YAZILIM YAŞAM DÖNGÜ SÜREÇLERİ VE ÇEVİK YAZILIM GELİŞTİRME

## Necibe CAN

### ÖZET

Bu yazıda yazılım yaşam döngüsünün açıklaması ve modellerinden bahsedildi. Modelleri tek tek açıklandı ve avantajlarından ve dezavantajlarından bahsedildi. Bu sayede karşılaştırılmaları yapıldı. Günümüzde daha çok kullanılan ve gelişmiş olan çevik yazılım geliştirmeden bahsedildi. Modelleri açıklandı ve en çok kullanılan scrum modelinin üzerinde duruldu. Neden daha sık kullanıldığı anlatılmaya çalışıldı.

İçindekiler

[Necibe CAN 1](#_Toc35278780)

[ÖZET 1](#_Toc35278781)

[Şekil Tablosu 3](#_Toc35278782)

[1.GİRİŞ 4](#_Toc35278783)

[2.YAZILIM YAŞAM DÖNGÜSÜ 4](#_Toc35278784)

[2.1. Planlama 5](#_Toc35278785)

[2.2. Tanımlama 5](#_Toc35278786)

[2.3. Tasarım 5](#_Toc35278787)

[2.4. Geliştirme 5](#_Toc35278788)

[2.5. Entegrasyon ve Testler 5](#_Toc35278789)

[2.6. Uygulama 5](#_Toc35278790)

[2.7. 6](#_Toc35278791)

[3. SDLC YAKLAŞIMININ DEZAVANTAJLARI 6](#_Toc35278792)

[4. YAZILIM SÜREÇ MODELLERİ 6](#_Toc35278793)

[4.1 Kodla ve Düzelt (Code and Fix) 6](#_Toc35278794)

[4.1.1. Avantajları 6](#_Toc35278795)

[4.1.2. Dezavantajları 7](#_Toc35278796)

[4.2 Barok Modeli 7](#_Toc35278797)

[4.3 Şelale (Çağlayan) Modeli 7](#_Toc35278798)

[4.3.1 Avantajları 7](#_Toc35278799)

[4.3.2 Dezavantajları 7](#_Toc35278800)

[4.4. V Süreç Modeli 8](#_Toc35278801)

[4.4.1 Avantajları 8](#_Toc35278802)

[4.4.2 Dezavantajları 8](#_Toc35278803)

[4.5. Evrimsel Geliştirme Modeli 8](#_Toc35278804)

[4.5.1. Avantajları 9](#_Toc35278805)

[4.5.2. Dezavantajları 9](#_Toc35278806)

[4.6 Helezonik (Spiral) Model 9](#_Toc35278807)

[4.6.1 Avantajları 10](#_Toc35278808)

[4.6.2 Dezavantajları 10](#_Toc35278809)

[5.ÇEVİK YAZILIM GELİŞTİRME 10](#_Toc35278810)

[5.1. Extreme Programming (XP) 11](#_Toc35278811)

[5.2. Scrum 12](#_Toc35278812)

[5.2.1 Scrum Neden Bu Kadar Popüler? 13](#_Toc35278813)

[6. Sonuç 14](#_Toc35278814)

# Şekil Tablosu

[Şekil 1 Yazılım Yaşam Döngüsü 3](#_Toc34672624)

[Şekil 2 Kodla ve Düzelt 5](file:///C:\Users\lenova\Desktop\Makale%20Araştırması\YAZILIM%20YAŞAM%20DÖNGÜSÜ%20(Software%20Development%20Life).docx#_Toc34672625)

[Şekil 3 Şelale Modeli 6](file:///C:\Users\lenova\Desktop\Makale%20Araştırması\YAZILIM%20YAŞAM%20DÖNGÜSÜ%20(Software%20Development%20Life).docx#_Toc34672626)

[Şekil 4 V Süreç Modeli 7](#_Toc34672627)

[Şekil 5 Spiral Model 8](#_Toc34672628)

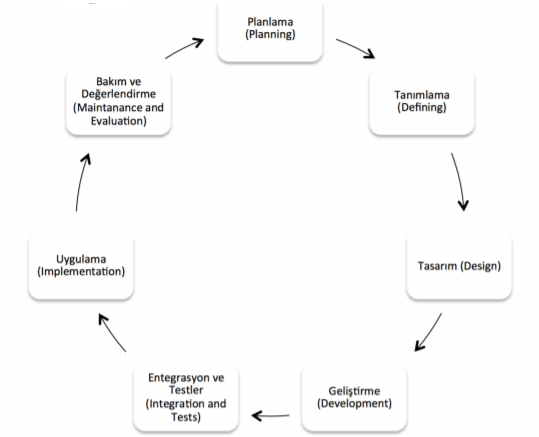
[Şekil 6 XP Modeli 10](#_Toc34672629)

# 1.GİRİŞ

Yazılım, bugünün dünyasında çok geniş bir uygulama alanına sahiptir. Hemen hemen her iş alanında yazılıma ihtiyaç duyulmakta. Bu yüzden kaliteli ve kullanışlı bir yazılımı üretebilmek büyük bir öneme sahiptir. Başarılı bir yazılım geliştirebilmek için yazılım yaşam döngüsü basamakları dikkate alınarak bir çalışma yapılmalıdır. Tüm basamakların dikkate alındığı ve verimli gözüken bir çalışma yine de başarısız olabilir. Bu yüzden en önemli noktalardan biri de hangi yazılım yaşam döngüsü modelinin yapılan iş için doğru olduğuna karar verebilmektir. Bu kararı verdikten sonra dikkatlice, seçilen modelin adımları uygulamaya geçmelidir. Bu modellemeler yazılım ürününü kalite, zaman, bütçe ve gereksinimleri yerine getirme gibi yönlerden önemli derecede etkilemektedir. Yazılım yaşam döngüsünde önemli noktalardan biride oluşan hataların hangi adımda yapıldığını anlayabilme ve geri dönüş yapılabilmesidir. Hatanın hangi adımda olduğunu anlayabilmek ve o adıma geri dönüş sağlayıp hatayı düzeltmek ürünün gelecekte daha büyük sorunlara yol açmasını engeller. Adımlar arasında geri dönüş sağlanamaması, sorunun hangi adımda olduğunu anlayamamak ürün maliyetini ve çıkış süresini artırır. Bu kullanılan yazılım yaşam döngü modelinin yeterli olmamasından kaynaklanabilir. Tam da bu sebepten yazılım yaşam modelleri günümüze gelene kadar gelişerek bir çok modele ayrılmıştır.

# 2.YAZILIM YAŞAM DÖNGÜSÜ

Yazılım aslında bir üründür ve her ürünün bir yaşam süreci vardır. Her ürün fikir aşamasından başlayıp ürün olma yolundan ilerlerken bazı adımlardan geçer. Bu adımlar her ürün için farklı olabilir. Hatta aynı ürün için farklı adımlar da seçilebilir. Yazılım yaşam döngüsü (SDLC) adımları aşağıdaki şekilde sıralanabilir.

1. Planlama

2. Tanımlama

3. Tasarım

4. Geliştirme

5. Entegrasyon ve testler

6. Uygulama

7. Bakım

Yukarıdaki liste çoğu kaynakta bulunan ortak bir liste olsa da bazı kaynaklar farklı adımlar ekleyip çıkarmış olabilir. Bazı kaynaklarda 4-5 adımdan bahsedilirken bazı kaynaklarda 10 adım üstüne bile çıkabilir.

Şekil Yazılım Yaşam Döngüsü

Her ne kadar farklı kaynaklarda farklı kullanımlar olsa da anlatılmak istenilen şey aynıdır. Her ürünün hatta canlıların kendi arasında ve hayatında bir döngüsü vardır. Asında bu döngüleri küçüklüğümüzden beri hep görüyoruz ve öğreniyoruz. Su buharlaşır, bulut olur ve yağmur yağar bu döngü sürekli tekrarlanır. Canlılar doğar büyür ve neslinin devamını sağlar daha sonra ölür. Öldükten sonra toprak olur ve başka canlıların yaşamına katkı sağlar. Ürünlerinde planlama aşamasından tamamlanma aşamasına kadar bu tarz döngüleri vardır. Bu döngüdeki adımları iyi anlamak ve iyi uygulamaya geçirmek ürünün kalitesini her yönden etkiler. Bu döngülerde gördüğümüz bir diğer şey sürekli başa dönüyor olmasıdır. Yani tüm aşamalar tamamlandıktan sonra başa dönülür ve tekrarlanarak devam eder. Bunu piyasadaki ürünlerin bir zaman sonra ömürlerinin bitmesi ve yeni yazılım ürünlerinin ortaya çıkmasıyla açıklayabiliriz. Aslında tüm yazılım ürünlerinin de bir ömrü vardır. Yeni ihtiyaçlar, yeterli olamayan ürünler nedeniyle bu ihtiyaçların karşılanması için bu adımlar tekrarlanarak bir yeni ürün elde edilir. Şimdi bu yazılım yaşam döngüsünün adımlarını daha ayrıntılı inceleyelim, avantajlarını ve dezavantajlarını tartışalım.

## 2.1. Planlama

Bu aşama yazılım yaşam döngüsünün (SDLC) ilk adımıdır. İlk olarak ne istenildiğini anlama ve planlama yapılmalıdır. Fikrin ortaya atıldığı ve tartışıldığı adımdır. Bu döngünün nasıl kontrol edileceği ve neler yapılacağı tartışılmalı, ortamdaki ihtiyaçların belirlenmesi yapılmalıdır. Aslında bu adım tüm adımların başlangıcı niteliğindedir. Bu yüzden oldukça önemli bir adımdır. Geliştirilecek olan sistemin hangi hizmetleri sağlayacağı ve bu hizmetlerin durumlarını (zaman , güvenlik vb.) ve dış dünyaya nasıl açılacağını bu adımda belirleriz. Projenin maliyeti, personeller, kurumun mevcut bilişim seviyesi, uzun vadede sistemin bakımı gibi bir çok parametre bu adımda incelenir.

## 2.2. Tanımlama

Gereksinimler belirlenir. Buna piyasa araştırması yapılarak başlanılabilir. Bu gereksinimler yazılım gereksinimleri şartnamesi adlı bir belgede kayıt altına alınır. Bu belge ilerideki geliştirme sürecinde sorunları çözecek olan ilgili bütün taraflar arasında yapılan bir anlaşmadır. Bu şart namenin önemli olduğu IEEE ve Birleşik Devletler Savunma Bakanlığı tarafından ispat edilmiştir. Planlanan projenin tanımlamaları yapılır. İstenilen fikrin ne olduğu ve temel tanımları üzerine tartışılır.Bu adım analiz aşaması olarak da adlandırılabilir.

## 2.3. Tasarım

Yazılımımızın tasarımları yapılır. Projeler çizilir, planlamaya göre tasarımlar çizilir. Kararlar verilir bir sonraki aşama için karar verilecek bir durum kalmamalıdır. Yazılım kullanıcısının ihtiyaçlarını derleme ve analiz sürecini kapsar. Yazılım sisteminin iç yapısı bu aşamada belirlenir. Tasarım aşamasının sonucunda yazılacak yazılımın tüm detayları belirlenmiş olur. Örneğin yazılım ürünün ekranı, ekranlar arası nasıl geçiş yapılacağı gibi olaylar belirlenir. Yazılım ürünü değil de bir ofis binası inşaat edecek olsaydık o zaman tasarım aşamasında ofisin gereksinimlere cevap veren bir taslağı ve projesi çizilirdi. Yazılım mühendisliğinde bu taslakları oluşturabilmek için bir çok yola başvurulmuştur. En çok kullanılan yollardan biri UML’dir.

## 2.4. Geliştirme

Bu aşama kodlamaya başlanıldığı aşamadır. Bir önceki aşamalarda verilen kararlara göre kodlama yapılmaya başlanır. Başka bir örnekle projeleri tasarımı yapılan binanın yapımına başlanması aşamasıdır. Yani projenin gerçekleştirilmesine başlanır.

## 2.5. Entegrasyon ve Testler

Geçmişte bu aşama yazılımın hatalarını bulup giderme olarak görülmüştür fakat günümüzde bu tanım yeterli bulunmamaktadır. Bu aşamada ürün artık gerçek hayatın içinde yerini alır. Eğer bir bağlantısı bulunuyorsa bağlantılarıyla entegre edilir. Aslında bu test aşaması her adımın sonunda uygulanması gereken bir aşamadır. Bu adımı yazılım yaşam döngüsünün tamamına yaymak en doğru seçim olacaktır. Ne yazık ki modern dünyamızda hala büyük hatalar oluşmaktadır. Bu hataların çoğu fark edilemiyor ya da bu hatalar çok büyük aksaklıklara neden olabiliyor. Bu demektir ki hala araştırılması ve düzeltilmesi gereken bir çok şey var

## 2.6. Uygulama

Ürün artık kullanılmaya başlamıştır. Kullanım sonucunda sorunlar çıkabilir. Bu sorunlar için çözüm aranır. Çözümler tartışılır ve uygulamaya alınır ve bu aşamadan sonra bakım aşamasına geçilir.

2.7. Bakım

Projenin bakımı yapılırken farklı aşamalardan geçer yeni tanımlar, tasarımlar ortaya çıkabilir. Bu döngü sürekli halde devam eder. Yazılım ürünü yaşayan bir organizmadır. Bir gün gelir ve ömrü biter. İşte tam da burada yazılım yaşam döngüsünü anlamamız gerekir. Hiç durmayan ve sürekli devam eden bu döngüye yazılım yaşam döngüsü diyoruz. Yazılım yaşam döngüsü yapılan bu adımların nasıl yapılacağını ve düzenlemesini yapar.

# 3. SDLC YAKLAŞIMININ DEZAVANTAJLARI

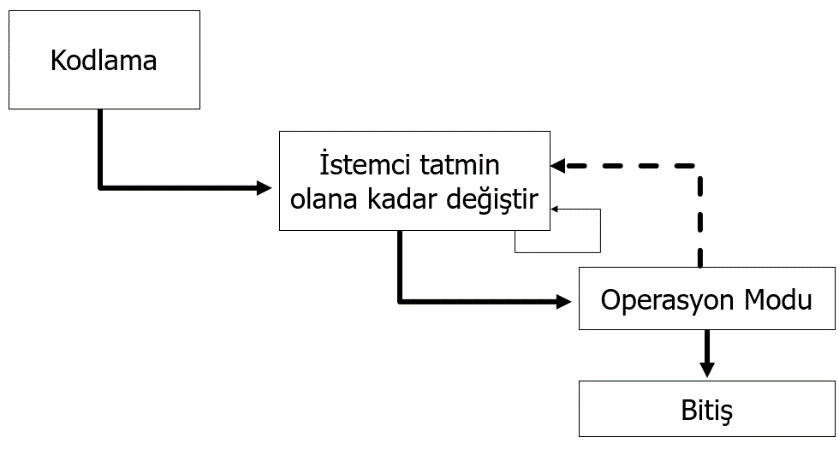
Buraya kadar anlattıklarımızda SDLC’nin bir çok avantajlı tarafına değindik. Şimdide bazı eksikliklerinden ve dezavantajlarından bahsedelim. Öncelikle uzmanlık gerektirir. Yönetimin tecrübeli olması beklenmektedir. Küçük sistemler için uygun değildir. Bir yazılımcı hesap makinesini 3-4 saat içinde yapabilmektedir. Fakat SDLC adımlarını takip etmek istemesi ona dezavantaj olarak dönecektir. Zaman artışını sağlar buda maliyet artışına sebep olur. Diğer bir dezavantajı ise test aşamasında bahsettiğimiz gibi projenin sonunda test yapılmasını bekler. Fakat her adımın sonunda müşteriyle iletişim kurularak bir test gerçeklemesi çok daha doğru bir yaklaşımdır.

# 4. YAZILIM SÜREÇ MODELLERİ

Aslında günümüz dünyasında çokça kullanılan bir yol olsa da kendiside bir yaşam döngüsüne sahiptir. Günümüz dünyası giderek modernleştiği gibi SDLC’yi de modernleştirmiş ve bir çok dala ayırmıştır. Bunlardan bazıları şunlardır:

* Kodla ve Düzelt (Code and Fix)
* Barok Modeli
* Şelale (Çağlayan) Modeli
* V Süreç Modeli
* Evrimsel Geliştirme Modeli
* Helezonik (Spiral) Model

## 4.1 Kodla ve Düzelt (Code and Fix)

 Genellikle öğrenciler tarafından kullanılan. Dokümantasyon gerektirmeyen projelerde kullanılır. Ve sonucunda coğunlukla spagetti kod elde edilir.

Şekil Kodla ve Düzelt

### 4.1.1. Avantajları

1. Herhangi bir plana ihtiyaç duymaz
2. Uzun aşamalardan geçilmediği için kıza zamanda proje bitimi sağlanır.
3. Kısa vadeli işler için uygundur.
4. Herkesin kullanabildiği bir modeldir. Uzmanlık gerektirmez.

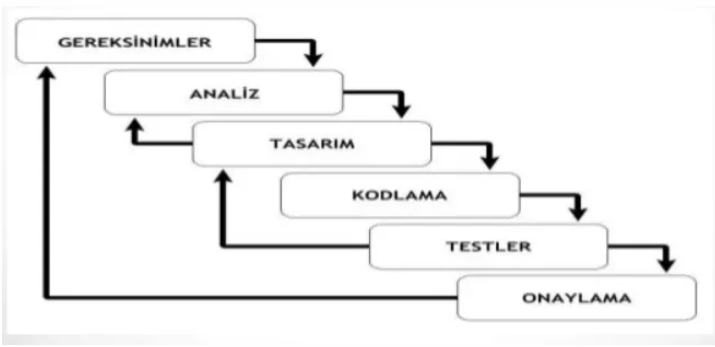
### Dezavantajları

1. Kontrollü değildir
2. Bitiş süresi belli değildir
3. Esnek değildir, hataların değişmesi zordur.
4. Kodları düzeltmek maliyetli olabilir.
5. Her ihtiyaca karşılık vermeyebilir.

## 4.2 Barok Modeli

Barok modeli 70’li yıllarda kullanılan ilkel bir modeldir. Bu modelde belgeleme atrı bir süreç olarak ele alınır. Gerçekleştirim döngüsüne önem verir. Adımlar arasında doğrusal bir ilerleme vardır ve adımlar arasında geri dönüşü sağlamaz. Günümüzde tercih edilen bir model değildir.

## 4.3 Şelale (Çağlayan) Modeli

 Şelale modeli genellikle ilk anlatılan modellerden birisidir. Şelale modeli statik bir modeldir. Örneğin bir bina yapımını ele alırsak. Beklentileri karşılayan bir bina yapılır ve kullanılmaya başlar. Bina kullanılmaya başlandıktan sonra sıkça bir değişiklik gerçekleşmez. Böyle bir değişimde beklenmez. Her gün farklı bir talepte bulunulmaz. Fakat yazılım ürünleri için işler daha dinamiktir. Her gün farklı bir istekte bulunulabilir. Talepler her geçen gün artarak devam edebilir. Bu ürünler dinamiktir. Oysa ki bir bina yıllarca değişmeden kullanılabilir. Şelale modeli bu tarz projeler için daha uygundur. Şelale modeli yazılım ürünleri için kötü bir yaklaşımdır. Fakat yine de basitliği ve diğer modellemelere temel olması açısından önemlidir.

Şekil Şelale Modeli

Şelale modeli geçmişte oldukça popüler olan bir modeldir. Geleneksel yazılım geliştirme modeli olarak da bilinmektedir. İşler aşama aşama yapılır. Bir aşamanın bitimi için o aşamanın dokümante edilmesi beklenmektedir. Bir aşama bitmeden diğerine geçiş sağlanamaz. Barok ile kıyaslanacak olursa bu modelde belgeleme daha kısa bir süreç olarak ele alınır. Ayrı bir süreç yerine döngünün doğal bir sonucu olarak görülür. Ayrıca bu modelde adımların arasında ki geri dönüşün nasıl olacağı da açıklanmıştır.

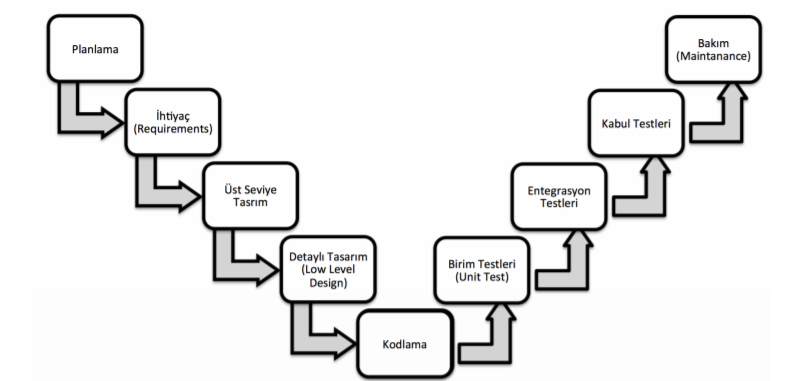
### 4.3.1 Avantajları

1. Kullanımı ve anlaması basittir.
2. Yönetimi kolaydır
3. Kısa vadede tamamlanacak ve gereksinimleri az olan projeler için uygundur.

### 4.3.2 Dezavantajları

1. Nesne yönelimli karmaşık projeler için uygun değildir.
2. Uzun vadeli projeler için uygun değildir.
3. Bir çok geri dönüş olduğu için iki üç adımlık geri dönüşler büyük maliyetlere yol açabilir.
4. Değişime ve gelişime açık bir model değildir. Bu yüzden müşteri memnuniyeti sağlamak zordur.
5. Yapılmak istenilen her değişikliğin maliyeti oldukça fazla olabilir.
6. Eğer gereksinimler iyi anlaşılmasa ürün tamamlandıktan sonra tamamen iptal olabilir.

## 4.4. V Süreç Modeli



Şekil V Süreç Modeli

Süreç V şeklini oluşturduğu için V süreç modeli ismini almıştır. Sol taraf üretim sağ taraf test adımlarını gösterir. Aslında şelale modeline benzerlik gösterir. Fakat şöyle bir farkı bulunur. Üst seviye tasarımların daha sonra detaylı tasarımları yapılır. Yani daha alt seviyelerin açıldığı girdilerinin çıktılarının ve beklentilerinin yazıldığı söylenebilir. Buraya kadar şelale modeline benzerlik gösterir. Aşağı doğru doğrusal ilerleyen bir model söz konusudur. V modeli farklı olarak bu inişin sonunda kodlamadan sonra yukarı doğru test aşamalarını bulundurur. Her üretilen modülün testi yapılır. Daha sonra modüllerin birbiri arasındaki uyum için entegrasyon testleri yapılır. Test aşamasından müşteri memnuniyeti sorgulanır ve eğer bir sorun çıkmazsa bakım aşaması başlar.

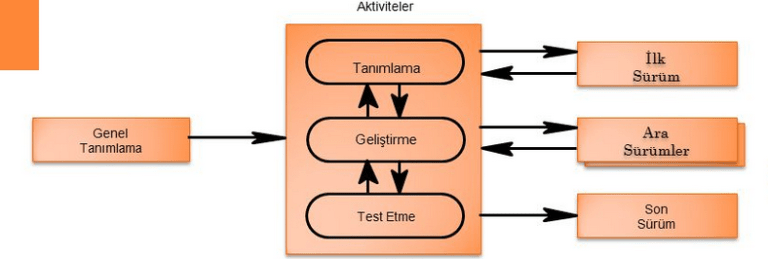
### 4.4.1 Avantajları

1. Proje yönetimi kolaydır
2. Kullanımı kolaydır
3. Projenin amacını doğrular.
4. Yapılan işler doğrulanarak devam eder.

### 4.4.2 Dezavantajları

1. Risk çözümleme ile ilgili aktiveteleri içermez.
2. Aşamalarda tekrarlama yoktur.

## 4.5. Evrimsel Geliştirme Modeli



Şekil Evrimsel Geliştirme Modeli

Model tam olarak “ne istediğimi bilmiyorum ama görsem tanırım” modelidir. İlerleyişi diğer modellere göre daha yavaştır. Başarısı ilk geçirdiği evrimin başarısına bağlıdır.

### 4.5.1. Avantajları

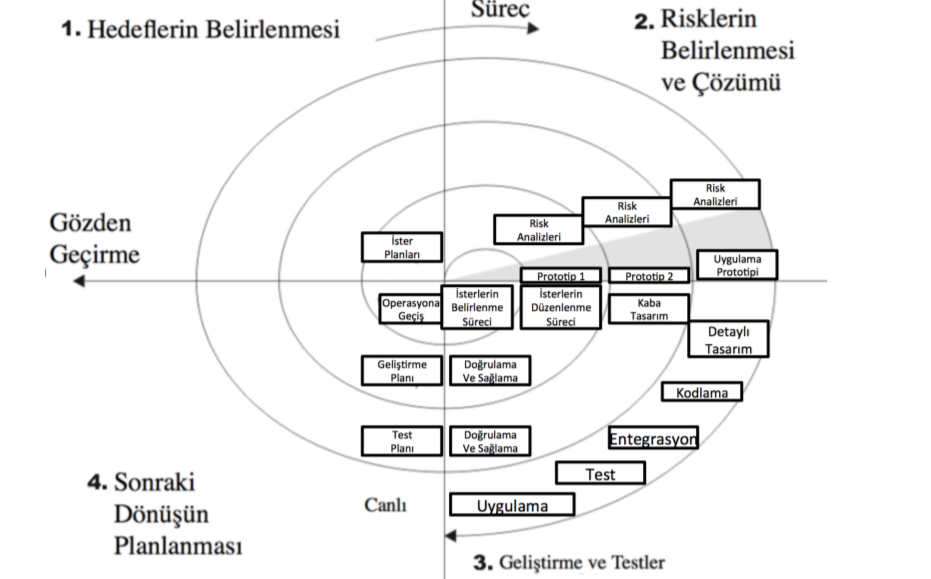
1. Risk azdır.
2. Hata azdır.
3. Gereksinimi anlamayı kolaylaştırır.

### 4.5.2. Dezavantajları

1. Düzenli bir ürün oluşumu yoktur.
2. Bakımı zordur.
3. Gereksinimler sürekli yenilenebilir.

## 4.6 Helezonik (Spiral) Model

Aynı adımlara geri dönülmesini vurgular. Şelale modelinde yok sayılan riskleri göz önüne bulundurur. Çağdaş modellere oldukça yakın bir modeldir.



Şekil Spiral Model

Bölgeleri dörde ayırdığımızı düşünebiliriz. Spiral şeklinde dönerek adımları ilerliyoruz. Bu adımları ilerlerken her adım sonrasında prototipler çıkartıyoruz ve bu prototipleri müşteriye sunuyoruz. Müşterinin yapılan iş hakkında fikir sahibi olmasını sağlıyoruz. Müşterinin yorum yapmasını sağlıyoruz. Bu yorumlar sayesinde daha emin adımlarla bir sonraki aşamaya geçiyoruz. Bunu gibi 3-4 veya daha fazla prototip çıkartılabiliyor. Bu projenin gereksinimlerine göre değişebiliyor. Sadece bir prototipte çıkartılabilir. Bundan sonra işler şelale modelindeki gibi ilerliyor. Bu spiralin 4 aşaması vardır:

* 1. Hedeflerin belirlenmesi.
  2. Risklerin belirlenmesi ve çözüm.
  3. Geliştirme ve testler.
  4. Sonraki dönüşün planlanması.

Bu döngü içerisinde sürekli dönülür. Yukarı doğru gittikçe maliyetin büyümesi ve sola doğru gittikçe de gözden geçirme süreçlerinin arttırılması görülür.

### 4.6.1 Avantajları

1. Kullanıcılar sistemi erken görebilir.
2. Risk ihtimallerini göz önünde bulundurur.
3. Hataların erken fark edilmesin ve çözülmesini sağlayabilir.
4. Gelişimi küçük parçalara böler.

### 4.6.2 Dezavantajları

1. Spiral sonsuz bir döngüye girebilir.
2. Karmaşıktır, anlaşılması zordur.
3. Küçük projeler için uygun değildir.
4. Ara adımların fazlalığından dolayı fazla dokümantasyon gerektirir.

# 5.ÇEVİK YAZILIM GELİŞTİRME

Yazılım süreci oldukça zor bir süreçtir. Bu yüzden çoğu proje kısmi başarı ve memnuniyetle tamamlanabilmekte. Yazılım sektöründe gün geçtikçe görülen yazılımın zamanında ortaya çıkmaması, güncellemelere uygun olmaması, hataların geç fark edilmesi gibi sorunlar ortaya çıkmıştır. Bu sorunların çözülmeye çalışılması sonucu 1990’lı yılların sonuna doğru çevik (agile) yazılım geliştirme süreci ortaya çıkmıştır. Ortaya çıkan bu süreç gereksinimlere daha çabuk cevap verebilme, daha hızlı ürün elde etme gibi amaçlar güder. Çevik yazılım geliştirme daha esnek, hızlı ve güçlü bir yazılım geliştirme metodolojisidir. Bu metodoloji projenin uzunluğu ne olursa olsun küçük bölümlere ayrılır ve her bölüm başlı başına bir proje gibi ele alınır. Her bölüm bir proje gibi ele alındığı için her bölümün sonunda müşterinin eline bir taslak geçmiş olur. Müşteri memnuniyetini artıran bu olay her adımın hata oranını da oldukça azaltmaktadır. Sürekli müşteriyle ve diğer çalışanlarla iletişim halinde olmayan hedefleyen bu metodoloji başarı oranını oldukça artmaktadır. Hatta bu başarı oranı %55 oranında artabilmektedir. Çevik yazılım yöntemlerinin temel maddeleri olarak:

* Bireyleri ve etkileşimi tercih etmek
* Yazılımın çalışmasını, detaylı bilgilendirmeye tercih etmek
* Sözleşme için görüşme yerine müşteri ile işbirliği
* Bir plan üzerinden gitmek yerine değişikliklere açık olma şeklinde belirlenmiştir.

Devam eden çalışmalar neticesinde 2001 yılında Kent Beck ve 16 arkadaşı çevik manifesto oluşturulmuştur. Çevik yazılım süreci yeni özellikler kazanarak 12 maddeye göre şekillenmiştir.

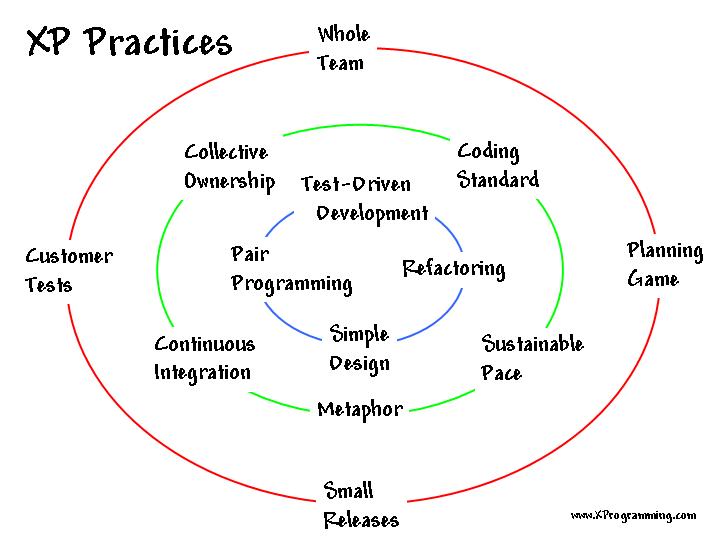
* + - * + Birinci öncelik müşteriye sürekli olarak kullanılabilir ürünler sunmaktır.
        + Esneklik oldukça önemlidir. Projenin ilerleyen safhalarında da yeniliklere açık olmalıdır.
        + Kısa zaman aralıklarında çalışan kaliteli yazılım ürünü ortaya koyar.
        + Tüm ekip elemanları iş brliği içersisindedir.
        + Motivasyonu yüksek tutmayı amaçlar.
        + Yüz yüze iletişime önem verir. Kaliteli bir iletişim olmasını amaçlar.
        + Çalışan ve kaliteli bir ürün sunar.
        + Küçük toplantılarla ekip sürekli güncel bilgiye sahiptir.
        + Basitliğe önem verir.
        + Sürdürülebilir geliştirmeyi destekler.
        + Sağlam teknik altyapı ve tasarım çevikliği arttırır.
        + Kısa sürede müşteri memnuniyetini sağlar.

Sıkça kullanılan çevik yazım modellerinden bazıları şöyledir:

* + Extreme Programming (XP)
  + SCRUM
  + Agile Unified Process
  + Feature-Driven Development (FDD)
  + Test-Driven Development (TDD)
  + LEAN Development
  + Dynamic System Development Methodology (DSDM)

## Extreme Programming (XP)

Kent Beck tarafından 1999 yılında yazılım geliştirme disiplini olarak ortaya çıkmıştır. İşbirliği ile, hızlı ve erken yazılım geliştirmeyi hedefler. Yazılım sektöründe sıklıkla kullanılır.



Şekil XP Modeli

Genel olarak 4 temel prensip üzerinde durur; İletişim, basitlik, geri bildirim, cesaret.

XP modeli iletişim konusunda oldukça önem vermektedir. Müşteri ve ekibin kendi içindeki iletişimin olmasının hataların daha çabuk fark edileceğine ve daha hızlı düzeltilebilirliğini savunur. Mümkünse bu görüşmeler yüz yüze olmalıdır. Tabi ki yüz yüze olan görüşmeler daha sağlıklı bir görüşme olacaktır. Bu yanlış anlaşılmaların önüne geçmek içinde iyi bir yoldur.

Basitlik ilkesi en zor ilkelerinden biridir. Her ne kadar kelime anlamından dolayı kolay gözükse de zor bir aşamadır. Sadece o an gerekli ve zorunlu olan bir işi yapmayı amaçlar. Karmaşıklıktan uzaktır. Esnek ve basit işlemler yapmayı hedefler.

Geri bildirim oldukça önemli bir konudur. Ortaya çıkabilecek sorunların çok daha kolay halledilmesini sağlar. Bu model müşteriyi ekibin içinde gibi görerek memnuniyeti arttırır ve hata oranını en aza indirger.

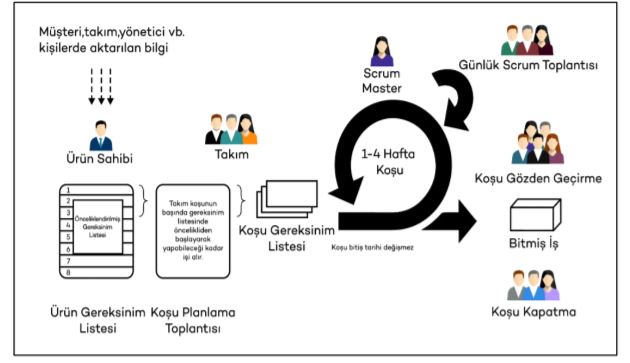
Xp modelinin en zor aşaması cesarettir. Xp başarısızlıktan korkmak yerine onun daha çok üstüne gitmeyi önerir. Başarısızlıktan korkmanın hiçbir faydası yoktur aksine başarısızlıkların üstüne gitmenin ve nedenlerini sorgulamanın gelişime çokça katkısı vardır. Bu yüzden xp korkusuzca, yılmadan yanlış yapılanın üstüne gitmeyi ve gerekirse gelinen yere kadar olan kodu silip baştan başlamayı önerir. Buda insan için oldukça zorlayıcı bir aşamadır.

Yazılım geliştirmede kolaylığı ve esnekliği sağlamak için, XP 12 farklı niteliği ön görür:

* 1. Planlama Oyunu,
  2. Ekipte Müşteri,
  3. Önce Test,
  4. Basit Tasarım,
  5. Çiftli Programlama,
  6. Sürekli Entegrasyon,
  7. Kısa Aralıklı Sürümler,
  8. Yeniden Yapılandırma,
  9. Ortak Kod Sahiplenme,
  10. Metafor,
  11. Kodlama Standardı,
  12. Haftada 40 Saat.

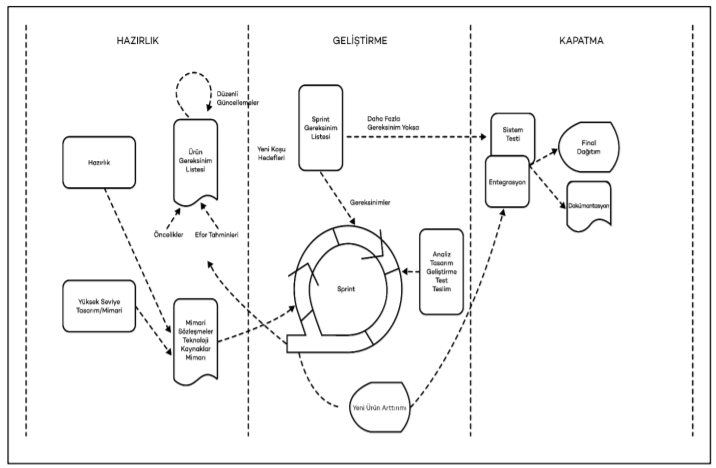
## Scrum

Scrum çevik yöntemler arasında en yaygın olarak kullanılan yöntemdir. Karmaşık yapıları küçük birimlere ayırmayı öngürür. Günlük 15 dakikalık toplantılarla sürekli bir proje takibi yapılmasını amaçlar.



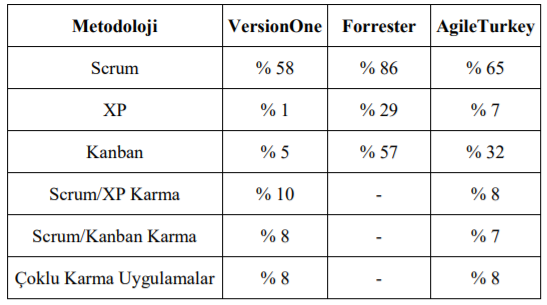
Şekil Scrum1

Scrum yönteminin doğru şekilde işleyebilmesi için şeffaflık, gözlem ve adaptasyon sağlanabilmesioldukça önemlidir. Şeffaflık, yapılan işteki ilerlemenin ortamdaki herkes tarafından görülmesini sağlayacaktır. Gözlem ise ürün parçalarının belirli aralıklarla müşteriye teslim edilerek gerekli düzeltmelerin yapılması ve kalan işlerin sürekli güncel tutulmasını sağlamaktadır. Adaptasyon ise geleneksel yöntemlerde olan gereksinimlerin bir kere belirlenmesi yerine, her adımın sonunda belirlenerek sürekli güncel tutulmasını sağlar. Scrum yöntemi temel olarak üç aşamadan oluşmaktadır. Bunlar hazırlık, geliştirme ve kapatma aşamalarıdır. Hazırlık aşamasında ürün gereksinim, kullanılacak mimari, teknik detaylar belirlenmektedir. Geliştirme aşamasında ürün parçalar halinde müşterilere sunulmaktadır. Kapatma aşamasında ise son testlerle birlikte müşteriye ürünün sunumu yapılmaktadır.



Şekil Scrum2

### 5.2.1 Scrum Neden Bu Kadar Popüler?

Scrum, proje yönetimini her sektörde, her işte ve hatta genel olarak yaşamda dönüştürme gücüne sahiptir. Scrum'ı kullanarak, daha hızlı tepki vermeyi ve yolunuza çıkan kaçınılmaz değişime nasıl daha doğru tepki vereceğinizi keşfederek daha Çevik olacaksınız. Ve odaklanarak, işbirliği yaparak ve iletişim kurarak, gerçekten yapılması gerekeni başarıyla gerçekleştirebilirsiniz.  
  
En önemlisi, Scrum kanıtlanmamış bir hype değildir. Bu, çeşitli proje ve ekiplere uygulanan sağlam ve başarılı bir Çevik çerçeve.Üniversiteler  müşterilere değerli projeler sunmak için Scrum'ı kullanır. Askerler gemileri konuşlandırmaya hazırlamak için Scrum'a güveniyorlardı. Gelen otomotiv dünyasında , Takım Wikispeed az $ 20,000 satmak gereken bir hızlı, ekonomik, ultra verimli, güvenli banliyö araba oluşturmak için Scrum kullanıyor.  
  
Bir sonraki akıllı telefon uygulaması üzerinde çalışıyor, bir mağaza için lojistiği yönetiyor veya bir yardım etkinliği planlıyor olun, Scrum'ı daha yakından incelemeniz gerekir. Scrum Alliance size başarıyı başarmak için ihtiyacınız olan kanıtlanmış çerçeveyi, en iyi uygulama uygulamalarını ve destekleyici rehberliği sunabilir

Şekil Çevik Yazılım Modelleri Kullanım Sırası

# Sonuç

Bu çalışmada, geleneksel yazılım geliştirme modelleri avantajları ve dezavantajları ile açıklanıp karşılaştırılmalı yapılmıştır. Daha sonra geleneksel yöntemlerin yeterli kalmamasından oluşan çevik yazılım geliştirme açıklanmıştır. Çevik yazılım geliştirmenin çok daha kullanışlı ve günümüz de daha çok kullanıldığı anlatılmaya çalışılmıştır. Çevik yazılım geliştirme modellerinden scrum tanımı ve neden yaygınca kullanıldığı açıklanmış ve karşılaştırma tablosuyla anlaşılmasının kolay anlaşılması hedeflenmiştir. Sonuç olarak geçmişten günümüze yazılım geliştirmek için başvurulan tüm yollar açıklanmış ve karşılaştırmalar yapılmıştır.

# KAYNAKÇA

<https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742>

<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://earsiv.medeniyet.edu.tr:8080/xmlui/bitstream/handle/123456789/287/ERHAN%20%c3%96ZL%c3%9c-TEZ.pdf?sequence=1&isAllowed=y>

<http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yaz%C4%B1l%C4%B1m-Geli%C5%9Ftirme-Modelleri-Yaz%C4%B1l%C4%B1m-Ya%C5%9Fam-D%C3%B6ng%C3%BCs%C3%BCSDLCYBS.pdf>

<https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>

<https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>

BROOKSHEAR J.G. Bilgisayar Bilimine Giriş. ISBN 978–605–320–361–2.

Doç. Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş Dersi 2. ve 3. Hafta Sunumları