Ali Efe Çoban

190601006

**Yazılım yaşam döngü modelleri**

**ve**

**çevik yazılım geliştirme modelleri**

# Yazılım Geliştirme Modelleri

Yazılım Geliştirme Modelleri ve Yazılım Yaşam Döngüsü, yazılımın geliştirilmesi ve bakımının gerçekleştirilmesi süreçlerini belirli aşamalara bölerek sürecin yönetilmesini düzenler. Bazı modellerde geliştirme ve bakım süreci belirli bir aşamadan sonra biterken bazı modellerde bu süreç bir döngü olarak kendini tekrar eder.

Modellerin hepsi temel olarak bir yazılıma sıfırdan başlayıp o yazılımı gerçekleştirmeyi hedefler. Yazılımın *gereksinimleri toplanır*, *analiz yapılır*, *tasarım geliştirilir* ve *gerçekleştirme* ile yazılım ortaya çıkar. Genelde yazılım geliştirme doğrusal ilerleyemez. Bunun iki temel nedeni vardır.

1. Yazılımı geliştiren insandır ve hata yapabilir.
2. Müşterinin gereksinimleri değişebilir.

# Yazılım Yaşam Döngüsü

Bir yazılımın geliştirilmesi ve bakımı sırasında geçirdiği tüm aşamalara verilen tanımdır. Geliştirme ekiplerine göre değişiklik gösterebilir. Örnek olarak bazı ekiplerde 4 aşamaya kadar inerken bazı ekiplerde 10 aşamaya kadar çıkabilir. Genel olarak aşağıda yer alan aşamalardan oluşur.

## Planlama - Gereksinim - Analiz

Müşteri gereksinimleri elde edilir. Projenin temel ihtiyaçları belirlenir. Sistem gereksinimleri ve işlevleri ayrıntılı olarak incelenir. Detaylı dokümanlar hazırlanır. Müşteri ve Yazılım Mühendisi arasında iletişimin iyi tutulması gereklidir. Analiz aşaması problemin ne olduğunun belirlendiği aşamadır.

## Tasarım

Yazılımın temel yapısı bu aşamada oluşturulur. Müşterinin gereksinimleri ve istekleri birleştirilerek yazılımın ne gibi özelliklere sahip olacağına karar verilir. Bu aşama Mimari Tasarım ve Detaylı Tasarım olarak ikiye ayrılır. Mimari Tasarım’da modüller ve akış şemaları gibi sistemin genel yapıları tasarlanır. Detaylı Tasarım’da bileşenler ve bu bileşenlerin ayrıntıları yer alır. Mimari Tasarım detaylandırılır ve Mimari Tasarım revize edilir. Kısaca, Tasarım aşaması problemin nasıl çözüleceğinin belirlendiği aşamadır.

Tasarım hazırlanırken **Soyutlama** (*Abstraction*) **Yöntemi** ile bazı durum ve olaylar görmezden gelinerek problemin çözümüne ulaşmak daha kolay olur. Tasarım aşamasında **Statik** ve **Dinamik Modelleme** yapılır. Statik Modelleme, yazılımın değişmezlerini ifade ederken Dinamik Modelleme, yazılımın işleyişini ifade eder.

## Gerçekleştirme

Yazılımın ve modüllerin kodlandığı, testlerin yapıldığı ve kurulum çalışmalarının yapıldığı aşamadır. Bu aşamada yazılan kod ne kadar temiz, standartlara uygun (takımın veya şirketin) ve basit ise daha sonra tekrar düzenlenmesi gerektiğinde o kadar kolay ve hızlı düzenlenir veya bakımı gerçekleştirilir. Yazılım hazırlandıktan sonra ne kadar erken test edilirse bulunacak sorunların tespiti ve düzeltilmesi o kadar kolay olur.

## Teslim – Bakım

Testlerden başarıyla geçen yazılım artık teslim aşamasına ulaşır. Bu aşamada kullanıcılar içinde bir döküman oluşturulması gereklidir. Teslim ile birlikte yeni bir aşama olan bakım aşaması da başlar. Teslim edilen ürünün ortaya çıkan hataları, iyileştirmeler, yeni özellikler veya eski özelliklerin çıkartılması gibi bakım faaliyetleri vardır.

# Yazılım Yaşam Döngü Modelleri

## **Gelişigüzel Model**

Gelişigüzel Model aslında bir model değildir çünkü herhangi bir modeli veya yöntemi yoktur. Geliştiren kişinin izlediği yollardan oluşur ve kişiden kişiye göre değiştiği için bir standart yoktur. Bakımı zordur.  
 Bu model, yazılıma yeni başlayanların kullandığı adımlara verilen model adıdır.

## **Barok Model**

Temel adımları tanımlanmışken bu adımlar arasında geri dönüş tanımlanmamıştır. Temel adımlar doğrusal olarak ilerler. Diğer modellerden en büyük farkı, dokümantasyon aşamasının yazılımın geliştirme ve testlerden sonra yapılmasını söyleyen ayrı bir süreci vardır.

## **Şelale/Çağlayan Yaşam Döngü Modeli**

En temel ve en popüler yazılım geliştirme modellerinden biridir. Geleneksel yazılım geliştirme modeli olarak da bilinir. Bu modelde aşamalar en az bir kez tekrarlanır. Günümüzde yazılım alanında kullanımı azalmaktadır.

Şelale modelinde işlemler doğrusal olarak ilerler. Her aşamada dokümantasyon hazırlanması ve testlerin yapılması gereklidir. Dokümantasyon ve testler sürecin içine dağılmış durumdadır. Barok Modelinden farklı olarak aşamalar arasında geçiş vardır. Analiz aşamasında müşteri ile sistem gereksinimleri ve istekler en ince ayrıntısına kadar anlaşılmış ve belirlenmiş olması gereklidir. Müşteri ile projenin başında ve sonunda görüşüldüğü için yani müşteri geliştirme aşamasının içinde olmadığı için sonradan geri dönüşler artabilir ve düzeltmelerin sonradan yapılması hem maliyeti arttırır hem de geliştirme ekibini yorar.

Çok iyi tanımlanmış, küçük ve kısa zamanda geliştirilmesi gereken yazılımlar için kullanılır. Uzun süreli projelerde daha esnek yapılı modellere başvurulması gerekir.

## **Fıskiye Modeli**

Şelale Modeli’nden türemiş Fıskiye Modeli’nde her bir adım kendi döngüsüne sahiptir. Bu döngü sayesinde kendisinden önceki aşamaya geri dönülmesi daha kolaydır. Aşağıdan yukarıya doğru ilerleyen bu modelde yayınlanmadan sonra genelde 2 ayrı ekip süreci devam ettirir. Bir ekip bakım işleri ile uğraşırken diğer ekip sisteme eklenecek yeniliklerle uğraşır.

## **V Modeli**

Şelale Modeline benzer fakat ondan farklı aşamaları olan bir modeldir. Yukarıdan aşağıya doğru planlama, ihtiyaçların belirlenmesi, üst seviye tasarım, alt seviye tasarım ve kodlama aşamaları yapılır. Buraya kadar Şelale Modeli ile aynıdır. Buradan sonra aşağıdan yukarıya doğru çıkılmaya başlanır. Sırasıyla; birim, entegrasyon, kabul testleri yapılır. Kabul testi müşteri tarafından yapılır. Müşteri kabul ederse artık bakım aşaması başlar.

Önce yukarıdan aşağıya doğru akan bir şelale gibi ilerleyen daha sonra aşağıdan yukarıya doğru çıkan bu modelde yukarıdan aşağıya inilen süreç üretim, aşağıdan yukarıya çıkılan süreç test bölümleri olarak adlandırılır. Kısaca V’nin sol tarafı üretim, sağ tarafı test işlemleridir.

Bu modelde 3 alt model vardır. Kullanıcı Modeli, Mimari Model, Gerçekleştirim Modeli. Her model sol da üretimi yapılan aşamanın sağ da testinin yapılmasını söyler.

İhtiyaçlar, Kabul Testi ile test edilir. Üst Seviye Tasarım, Entegrasyon Testleri ile test edilir. Detaylı Tasarım, Birim Testleri ile test edilir. Kısaca V Modeli yapılan işin karşılığının doğru olup olmadığı önemli olan bir modeldir.

Belirsizliklerin az olduğu, istenilenin zamanla değişmeyecek olduğu Bilgi Teknolojileri gibi projeler için uygun bir modeldir.

## **Helezonik Model**

Planlama, Risk Analizi, Üretim ve Kullanıcı Değerlendirmesi olarak 4 ana gruptan oluşur. Değişik döngülerden, aşamalardan ve risk analizlerinden geçer. Bu model ile birlikte Risk analizi ön plana çıktı. Her döngü faz olarak ifade edilir. Her döngüde bir prototip ile programın son haline yavaş yavaş yaklaşılır. Prototipler ile birlikte kullanıcılar son yazılıma yavaş yavaş hazırlanır. Projenin dışarıdan izlenebilirliği prototipler ile birlikte artar. Hem müşteri hem yöneticiler ilerlemeleri görebilir. En son hazırlanan prototip hem geliştirici ekibinin hem de müşterinin onayından sonra Şelale modeline benzer bir yapıya girer ve en sonunda canlıya çıkar.

## **Artımsal Model**

Bu modelde problemler ve yeni planlar ile döngü beslenir ve artırım sağlanır. Geliştirme ve teslim parçalara bölünür ve her teslim ile son ürüne yaklaşılır. Kullanıcının gereksinimleri öncelikli olan bu modelde öncelikli olan gereksinimler erken teslimlere dahil edilerek geliştirme sürdürülür. Her yinelemede başka bir gereksinim veya gereksinimler eklenir ve ilk başta da söylediğimiz gibi artırım sağlanır. Son çıkan her sürüm bir öncekini kapsar ve giderek genişleyen bir yapıdadır. Bu modeldeki erken teslim mantığı prototip yapısına da benzemektedir.  
 Uzun zaman sürecek projeler için uygun bir yapısı vardır. Bu yapıya benzer bir örnek İşletim Sistemleri verilebilir. Teslim yapılıp kullanıcıların yazılımı kullanması sağlanırken aynı zamanda geliştirme de sürmektedir.  
 Bir sonraki artırımın geliştirilmeye başlanması için bir önceki artırımın bitmesine gerek yoktur. Öncelikli olan gereksinimler ilk teslimlerde teslim edildiği için daha sonraki teslimlerde de testleri sürmektedir.

## **Kodla ve Düzelt Modeli**

Bu model ile birlikte ilk önce yazılım ürünü gerçekleştirilir. Sistem istenilen haline gelene kadar geliştirmeler devam eder ve bu sırada dokümantasyon yapılmaz. Yazılım geliştirmenin en kolay yollarından biridir. Bakım çok zordur ve yazılımın emeklilik safhası vardır.   
 Küçük çaplı veya kısa ömürlü prototip projelerde kullanılmaya uygundur.  
  
**Ayrıca:** Yazılım Yaşam Döngüsü (*SDLC = Software Development Life Cycle*) sırasında bir aşamada sorun çıkarsa o sorun diğer aşamalara da aktarılıyor ve döngü tekrar edene kadar o sorun çözülemiyor. SDLC ortaya çıktıktan sonra bu yaklaşımın yazılımdan çok sistem geliştirme döngüsü olarak kabul görmeye başladı ve adı Systems Development Life Cycle olarak değiştirildi.

# Çevik Yazılım Geliştirme (Agile)

Yazılım Yaşam Döngü Modelleri başarılı modeller olmasına rağmen yazılımda genel başarısı çok iyi olamadı. Sistem ne kadar büyürse kontrol edilebilirliği ve hareket kabiliyeti o kadar zorlaştı. İnsanlar bu metotların bir yerden sonra yeterli olmadığına karar verdiler. Bunun üzerine yeni metotlar geliştirilmeye başlandı ve geliştirilen bu yeni metotları kullanmaya başladılar.

Geliştirilen yeni metotlar aslında Çevik Yazılım Geliştirme Sürecine(Agile Programming) ve XP metotlarına çok benziyor fakat ortak bir çatı altında toplanamamıştı. Çevik Yazılım Geliştirme ortaya çıktığında bu kullanılan yeni metotları ortak bir çatı altında toplayarak belirli standartlara oturttu.

Çevik Yazılım Geliştirme, asıl işin programlama olduğunu öne sürdü. Önceki metotlarda %30 gibi bir oranda programlama yapılıyordu. Geriye kalan %70’lik aşamaları ne kadar hızlandırırsak asıl işimiz olan programlamaya o kadar odaklanabiliriz denildi ve bununla birlikte müşterinin geliştirme sürecine entegre edilmesi gerektiğine karar verildi.

Çevik Yazılım Geliştirme’nin ana odağı **yazılım** ve **müşteri** oldu. Asıl amaç hızlı, esnek, hata oranı az olan yüksek verimli yazılımların geliştirilmesi ve bu yazılımların sürdürülebilmesi oldu.

Çevik Yazılım Geliştirme ile yazılım geliştirme artık küçük yinelemeler ile geliştirilmeye başlandı. Bu yinelemeler 2 ile 4 hafta gibi bir süre içerisinde sürüyor ve sıradaki yinelemeye geçiliyor. Bu geliştirme sırasında ekip içerisindeki iletişim ne kadar iyiyse geliştirme aşaması o kadar verimli geçiyor yani geliştirme sırasında asıl amaç bir ekip olarak çalışmak ve iletişimi güçlü tutmak. Her yinelemede ortaya bir ürün çıktığı için bu ürün müşteriye gösterilerek proje hakkında müşterinin aklındaki soru işaretlerini giderir ve müşteriyi de geliştirme sürecinin içine alır.

Çevik Yazılım Geliştirme ile Süreç ve Araçlar yerine Bireyler ve Etkileşimler, Kapsamlı Belgeler yerine Çalışan Yazılım, Sözleşme Görüşmeleri yerine Müşteri İlişkileri, Plan İzleme yerine ise Değişikliğe Açıklık başlıkları ön plana çıktı. Bunların yanında *basitliğin* önemine vurgu yapıldı. Ekipler kendilerini organize edebilmeli ve kendilerini gözden geçirip verimliliği arttırmak için süreç iyileştirmeleri yapmalıdır.

Çevik Yazılım Geliştirme’nin bir de Çevik manifestosu vardır ve bu manifestoya göre bazı temel prensipler vardır:

1. Hızlı ve sürekli olarak kullanılabilir yazılım teslimatı.
2. Gereksinimlerin değişikliği kodlama aşamasının ilerleyen safhalarında bile yapılabilir.
3. Kısa zaman aralıklarında çalışan, kaliteli yazılım teslimatı.
4. Tüm ekip elemanları arasında iletişim yüksek tutulmalı ve birlikte çalışılmalı.

# Çevik Yazılım Geliştirme Modelleri

## **Extreme Programming (XP)**

Ekip içi iletişime önem veren, geri dönüşlere daha çok imkan sağlayan bir yöntemdir. Bu metotta değerler ve pratikler vardır.

**Temel Değerleri**; *iletişim*, *basitlik*, *geri bildirim* ve *cesaret*tir.  
 **İletişim**: Projelerdeki en büyük sorun, ekip içindekilerin bir biri ile olan anlaşmazlıkları veya iletişim kopukluklarıdır. Bu yüzden XP modeli, ekip içi ve ekip ile müşteri arasındaki iletişimi çok önemli görür. İletişim yüz yüze ve sıkı olmalıdır. Eğer bir konu hakkında bir tarafın bilgiye ihtiyacı varsa en kısa sürede o bilgiyi bu sıkı iletişim ile sağlamalıdırlar.  
 **Basitlik**: Basitlik aslında zor bir konudur. Geliştirme sırasında gereksinime en basit çözüm kullanılmalıdır. Karmaşıklık XP’ye aykırı bir kavramdır.

“*UNIX is very simple, it just needs a genius to understand its simplicity.*” -**Dennis Ritchie**

**Geri Bildirim**: Yazılımcılar, daha sonradan ortaya çıkabilecek hataları ortadan kaldırmak için geri bildirim sistemini kullanırlar. Mantıksal yapı içeren birim testler ile sistem hakkında somut bilgiler elde edilir. Müşteri, yönetici ve diğer proje çalışanlarının görüş ve fikirleri önemlidir. Müşteri; proje grubunun bir üyesi durumdadır. Burada gerçekten projeyi isteyen kişiler müşteri olmak zorunda değildir. Müşteri, projeyi isteyen gerçek müşteri gibi projeyi denetler, her ayrıntısı hakkında bilgisi vardır ve proje grubu içinde bulunur. Bu sırada tekrarlamalar ile 2-4 hafta aralıklarla çıkan sürümler ile sistem kontrol edilir. Müşteri ve ekip birlikte projenin durumu hakkında belirli zamanlarla toplanıp görüşürler. Bu sayede müşteri ve ekip arasındaki iletişim ile daha sonradan çıkabilecek anlaşmazlıklar çözülmüş olur.  
 **Cesaret**: Projeler geliştirilirken ekip üyeleri yılmadan, başarısızlıktan korkmadan projedeki zorluklar ve ortaya çıkan başarısızlığa neden olan sebeplerin üstüne gidilmelidir. Yazılım ekibi çıkardığı veya çıkartacağı ürünün gidişatından memnun değilse projeyi tamamen baştan yazabilmelidir. Eğer ekip kendinde yeterli cesareti bulamıyorsa o zaman ortaya çıkacak olan ürünün başarısız olması veya zamanında teslim edilememesi gibi sorunların ortaya çıkması yüksek ihtimalle gerçekleşir.

**Pratikleri** 12 tanedir ve 3 parçadan oluşan “*soğan*” olarak adlandırılabilir bir yapısı vardır. Bu yapı içten dışarıya doğru *bireysel*, *takım*, *organizasyon* olarak gruplandırılır.  
 **Planlama Oyunu**: Müşterinin belirlediği her bir yineleme için, müşterinin de bulunduğu toplantıda yazılım ekibi projenin ne kadar süreceği hakkında tahminlerde bulunur. Bu sırada Planlama Kartı adı verilen kartlar ile her bir ekip üyesi fikir belirtir. Eğer yakın tahminler yapıldıysa proje benzer sürede yapılabilir. Fakat tahminler arasında çok fark varsa burada bir sıkıntı vardır. Bu toplantıda müşteri kendi işinin etki alanları içinde gerekli bilgileri ve fikirlerini paylaşırken yine ekip üyeleri kendi alanlarında fikir belirtir ve söz söyler. Ne müşteri ekip üyelerine ne de ekip üyeleri müşteriye bilmediği konuda söz söyleyemez veya ne yapması gerektiğini söyleyemez. Her iki taraftan kendi alanları hakkında konuşmalı, müşteri isteklerini ve gereksinimlerini söylemeli, ekip üyeleri de bu iş hakkındaki görüş ve bilgilerini paylaşmalı, projenin nasıl, ne kadar sürede yapılacağı konusunda konuşmalı. Her iki tarafta proje için fayda sağlamalıdır. Bazen bu toplantılarda 2 geliştirme ekibi birden yer alır ve hangi ekip müşterinin istediği projeyi daha hızlı ve doğru şekilde hayata geçirirse o ekip bu oyunu kazanmış olur.  
 **Metafor Kavramı**: İletişimi daha etkili kurmak için ortaya çıkmış bir kavramdır. Benzetme, temsil gibi kavramların kullanılmasını sağlar.  
 **Ekipte Müşteri**: Müşterinin proje geliştirme içindeki varlığı önemlidir. Müşteri yerine Müşteri temsilcisi, yazılım ekibi ile birlikte projenin geliştirilmesi aşamasında aynı ortamda bulunur. Bu sayede müşteriye ihtiyaç duyulduğunda kısa sürede müşteri veya müşteri temsilcisi ile görüşülebilinir.  
 **Önce Test**: Asıl programdan önce güvenli yazılım geliştirilmesi ve sorunların tespiti için test yazılımı geliştirilir.  
 **Basit Tasarım**: Müşterinin gereksinimlerini en basit tasarım ile gerçekleştirmek gerekir. Bu yöntem ile gereksinimler kısa sürede anlaşılır, yönetilebilir ve değiştirilebilir.  
 **Çiftli Programlama**: İki yazılımcı tek bilgisayar üzerinde beraber kod yazar. Yeni başlayan çalışanları eğitmek için güzel bir yol olabilir fakat uzun vadede sıkıntılı bir sürece dönüşür. Burada iki yazılımcı arasında güven ve iletişim oluşur. Yeni gelen yazılımcı, ekibin standartlarını ve kültürünü öğrenir. Bu sırada kaliteli kod geliştirilir, eğlenceli olabileceği gibi kavgalı bir süreçte olabilir, yazılım daha hızlı geliştirilir ve biri o sırada bir bölümü geliştirirken diğeri sistemin genelini görür ve bağlantıların nasıl olması gerektiğini görür. Bu sırada yazılımı yazan kişi bir hata yaparsa diğeri hemen hatayı giderebilir. (Mentor ve Mentee yapısına benzer. Beraber öğrenirler. Biri daha bilgili bir geliştiricidir, bilgi paylaşımı yapar.)  
 **Sürekli Entegrasyon**: Yen bileşenler ve değişiklikler sisteme entegre edilir ve bütün sistem ile test edilebilir. Sistemin sürekli olarak entegre edilmesi ile diğer yazılımcılar da oluşabilecek sorunları görüp erken safhada sorunları düzeltebilirler.  
 **Kısa Aralıklı Sürümler**: Proje zaman dilimlerine bölünür ve her birinin bir teslim süresi vardır. Müşteri projenin ilerleyişini ve sona nasıl ulaştığını görebilir.  
 **Yeniden Yapılandırma (***Refactoring***)**: Yeni bir ekleme yapılacağı zaman önceden yazılmış koda eklemeler veya düzeltmeler yaparak o yeni eklemenin sisteme yapılmasını sağlar. Bu sırada bir sınıf, fonksiyon veya bir yapı sisteme yapışmaya başlaması(o kısım çıkartılırsa programın çalışmaması gibi) istenmeyen bir durumdur. Bu sorunun yanında birden fazla defa aynı işi yapan sistemler yazılması da başka bir istenmeyen durumdur.  
 **Ortak Kod Sahiplenme**: Geliştirilen yazılım bütün ekip üyelerinin ortak ürünüdür ve bütün ekip üyeleri bu kodu içselleştirmelidir. Bir ekip üyesi başka bir ekip üyesinin yazdığı kodda düzenlemeler yapabilir.  
 **Kodlama Standardı**: Ekip üyeleri ortak bir kodlama standardı kullanmalıdır. Burada amaç yazılan kodun karmaşıklığını azaltmak ve ekip tarafından kolaylıkla anlaşılması sağlanır.  
 **Haftada 40 Saat**: Bir geliştirici kafasını ne kadar rahat ve temiz tutarsa proje geliştirmesinde o kadar verimli olur ve hataların ortaya çıkması azalır. Bu yüzden bir geliştirici haftada 40 saat çalışmalı. Fazla mesai yapması verimliliği düşürür ve hataları arttırır.

## **SCRUM**

Çevik bir yapıdır. Proje küçük proje parçaları halinde ele alınır. Yeni gereksinimler Şelale modeline göre daha hızlı ele alınır. İş sürekli olarak ilerler. Gereksinimlerin değişmesi bu modeli olumsuz etkilemez, aksine bu tip değişiklik yapılan projeler için geliştirilmiş bir modeldir. Gereksinimlerin toplanması, tasarım, programlama ve test aşamaları neredeyse iç içe geçmiştir.

**Ara Not**: İstenen iş ve kullanılacak teknoloji kompleksleştikçe proje içinden çıkılmaz bir hal alır ve sonuçta kontrol edilemez ve bitirilemez bir hal alır.

Bu modelde günlük 15 dakikalık toplantılar içeren ve 15 ile 30 gün arasında süren sprintler (periyotlar) vardır. Her sprintten sonra Product Backlog’tan yeni Sprint Backlog’lar alınır ve bunlar parçalara ayrılarak yeni bir sprint koşulur. En sonunda proje yayınlanır.  
 SCRUM Master, Müşteri (veya Müşteri Temsilcisi, müşteriyi müşteri gibi temsil eder.) ve ekip üyelerinden oluşan bir ekip yapısı vardır. SCRUM Master, yöneticiden çok ekibin lideridir ve ekibin problemlerini çözerek projenin başarıya ulaşmasına çabalar.

SCRUM’da Temel Kavramlar vardır; roller, toplantılar, bileşenler/araçlar.

**Roller**: SCRUM ekip üyelerine verilen rollerdir.  
 **Ürün Sahibi**: Projenin sahibidir.  
 **SCRUM Yöneticisi**: Takımın organizasyonunu sağlar. Geliştirme aşamasındaki sorunları çözer.  
 **SCRUM Takımı**: Genelde 5 ile 9 kişi arasında olabileceği gibi 3 kişilik takımlar veya 11 kişilik takımlar da olabilir. Hep beraber tek bir hedefi gerçekleştirmeye çalışırlar.

**Toplantılar**: Sprint Planlama, Sprint Gözden Geçirme, Günlük Scrum Toplantısı, Geçmiş Sprint Değerlendirmesi olmak üzere 4 ana toplantı vardır.   
 **Sprint Planlama**: Sprintten önce yapılır. Geniş kapsamlı olarak gereksinim listesi hazırlanır, dağıtım gereksinimleri belirlenir. Dağıtımlar için gereksinimlerin eşleştirilmesi yapılır. Dağıtımlar için takımlar belirlenir ve risk analizi yapılır. Maliyetler hesaplanır. Geliştirme araçları ve altyapısı onaylanır. Gözden geçirilir ve onay verilir.   
 **Sprint Gözden Geçirme**: Sprintten önce planlama toplantısı yapılır. Ürün sahibi ve geliştirme ekibi gereksinimleri gözden geçirir, hedef ve içerikleri belirler. İkinci aşamada gereksinimler önem sırasına göre sıralanır ve öncelikli olanlardan başlanarak Sprint Gereksinim Listesi oluşturulur. Scrum ekibi gereksinimlerin ne kadarının gerçekleştirilip gerçekleştirilemeyeceğine karar verir.  
 **Günlük Scrum Toplantısı**: Her günün başında 15 dakika civarında bir toplantı yapılır. *Dün ne yaptın?*, *bugün ne yapacaksın?*, *bir problemin var mı?* Bu soruların sonunda bir problem bulunursa toplantı sonrası Scrum Yöneticisi ile bu sorun görüşülür ve gerekirse tüm ekibe haber verilir. Günlük toplantılarda genelde herkese 2 dakika kadar konuşma süresi verilir. Bazı şirketlerde bu günlük toplantılara geç kalanlara eğlenceli cezalarda verilir.   
 **Geçmiş Sprint Değerlendirmesi (***Sprint Retrospective***)**: Sprint sonunda Scrum takımının kendilerini değerlendirdiği bir toplantıdır. Neleri, nasıl daha iyi yapabileceklerini ararlar. Nerelerde doğru nerelerde yanlış yapıldığına bakılır. Takımın sürekli gelişimi ve sıradaki sprinte hazırlanmasını sağlar.

**Bileşenler/Araçlar**: Product Backlog, Sprint Backlog, Burndown Chart olmak üzere 3 başlıktan oluşur.  
 **Ürün Gereksinim Dokümanı (***Product Backlog***)**: Proje boyunca gereken iş elemanlarının listesidir. Süreç boyunca yeni elemanlar eklenebilir veya çıkartılabilir. Genellikle **kullanıcı hikayelerinden** (*user story*) oluşur. Kullanıcı bakış açısından bakılır.  
 **Sprint (***Koşu***) Dokümanı (***Sprint Backlog***)**:

Mevcut koşulan Sprint için Product Backlog’tan alınan iş ve görevlerdir. Sadece takım tarafından değişiklik yapılabilir. Son ürün çıkmadan önce proje parçaları olarak geliştirilen çalışan küçük parçalar elde etmek için kullanılır.  
 **Sprint Kalan Zaman Grafiği (***Burndown Chart***)**: Sprint boyunca işlerin ne kadar kaldığını gösteren bir grafiktir. Küçük artma ve azalmalar bir sorun teşkil etmez ve genel olarak grafik aşağı doğru doğrusal olarak inmesi gerekir. Eğer grafik çok yavaş aşağı iniyorsa veya grafik çok hızlı aşağı iniyorsa Sprint Backlog’ı gözden geçirilmelidir. Eğer grafik doğrusala yakın veya yukarı doğru ilerliyorsa Scrum Yöneticisi’nin durumla ilgilenmesi gerekir.

## **Scrum günümüzde neden popüler?**

SCRUM, Sadece yazılım geliştirme değil, proje yönetimi alanlarında da kullanılabilecek esnek yapılı bir modeldir. SCRUM’da amaç büyük bir görevi daha küçük parçalara ayırarak takım odaklı bir şekilde küçük parçaları tamamlayarak ilerlenen ve en sonunda son ürüne ulaşılan bir model. İşlerin yapılmasını planlama, daha sonradan eklenebilecek yeni gereksinimlere açık olması, takım odaklı olması ve müşterilere ürün geliştirme aşamasındayken belirli adımlarla da ürünün küçük parçalarını sunarak ilerleyişi göstermeye yarıyor.

Kısaca; *esnek yapısı*, *süreç sırasında değişikliklere uğrayabilmesi*, *takım odaklı olması*, *müşteriyi sürecin içine katması* ile Dünya genelinde kabul görmüş bir yaklaşımdır.

## **MVP (MİNİMUM VİABLE PRODUCT)**

Minimum Uygulanabilir (Faydalı) Ürün olarak adlandırılır. Son ürüne varana kadar benzer işlevliğe sahip daha basit ve minimum özelliklere sahip ürünler elde etmektir. Maliyeti az olduğu için erkenden pazara çıkarılabilir. Örnek olarak; günümüz modern bilgisayarlarına gelene kadar öncelikle sadece belirli işlemleri yapan makineler üretildi, daha sonra tüplü programlanabilir bilgisayarların üretimi, delikli kartlar ile programlanabilen bilgisayarlardan günümüz modern bilgisayarlarına geliş süreci.

#### Yararlanılan Kaynaklar

* <https://medium.com/@denizkilinc/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-temel-a%C5%9Famalar%C4%B1-software-development-life-cycle-core-processes-197a4b503696>
* <http://ybsansiklopedi.com/>
* <https://www.academia.edu/20189499/Yaz%C4%B1l%C4%B1m_Geli%C5%9Ftirme_Hayat_D%C3%B6ng%C3%BCs%C3%BC>
* <https://en.wikiquote.org/wiki/Dennis_Ritchie>