Mehmet Ege Çoban

190601016

**Yazılım Yaşam Döngü Modelleri ve**

**Çevik Yazılım Geliştirme Modelleri**

# Yazılım Yaşam Döngüsü Modellerine Giriş

Bir yazılım ürününün soyut bir fikirden somut bir ürün haline gelişini ve gelişimini içeren sürecin işleyişini belirleyen, yazılımcılara yazılımı oluştururken planlı bir şekilde ilerlemelerini sağlayan modellere Yazılım Yaşam Döngüsü Modelleri denir.

Yazılım ürünlerinin sahip olması beklenilen birkaç özelliği vardır;  
 **1- Güvenilirlik:** Yazılımın sıfır hatayı hedeflemesi ve en az hata ile çalışması istenir.  
 **2- Verim:** Yazılımın sistem kaynaklarını verimli ve sadece gerektiği kadarını kullanması istenir.  
 **3- Sonradan Bakım Yapılabilir:** Bir yazılımın çıkışından sonra bakımı yapılabilir şekilde olması ve bu yüzden temiz kodlanmış, düzgün belgelenmiş şekilde olması istenir.

Bir yazılım ürününün yukarıdaki özelliklere sahip olabilmesi için arka planda sıkı bir planlama gerektirdiğini unutmamak gerekir.

# Yazılım Yaşam Döngü Temelleri

Yazılım Yaşam Döngüsünün Temelinde şu aşamalar yatar:

## Gereksinim, Planlama ve Analiz:

Gereksinimlerin analiziyle başlanır. Burada müşteri ile etkili ve iyi bir iletişim kurulmalıdır ki Yazılım için gereksinimler düzgün ve net bir şekilde belirlenebilsin. Bazı durumlarda gözden kaçabilen gereksinimler olabileceği için bu adımda çok dikkatli ve net olunması gerekir.

Gereksinimler belirlendikten sonra personeller, donanımlar ve fizibilite raporları hazırlanır ve kabaca bir proje planı ortaya konur.

Sistemin işlevleri ve kesin gereksinimleri belirlendikten sonra dokümanlar oluşturulur. Dokümanlar yardımıyla müşterinin net anlaşılıp anlaşılmadığı ve oluşacak yazılım ürününün dokümante edilmiş fikri ortaya konmuş olur. Bu aşamanın sonunda bir şartname dokümanı da oluşturulur ve artık kesin bir proje planı ortaya konmuş olur.

Bu aşamalar sırasında yavaş yavaş dokümanları test etmeye yani ileride karşılaşılabilecek hataları tahmin etmeye ve düzeltmek için adımlar atmaya başlamalıyız.

## Tasarım:

Bu aşamada yazılım ürününün nasıl çalışacağıyla ilgili dokümanlar hazırlanır ve ürünün tasarımı ortaya konmuş olur. Buradaki dokümanların Analiz aşamasındaki dokümanlardan farkı şudur; Analiz’de Ne istendiği ile ilgili dokümanlar varken Tasarım’da yazılımın Nasıl işleyeceğiyle ilgili dokümanlar vardır.

Mimari Tasarım ve Ayrıntılı Tasarım olarak iki tür tasarım vardır. Mimari Tasarım (Üst Seviye ve Mimari Tasarım) kaba taslak plan ve modülleri içerirken Ayrıntılı Tasarım; Veritabanı tasarımları, algoritmalar, sınıf, aktivite, etkileşim diyagramlarını içerir.

Tasarım yapılırken **Soyutlama** (Abstraction) yöntemi kullanılarak bazı olaylar/nesneler görmezden gelinir ve yazılım ürününün en önemli kısımlarına odaklanılması sağlanır.

Tasarımda aynı zamanda **Modelleme** kullanılır. Statik Modelleme, yazılımın değişmeyen özelliklerini modellerken Dinamik Modelleme, yazılımın işleyişini modeller.

## Gerçekleştirim:

Artık kodlar yazılmaya başlanır. Kodlarda önemli olan şey yazılan kodların bir standartının olması ve temiz kodlanmasıdır. Kodumuz ne kadar temiz ve bir standarta sahipse ileride bize bakım aşamasında o kadar zaman tasarrufu sağlayacaktır. Aslında bakıma gelmeden de temiz kodlamanın önemini şu şekilde anlayabiliriz. Üstünden biraz zaman geçen kodlar temiz kodlanmamışsa birkaç hafta sonra dönüp bakıldığında anlaşılmaz bir hal almış olabilir ve anlaşılması uzun sürebilir.

Kodlama aşamasında ve bu aşamadan hemen sonra en önemli bir diğer şey TESTtir. Kodlarımızı test etmeliyiz fakat kendi yazdığımız kodda kendimiz hataları bulamayabiliriz bu yüzden ekipteki herkes birbirinin kodlarını ve yazılımın tamamını test etmelidir.

## Teslim ve Bakım:

Yazılım tamamlanıp sahaya çıkabilir olduğunda yazılımın yanında dökümanların da tamamen hazır olması gerekir. Yazılımımız tamamlandıktan sonra bakım aşaması başlar ve artık yazılımın iyileştirilmesi, hata düzeltmeleri gibi çeşitli bakım işlemleri yapılmaya başlanır. Teorik olarak bakım aşaması yazılımın yaşamı boyunca devam eder.

# Yazılım Yaşam Döngü Modelleri

## Gelişigüzel Model

Bu model aslında bir model değildir. Bu aslında yöntem ve yol belirtmez ve tamamen geliştiriciye bağlıdır. 1960’larda basit programlar için kullanılmıştır.

## Barok Modeli

1970’li yıllarda ortaya çıkan ve ileride bahsedilecek olan Şelale(Waterfall) Modele benzer şekilde olan bir modeldir.

Başlangıçtan sonuca doğru doğrusal ilerler ve aşamalar arasında geri dönüşlerin nasıl yapılacağıyla ilgili bir bilgi içermez. Aynı zamanda bu modelde diğer modellerden farklı olarak test aşaması bittikten sonra, yani en son, dokümantasyon yapılır.

## Şelale/Çağlayan (Waterfall) Modeli

En çok bilinen ve en temel modeldir. Baştan sona doğru doğrusal bir modeldir ancak aşamalar arasında geriye dönüşler yapılabilir.

İlk başta Gereksinimler belirlenir, Analiz edilir, Tasarım yapılır, Uygulama ve Geliştirim ile ürün sahaya çıkar. Her aşamada dökümantasyon ve test olur. Bir aşama bitmeden öbür aşamaya geçilemez. Bundan sonra her bakımda bütün aşamalara tekrar dönülebilir fakat bu da bir maliyete neden olmaktadır.

Bu modelde Gereksinim ve Tasarım aşamaları çok önemli ve en fazla zamanın harcanacağı yerlerdir çünkü temeli sağlam bir şekilde atılmadan kodlamaya başlanmamalıdır, yoksa hem maliyet açısından hem de çalışanların istek-isteksizliği açısından aşamalar arasında geriye dönüşlerin azaltılması gerekir.

Bu modelde hem müşteri ile iletişimin çok az olması hem de sonuca ulaşma zaman aldığı için günümüzde popülerliği giderek azalmış ve yazılımcıları mutsuz eden bir modeldir.

Şelale modeli günümüzde daha çok Hastahane sistemleri gibi her gün gelişip değişmesi gerekmeyen ve iyi dökümantasyon gerektiren sistemlerde kullanılmaya daha elverişlidir.

## Fıskiye Modeli

Şelale modelinden esinlenilerek oluşturulmuş bir modeldir. Şelale modelinde farkı döngüleridir. Koddan tasarıma, tasarımdan gereksinimlere dönüşlerin olduğu hatta çıkışını gerçekleştirdikten sonra tekrar testlere dönüşün olduğu bir modeldir.

Çıkışını gerçekleştiren yazılımdan sonra ekip 2’ye ayrılır. Bir kısım yeniliklerle ilgilenirken bir kısım bakımla ilgilenmeye başlar.

## V Modeli

Üretim ve Sınama(Test) olarak ikiye ayrılır. İlk başta Üretim bölümü gelir, Üretim; Planlama, İhtiyaçların Belirlenmesi, Üst Seviye Tasarım, Alt Seviye Tasarım ve Kodlama aşamalarından oluşur. Bundan sonra Sınama(Test) bölümü gelir; Birim Testleri, Entegrasyon Testleri, Kabul testleri ve Bakım.

Bu modelin 3 alt modeli vardır. Kullanıcı, Mimari ve Gerçekleştirim Modeli. Bu 3 modelde Üretim ve Sınama bölümlerinin belirli kısımlarını içinde barındırır.

Kullanıcı Modeli’nde geliştirme sürecine kullanıcı da dahil edilir ve gereksinimler, planlar ortaya konur. Planlar ve Gereksinimler Kabul Testi ile Test Edilir.   
Mimari Model’de; Üst Seviye Tasarım, Entegrasyon Testleri ile test edilir, Detaylı/Alt Seviye Tasarım, Birim Testleri ile test edilir.   
 Gerçekleştirim Modeli’nde kodlar yazılır ve kodlar test edilir.

İş tanımlarının kesin olduğu Bilgi Teknolojileri Projeleri için uygun bir modeldir.

## Spiral Model

Planlama, Risk Analizi, Üretim ve Kullanıcı Değerlendirmesi olarak 4 büyük parçaya ayrılır. Spiral modelde bu 4 büyük parça ana ürün oluşana kadar sürekli döner ve aralarda ara ürünler oluşturulur. Aynı zamanda şu ana kadar bahsettiğimiz modellerde olmayan Risk Analizi kısmında riskler ve seçenekler araştırılıp belirlenir. Yinelemeli artımsal bir yaklaşımı vardır.

Üretim süresi boyunca ara ürünlerin oluşması ve bunların kullanıcı tarafından değerlendirilmesi kullanıcıyı etkin kılar. Aynı zamanda ara ürünlerin oluşması ve ilerleyişin görünebilir olması hem planlamaları kolaylaştırır hem testlerin kısa sürelerle çok defa yapılmasını hem de yazılımcıların bu ilerleyiş karşısında işten kopmamasını sağlar.

Bunca iyi özelliğinin yanı sıra spiral model kısa/küçük projeler için uygun değildir. Çünkü projenin normalden daha uzun sürede çıkmasına neden olur.

## Artımsal Geliştirme Süreci Modeli

Her zamanki gibi bir planlama aşamasıyla başlar fakat bu modelde sistemi tek seferde teslim etmek yerine geliştirme ve teslim olarak iki parçaya bölünür ve her teslimde öncelik sırasına göre kullanıcı gereksinimlerinden yapılması istenen bir işlevsellik yapılmış olur, teslim edilir ve bu şekilde ilerler. Eğer değişiklik gerekiyorsa bir sonraki teslimde ele alınır.

Gereksinimlerin belirlenmesi, analiz yapılması, tasarım, gerçekleştirim ve test aşamaları ilk geliştirmeden son teslime kadar birbiri içinde dağılmış şekilde ilerlemektedir.

Spiral Modele benzer yanı her teslimin bir prototip sayılabilmesidir ve bu nedenle hem müşteri hem geliştirici ekip prototipler elde etmekten mutlu olurlar.

Her yazılım projesi bu modele uygun değildir. Eksik modüllerle ilerleyebilecek projeler için tasarlanmış bir modeldir.

## Kodla ve Düzelt Yaşam Döngü Modeli

Küçük projeler için önerilen bir modeldir. İlk başta bir sürüm geliştirilir ve istenilen şekle gelene kadar bu sürüm sürekli düzeltilerek ilerlenir. Dokümantasyon yapılmaz ve bu yüzden bakımı çok zordur. Küçük projeler dışında kullanılırsa pahalı bir yol olur.

# Çevik Yazılım Geliştirme

Yazılım Yaşam Döngü Modelleri genel olarak günümüz yazılımları için çokta uygun sayılmazlar. Bunun için Çevik Yazılım Geliştirme adı altında yeni modeller ve metotlar kullanılmaya başlanmıştır. Çevik Yazılım Geliştirme (Agile) aslında ortaya çıkmadan önce de aynı tarzda modeller kullanan geliştiriciler vardı ama hiç kimse bu işin adını koymamıştı.

Çevik (Agile) Yazılım Geliştirme Metotlarıdnan önce proje geliştirme sürecinin %30’u kodlamaktı. Kalan %70’i hızlandırılmasının gerektiği düşünülmeye başlandı. Bunu yapabilmek için müşteri/kullanıcıda yazılım sürecine dahil edilmeliydi. Artık doküman yerine ana hedef Yazılım ve Müşteri/Kullanıcıydı. Süreç ve Araçlar yerini birey ve etkileşimlere, belgeler yazılımlara, görüşmeler müşteri ilişkilerine, plan izlemeler değişikliğe açık olamaya evrildi.

Çevik (Agile) Yazılım Geliştirme’nin asıl amaçları hızlı, esnek, ucuz, hata oranı düşük, verimi yüksek ve sürdürülebilir geliştirmeyi destekler yazılımlar geliştirmeyi sağlamaktır.

Bu metotlar artımlı modele benzerlik gösteren bir özelliğe sahip, proje küçük parçalara bölünüp 2-4 hafta gibi sürelerde bu parçaları yapmayı ve sıradaki parçalara geçmeyi ön gören yani yinelemeli ve artımlı yeni bir metodu ortaya koydu.

Genelde Çevik Yazılım Geliştirme metodu maliyetleri düşürürken üretkenlik, yazılım kalitesi, ekip motivasyonu, kısa sürede müşteri memnuniyetini ve yazılım projelerinin başarısını arttırdı.

## Extreme Programlama (XP)

XP, bir Çevik Yazılım Geliştirme modelidir. XP başta İletişim, Basitlik, Geri Bildirim ve Cesaret olarak 4 temel değeri barındırır.

**İletişim:** Eğer yazılım ekibi ile kullanıcı/müşteri arasında iyi bir iletişim kurulabilirse projedeki sorunlar, hatalar önceden tespit edilebilir ve düzeltilebilirdir.

**Basitlik:** XP’ye göre önce proje için zorunlu işler yapılmalı ve en basit yöntemler kullanılmalıdır. Karmaşık çözümlerden, yöntemlerden uzak durulmalıdır.

**Geri Bildirim:** İletişimin yanında doğru ve etkili geri bildirimler de yazılımda ortaya çıkabilecek hataların erkenden keşfedilmesini ve düzeltilmesini sağlar. Mantıksal yapı içeren birim testler yapılarak sistem ile ilgili bilgiler elde edilir. Müşteri, yönetici ve diğer proje çalışanlarının görüş ve fikirleri burada önemli yer alır. Yinelemelerden (2-4 haftalık yinelemeler) elde edilen sürümler ile sistem kontrol edilir.

**Cesaret:** En az İletişim, Basitlik ve Geri Bildirim kadar önemli olan bir diğer temel değer Cesarettir. Yazılım ekibi en zor yazılım projesinde bile yılmadan ve başarısızlıktan korkmadan devam etmeli gerekirse projenin bir kısmını çöpe atarak o bölümü daha iyi bir şekilde yeniden yazabilmelidir.

XP’de planlama aşamasında proje ile ilgili hikaye kurulur. Müşteri veya müşteri temsilcisi, müşteri tarafıyla ilgili fikirler ve bilgiler verirken teknik taraftaki yazılımcılar teknik kısımlarla ilgili tahminler sunar. Bir önem sırası belirlenir. Plandaki detaylar kesinleşene kadar bu aşama sürer. Bu aşamada en önemli iki şey müşterinin neyi önemsediği (kalite yönetimi) ve projenin yapımının bitmesidir (şu şartları sağlıyorsa artık tamamdır). (Gerekli ama müşterinin isteği dışındaki parçalar hikayede yer almaz. Örn: Güvenlik duvarı)

Hikaye aşamasında hikaye kartları kullanılabilir.

XP esnek kod geliştirme için 12 farklı pratiği vardır;  
 **Planlama Oyunu**: Bir projenin her yinelemesinin ne kadar süreceğini tahmin etmeye çalışıldığı bir toplantı oyunudur. Bu oyunda müşteri de yazılım ekibi de aynı ortamdadır. Böylece proje için öncelikli yinelemeler ve yaklaşık bir süre ortaya konabilir. Aynı şekilde burada 2 ekip yarıştırılarak normalden daha hızlı bir şekilde proje parçası/ları elde edilebilir ve bunları rakip ekip(ler)e kontrol ettirerek test işini de hızlandırabiliriz.

**Müşterinin Ekibe Dahil Olması:** Müşteri veya onu temsilen bir müşteri temsilcisinin de yazılım ekibinin yanında aynı ortamda bulunması gerekir ki iletişim ve geri bildirimler hızlı gerçekleşebilisin.

**Testin Önemi:** Asıl proje/proje parçası yazılmadan önce bir test programı yazılmalıdır. Sorunların daha erken tespiti ve daha güvenilir bir yazılım yazmak için bu adım önemlidir.

**Basit Tasarım:** Müşterinin gereksinimlerini karşılayabilecek basit tasarımın yapılmasıdır. Amaç kısa sürede anlaşılması, değiştirilebilmesi ve yönetilebilmesi kolay yazılım geliştirmektir.

**Çiftli Programlama (Pair Programming):** İki yazılımcının birbirlerini denetleyerek, birbirlerine yeni şeyler öğreterek, standartlara bağlı kalınarak iki yazılımcının da birbirine bir şeyler katması beklenir. İki yazılımcının tek bilgisayar üzerinde geliştirme yapmasıdır. Çoğu zaman yazılımcıların projeye odaklanmasını sağladığı için projeye hız kazandırabilir. İki yazılımcı arasında güven oluşur. Genelde işe yeni başlayan programcılara standartları öğretmek için uygulanan bir metottur.

**Sürekli Entegrasyon (Continuous Integration):** Geliştirilen yazılımın sisteme olabildiğince hızlı güncellenerek eklenmesidir. Burada amaç hataları erken tespit etmenin yanı sıra bir yazılımcının yaptığı değişiklikleri de görebilmektir.

**Kısa Aralıklı Sürümler:** Proje parçalara bölünür ve bu parçalara belirli zaman dilimleri verilir (2-4 hafta arası). Bu parçalar bu zaman dilimlerinde yazılarak teslim edilir. Artımlı modele benzer yanı da budur. Parçalar tamamlanarak en sonda tam bir ürün oluşur.

**Yeniden Yapılandırma (Refactoring):** Temelde amaç, zaman içinde kodun değiştirilerek veya eklenerek daha optimize ve daha düzgün yazılmasıdır. Bu sırada müşteriyi memnun etmek için de kod geliştirilebilir. Burada önemli bir detayda şudur, o kod başka kodları da etkileyecekse yani diğer kodlarla içiçe geçtiyse kodun refactor edilmesi zorlaşır, sisteme ilerleyen zamanlarda sorun oluşturabilir, buna dikkat edilmelidir.

**Kodu Ortak Sahiplenmek:** Kodu yazan ekibin yazılan kodların tamamını ortak olarak sahiplenmesi durumudur. Bu ekibin birbiri ile bağlarını güçlendirebilir. Aynı zamanda bir geliştirici başka bir geliştiricinin kodunu daha iyi bir şekle sokmak için düzenlemeler yapabilir.

**Metafor / Benzetme:** Yazılımın tamamı veya parçalarını daha iyi anlatabilmek için başka sistemlere veya başka nesnelere benzetilmesidir. İletişimi daha etkili kurmak için kullanılabilir.

**Kodlama Standardı:** Kodlarda ekiplerin bir standartları olmalıdır. Hizalama, genel isimlendirme, yorum satırları ve açıklamalar, sınıf tanımlama ve kullanma, dizi kullanma, özel durum ve hata yönetimi, genel önerilerin birleşmesidir. Amaç kodun karmaşıklığını azaltıp anlaşılabilirliğini arttırmaktır.

**Haftalık 40 Saat İş:** 1 haftada işler toplamda 40 saatlik zamanda bitirilir. Fazla mesai olmamasına özen gösterilmelidir. Amaç konsantrasyon ve verimin düşmemesini ve tam olarak kullanılmasını sağlamaktır.

Bu 12 pratik XP’nin özel pratikleridir.

## SCRUM

Çevik Yazılım Geliştirme Modellerinin en çok tercih edilenlerinden birisi de SCRUM’dır. Yazılım dışında da kullanabilir bir yönetim yaklaşımıdır. Bir projeyi küçük birimlere yani sprintlere bölerek adım adım yazılım geliştirmeyi amaçlar. Küçük parçalara bölündüğü için projeye yön vermek daha kolay ve hızlıdır. Günümüzde Scrum’ın bu hızı bir çok geliştirici ekibini ve şirketi etkilemiştir.

Günlük 15er dakikalık toplantılar içeren ve 2-4 hafta arasında süren sprintlere sahip bir modeldir. Gereksinimlerin kesin olarak tanımlanamadığı projelerde de kullanılmaya uygun bir yöntemdir. Gereksinim toplanması, tasarım, programlama ve testin neredeyse içiçe geçtiği bir yöntemdir.

Scrum’da müşteri veya müşteri temsilcisi ekiple içiçedir ki iletişim yüksek seviyede olsun.

Scrum’da aynı zamanda Scrum Master adında bir çeşit takım lideri vardır. Asıl amacı takımın sorunlarını çözmek ve projenin ilerleyişini sağlamaktır. Ekibe hizmet eder.

Scrum’da birden fazla takıma aynı anda dahil olunabilir veya daha sonradan takım değiştirmeler yaşanabilir.

Scrum’da 3 temel kavram vardır;  
**Roller**  
-> Ürün Sahibi (Product Owner)   
-> Scrum Yöneticisi (Scrum Master): Takımın Scrum’a adapte eder, Scrum’ın temel değerleri ve kurallarına bağlı kalmalarını sağlar.  
-> Scrum Takımı (Scrum Team): Devamlı iletişim halindeki ve tek hedefe odaklanmış 5-10 kişilik ekip.

**Toplantılar**  
-> Sprint Planlama (Sprint Planning): Gereksinim listesi çıkarılır ve uygun bir dağıtım gereksinimleri belirlenir. Dağıtımlar için takımlar belirlenir. Risk değerlendirmesi ve kontrolleri belirlenir. Gözden geçirme yapılır ve olası gereksinim değişiklikleri belirlenir. Geliştirme araçları ve altyapısı onaylanır. Dağıtım, geliştirme ve pazarlama maliyetleri hesaplanır. Bir kez daha gözden geçirme ve onaylama yapılır.

-> Sprint Gözden Geçirme (Sprint Review): Her sprint başlangıcında bir planlama toplantısı yapılır. Ürün sahibi ve ekip, gereksinim listesini gözden geçirir ve hedefler ile içerikler belirlenir. İkinci kısıma geçildiğinde gereksinim listesinin en üstünden sprint sonucunda gerçekleştirilecek gereksinimlerden oluşan Sprint Gereksinim Listesi oluşturulur ve **bu Scrum’ın anahtar uygulamasıdır.** Scrum ekibi bu gereksinimlerin ne kadarını yapabileceklerini belirler ve taahhüt ederler.

-> Günlük Scrum Toplantısı (Daily Scrum): Bu uygulama da Scrum’ın anahtar uygulamalarındandır. Günlük olarak genelde sabahları 15-30dklık küçük bir toplantı yapılır ve tüm ekip katılır. İlerleyiş ve engelleri görmek için önemlidir. Dün ne yaptın?, Bugün ne yapacaksın? Engeller ve karşılaştığın sorunlarneler? Gibi sorulara cevap verilir.

-> Geçmiş Sprint Değerlendirmesi (Sprint Retrospective): Scrum takımının Sprint sonunda gerçekleştirdikleri kendilerini değerlendirme ve sürekli gelişimi devam ettirme toplantısıdır. Neyi daha iyi yapabiliriz? Nasıl daha iyi olabiliriz? Neyi yanlış yaptık? Gibi sorulara cevap aranır. Bir sonraki Sprint’e daha iyi hazırlanmayı sağlayan bir toplantıdır.

**Bileşenler/Araçlar (Artifacts)**  
-> Ürün Gereksinim Dökümanı (Product Backlog): Ürünle ilgili özelliklerin olduğu, önceliklendirilmiş yüksek seviyeli gereksinim listesidir. Zamanla ekleme ve küçük parçalara ayrıştırma yapılırken aynı zamanda anlamsız bazı gereksinimler silinebilir ve her gün değişiklik gösterebilir. Genelde kullanıcı hikayelerinden ve kullanıcı bakış açısından yararlanılır.

-> Sprint Dokümanı (Sprint Backlog): Product Backlog’taki iş ve görevleri kapsar. Amaç bu görev ve işleri sprint sonuna yetiştirmek ve tamamlamaktır. Sadece takımı igilendirir. Her sprint sonunda küçük teslimler yapılır.

-> Sprint Kalan Zaman Grafiği (Burndown Chart): İşin ne kadarının yapılıp ne kadarının kaldığını gösteren bir grafiktir. Aşağı doğru inen bir grafik olması gerekirken bazen düz veya yukarı doğru ilerler. Bu gibi durumlarda işin tekrar planlanması veya iptal edilmesi gerekebilir.

Scrum, takım odaklı bir Çevik Yazılım Geliştirme modelidir. Kısa zamanlarda küçük parçalar halinde yazılımı tamamlamayı amaçlar. Dokümanlar ikinci plandayken iletişim ve takım birinci plandadır.

## Scrum GÜNÜMÜZDE neden popüler?

Scrum, Çevik Yazılım Geliştirme’de en hızlı ve ekip arasında iletişimin en yüksek olduğu, projenin parçalara bölünerek yapılmasından kaynaklı olarak projenin değiştirilmesinde ve tekrar yön verilmesinin kolay olduğu, herhangi bir hatanın oluşumunda bunun giderilmesinin daha kolay olduğu ve proje kodlarınının içiçe girmesinin çok daha az olması sonucunda modülerlik kazanması, yazılım şirketlerinin ve ekiplerinin Scrum’a yönelmesinin en önemli sebeplerindendir.

## MVP (Minimum Viable Product)

Türkçe olarak Minimum Uygulanabilir (Faydalı) Üründür. Olayı, ürünün önemli, işe yarayan ama minimum özelliklerdeki basit, son ürünün en temel özelliklerini içerek yapıda olmasıdır.

Maliyeti az, erkenden pazara çıkabilir, yinelemeli versiyonlar ile ilerleyen bir özelliği vardır.

Örneğin: 3 boyutlu gerçekçi görünen bir oyunu yapmak için önce 3 boyutlu modeller sonra fizik sistemi sonra grafik işleme vs. şeklinde ilerlemek yerine yazı tabanlı oyun, 2 bouytlu oyun ve daha sonra 3 boyutlu oyun şeklinde ilerlemek.

### 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>