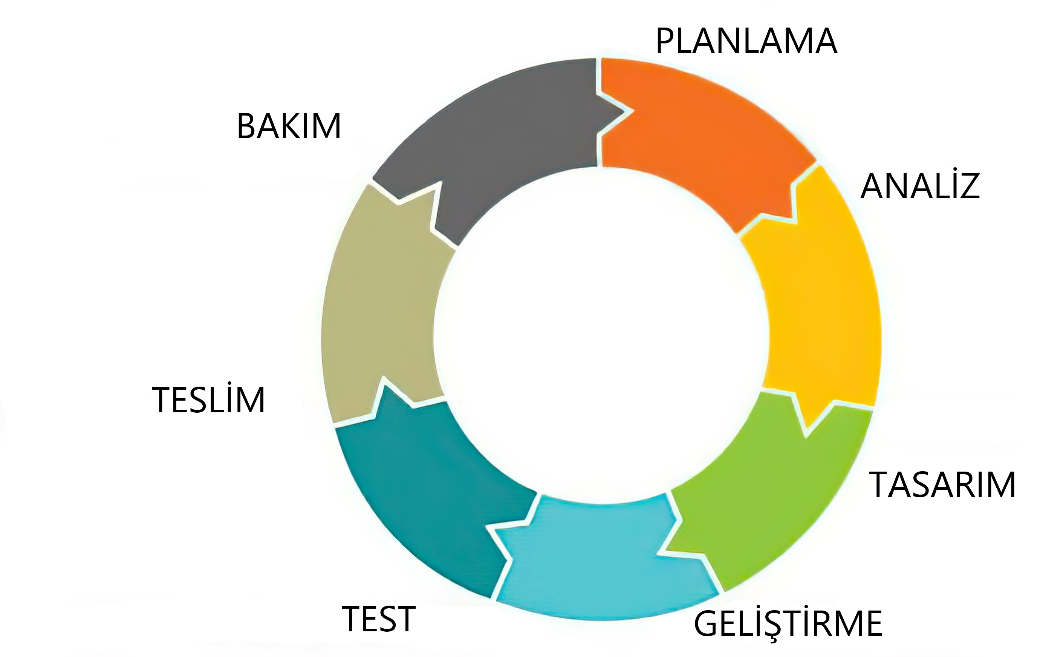
**Yazılım Yaşam Döngü Modelleri**

**Yazılım Yaşam Döngüsü (SDLC) Nedir?**

Yazılım yaşam döngüsü, yazılım endüstrisi tarafından yüksek kaliteli yazılımları müşterinin gereksinimleri doğrultusunda tasarlamak, geliştirmek ve test etmek için kullanılan bir dizi aşamalardan oluşan bir süreçtir. Yazılım yaşam döngüsünün temel amacı, verilen bütçe ve zaman dahilinde, tüm aşamalarda müşterilerin gereksinimlerini karşılayacak yüksek kaliteli ürünün planlı ve etkili şekilde üretilmesini sağlamaktır.

**Yazılım Yaşam Döngüsünün Temel Aşamaları Nelerdir?**

Yazılım yaşam döngüsünün 7 temel aşaması vardır ve her biri yazılım geliştirme sürecinde önemli bir rol oynar.



**1-Planlama**

Planlama, yazılım yaşam döngü sürecinin en önemli aşamalarından biridir. Proje için kapsamlı bir eylem planı belirlemek için gereksinimler ve temel ihtiyaçlar doğrultusunda tüm yazılım yaşam döngüsü boyunca ne olacağını ana hatlarıyla belirtilir. Bu aşamada alınan kararlar projenin başarısında kritik öneme sahiptir.

**2-Analiz**

Yazılım projesi için ortaya konulan işler, sistem gereksinimleri ve problemler araştırılır, öne çıkarılıp ayrıntılı olarak incelenir. Bu analiz işlemi sonrasında beklentiler kesinleştirilir ve belgelenir. Bu aşamada ihtiyaçlar net bir şekilde belirlenmesi gerekmekte bu sebeple müşteri ile mühendisler arasında sağlıklı bir iletişimin kurulması önem arz etmektedir.

**3-Tasarım**

Planlama ve analiz aşamalarının tamamlanmasıyla birlikte ortaya konulacak yazılım ürünün tasarımına başlanır. Bu aşamada, sistemin mimarisi, kullanıcı arayüzü, platformu, güvenliği gibi birimler müşterinin istekleri ve sistem gereksinimleri ele alınarak tasarlanır. Oluşturulan tasarımın, ihtiyaçları karşılaması ve geleceğe yönelik yenilenebilir yapıda olması projenin başarısını etkileyecektir.

**4-Geliştirme**

Tasarım sürecinin sonlanmasıyla birlikte geliştirme süreci başlar. Geliştirme süreci diğer aşamalara göre bir nebze daha yoğundur. Bu aşamada, yazılım mühendisleri ortaya konulacak ürünün gereksinimlerini ve tasarım ögelerini koda dönüştürür. Burada yazılım mühendislerinin kodlarını belli bir kalite çerçevesinde yazması, bir başka yazılım mühendisi tarafından anlaşılabilir olması ve gelecekte güncellemeye açık olması önem arz etmektedir.

**5-Test**

Bu aşama, geliştirilen ürünün beklenildiği gibi çalışıp çalışmadığını ve gereksinimleri tam olarak karşılayıp karşılamadığı değerlendirmek için sistemin detaylı bir şekilde test edilmesidir. Ürünün belli bir kalite standartlarına ulaşması için ürün hatalarının açık bir şekilde rapor edilmesi, takip edilmesi, düzeltilmesi ve yeniden test edilmesi önemlidir.

**6-Teslim**

Ürün kapsamlı bir testten başarılı olarak geçtikten sonra ürünün müşteriye teslim edilmesidir. Teslim sırasında ürün hakkında bilgilendirici dokümanları ve kullanım bilgisini müşteriye iletmek önemlidir.

**7-Bakım**

Yazılım ürününün son kullanıcıya ulaştıktan sonraki dönemlerde ürünün hatalara karşı ve kullanıcı isteklerine göre küçük veya büyük güncellemeleri alması gerekir. Bu gereksinimler bakım aşamasında gerçekleştirilir.

**Yazılım Yaşam Döngüsü Modelleri Nelerdir?**

**1-Gelişigüzel Model**

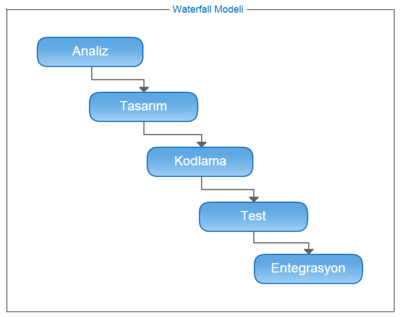
1960’lı yıllarda ortaya çıkmıştır. Herhangi bir model veya yöntemi yoktur. Çoğunlukla tek kişilik ve basit projelerde kullanılır. Bu modelle oluşturulan projelerin takip edilebilirliği ve bakımı oldukça zordur. Genellikle basit programlama içerir.

**2-Barok Modeli**

1970’li yıllarda ortaya çıkmıştır. Aşamalar doğrusal bir biçimde geliştirilir. Bu modelde dokümantasyon ayrı bir süreç gibi işlenir. Aşamalar arasında geri dönüş tanımlı değildir. Daha çok projenin gerçekleştirme evresine ağırlık verilir. Bu nedenlerden ötürü günümüzde kullanılmaz.

**3-Çağlayan Yaşam Döngü Modeli**

Listedeki en eski modellerden biri olan Çağlayan yaşam döngü modelinin tarihi 1960’lara dayanıyor fakat günümüzde hala popülerliğini koruyan bir model. Çağlayan yaşam döngü modeli analiz, tasarım, kodlama, test, entegrasyon gibi aşamalardan oluşur. Bu aşamalar doğrusal bir şekilde ilerler. Bu, geliştirme sürecindeki herhangi bir aşamanın yalnızca önceki aşama tamamlandığında başlayacağı anlamına gelir. Her aşamanın sonunda doküman oluşturulur. Müşteri ve kullanıcılarla yalnızca başlangıç aşamasında diyaloğa girilir.

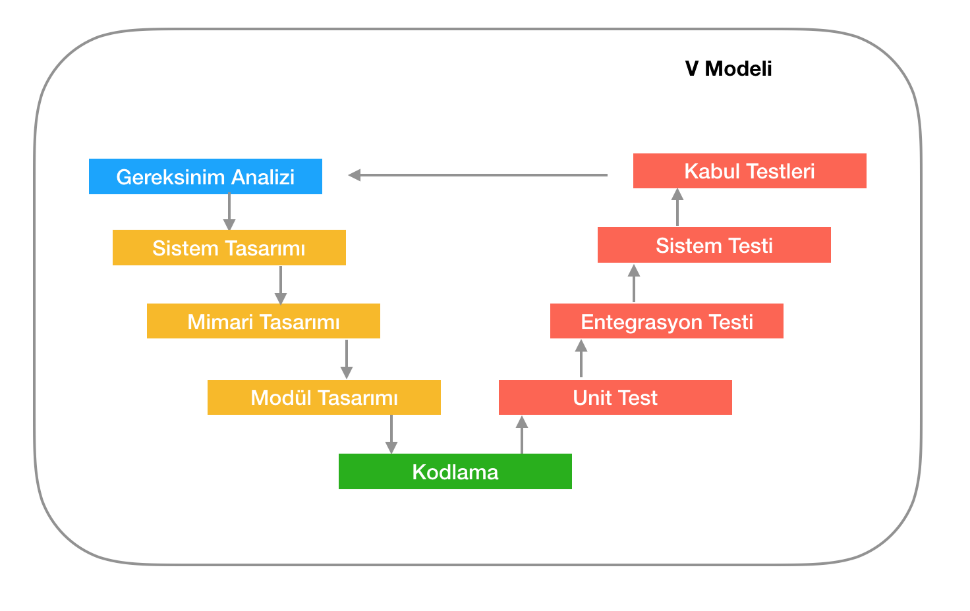


**Avantajları**

* Anlaması ve kullanması kolaydır.
* Proje planlaması ve aşamalar net bir şekilde belirtilir.
* Her aşamanın belirli bir sonucu olduğu için yönetimi kolaydır.
* Küçük ve gereksinimleri net bir şekilde belirlenmiş projelerde kullanılması başarıyı artırır maliyeti azaltır.
* Hataların her aşama sonunda doğrulanması hataların erken tespitini ve düzeltilmesini sağlar.
* Küçük ve gereksinimleri net olan projelerde kısa sürede sonuca ulaşılabilir.

**Dezavantajları**

* Tamamlanmış herhangi bir aşamaya geri dönmek zordur ve genelde projelerde önceki aşamalara dönmek zorunda kalınabilir.
* Dokümantasyon işlemi çalışanların çok fazla zamanını alır. Çalışanlar bir an önce kod yazma eğiliminde oldukları için ekip moralini de etkileyebilir.
* Genelde projenin tamamlanması zaman alır.
* Müşteri ve kullanıcı gereksinimleri başlangıçta alınması ilerde sorun yaratabilir. Müşterinin gereksinimleri değişebilir ve bu yüzden değişen müşteri gereksinimlerini hayata geçirmek maliyeti oldukça arttırabilir.
* Hatalar yalnızca bulunduğu aşamada giderilmesi ileride sorun yaratabilir.
* Karmaşık projeler için uygun değildir.

**4-V Süreç Modeli**

V-süreç modeli, aşamaların V şeklinde sıralı bir şekilde yürütüldüğü bir yazılım yaşam döngü modelidir. V süreç modelinde sol taraf üretim, sağ taraf sınama işlemleridir. V süreç modelinde, çağlayan yaşam döngü modelindeki gibi bir aşamadan sonraki aşamaya geçmek için tüm işler netleşmiş veya bitmiş olmalıdır. Bu modelde şelale modelinden farklı olarak geliştirme döngüsündeki her bir aşama için doğrudan ilişkili bir test aşaması vardır. Buradaki amaç sistemin düzgün ve kaliteli çalıştığından emin olarak ilerlemektir.

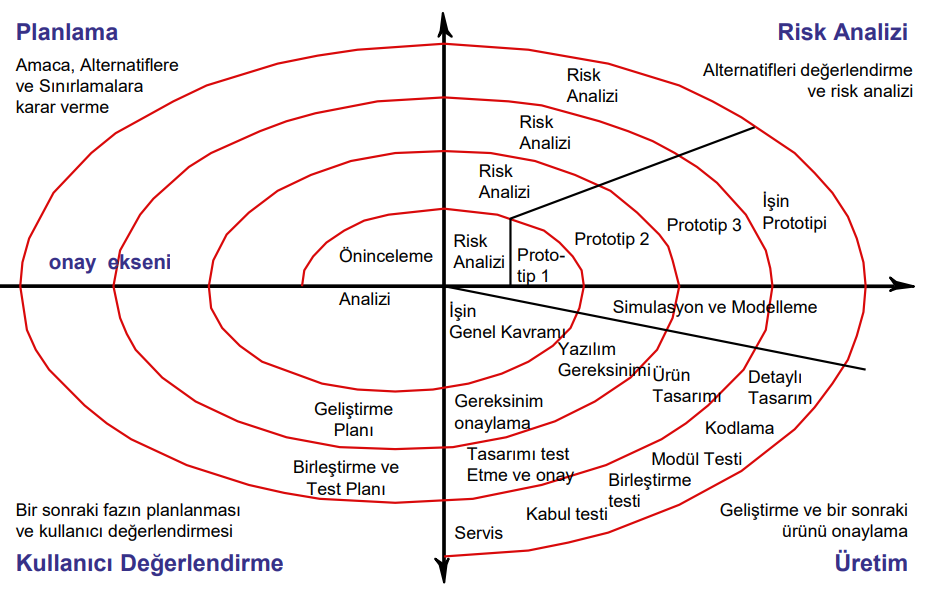
**Avantajları**

* Tüm süreci daha kolay kavramayı sağlayan çok sayıda planlama var.
* Proje yönetimi tarafında takibi kolaydır.
* Anlaması ve kullanması kolaydır.
* Gereksinimlerin çok iyi anlaşıldığı küçük projeler için iyi sonuç verir.
* Bu model oldukça disiplinli bir modeldir ve aşamalar birer birer tamamlanır. Bu nedenle yönetimi daha kolaydır. Her aşamanın belirli çıktıları ve inceleme süreci vardır.

**Dezavantajları**

* Çağlayan yaşam döngü modeli gibi çok esnek değil.
* Karmaşık ve nesneye yönelik projeler için iyi bir model değil.
* Bir uygulama test aşamasındayken, geri dönüp bir işlevi değiştirmek zordur.
* Ürün kodlama aşamasında geliştirildiğinden dolayı yazılımın erken prototipleri üretilmez.
* Uzun ve karmaşık projeler için oldukça zaman harcar aynı zamanda maliyeti de arttırır.

**5-Helezonik (spiral) Model**



Gereksinimlerin belirsiz, projenin karmaşık ve büyük olduğu durumlarda risk oldukça yüksek olur. Bu ihtiyaç doğrultusunda helezonik model ortaya çıkmıştır. Bu modelde risk analizi ve yönetimi ön plandadır. Bu durum, başta bahsettiğimiz projeler gibi projeler için riski azaltır. Planlama, Risk Analizi, Üretim ve Kullanıcı Değerlendirme fazlarından oluşur.

**1-Planlama**

Bu aşamada müşteri gereksinimleri incelenir. Bu gereksinimlere göre hedef belirlenir ve bu hedeflere göre çözüm üretilir. Kısaca proje ana hatlarıyla planlanır.

**2-Risk Analizi**

Bu aşamada risk analizi yapılır. Çözümler araştırılır ve en iyi çözüm yolu seçilir. Bu konuda ürünün bir prototipi geliştirilir.

**3-Üretim**

Seçilen çözüm yoluna göre bir ara ürün ortaya konur ve testleri yapılır. Bir sonraki faza hazırlanır.

**4-Kullanıcı Değerlendirme**

Ortaya konulan ara ürün kullanıcıya sunulur ve kullanıcı tarafından yapılan değerlendirmeler ele alınır.

**Avantajları**

* İş ilerledikçe proje hakkında yapılan tahminler daha gerçekçi hale gelir çünkü önemli konular daha erken ele alınır.
* Risk yönetimi, projeyi parçalar halinde ele alarak daha kolay hale getirilir.
* Yazılım sürekli geliştirilir. Değişime açıktır.
* Her aşamada riski tespit etmek kolaydır.
* Kullanıcılar sistemi erken görebilirler.
* Gereksinimlerin belirsiz olduğu karmaşık ve riskli projeler için oldukça uygundur.

**Dezavantajları**

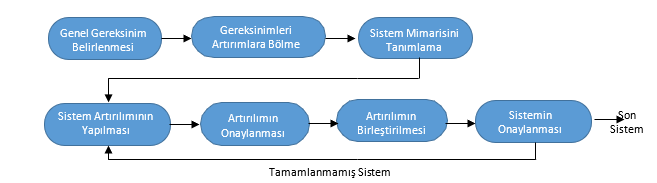
* Projeyi bitirmek için oldukça zamana ve paraya ihtiyaç vardır.
* Küçük ve karmaşık olmayan projeler için gereğinden fazla maliyetli bir çözümdür kullanılması önerilmez.
* Diğer modellere göre karmaşıktır.
* Çok fazla dokümantasyon gerektirir.
* Spiral sonsuza kadar gidebilir.
* Başlangıçta planlanandan daha farklı bir proje ile karşılaşılabilir.
* Riskleri ve varsayımları değerlendirmek için özel becerilere ihtiyaç duyar.

**6-** **Artımsal Geliştirme Süreç Modeli**

Artımsal geliştirme süreç modeli, müşteriye yazılımı parça parça geliştirip teslim etmeye dayanır. Müşterinin gereksinimleri belirlenir ve bu gereksinimler, birden çok bağımsız modüle bölünen yazılım geliştirme sürecine dahil edilir. Her bir yazılım bir öncekinin üstüne bazı ek işlevler ekler. Bu modelde bir taraftan üretim bir taraftan da kullanım yapılır. Her bir yazılım parçasının birbirinden olabildiğince ayrı olması ve birbirleri ile entegrasyon ihtiyaçlarının az olması sayesinde model başarılı olur.

Bu model;

* Sistem gereksinimleri açıkça anlaşılır.
* Ürünün erken piyasaya sürülmesi gerektiğinde
* Yüksek riskli ve karmaşık hedefler söz konusu olduğunda
* Yazılım mühendisliği ekibinin yetersiz veya sayısı az olduğunda

Kullanıma uygundur.

**Avantajları**

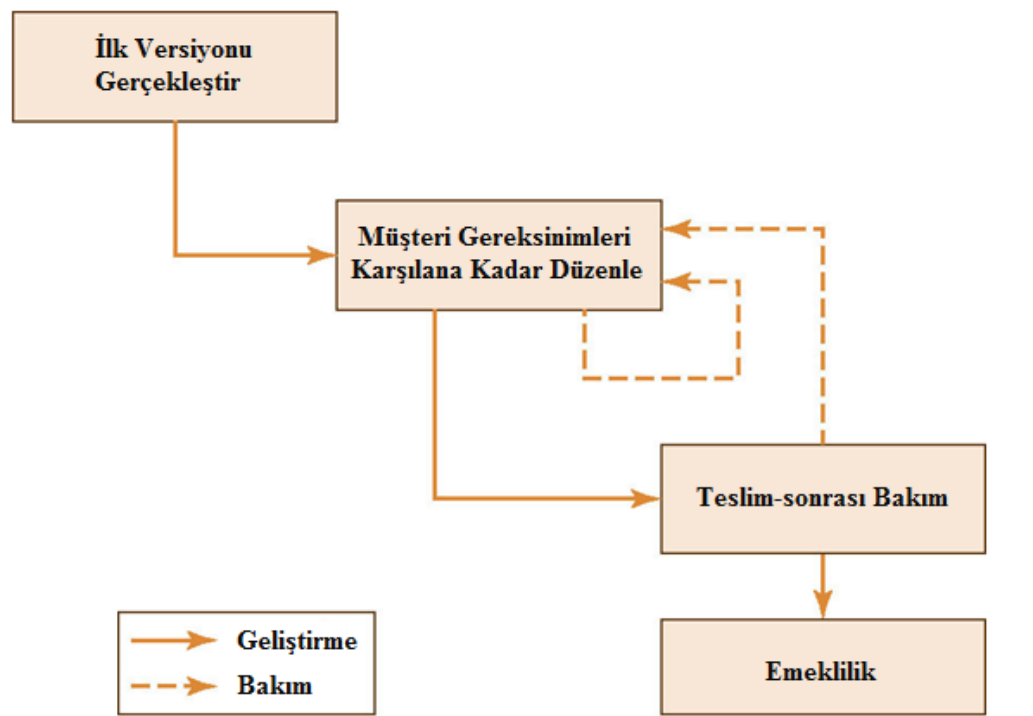
* Yazılım hızlı bir şekilde üretilir.
* Gereksinimler müşterilerle belirlenir.
* Gereksinimler önemine göre sıralanır ve önce en önemli gereksinimi karşılayan bir yazılım geliştirilir.
* Gereksinimleri ve kapsamı değiştirmek esnektir ve daha ucuzdur.
* Projenin başarısız olma riskini azaltır.
* Hataları tanımlaması kolaydır.

**Dezavantajları**

* Kusursuz ve iyi bir planlama gerektirir.
* Yazılım yaşam döngüsünün tamamı için tüm gereksinimler önceden toplanmadığı için sistem mimarisi nedeniyle ilgili sorunlara neden olabilir.
* Deneyimli personel gerektirir.
* Her artırım aşaması katıdır ve birbiriyle örtüşmez.
* Tek bir modülde bir problemi gidermek, tüm modüllerde düzeltme gerektirebilir. Bu nedenden ötürü sistem çok zaman ve para harcayabilir.

**7-Kodla ve Düzelt Yaşam-Döngü Modeli**

Kodla ve düzelt yaşam döngü modelinde en kısa sürede prototipleyerek sonuca ulaşma amaçlanır. Kodla ve düzelt modeli cowboy coding olarak da adlandırılır. Diğer modellere kıyasla proje gerçekleştirme öncesi planlama, analiz gibi aşamalar üzerinde çok vakit harcanmaz. Ana problemlere odaklanılır.



**Avantajları**

* Planlama ve analize çok zaman harcanmaz.
* Projeyi bir an önce gerçekleştirmek için kullanılabilir.
* Kişisel kullanıma uygundur.
* Çok küçük ve basit projeler için kullanılabilir.
* Amatör ve yeni başlayan mühendisler için kullanması en kolay modeldir.

**Dezavantajları**

* Sisteme ait dokümantasyon olmadığı için bakım aşaması zordur.
* Plansız olduğu için kontrollü değildir.
* Kodlama aşaması planlanmadığından sonradan yapılmak istenen değişiklikler maliyetli ve değiştirilmesi zor olur.
* Projenin bitiş süresi tam kestirilemez

**Çevik Yazılım Geliştirme (Agile Software Development) Nedir?**

2001’de Utah’ta yaşayan 17 yazılım mühendisi tarafından geliştirilen ve en ünlü SDLC metodu olan Çevik yazılım geliştirme modeli, hızla değişen dünyamıza ayak uyduran, yazılım geliştirmeye yönelik insan ve sonuç odaklı bir yaklaşımdır. Çevik yazılım geliştirme modelinin bir de manifestosu vardır. Bu manifestoda:

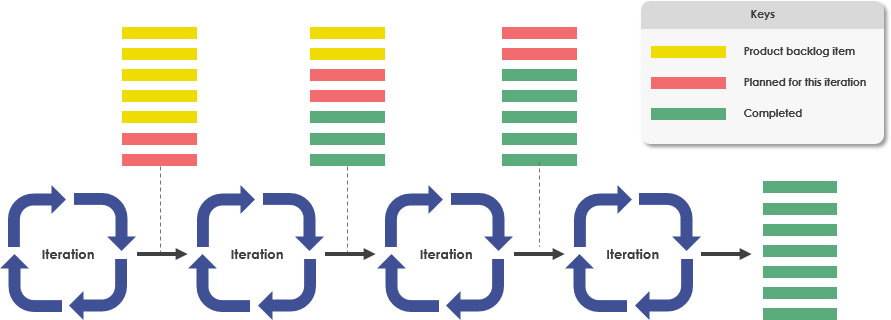
Süreçler ve araçlardan ziyade **bireyler ve etkileşimlere**

Kapsamlı dokümantasyondan ziyade **çalışan yazılıma**

Sözleşme pazarlıklarından ziyade **müşteri ile işbirliğine**

Bir plana bağlı kalmaktan ziyade **değişime karşılık vermeye,**

Değer verildiği vurgulanmıştır.



Çevik yazılım geliştirme modelinde büyük projeler daha küçük parçalara bölünür bu parçalar yinelemeler (iterations) olarak adlandırılır. Her yineleme, çevik yazılım geliştirme sürecinin tipik aşamalarından geçer. Bu aşamalar planlama, tasarım, gerçekleştirme, test, dağıtım ve bakımdır. Her yinelemenin tamamlanması yaklaşık olarak 2-4 hafta sürer ve sonra piyasaya sürülüp geri bildirimler alınır. Bu yönleriyle çevik yaşam geliştirme modeli verimli, esnek, hızlı ve oldukça kullanışlıdır. Çevik yazılım geliştirme modeli kapsamına giren bir çok metodoloji vardır. Bunlardan en popülerleri Extreme Programing(xp) ve Scrum’dır.

**Extreme Programing(XP)**

Extreme Programing(XP), en popüler çevik yazılım geliştirme metodolojilerinden biridir. Yazılım kalitesini arttırmak ve müşteri gereksinimlerini en iyi şekilde yanıt vermek için kullanılır. Geri bildirim ve iletişime oldukça önem verir. XP’nin 5 temel değeri vardır. Bunlar:

1. Basitlik
2. İletişim
3. Kullanıcı geri bildirimi
4. Cesaret
5. Saygı

**Basitlik**

XP de kullanılacak en basit ve en verimli yöntem seçilmelidir. Karmaşık çözümler XP’ye aykırıdır. Yazılımın o anki gereksinimlerini minimum şekilde karşılayacak en basit çözüm doğrudur.

**İletişim**

İletişim projenin başarılı olması için gerekli olan en önemli faktörlerden biridir. Ancak doğru ve karşılıklı anlaşılan bir iletişim sorunların çözümüne yardımcı olacaktır. Bu yüzden XP, yüzyüze iletişime oldukça önem verir. Ürünün üreticileri ve kullanıcıları arasında sağlıklı bir iletişimin sağlanması ürünün başarılı,gereksinimleri tam olarak karşılama olasılığını arttır

**Kullanıcı geri bildirimi**

Ürünün son kullanıcıya ulaşması ile birlikte kullanıcılardan geri bildirim almak oldukça önemlidir. Bu geri bildirimler sayesinde ortaya koyulan yazılım ürününün, hatalarından arındırılması ve ürüne yeni özelliklerin eklenmesi işleri kolaylaştırır.

**Cesaret**

Cesaret belki de XP’nin en zor değerinden biridir. Cesaret, başarıyı sağlamak için oldukça önemli bir adımdır. Yazılım mühendisleri yeni teknolojilere, yeni hatalara, yeni sorunlarla baş etmek için cesaretli olmalıdır. Hata yapmaktan korkmamalı ve hatanın kaynağını araştırmalıdır.

**Saygı**

Çalışanların kendilerine saygı duyması yapılan işin kalitesini arttırır. Saygı, ekip arkadaşlarınızla bağlanmanıza ve daha iyi bir çalışma ortamının oluşmasını sağlar. Bu ortaya konulan ürünün kalitesini de etkileyecektir.

**Scrum**

Scrum, büyük, riskli ve karmaşık yazılım projelerini yinelemeli bir şekilde geliştirmeyi öngören bir çevik yazılım geliştirme metodolojisidir. Scrum, yazılım işlerini **sprint** adı verilen zaman dilimlerine böler. Her sprint 2-4 hafta arası sürer. Bu şekilde karmaşık ve yönetilmesi zor olan projeler parçalara ayrılarak yönetilmesi kolay hale getirilir. Scrum metodolojisinde 3 önemli rol vardır. Bunlar ürün sahibi, scrum yöneticisi ve scrum takımıdır.

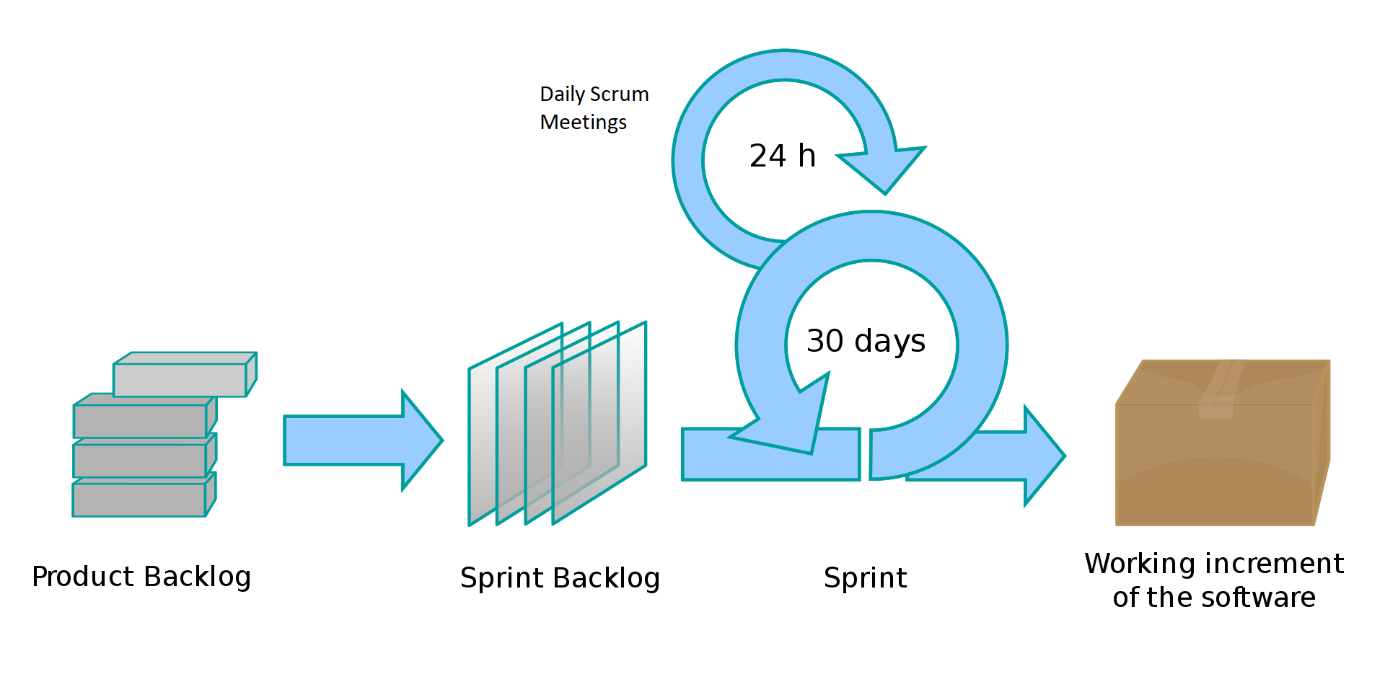
**Ürün sahibi**

Ürün sahibi, müşterilerden ve kullanıcılardan geri dönüşleri alır. Gereksinimleri, öncelikleri ve Product Backlog’u belirler.

**Scrum Yöneticisi**

Scrum’un kurallarına, yapısına hâkim ve bunları uygulamasından sorumlu kişidir.

**Scrum Takımı**

Her sprint başında belirlenen hedefe birlikte yürüten, projeyi geliştiren ve gerekli teknik bilgiye sahip bir grup profesyonel kişiler.

**Scrum Daily Meetings (Günlük Scrum Toplantısı)**

Scrum Daily Meetings, her gün aynı yerde ve aynı saatte ayakta gerçekleştirilen kısa bir toplantıdır. Bu toplantıda takım, bir önceki gün yapılan işleri gözden geçirir ve sonraki 24 saatte için hangi işlerin yapılıp yapılmayacağı planlanır. Yaşanan sorunlar hakkında tartışma yapılır ve çözüm aranır.

**Product Backlog (Ürün Biriktirme Listesi)**

Product Backlog, birikmiş işlerin müşteri ile anlaşılıp yüksek gereksinimli işlerin önceliklendirildiği bir listedir.

**Sprint backlog (Sprint Biriktirme Listesi)**

Geliştirme takımı tarafından, üzerinde çalışacakları sprint sırasında gerçekleştirmek için önceliklerine göre sıralanmış bir yapılacaklar listesidir.

**KAYNAKLAR**

* [**https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696**](https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696)
* [**https://www.tutorialspoint.com/sdlc/sdlc\_overview.htm**](https://www.tutorialspoint.com/sdlc/sdlc_overview.htm)
* [**https://stackify.com/what-is-sdlc/**](https://stackify.com/what-is-sdlc/)
* [**https://blog.codegiant.io/software-development-life-cycle-the-ultimate-guide-2020-153d17bb20fb**](https://blog.codegiant.io/software-development-life-cycle-the-ultimate-guide-2020-153d17bb20fb)
* [**https://blog.codegiant.io/software-development-life-cycle-the-ultimate-guide-2020-153d17bb20fb**](https://blog.codegiant.io/software-development-life-cycle-the-ultimate-guide-2020-153d17bb20fb)
* [**https://www.intellectsoft.net/blog/what-is-system-development-life-cycle/**](https://www.intellectsoft.net/blog/what-is-system-development-life-cycle/)
* [**https://www.konstantinfo.com/blog/software-development-life-cycle/#:~:text=SDLC%20Methodologies%20are%20used%20to,primitive%20approach%20for%20software%20development**](https://www.konstantinfo.com/blog/software-development-life-cycle/#:~:text=SDLC%20Methodologies%20are%20used%20to,primitive%20approach%20for%20software%20development)**.**
* [**https://www.codex.com.tr/yazilim-gelistirme-modelleri**](https://www.codex.com.tr/yazilim-gelistirme-modelleri)
* [**https://medium.com/architectural-patterns/yazılım-geliştirme-modelleri-62915545c51e**](https://medium.com/architectural-patterns/yazılım-geliştirme-modelleri-62915545c51e)
* [**https://agiletech.vn/top-software-development-life-cycle-models/**](https://agiletech.vn/top-software-development-life-cycle-models/)
* [**https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/**](https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/)
* [**https://existek.com/blog/sdlc-models/**](https://existek.com/blog/sdlc-models/)
* [**https://www.digite.com/agile/scrum-methodology/#:~:text=Scrum%20is%20an%20agile%20development,the%20development%20of%20the%20project**](https://www.digite.com/agile/scrum-methodology/#:~:text=Scrum%20is%20an%20agile%20development,the%20development%20of%20the%20project)**.**
* [**https://agilemanifesto.org**](https://agilemanifesto.org)
* [**https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696**](https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696)

**Medium:** [**https://sedatkimki.medium.com/yazılım-yaşam-döngü-modelleri-49c0225a0812**](https://sedatkimki.medium.com/yazılım-yaşam-döngü-modelleri-49c0225a0812)

**Github:**

**Sedat Korkmaz**

**200601046**