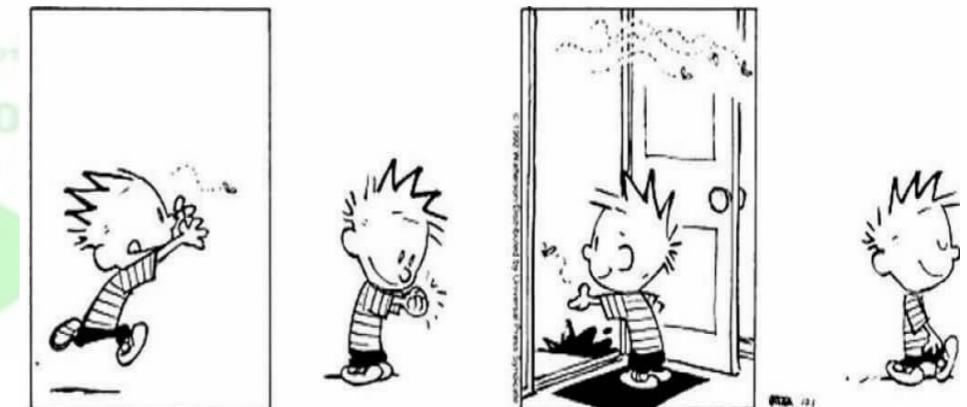
Kısmi Regresyon, kodda değişiklikler yapıldığında ve bu birimin değişmemiş veya halihazırda mevcut olan kodla entegre edildiğinde bile kodun iyi çalıştığını doğrulamak için yapılır.

## 2) Tam Regresyon

Tam Regresyon, kodda bir dizi modülde bir değişiklik yapıldığında ve ayrıca başka herhangi bir modüldeki bir değişikliğin etkisinin belirsiz olması durumunda yapılır. Değiştirilen kod nedeniyle herhangi bir değişikliği kontrol etmek için ürün bir bütün olarak test edilir.

Regression:

"when you fix one bug, you introduce several newer bugs."



**Smoke Testing:** Bir tür kabul testi olan smoke testi, yeni bir yazılım yapısının ve kritik işlevsellüğünün kararlı olduğuna dair bir ilk kontrol sağlar. Smoke testleri başarılı olursa, yapı daha fazla teste tabi tutulabilir. Verification testing (yapı doğrulama testi) olarak da adlandırılan duman testi, genellikle yeni veya kritik işlevlerin amacını karşılayıp karşılamadığını kontrol eder. Testler geçemezse, hata var demektir ve ek geliştirme çalışması gereklidir.

**Örnek:** Bir sigorta şirketi için bir web uygulaması, bir talep durumu sayfası ekler. QA'er, bir kullanıcının başarılı bir şekilde oturum açıp açamayacağı, talep durumu sayfasına gidip gidemediği ve belirli bir talebin durumunu uygulama çökmeden veya arızalanmadan alıp alamayacağı gibi mevcut yapının temel düzeyde çalıştığını doğrulamak için duman testleri uygular.

**Sanity Testing:** Bir tür regresyon testi olan Sanity testi; uygulamaya yeni bir özellik getirildiğinde özelliğin çalışıp çalışmadığını kontrol eder. Yazılıma yeni bir versiyon eklendiğinde gelen özelliklerin çalışıp çalışmadığı test edilir. Smoke testine benzer olsa da, sanity testi bir kod değişikliğinden dolayı olan hataları hedef alıyor.

**Örnek:** Bir tele sağlık sağlayıcısının web sayfası, akıl sağlığı sayfası için 404 hatası veriyor. Developerlar sorunu düzeltir, ardından QA, söz konusu sayfa için temel işlevlerin ve gezinmenin amaçlandığı gibi çalışıp çalışmadığını belirlemek için sanity test gerçekleştirir.

## **System Testing (End To End Testing)**

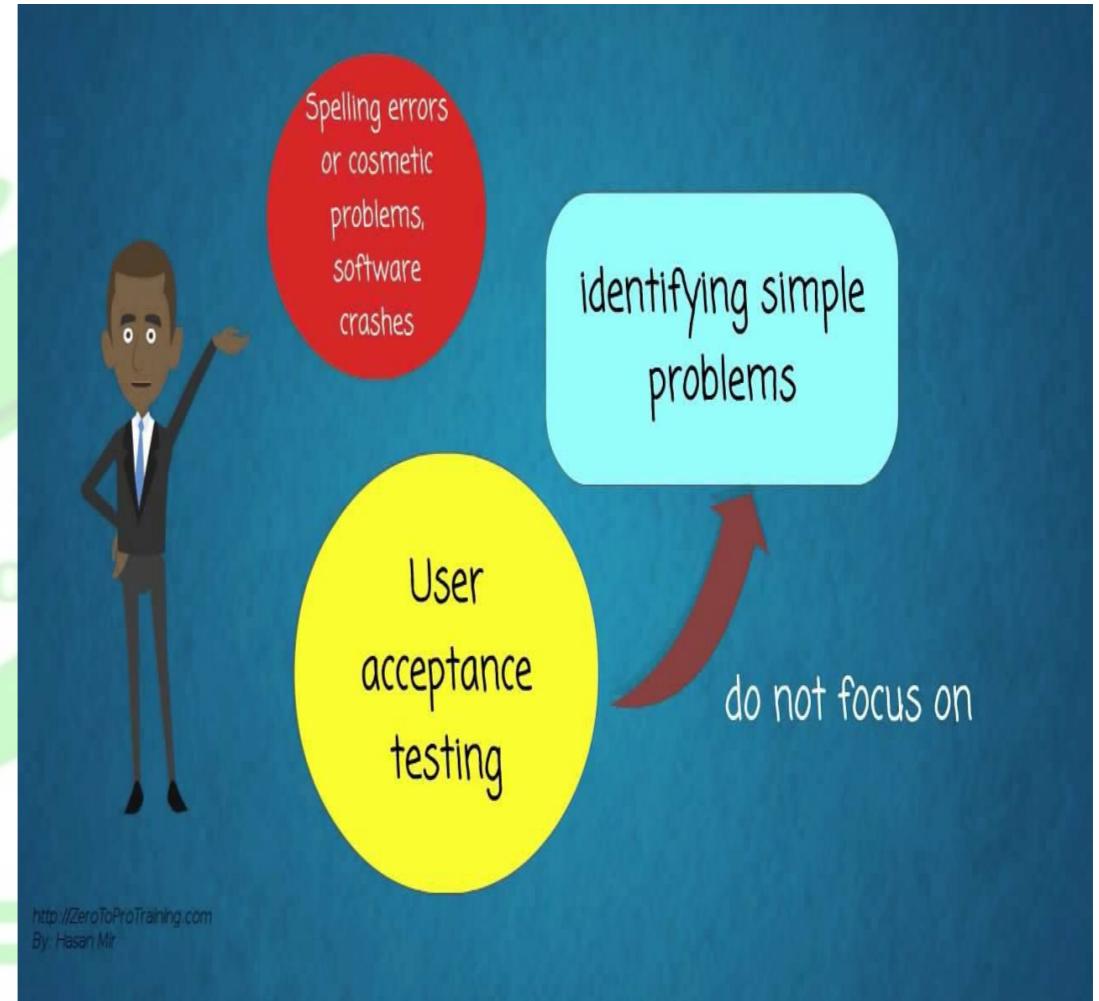
- Sistem Testi, yazılım gereksinim testlerinin tamamlanmasından sonra, sistem gereksinimlerine göre oluşturulan testleri kapsar.
- Yazılım tarafında yapılan Birim Testi(Unit Test) ve Entegrasyon Testi(Integration Test) adımlarından sonra yapılan sistem testleri, daha çok işlevi tamamlanan yazılımın, güvenlik, güvenilirlik, performans gibi faktörler altında yapılan test işlemlerini kapsar.
- Sistem testinin amacı, bir uygulamanın tasarlandığı gibi işlevleri gerçekleştirken doğruluğunu ve eksiksizliğini doğrulamaktır.
- Gerçek sonuçlar ve beklenen sonuçlar sıraya girdiğinde veya farklılıklar, müşteri girdisine göre açıklanabilir veya kabul edilebilir olduğunda sistem testi tamamlanmış olarak kabul edilir.

## **Ad-hoc Testing**

- Rastgele Test veya Maymun Testi olarak da bilinir, herhangi bir planlama ve belge olmadan bir yazılım testi yöntemidir.
- Testler herhangi bir resmi prosedür veya beklenen sonuç olmadan gayri resmi ve rastgele yapılır.

## User Acceptance Testing UAT (Kullanıcı Kabul Testi)

- Son kullanıcının veya müşterinin spesifikasyonlarına veya son kullanıcılar/müşteriler tarafından sınırlı bir süre boyunca kullanıma dayalı nihai test.
- Son Asama olarak her şey gözden geçirilir.
- Beta testi olarak da adlandırılır.
- Hiçbir sistem mükemmel değildir, bu yüzden sistem kurulmadan önce “kabul edilebilir” olması sağlanmalıdır. Kullanıcı Kabul Testi son kullanıcılarla, ürünün nihai halini gerçekten kabul edip etmediklerini belirleme fırsatı verir.



T E C H P R

## Equivalence Partitioning (Eşdeğer Aralık) Test

Aynı bölgedeki değerler girdi olarak kullanıldığında aynı sonucu verir ön koşulundan çalışmaktadır. Ornegin mail üzerinden çalışan bir testte farklı mail çeşitleri olabilir(Hotmail, gmail,yahoo gibi..) her gruptan en az bir mail ile test yaptığınızda diğerlerinin de aynı sonucu vereceği değerlendirilir.

## Boundary Value Test (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.

Yaslara göre insanları grupladığımızı düşünduğumuzda, 18 yaşın üstü genç diye bir tanım varsa 17 ve 19 yaşları denenerek sistemin doğru çalıştığı test edilebilir.



EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
<=17	18-56	>=57

# **SDLC**

## **Software Development Life Cycle**



**Bug Life Cycle**  
**(Hata Yaşam Döngüsü)**

**7. Ders**

**18/12/2021**

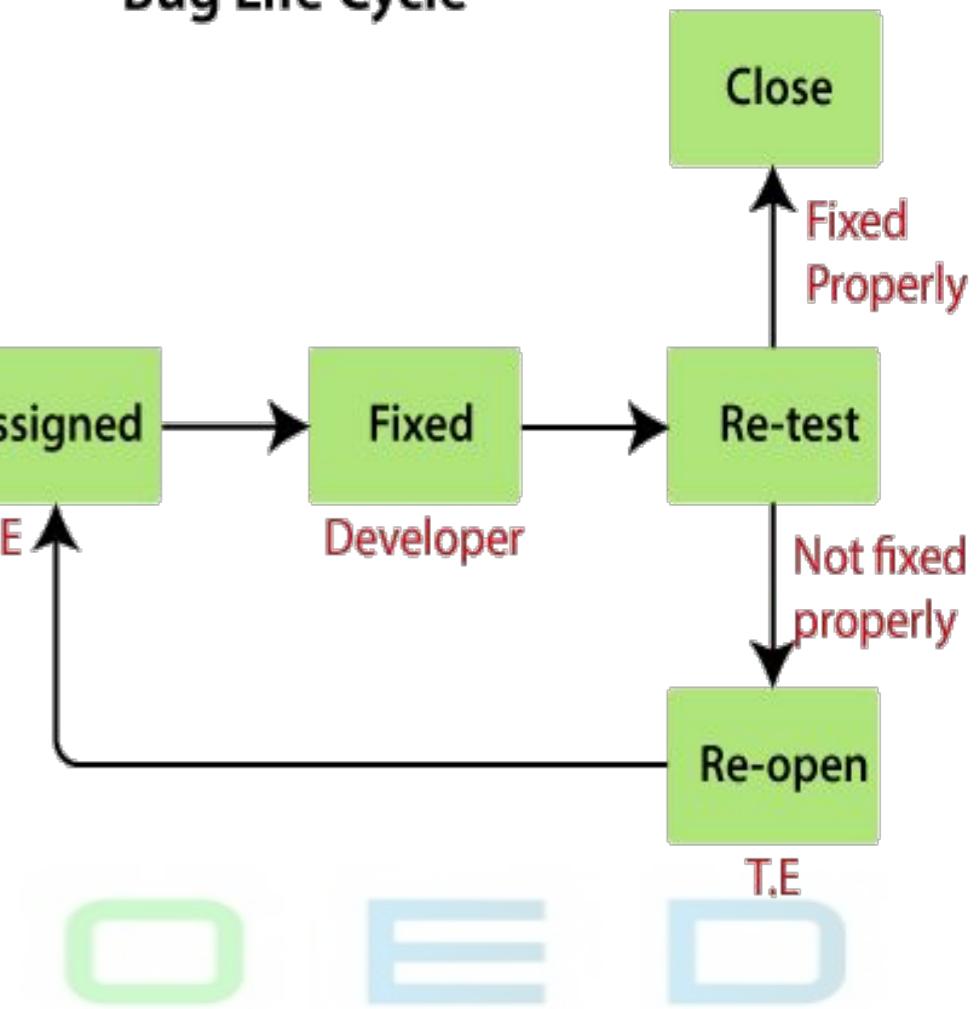
**T E C H P R O E D**

# Bug Life Cycle

Hata Yaşam Döngüsü (Bug Life cycle), bir hatanın açılışından kapanmasına kadar geçirdiği evreler olarak tanımlanabilir. Yazılım projelerinde hatalar, projenin doğasına uygun sistematik bir yaklaşımla ele alınmalıdır. Bunun için projede, tespit edilen hataların çözümünün, kontrol altında tutulabileceği bir “hata yönetimi yaşam döngüsü” kurulmalıdır.

“Yazılım geliştiricilerinin, yazdıkları programların içerisinde hem fiziksel hem de mantıksal bozukluk yaratabilen yazılım bozukluğu; **defect**, ya da **bug** olarak adlandırılır.”

Bug Life Cycle



**Bug:** Yazılım hatası (bug), bir bilgisayar programının veya yazılım sisteminin yanlış veya beklenmeyen bir sonuç üretmesine ve istenmeyen şekilde davranışına neden olan bir hata, kusur ya da arızadır. Hataları bulma ve düzeltme süreci "hata ayıklama" olarak adlandırılır. Bug, bir kodlama hatasının sonucudur.

**Defect:** En anlaşılır şekilde şöyle bir örnekle açıklanabilir. Koddaki hata error olarak tanımlanır, tester tarafından bulunan bu error, defect olarak tanımlanır, designer/developer tarafından kabul edilen bu defect'e ise bug denir. Defect, gereksinimlerden bir sapmadır.

defect

failure

bug  
error

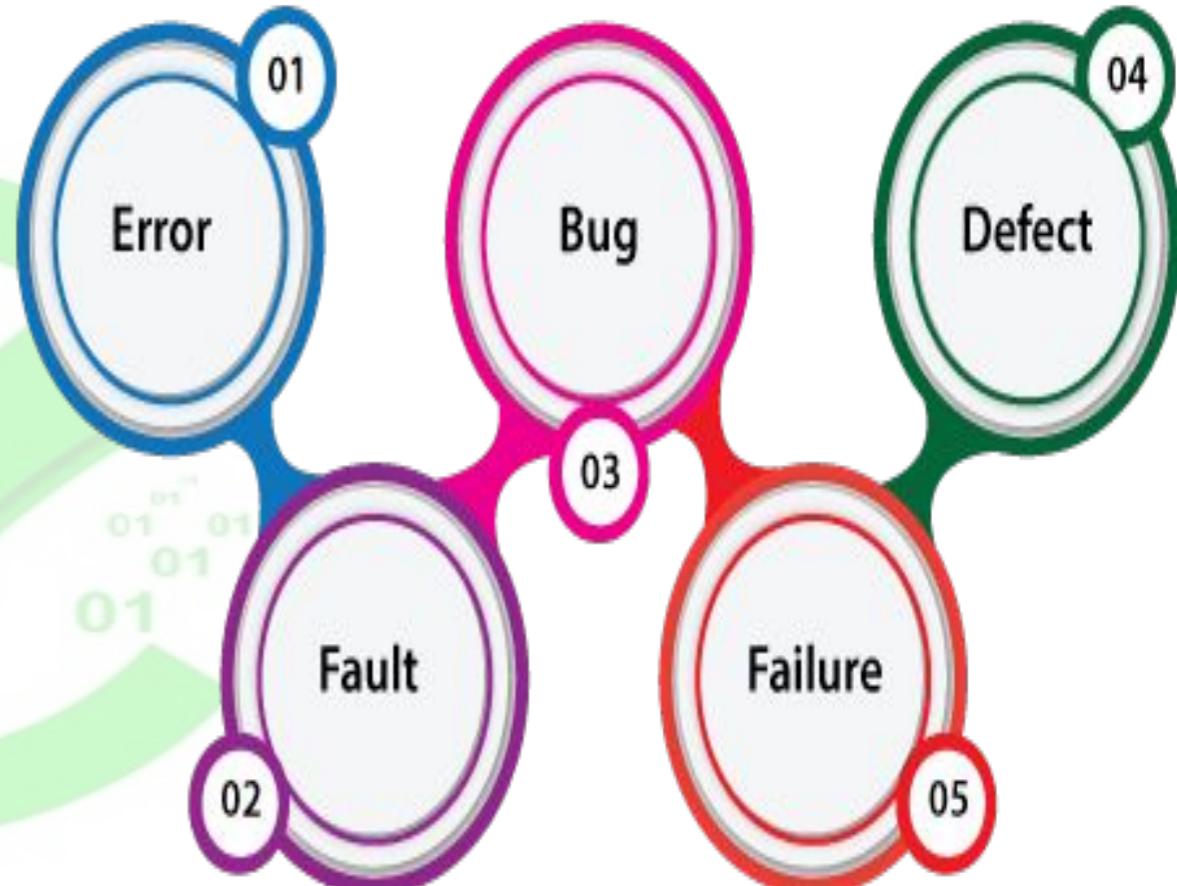
**Bug:** Kusur için belirtilen resmi olmayan bir addır.

**Defect:** Gerçek sonuçları ve beklenen çıktılar arasındaki faktır.

**Error:** Kod'da yapılan hatadır; bu yüzden kodu çalıştırılamaz veya derleyemeyiz.

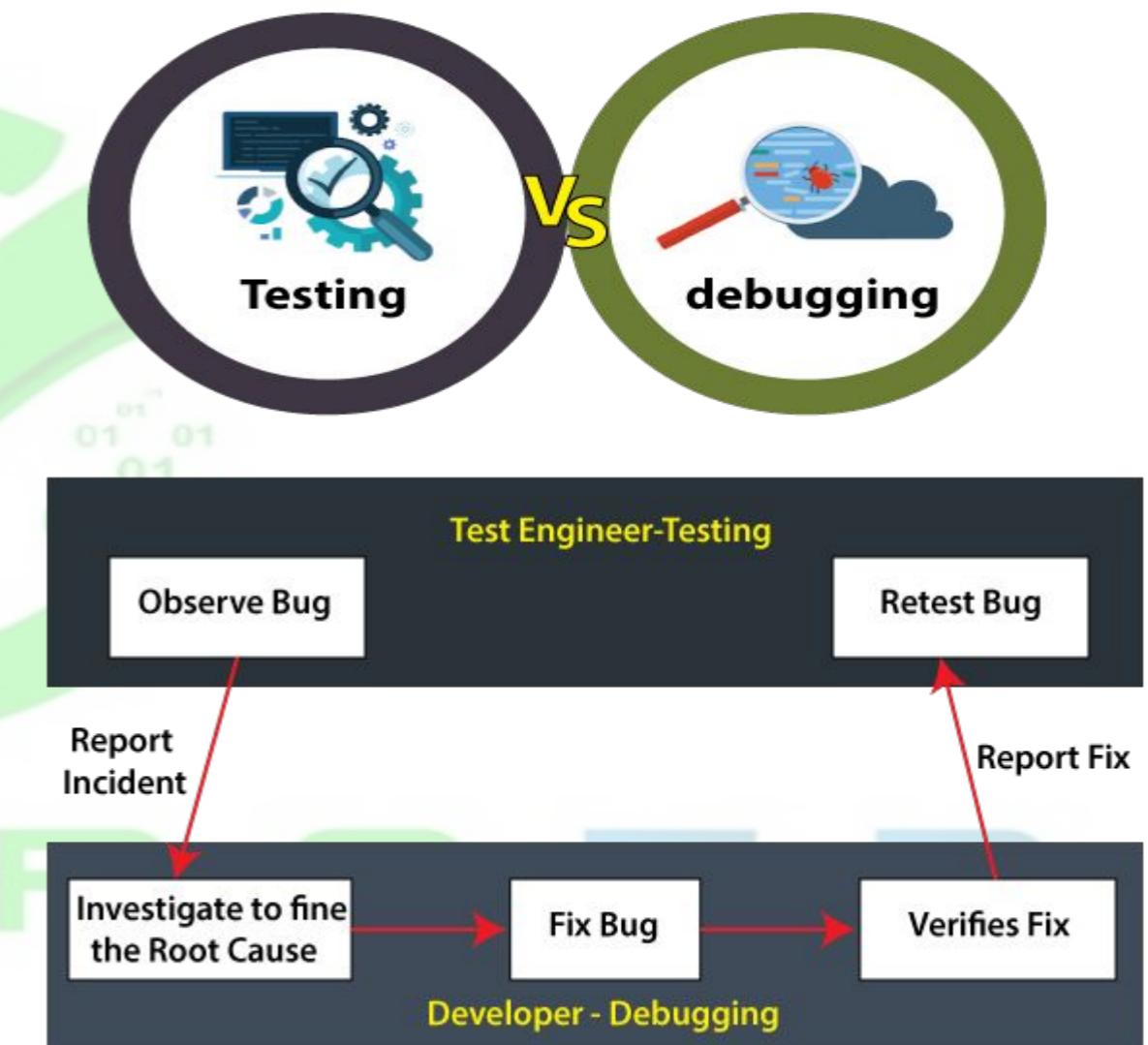
**Fault:** Yazılımın temel işlevi yerine getirmek için başarısız olması durumudur.

**Failure:** Yazılımın çok sayıda kusuru varsa, başarısızlığa neden olur.



# Test Etme (Testing) ve Hata Ayıklama (Debugging)

- Yazılım testi , yazılım ürünündeki kusurları belirleme sürecidir. Gereksinimlere kıyasla yazılımın veya uygulamanın davranışını doğrulamak için gerçekleştirilir.
- Hata Ayıklama, Test Etmenin aksine, geliştirme ekibinin veya bir geliştiricinin, yazılımdaki hatalarla ilgili test raporunu test ekibinden aldıktan sonra uyguladığı eylemdir.
- Yazılım geliştirme sürecinde hata ayıklama, bir yazılım programındaki kod hatalarının saptanmasını ve değiştirilmesini içerir.



<b>Test Etme (Testing)</b>	<b>Debugging (Hata Ayıklama)</b>
Yazılımın kusurları tespit etmek amacıyla uygulanmasıdır.	Kusurları düzeltme ve çözme işlemi, hata ayıklama olarak bilinir.
Test, manuel olarak veya bazı otomasyon araçları yardımıyla gerçekleştirilebilir.	Hata ayıklama işlemi otomatikleştirilemez.
Bir grup test mühendisi testi yürütür ve bazen geliştiriciler tarafından gerçekleştirilebilir.	Hata ayıklama, geliştirici veya programcı tarafından yapılır.
Test mühendisleri, uygulama üzerinde manuel ve otomatik test senaryoları gerçekleştirir ve herhangi bir hata veya hata tespit ederse, düzeltilmesi için geliştirme ekibine geri bildirimde bulunabilirler.	Geliştiriciler yazılım hatalarını bulur, değerlendirir ve kaldırır.
Test sürecini gerçekleştirmek için programlama bilgisi gerekli değildir.	Programlama dilini anlamadan hata ayıklama işlemine devam edemeyiz.
Kodlama aşaması tamamlandıktan sonra test sürecine geçiyoruz.	Test senaryosunun uygulanmasından sonra Hata Ayıklama işlemine başlayabiliriz.

## Bug/Defect bulduğumuzda ne yapacağız?

- Her şeyden önce, testerlar buldukları hatanın geçerli bir hata olduğundan emin olmalıdır.
- Bunun için testini tekrar etmelidir.
- Bir bug/defect bulduğunuzda, sorunu ekip içindeki diğer testerler ile tartışmalı (beraber gözden geçirmeli), böylece geçerli bir sorun olup olmadığı konusunda daha iyi bir fikre sahip olunmalıdır.
- Test ekibi olarak mutabık kalınırsa, raporlamadan önce developer ile görüşülebilir. Onun fikrini de alarak oluşturulacak Bug Ticket (hata raporu) daha verimli olacaktır.



**Başlık (Title):** 'Kaydol' düğmesine tıklandığında uygulama kilitleniyor

**Açıklama (Description):** Bir kullanıcı bir hesaba kaydolmaya çalışlığında uygulama çöküyor ve kullanıcı hesap oluşturamıyor.

**Yeniden oluşturma adımları (Steps to reproduce):** Kayıt sayfasına gidin > gerekli alanları doldurun > 'Kaydol' CTA'sını tıklayın > uygulama kilitleniyor

**Beklenen sonuç (Expected result):** Bir kullanıcı kayıt sayfasındayken, gerekli alanları doldurduğunda ve 'Kaydol' CTA'sını tıkladığında, kullanıcı başarıyla bir hesap için kaydolmuştur.

**Gerçek sonuç (Actual result):** Bir kullanıcı kayıt sayfasındayken, gerekli alanları doldurup 'Kaydol' CTA'sını tıkladığında, uygulama çöküyor ve kullanıcı hesap oluşturamıyor.

**Görsel (Visual):** bir ekran görüntüsü, video veya herhangi bir görsel.

**Öncelik (Priority):** yüksek, orta, düşük vb.



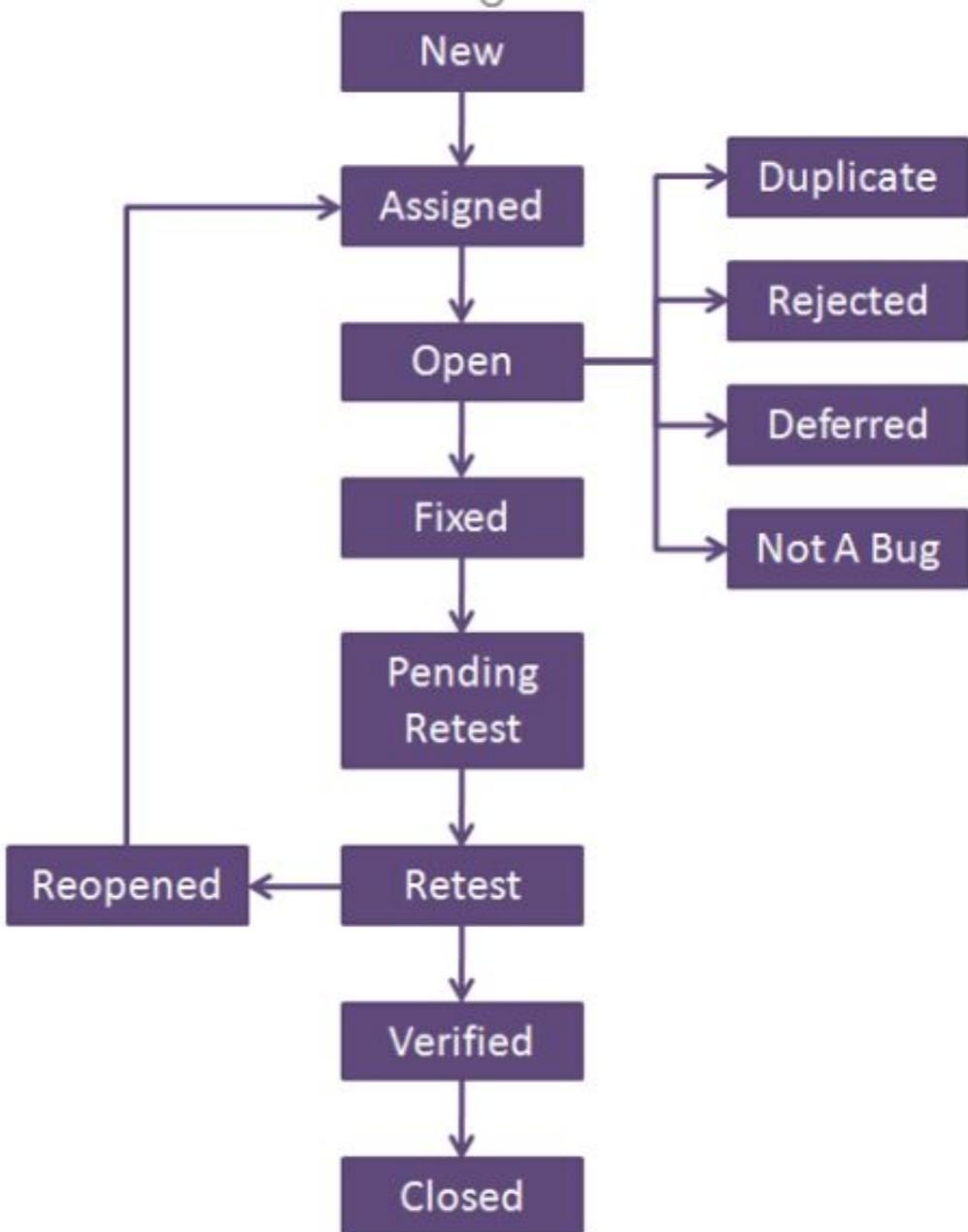
**New (Yeni):** Bir hata ilk kez yayınlandığında, durum Yeni olur. (Jira, Alm, VersionOne gibi boardlarda..)

**Assigned (Atandı):** Hata post edilince, test lead veya Proje lead, ilgili Developer'a "atanmış" olarak atayacaktır. (Meşru bir hata olup olmadığı ile alakalı Developerin kesinlikle bir girdisi olacaktır)

**Open (Açıldı):** Developer bu aşamada düzeltmek için hatayı analiz eder ve üzerinde çalışır.

**Fixed (Onarıldı):** Developer gerekli kod güncellemlerini yaptığında ve hatanın düzeltildip düzeltildmediğini doğruladığında, hata durumunu "Fixed" olarak değiştirebilir ve hata test ekibine aktarılır.

**Pending retest (Tekrar Test İçin Bekliyor):** Arızayı giderdikten sonra Developer, yeniden test için bu özel functionalityi deploy eder. Burada testler test aşamasında beklemeye olur. Ve durum tekrar test etmeyi gerektirir.

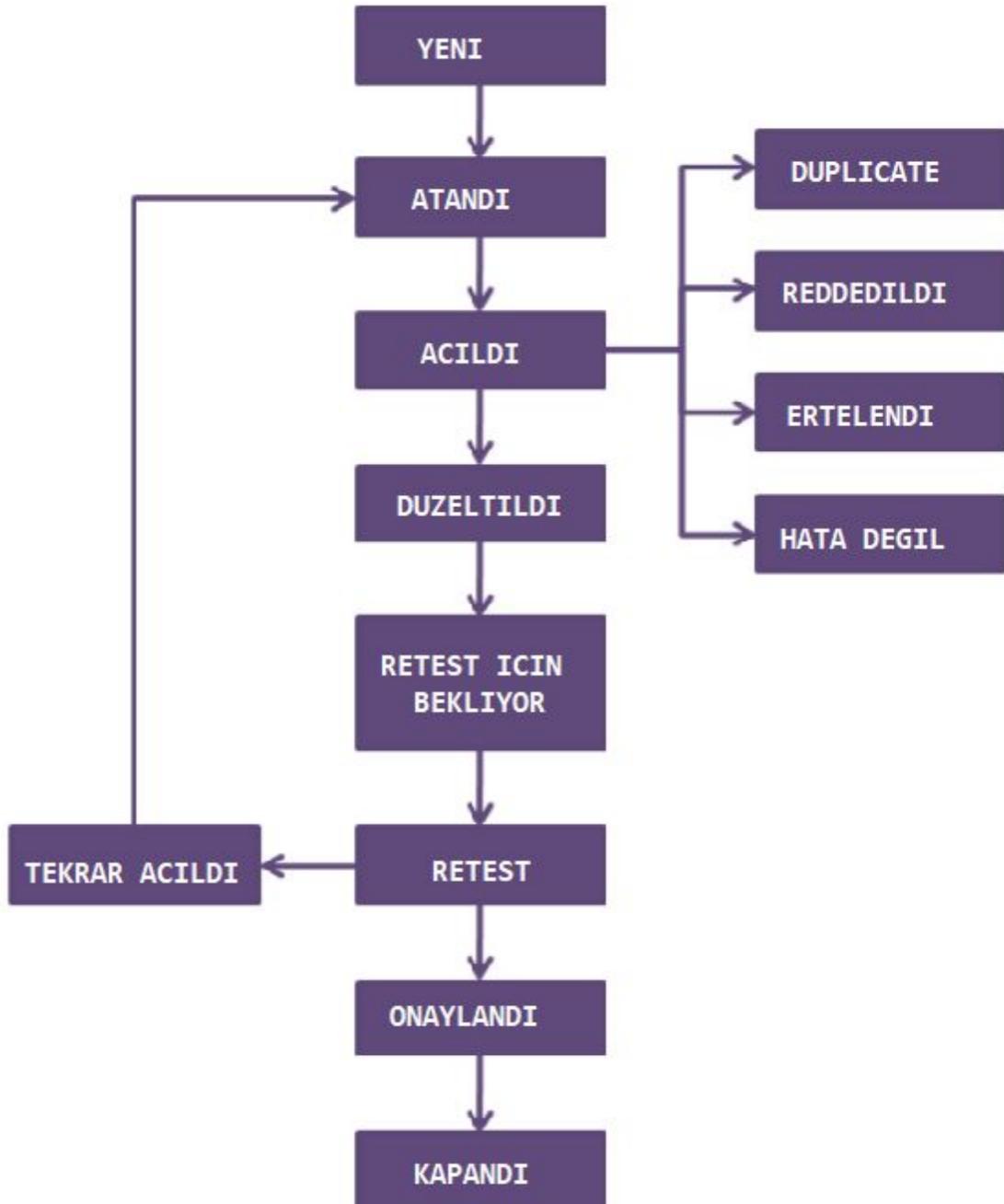


**Retest (Tekrar Test):** Bu aşamada, tester işlevselligi farklı test verileriyle tekrar test eder ve buldukları defect ilgili her şeyi kapsadığından emin olur.

**Verified (Onaylandı):** Tester doğrulama yaptıktan ve hata yazılımda görünmediğinde, hatanın düzeltildiğini onaylar ve durumu "Doğrulandı" olarak değiştirir.

**Reopen(Tekrar Açıldı):** Hata Developer tarafından düzeltildikten sonra bile hata hala mevcutsa, tester durumu "Yeniden Açıldı" olarak yapar. Bu yüzden hata aynı süreçten bir kez daha geçer.

**Closed (Kapandı):** Hata giderildikten sonra tester tarafından test edilir. Tester yazılımda defect bulunmadığından emin olursa, hatanın durumunu "Kapalı" olarak değiştirir. Bu, hatanın düzeltildiği, test edildiği ve onaylandığı anlamına gelir.

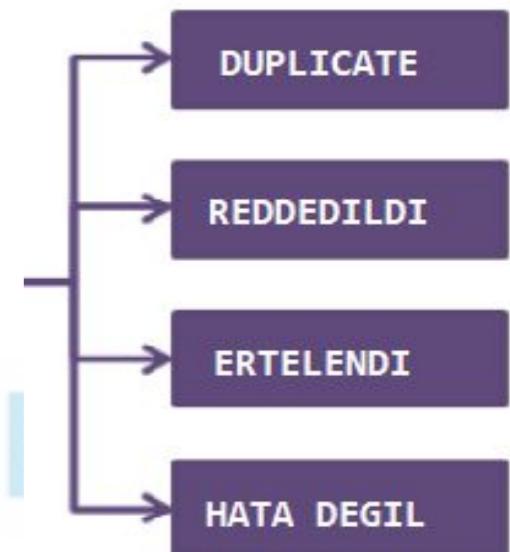
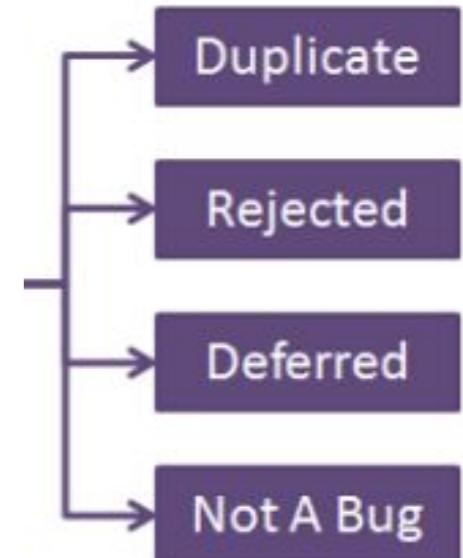


**Duplicate (Çiftleme):** Geliştirici, kusuru başka herhangi bir kusurla aynı bulursa veya kusurun kavramı başka bir kusurla eşleşirse, kusurun durumu geliştirici tarafından 'Çoğalt' olarak değiştirilir.

**Rejected (Reddedildi):** Kusur, geliştirici tarafından gerçek bir kusur olarak değerlendirilmezse, geliştirici tarafından 'Reddedildi' olarak işaretlenir.

**Defered (Ertelenmiş):** Geliştirici kusurun çok önemli bir önceliğe sahip olmadığını hissederse ve sonraki sürümlerde düzeltilebilirse, böyle bir durumda kusurun durumunu 'Ertelendi' olarak değiştirebilir.

**Not a Bug (Hata Değil):** Kusur uygulamanın işlevselliği üzerinde bir etkiye sahip değilse, kusurun durumu 'Hata Değil' olarak değiştirilir.



# Bu Dersten Sonra Resume veya CV'nize Neler Yazabiliriz?

SDLC (Software Development Life Cycle)

Agile Methodology

STLC (Software Testing Life Cycle)

BLC (Bug Life Cycle)

konularında bilgili, deneyimli, çalışmış, tecrubeli .....

T E C H P R O E D