# YAZILIM YAŞAM DÖNGÜSÜ ve MODELLERİ

**Yağmur Dilara PEKİN**

İzmir Bakırçay Üniversitesi, Mühendislik ve Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü

## ÖZET

Yazılım geliştirmenin kolaylaşmasını, anlaşılabilirliğini ve çözülebilirliğini artırmak için yazılım geliştirme modellerinin günümüzdeki önemi çok büyüktür. Yazılımın geliştirilebilmesi için en uygun ortamı oluşturmak ve bu ortamda yazılım geliştirmenin sürdürülebilirliğini sağlayabilmek için projenin gidişatına ve gereksinimlerine göre en doğru modelin seçilebilmesi gerekir. Modellerin günümüzden bu yana gelişimini, avantajlarını ve dezavantajlarını bilmek bu seçime yön veren bilgilerdir. Bazı metodolojilerin günümüzde daha çok tercih edildiği görülmüştür ve bunun nedenlerini, hangi ihtiyaçlardan dolayı ortaya çıktığı belirlenerek aktarılmaya çalışılmıştır.

**Anahtar Kelimeler:** Yazılım geliştirme, yazılım yaşam döngüsü, yazılım yaşam döngüsü modelleri, SCRUM, yazılım geliştirme süreci.

## 1. YAZILIM YAŞAM DÖNGÜSÜ (Software Development Life Cycle)

Bir yazılımın planlanma aşamasından teslimatına kadar geçirdiği sürece ve bu sürecin aşamalarından oluşan döngüye yazılım yaşam döngüsü denir. Yazılım yaşam döngüsü müşterinin istekleri doğrultusunda süre ve maliyet tahminleriyle beraber yazılımın nasıl geliştirilebileceği, korunacağı, sürdürülebileceği ve daha kaliteli hale getirilebileceğini kapsayan detaylı bir plan oluşturmaktır. Bu planlama sürecinde gereksinimler değişebileceğinden ve üründe hatalar ortaya çıkabileceğinden her zaman bu sorunları gidermek ve yazılımı geliştirmek amacıyla geriye dönüşüler yapılır. Bu geriye dönüşler sayesinde sorunlar çözülür ve yeni eklentiler yapılarak yazılım iyileştirilir. Süreç içerik olarak birbirinden ayrılmış fakat sürekli ilişki içerisinde olan evrelere ayrılır ve bir döngü halini alır. Ayrıca ISO/IEC 12207, yazılım endüstrisi tarafından kaynak gösterilen, yazılım yaşam döngüsü evreleri için ortak bir çerçeve oluşturur ve uluslararası bir standarttır.

### 1.1. Yazılım Yaşam Döngüsü Temel Aşamaları

Yazılım geliştirme evreleri 4 ila 10 aşama arasında değişmektedir. Bu makalede temel olarak 5 aşamadan bahsedeceğim. Bunlar planlama, analiz, tasarım, gerçekleştirme ve bakım aşamalarıdır.

#### 1.1.1. Planlama

Yazılım yaşam döngüsünün başlangıç aşamasıdır. İlk olarak gereksinim analizi yapılır. Gereksinim analizinde müşterinin istekleri ve projenin nasıl başarılı bir şekilde faaliyete geçirilebileceği belirlenmelidir. Bu doğrultuda proje planlamasının ilk adımları atılmış olur. Yazılım üretim aşamasında gerekli olan donanım ve personellerin belirlenmesinin yanı sıra teknik, finansal ve hukuki açıdan araştırmaların yapılacağı fizibilite çalışmasını içinde barındırır. Amaç projeyi minimum risklerle başarıya ulaştırmaktır.

#### 1.1.2. Analiz

Bu aşamanın amacı sistemin işlevlerini ve gereksinimlerini ayrıntılı olarak tanımlamak, belgelemek ve müşteriden veya piyasa analistlerinden onay almaktır. Bu aşamada müşteri ve yazılım mühendisinin çokça iletişim içerisinde olması ve istenilen gereksinimlerin kesinleştirilmesi gerekir. Her şey iki tarafça anlaşılıp kesinleştirildikten sonra dokümantasyon işlemi "**Yazılım Gereksinim Belgesi**" aracılığıyla yapılır. Kısacası analiz aşamasında projenin planlanan tüm detayları ortaya çıkarılır ve belgelenir.

#### 1.1.3. Tasarım

Gereksinimlerin tamamlanmasıyla birlikte sistem tasarımı aşamasına geçilir. İki adet yazılım tasarımı vardır: Mimari ve detaylı tasarım. Mimari tasarımda, analiz aşamasında ortaya çıkan proje detayları doğrultusunda proje bileşenlere ayrılır ve proje içerisinde yapılacak işlemler adım adım belirlenir. Bu adımlar **Tasarım Belge Şartnamesinde** belgelenir. Detaylı tasarımda ise bu belge detaylı bir şekilde analiz edilir. Tasarım dokümanında projenin amacı, kapsamı, sistem tasarım bilgileri, tasarım detayları, veri modeli, kullanıcı arayüz tasarımları ve UML diyagramları belirtilir. Bu belgenin tutulma amacı ise yazılım geliştirilirken bir referans sağlaması ve sonradan dahil olacak yazılımcıların işini kolaylaştırabilecek olmasıdır.

Yazılım tasarım aşamasında en önemli adımlardan biri **"soyutlama"** dır. Ortaya çıkan problemlerin çözülebilmesi adına nesnelerin, olayların ya da durumların görmezden gelinmesidir. Problemleri basite indirgeyerek daha önemli kısımlara odaklanmamızı sağlar. Tasarımın modelleme kısmında ise iki farklı model mevcuttur. Statik model, programın değişmeyen özelliklerine odaklanırken dinamik model ise programın çalışma sırasındaki işleyişe odaklanır.

#### 1.1.4. Gerçekleştirim

Müşteriye teslim edilecek yazılım ürünü bu aşamada kodlanır. Bu aşamada herhangi bir analiz yapılmamalı ve proje tasarım aşamasında belirlenen kodlama yönergelerine uyularak kodlanmalıdır. Kodlama için farklı üst düzey programlama dilleri kullanılır ve programlama dili, geliştirilen yazılımın türüne göre seçilir. Yazılımın geliştirilmesi sırasında ve sonrasında müşteriye sunmadan önce test ekibi tarafından birtakım testler gerçekleştirilir. Erken aşamada testler uygulanmaya başlanırsa hata yapma olasılığı, maliyetler önemli derecede azalır.

#### 1.1.5. Teslim ve Bakım

Tüm test aşamaları tamamlandıktan sonra ürün piyasaya sürülür. Teslim çıktısı ürünün yanında kullanıcılar için kullanım kılavuzu oluşturulmalıdır. Proje yayına alındıktan sonra oluşabilecek hataların giderilmesi, yazılımın iyileştirilmesi ve yeni işlevlerin eklenebilmesi için bakım faaliyetleri gerçekleştirilir.

Tüm bu süreçlerin gerçekleştirilmesi amacıyla **Yazılım Yaşam Döngüsü Modelleri** ve **Yazılım Belirtim Yöntemleri** kullanılır.

### 1.2. Yazılım Belirtim Yöntemleri

Süreç Akışı İçin Kullanılan Belirtim Yöntemleri: Süreçler arasındaki ilişkileri ve iletişim durumunun gösterildiği yöntemlerdir. Örnek olarak veri akış şemaları ve nesne-sınıf şemaları verilebilir.

Süreç Tanımlama Yöntemleri: Gerçekleşen süreçlerin iç işleyiş durumlarını göstermek için kullanılan yöntemlerdir.

Veri Tanımlama Yöntemleri: Süreçler tarafından kullanılan verilerin tanımlanmasında kullanılan yöntemdir. Örnek olarak veri tabanı tabloları ve veri sözlüğü verilebilir.

## 2. YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ

Yazılım geliştirme süreci boyunca bu süreçlerin hangi sırayla ya da nasıl uygulanacağını belirleyen modellerdir. Sürecin modellerle şekillendirilmesi projenin karmaşıklaşmasını önleyerek yönetim ve işleyiş bakımından kolaylık sağlar. Her modelin avantajları olduğu gibi dezavantajları da vardır fakat her model her proje için uygun değildir. Projeden projeye değişkenlik gösterir.

### 2.1. Kodla ve Düzelt Model

Bu model bir ürün fikriyle başlayarak ürünün teslimine kadar geçen sürede sadece kodlama yapılarak devam edilmesini temsil eder. Kişiye özel bir prosedür izlenir. Tek kişilik üretimde ve basit programlar için kullanılır.

#### 2.1.1. Avantajları

- Uzman görüşüne ihtiyaç yoktur, herkes bu modeli kullanabilir.
- Planlama ya da analiz gibi evrelere ihtiyaç duyulmaz.
- Küçük projeler için kullanılabilir.

#### 2.1.2. Dezavantajları

- Kontrollü aşamalardan geçmediği için sağlıklı bir kod çıkarmak zordur.
- Hata ayıklamak zordur.
- Kodları düzenlemek maliyetli olabilir.
- Müşterinin beklentisini ve ihtiyacını karşılamayabilir.
- Bakım süreci düşünülmediği için değişikliğe açık değildir.

### 2.2. Barok Modeli

Bu model eski zamanlarda ortaya çıkmıştır ve yazılım yaşam döngüsünün temel adımlarını doğrusal bir şekilde gerçekleştirmeyi hedefler. Belgelenme ayrı bir süreç olarak ele alınır. Oysaki günümüzde belgeleme her aşama sonrasında ayrı bir evre oluşturmaksızın yapılır. Barok modelinin diğer yöntemlerle karşılaştırıldığında avantajdan ziyade dezavantajları çok fazladır.

#### 2.2.1. Dezavantajları

- Aşamalar arasındaki geri dönüşlerin nasıl sağlanacağı belirsizdir.
- Belgeleme sadece 1 aşama içerisinde yapılır bu sebeple karışıklıklar yaşanabilir.

### 2.3. Şelale Modeli

Şelale yönteminde yazılım geliştirme süreci planlama, analiz, tasarım, kodlama, test, dağıtım ve bakım gibi aşamalardan oluşur. Temel adımlar baştan sona en az birer kez incelenir. Bu modelde her faz, bir sonraki aşamaya geçmeden önce tamamlanmalıdır ve üretimin her bir aşamasında belgelemenin yapılması gerekir.

#### 2.3.1 Avantajları

- Anlaması ve kullanması diğer modellere göre daha kolaydır.
- Gereksinimleri net bir şekilde tanımlanmış ve kısa vadede sonlanması beklenen projeler için uygun bir modeldir.
- Her aşamanın sonunda yapılan incelemeler kullanıcı katılımını sağlar.
- Teknik açıdan deneyimsiz veya zayıf personel için iyi çalışan bir modeldir.

#### 2.3.2. Dezavantajları

- Gereksinim tanımlamaları netleştirilmezse hata ancak gerçekleştirme aşamasında belli olur ve bu hem zaman kaybına hem de maliyete yol açar.
- Planlama ve belgeleme aşaması zaman alır.
- Her türlü ihtiyacı önceden tahmin etmek zordur.
- Karmaşık ve nesne yönelimli projeler için uygun değildir.
- Proje, oluşabilecek değişimlere açık olmadığından değişim yapılmak istendiğinde maliyeti artırır.

### 2.4. V Model

V-Model, çağlayan modelin geliştirilmiş versiyonu olarak görülebilir. Bu modelde, diğer evreler kodlama evresinden sonra yukarıya doğru sıralanarak bir V şeklini oluşturur. Modelin sol kısmı üretim sağ kısmı ise sınama işlemlerini ilgilendiren evrelerinden oluşur. Modelde temel aşamalar olarak kullanıcı modeli, tasarım modeli ve gerçekleştirim model olarak üç alt model vardır. Her üretim aşamasında, tam karşısında bulunan sınama aşamaları yer almaktadır ve her aşama bu sınama aşamalarına bağlıdır. Bu sayede her adımda hata kaynağına ulaşılması çok daha kolaydır. Kısacası bu model sıklıkla onaylama ve doğrulama aşamalarından geçer. Bu model belirsizliklerin az ve proje gereksinimlerinin belirgin olduğu bilişim teknolojileri projeleri, istenen ürünün projeye başlandıktan sonra değişikliğe uğramayacağı kısa süreli projeler için uygundur.

#### 2.4.1. Avantajları

- Her bir aşaması detaylıca ve disiplinli bir şekilde oluşturulmuştur.
- Projeyi anlaması ve yönetmesi kolaydır.
- Her aşaması sınama ve doğrulamaya bağlı olduğu için hatalar erken süreçte fark edilebilir.
- Maliyet ve zaman kaybına neden olmaz.
- Kullanıcının isteği net olduğu için yapılması gerekenler bellidir ve bu nedenle kullanıcının projeye katkısını artırır.

#### 2.4.2. Dezavantajları

- Modelin aşamalarında tekrar bulunmaz bu yüzden yapılan hatalar geç fark edilirse geri dönüşü zordur.
- Karmaşık ve nesne yönelimli projeler için uygun bir model değildir.
- Uzun ve proje sırasında isteklerin ve gereksinimlerin değişebileceği projeler için uygun değildir.
- Risk çözümleme gibi planlamalar içermez, riskin yüksek olduğu projeler için uygun değildir.

### 2.5. Evrimsel Geliştirme

Bu model ilk tam ölçekli modeldir. Modelin başarısı ilk evrimin başarılı olmasına bakar. Müşterinin istekleri doğrultusunda taslak bir sistem oluşturulur. En netleşen gereksinimlerle başlanır ve sonrasında müşteri yeni özellikler talep ederse bu özellikler süreç boyunca projeye eklenir. İki çeşit evrimsel geliştirme vardır. Keşifçi geliştirmenin hedefi müşteri gereksinimlerini incelemek için müşteriyle birlikte çalışıp son ürünü teslim etmektir. Atılacak prototip çeşidinde ise hedef sistem gereksinimlerini anlamaktır, tam anlaşılmamış gereksinimlerle başlar. Büyük firmalar için önerilir. Müşterinin istekleri anlaşılmadığı zaman kullanır.

#### 2.5.1. Avantajları

- Her aşama sonunda üretilen ürün tam fonksiyonelliği içermektedir.
- Model, kullanıcının gereksinimlerinin ne olduğunu anlamasına yardım eder.
- Sürekli değerlendirme yapıldığı için hatalar ve geliştirme riskleri azalır.

#### 2.5.2. Dezavantajları

- Sürecin görünürlüğü ve izlenebilirliği azdır.
- Sürekli değişmeye müsait olduğu için yazılım zarar görebilir ve bakımı zorlaşabilir.
- Değişiklik yönetimine sahip değildir.

### 2.6. Prototip Oluşturma

Başlangıç olarak gereksinimler hızlıca toplanır. Müşteriler ve geliştiriciler tüm gereksinimleri detaylıca konuşulup ne yapılacağı hakkında iki taraf da uzlaşmaya çalışır. Sonrasında projenin müşteriye ne sunacağını gösteren bir tasarım örneği üretilir. Bu prototip müşterinin değerlendirmesine sunulur. Örnek üzerinde gerekli değişimler yapılır ve yeni örnek tekrardan değerlendirilir. Bunların sonucunda müşterinin isteği ürüne en yakın taslak elde edilmiş olur. Bu model, müşteri ve geliştiriciler arasında düzenli bilgi akışı sonucunda son halini alır.

#### 2.6.1 Avantajları

- Müşteri ve geliştiriciler arasında yanlış anlaşılma olasılığı ortadan kalkar.
- Risk analizi yapılır.
- Değişime açıktır, yeni özellikler rahatlıkla belirlenip projeye dahil edilebilir.
- Kullanıcı sistem gereksinimlerini görebilir.

#### 2.6.2 Dezavantajları

- Belgelendirme olmadığı için hızlı ve düzensiz prototipler ortaya çıkabilir.
- Müşteri son ürünün prototipteki gibi görünmesini ve etkili olmasını bekler.

### 2.7. Spiral Model

Spiral model önceki aşamalara geri dönülmesinin zorunluluğunu vurgular. Proje tekrar eden 4 temel aşamaya ayrılır ve her aşamanın riskleri ayrı ayrı incelenir. Bu aşamalar sırasıyla planlama, risk analizi, üretim ve kullanıcı değerlendirmesidir. Risk analizinin üzerinde duran bir modeldir. Ayrıca prototip yaklaşımı görülür.

#### 2.7.1. Avantajları

- Üretilen prototipler sayesinde yinelemeli artırımlı bir yaklaşımı vardır.
- Müşteriler sistemi başlangıçtan itibaren görebilirler.
- Müşterinin katkısı büyük bir yer kaplar.
- Risk analizine önem verilmesinden dolayı üretim esnasında hatalar erken fark edilerek zaman ve maliyet kaybı önlenir.
- Geliştirme aşaması daha küçük parçalara bölünebilir ve daha riskli kısımlar önceden geliştirilebilir.

#### 2.7.2. Dezavantajları

- Küçük ve düşük riskli projeler için pahalı kaçabilir.
- Yönetim ve süreç karmaşıktır.
- Spiral sonsuz bir döngüye girerse belgelemelerle birlikte karmaşıklık da artacaktır.

### 2.8. Formal Sistem Geliştirme

Bu model yazılım tasarım ve gerçekleştirilmesiyle ilgili matematiksel bir tekniktir. Modelin temelinde karışık programları geliştirmek yatar. Hedefi pahalı hata ayıklama işlemlerine girmemek için kodu ilk baştan itibaren doğru yazmak ve test aşamasında doğruluğundan emin olmaktır.

#### 2.8.1. Avantajları

- Hatasız yazılım geliştirmek için olanak sağlar.
- Karmaşık değildir.

#### 2.8.2. Dezavantajları

- Çok zaman alan pahalı bir yöntemdir.
- Deneyimsiz ve zayıf personeller varsa iletişimde ve geliştirmede sorunlara yol açar.

### 2.9. Yeniden Kullanıma Yönelik Geliştirme

Organizasyonlar tarafından daha önce hazırlanmış veya dışarıdan temin edilen yazılımların kullanılmasıyla son dönemde popülerlik kazanmış bir yöntemdir. Kısa süreli yazılımlar için uygundur.

#### 2.9.1. Avantajları

- Kaynak ve maliyet deneyimi yapmak mümkündür.
- Basit ve anlaşılırdır.

#### 2.9.2. Dezavantajları

- Başarı garantisi yoktur ve pahalıdır.

### 2.10. Artırımlı Geliştirme

Bu modelde proje küçük ve daha kolay yönetilen modüllere bölünür ve müşterinin önceliğine göre sıralanır. Bu modüllerin bittiği her seferde bir ara ürün geliştirilir, yani model her yenilemede ürüne yeni özellikler katar. Geliştirme sürecinde birden fazla döngü aynı anda gerçekleşebilir. Bu süreçte bir yandan ürün kullanılırken bir yandan da geliştirilmeye devam eder. Bu model uzun süreli ve işlevselliği eksik olabilecek projeler için uygundur.

#### 2.10.1. Avantajlar

- Hata riski aşırı düşüktür ve ürünün önemli hatları erken aşamalarda oluşmaya başlar.
- Kısıtlı sayıda çalışanlarla rahatlıkla yapılabilir.
- Ürün, fonksiyonellik sağlamış bir şekilde tüm döngülerin sonunda ortaya çıkar.
- Ara ürünler prototip gibi davranarak gereksinimlerin daha iyi anlaşılmasını sağlar.
- Fazlaca test etmek için olanak sağlar.

#### 2.10.2. Dezavantajları

- Bir ara ürün bitip diğeri başlayana kadar herhangi bir değişiklik yapılamaz, modülleri oluşturmak için sistemin detaylı bir şekilde tanımlanması gerekir.
- Gereksinimleri, doğru boyuttaki artımlara atamak zor olabilir.

### 2.11. Birleşik Süreç

Nesne tabanlı yazılım geliştirmek için kullanılan yöntemlerden edinilen deneyimlerle seçilen en iyi özelliklerin bir araya gelmesiyle oluşan bir yazılım geliştirme sürecidir. Yinelemeli, artırmalı ve evrimseldir. Aynı zamanda risk güdümlüdür. Önce çok daha riskli kısımlara odaklanılarak riskler önceden giderilebilir. Birleşik süreçte yazılım geliştirmesi başlangıç, ayrıntılandırma, tamamlama ve yayım aşamalarından oluşur. Başlangıçta fizibilite çalışması yapılır. Ayrıntılandırma aşamasında ise yüksek riskli parçaların yinelemeli bir şekilde oluşturulur. Tamamlama sürecinde daha az riskli kısımlara odaklanılır. En son beta testleri yapılarak piyasaya sürülür.

#### 2.11.1. Avantajları

- Proje adımlarla gerçekleştiği için sonradan değişebilecek isteklere uygundur.
- Büyük sistemleri çözmede kolaylık sağlar.
- Risklerin erken giderildiği bir modeldir.

#### 2.11.2. Dezavantajları

- Karmaşıktır.
- Dokümantasyon yükü ağır olduğundan maliyete neden olabilir.

## 3. ÇEVİK YAZILIM GELİŞTİRME

"Çevik" yazılım geliştirme metodu geliştirilen yazılımın teslim zamanına yetişememesi, değişen isteklere ve gereksinimlere uyum sağlayamaması ve hataların geç fark edilmesi gibi sorunların aşılmasına yönelik bir metot olarak ortaya çıkmıştır. Bu metot yinelemeli geliştirmeyi, basitliği ve sürdürülebilirliği temel alan, bazı yazılım geliştirme metodolojileriyle harmanlanmış yenilikçi bir metodolojidir. Piyasaya hızlı ürün teslim etmeyi, isteklere çabuk adapte olabilmeyi, hata oranı ve maliyeti düşük, verimliliği yüksek bir yazılım ürünü ortaya çıkarmayı hedeflemektedir. Çevik yazılım geliştirme sürecinde proje ne kadar uzun olursa olsun, küçük yinelemelere ayrılır ve detaylı bir şekilde geliştirilir. Her yinelemenin sonucunca müşteriye projenin durumu hakkında bilgi verilir. Takım çalışmasının, sıkı denetimlerin ve müşteri ile tam uyum sağlamanın önemini vurgulayan bir metodolojidir. **Extreme Programming** ve **SCRUM** en popüler iki yöntemidir.

### 3.1. Extreme Programming (XP)

XP, grup içi iletişime önem verir ve geri dönüşleri fazlalaştırmayı amaçlar. XP'nin 4 temel değeri vardır.

#### 3.1.1. İletişim

XP'de iletişim çok önemli bir yer kaplar. Öncelikle çalışan ekibin başarıya ulaşması için sağlıklı bir iletişim sağlanması gerekir. Günümüzde yüz yüze iletişim aksayabileceğinden artık çevrimiçi ortamlarda görüşme yapabilmek çok daha kolaylaşmıştır. Eğer iletişim iyi olursa hataların erken fark edilmesiyle birlikte kısa sürede bilgi alışverişi tamamlanarak yazılımda aksaklığa izin verilmeden devamlılık sağlanmış olur. Ekip çalışanlarının aynı disipline sahip olması birlikteliği ve verimliliği artırır.

#### 3.1.2. Basitlik

Her ne kadar basitlik sağlanması kolay gibi görülse de en zorlanılan konulardan biridir. Çevik yazılım geliştirmede müşteriden hızlı bir şekilde geri dönüş alınması beklendiği için istenilen bilgiler doğrultusunda, gereksinimlerin karşılanabildiği esnek ve basit bir çözüm sunmayı hedefler.

#### 3.1.3 Geri Bildirim

Müşteri, yönetici ve takım üyelerinin geri dönüşleriyle beraber ortaya çıkabilecek hatalar fark edilir. 2 veya 4 hafta aralıklı sürümler ile sürecin durumu belirlenir. Müşteriler ve yazılım ekibi belli zamanlarda buluşarak gelinen noktanın ve sonraki adımların analizini yaparlar böylece büyük anlaşmazlıklar çıkmadan giderilmiş olur.

### 3.1.4. Cesaret

En zor değer olarak görülen cesaret, projelerin geliştirilmesinde rol oynayan en önemli nokta olarak görülebilir. Karşılaşılan sorunlar ne kadar uğraştırıcı ve zorlayıcı olursa olsun üstüne ısrarla gidilmenin önemini vurgular. Çalışmayan bir kodla tüm zamanınızı harcamak yerine çöpe atmak ve yeniden başlamak konusunda tereddüt edilmemelidir, yoksa hem motivasyon kaybına hem de proje hızının düşmesine sebep olur.

## 3.2. SCRUM

Scrum, kelime anlamını rugby sporundan almıştır ve tüm oyuncularla birlikte karşı sahaya atak yapılması anlamına gelir. Scrum sadece yazılım geliştirme metodu değildir, her alanda her amaçla kullanılabilecek bir proje yönetim yaklaşımıdır. Scrum'ın genel çalışma mantığı, genellikle kullanıcı hikayelerinden oluşan, yapılacak işin listesi tutulur. Bu listeye **"Product Backlog"** denilir, parçalara bölünür ve iş sprint adı verilen küçük parçalara yani **"Sprint Backlog"**lara ayrılır. Her bir sprint genellikle 15-30 günlük zaman dilimlerine ayrılır. Bu süreçte Scrum ekibi her gün maksimum 30 dakikalık bir görüşme **"Scrum Daily Meeting"** düzenler ve gelinen genel durum hakkında bilgi alınır. Scrum'da üç temel kavram vardır. Bunlar **"Roller"**, **"Toplantılar"** ve **"Araçlar"** dır. Rollere gelinecek olursa öncelikle projenin değerlendirilmesi sırasında geri dönüş yapmakla sorumlu olan bir **"Ürün Sahibi"** bulunur, bu kişi müşterinin kendisi olmak zorunda değildir. Devamlı iletişim halinde olması gereken, aynı hedefler doğrultusunda birleşmiş 5-9 kişiden oluşan bir **"Scrum Takımı"**nın disiplinini sağlaması için **"Scrum Yöneticisi"** atanır. Günlük olarak yapılan, takımın işleyişi hakkında bilgi sahibi olmak için kısa süreli toplantılar düzenlenir. Araçlar, Product Backlog ve Sprint Backlog'u kapsar. Durumun analizi için çeşitli grafikler kullanılır. Örnek olarak **"Burndown Chart"** verilebilir. Bu grafik sprint boyunca işlerin ne kadar yapıldığı ve yapılması hedeflenen işi karşılaştırmak için kullanılır.

Scrum'ın günümüzde bu kadar popüler olmasının sebebi karmaşık ve büyük kapsamlı projeler dahil tüm projeleri kolaylaştırması olarak görülebilir. Hedef işin küçük birimlere ayrılması her anlamda geliştiricinin odaklanmasını artırır, bununla birlikte verimlilik de artar. Uzmanlık gerektiren ve maliyeti yüksek bir süreç olsa bile hızlı olması ve başarı garantisinin yüksek olması sebebiyle kullanımı artmıştır. Diğer modellere kıyasla esnekliğe en açık olan model, çevik modellerdir. Eğer müşteriden gelen talepler tahmin edilemiyorsa, detaylı yol haritası, tasarımın ve test aşamalarının süresini tahmin etmek güçse çevik model kullanımı en mantıklısıdır.

## 3.3. Çevik Modellerin Avantajları

- İnsanın doğal eğilime yatkın olduğu için öğrenimi ve adaptasyonu kolaydır.
- Sağlıklı iletişim sayesinde takım içi motivasyon genellikle yüksektir.
- Projede planlama ve yürütme bir arada olduğundan, tek seferde plan aşamasında ayrıntılı plan yapmak yerine her yineleme sonrası tekrardan plan yapılır.
- Değişime açıklık ve esneklik çok yüksektir.
- Kısa sürede müşteri memnuniyeti sağlanır.

## 3.4. Çevik Modellerin Dezavantajları

- Sürekli ihtiyaç değişimlerinden dolayı aşırı mesai yapılabilir, bu yüzden yoruculuğu fazla olabilir.
- Yorgun personelin verimliliği düşerek zaman kaybına yol açabilir, bu çevik modellerde en son istenen şeydir. Proje başarısı riske girer.
- Dokümantasyon, sürekli değişen istekler karşısında güncelliğine koruyamayabilir. Bu eksiklik sebebiyle yeni ekip üyesinin projeye dahil edilmesi zor olabilir.

**REFERANSLAR**


- **https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696**
- Doç. Dr. Deniz KILINÇ, Bakırçay Üniversitesi Yazılım Mühendisliği Temelleri Dersi (2021-2022) 3. ve 4. Ders Notları
- **https://www.codex.com.tr/yazilim-gelistirme-modelleri**
- **https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/**
- **https://webdosya.csb.gov.tr/db/cbs/icerikler/salihsoylu_tez_v10-20180925134450.pdf**
- **https://zeynepaygun.wordpress.com/2017/05/29/what-is-sdlc-sdlc-nedir/**
- **https://hayririzacimen.medium.com/yazılım-yaşam-döngüsü-ve-süreç-modelleri-70fdfb2f8f77**
- **https://medium.com/@omerharuncetin/yazılım-yaşam-döngü-modelleri-543c7879a742**
- **https://www.bayramucuncu.com/scrum-kavramlari/**


**Hesaplarım**

- **https://github.com/yagmurdilara**
- **https://www.linkedin.com/in/yağmur-dilara-pekin-023759233/**
- **https://medium.com/@yagmurdilara**