**Yazılım** **Geliştirme Yaşam Döngüsü Nedir?**

YazılımGeliştirme Yaşam Döngüsü (Software Development Life Cycle, SDLC), bir yazılım organizasyonu içerisinde yani bir yazılımın geliştirilmesi ve kullanım aşamasında, yazılım projesi için izlenen süreçtir. Belirli bir yazılımın; nasıl geliştirileceğini, korunacağını, değiştirileceğini veya iyileştirileceğini açıklayan ayrıntılı bir plan dizisinden oluşur. Yaşam döngüsü, yazılım kalitesini ve genel geliştirme sürecini iyileştirmek için gereken tüm yöntemlerin birleşimidir. Ayrıca, bu yöntemlerin uygulanacağı yapıyı da ele alır. Başka bir deyişle, bir yaşam döngüsü modeli; yazılımın, bir fikir olarak doğmasından kullanım dışı kalana kadar gerçekleştirilen çeşitli aşamaları içerir. Yazılımın üretildiği andan itibaren işlevsellik ile gereksinimleri sürekli değişecek ve bu değişiklikler de yazılımın genişlemesine neden olacaktır. Herhangi bir aşamada geriye dönmek, geliştirmeyi yapmak ve tekrar ilerlemek mümkün olduğundan bu aşamalar bir döngü olarak ifade edilmelidir.

Temel Yazılım Yaşam Döngüsü 6 aşamadan oluşur:



**Planlama ve gereksinim analizi (Planning and Requirement Analysis Phase):** İlk olarak gereksinim analizi ile yani müşterinin istekleri doğrultusunda ortaya çıkacak olan yazılım ile ilgili bilgilerin alındığı adım ile başlanır. Bu adımda müşteri iyi bir şekilde dinlenmeli, istekler tam ve net olarak belirlenmelidir. Daha sonra, yazılımın üretim aşamasında ihtiyaç duyulacak personel, donanım ve yazılım gereksinimleri çıkarılır, fizibilite çalışması yapılır ve proje planı oluşturulur.

**Analiz (Analysis Phase)**: Yazılım işlevleri ile gereksinimlerin ayrıntılı olarak çıkarıldığı aşamadır. Temel amaç, bir yazılım mühendisi gözüyle mevcut yapıdaki işlerin ortaya çıkarılması ve doğru olarak algılandığından emin olmaktır. Gereksinimler açık ve kesin olarak belirlendikten sonra belgelenir. Ayrıca bu belge alışıldık ve alışılmadık kullanım senaryolarını da içerir.

**Tasarım (Design Phase)** Analiz aşamasından gelen belge göz önüne alınarak gereksinimlere çözüm olacak yazılımın tasarımına başlanır. Kısaca analiz çalışması sonucunda üretilen mantıksal modelin fiziksel modele dönüştürülme çalışmasıdır. Sistem mimarisi tasarımı ve uygulama planı da diyebileceğimiz daha detaylı tasarım şeklinde iki tür tasarım vardır. Sistem mimarisi tasarımında sistem bileşenleri, ara yüzleri ve davranışları tanımlanır. Bu aşamada liste, yığın, ağaç, grafik gibi ilkeller kullanılabilir. Detaylı tasarım da ise kullanılacak program dilleri, platformlar, veri yapıları, ekran tasarımları gibi ayrıntılı uygulama detayları belirlenir.

**Uygulama/Geliştirme (Coding Phase):** Tasarım aşamasında alınan kararlar doğrultusunda kodlama yapılır.

**Test (Testing and Integration Phase):** Kodlama tamamlandıktan sonra, geliştirilen yazılım kapsamlı bir şekilde test edilir ve bulunan tüm hatalar (defect) düzeltilmesi için geliştiriciye atanır. Yeniden test (retesting) ve regresyon testi yazılımın hatasız çalıştığından emin olana kadar yapılır. Müşteri geliştirilen yazılımı test eder ve isteklerinin karşılandığından emin olur. Sonrasında ürün kurulumu yapılır.

**Bakım (Maintenance Phase):** Ürün teslimi ile bakım aşaması başlar ve yazılımın son kullanıldığı güne kadar devam eder. Yazılım kullanılırken karşılaşılan hataların çözümlenmesi, ürüne yeni özellikler eklenmesi gibi yazılımın güncel ve etkin kullanımının sürdürülmesi için geliştiriciler tarafından yapılan iyileştirmelerdir.

**Yazılım Geliştirme Yaşam Döngüsü Modelleri**

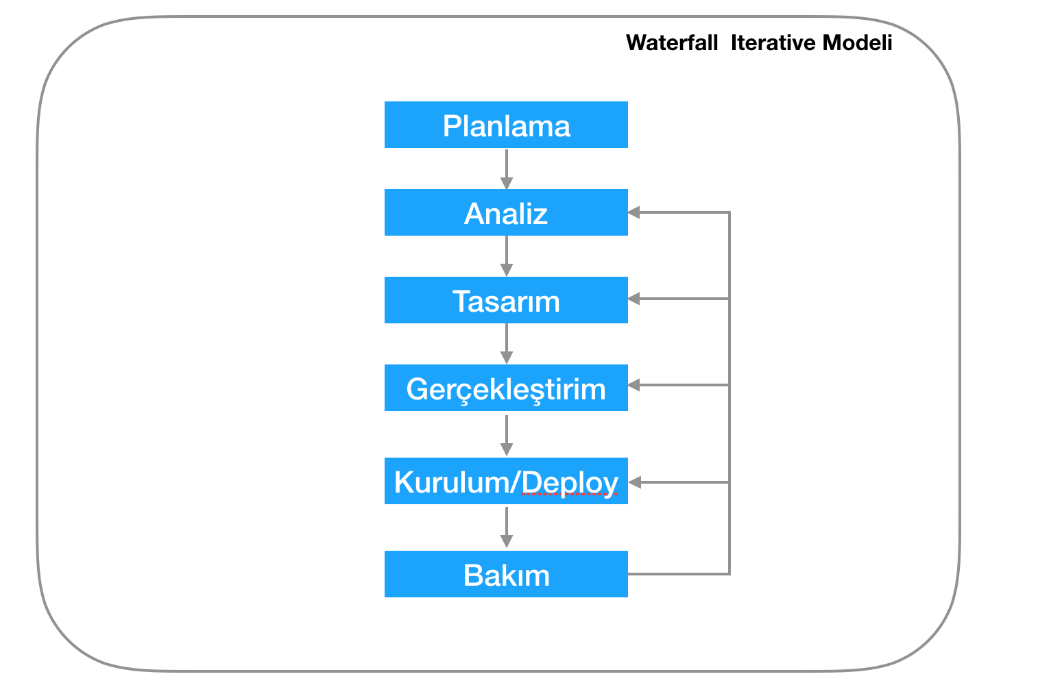
Yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlayan modellerdir. Her modelin kendine özgü avantaj ve dezavantajları vardır. Bu yüzden projenin gereksinimlerine göre doğru model seçilmelidir.

1. **Gelişigüzel Model**

Bu yazılım geliştirme modelinde belirlenmiş bir model ya da yöntem bulunmaz. Kişiye bağlı yazılım geliştirme şeklindir. Bu yüzden yazılımın takip edilebilirliği ve bakım yapılabilirliği oldukça zordur. 1960'lı yıllarda uygulanan bu yöntem, genellikle basit programlama içeren ve çoğunlukla tek bir kişinin yazılım geliştirirken kullandığı yöntemdir.

1. **Çağlayan Modeli (Waterfall Model)**

Çağlayan Modeli, yazılım geliştirme için en yaygın olarak kullanılan ve bilinen en eski modeldir. Geleneksel yazılım geliştirme modeli olarak da bilinir. Bu model sıralı, doğrusal bir akışta, aşağı doğru ilerlediği için çağlayan/ şelale modeli denmiştir.



Çağlayan modeli, tüm aşamaların adım adım yapıldığı kolay anlaşılabilen basit bir modeldir. Çağlayan Modelinde, belirtilen adımlar kendi içerisinde tamamlandıktan sonra bir sonraki aşamaya geçilebilir. Her adımın sonunda o adım test edilir ve belge oluşturulur. Gereksinim aşamasında müşterinin istekleri detaylı, açık ve kesin bir şekilde alınan ve kısa bir süre içerisinde yetişmesi gereken projeler için uygun bir modeldir. Sadece başlangıç aşaması olan Gereksinim Aşamasında müşteri ile iletişim kurulduğundan küçük bir sorun, proje ilerlediğinde düzeltilmesi zor boyutlara gelebilir. İlerleyen safhalarda projede değişiklik yapılırsa tüm aşamaların tekrarlanması, değişmesi gerekir. Bu da maliyetin artmasına neden olur.

1. **Barok Modeli**

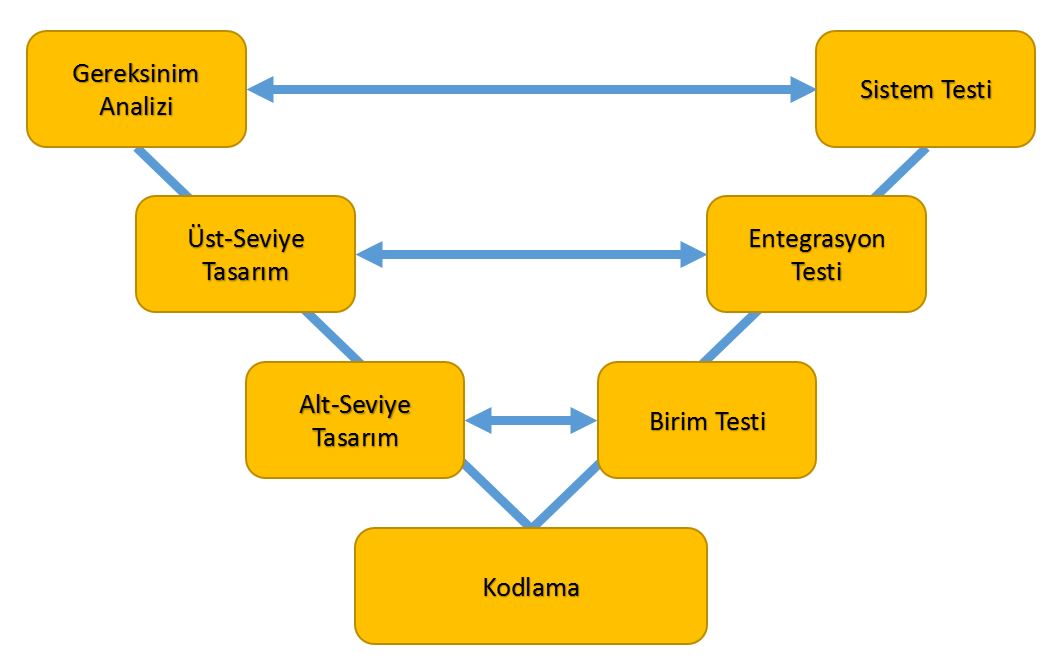
Temel Yaşam Döngüsü adımlarının döngü olarak değil, doğrusal bir şekilde geliştirildiği modeldir. Bu modelde aşamalar arasındaki geri dönüşlerin nasıl yapılacağı belirsizdir. Günümüzde kullanılan yazılım geliştirme modellerinden farklı olarak belgeleme ayrı bir süreç olarak ele alınır. Geliştirme/ Kodlama ve test aşamasından sonra ürün teslim edilmeden belgeleme yapılır. Günümüzde müşteri ile iletişim daha aktif olduğu için tercih edilmemektedir.

1. **V Süreç Modeli (V- Shaped Model)**

Tıpkı Çağlayan Modelinde olduğu gibi V Süreç Modeli, süreçlerin yürütülmesinin sıralı bir yoludur. Bir aşama tamamlanmadan bir sonraki aşama başlamaz. Bu modelde test, şelale modelinden daha fazla vurgulanmaktadır. Test prosedürleri, uygulamadan önceki aşamaların her birinde, herhangi bir kodlama yapılmadan önce geliştirilir.

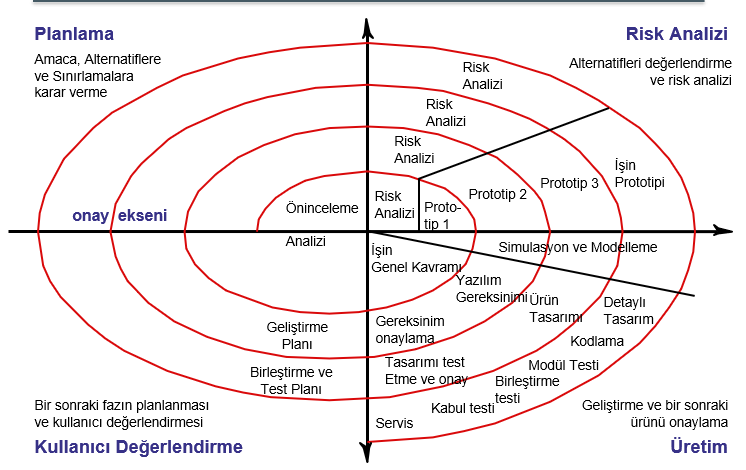
Şelale modeli gibi, gereksinimler V Süreç yaşam döngüsü modelini başlatır. Geliştirme başlamadan önce bir sistem test planı oluşturulur. Test planı, gereksinimlerin toplanmasında belirtilen işlevselliği karşılamaya odaklanır. Üst düzey tasarım aşaması, sistem mimarisi ve tasarımına odaklanır. Bu aşamada da yazılım sistemlerinin parçalarının birlikte çalışabilme yeteneğini test etmek için bir entegrasyon test planı oluşturulur. Düşük seviyeli tasarım aşaması; gerçek yazılım bileşenlerinin tasarlandığı ve bu aşamada da birim testlerinin oluşturulduğu yerdir. Kodlama aşamasında tasarlanan yazılım kodlanır. Kodlama tamamlandığında, daha önce geliştirilen test planlarının şimdi kullanıma sunulduğu V'nin sağ tarafından adımlar ilerler.

V Süreç Modeli, ilk aşamalarda test planlarının geliştirilmesi nedeniyle Çağlayan Modeline göre başarı şansı daha yüksektir. Belirsizliklerin az, iyi tanımlanmış Bilgi Teknoloji (BT) projeleri için uygun bir modeldir.



1. **Helezonik Model (Spiral Life Cycle Model)**

Süreç, spiralin merkezinden başlar ve dönerek Spiral Model görselindeki adımlarla ilerler. Bir defa istekler ve ihtiyaçlar belirlenir. Daha sonra risk analizleri yapılıp prototip çıkartılır. Bu prototipten sonra tekrar bunun ihtiyaçlarla ilgili kontrolü yapılıp doğrulama ve onaylama aşamasından geçirilip bir daha yazılım sürecinden geçirilir. Yine bir risk analizi yapılıp prototip 2 çıkarılır. Sonra bu prototipin test planı ve risk analizi yapılır. Sonrasında bir prototip daha çıkarılır. Projenin büyüklüğüne göre bu adımların sayısı artırılabilir veya tek bir prototip de yeterli görülebilir. Eğer son prototipimizin müşteri feedbacki ve risk analizleri istenilen gibiyse sırasıyla detaylı tasarım, kodlama, entegrasyon, testler ve uygulama süreci başlar. Bu son aşama Çağlayan modeline benzer bir yapıda işler. Spiral model buraya kadar bize değişik döngülerden geçen risk analizlerinin yapıldığı ve sonunda bir prototipin ortaya çıktığı, hem yazılım geliştirme ekibinin ve proje yönetim ekibinin hem de müşterinin üzerinde mutabık olduğu bir prototip ve sonrasında ürün ortaya çıkarır. Görseldeki oklar yukarı doğru gittikçe maliyetin büyümesi söz konusu ve sola doğru gittikçe de gözden geçirme süreçlerinin arttırılması söz konusudur.



Bu spirali 4’e böldüğümüzde 4 aşaması vardır.

1. Planlama

2. Risk Analizi

3. Üretim

4. Kullanıcı Değerlendirmesi

Her fazda, müşteriye ara prototiplerin gösterilmesi ve risklerin analiz edilmesi projenin istek ve ihtiyaçları karşılamasına yardımcı olur. Bu yüzden bu model büyük ve karmaşık projeler için uygun bir modeldir.

1. **Yinelemeli/ Artımsal Geliştirme Modeli (Iterative / Incremental Model)**

Artımsal Modelde, ürün tek seferde değil parça parça teslim edilir. Gereksinimler önemlerine ve birbirine bağlılıklarına göre sıralanır ve öncelikle istenilen gereksinimi karşılayabilecek işlevdeki sistem teslim edilir. Her yinelemede bunların bir kısmı tamamlanır. Sistemin sonraki her sürümü, tasarlanan tüm işlevler uygulanana kadar önceki sürüme işlev ekler. Eklenecek sürümlerin projenin geri kalanını kapsayacak bir biçimde tasarlanması önemlidir. İlk teslim edilen öncelikli gereksinimleri karşılayan sisteme, her yinelemede farklı işlevli yazılımlar ekleneceği için tasarımı önemlidir. Bu yüzden daha çok test edilir.

Birçok detay içeren, uzun bir proje tasarlanacak ise Artımsal Model sayesinde kısa bir sürede en gerekli ihtiyaçları karşılayan bir yazılım tasarlanabilir. Küçük gruplar halinde tasarlandığı için üzerinde rahatlıkla değişiklik yapılabilir, esnektir ve risk yönetimi daha kolay olur. Bunun sayesinde maliyet azalır. Tasarlanacak parçaları oluşturmak için sistemin detaylı bir şekilde tanımlanması gerekir.

Artımsal geliştirme yaşam-döngü modelindeki temel süreç akışı aşağıda verilen şekildedir.



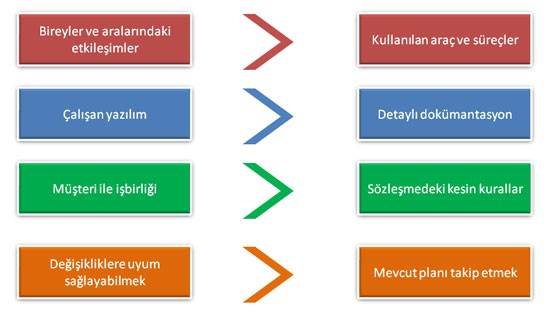
1. **Kodla ve Düzelt Yaşam-Döngü Modeli**

Kodla ve Düzelt modelinde yazılım yaşam döngüsünün ilk aşaması olan gereksinim analizinden hemen sonra kodlama aşamasına geçilir. Bu model üzerinde gerekli düzenlemeler, gereksinimler karşılanıncaya kadar devam eder. Yazılım geliştirmenin en kolay yoludur ancak herhangi bir yöntem izlenmeden kodlama yapıldığı için gereksinimleri fazla olan bir projede karmaşıklık çıkması doğaldır. Herhangi bir belgeleme de yapılmadığı için bu model bireysel geliştiriciler için daha uygundur. Ayrıca üzerine iyileştirici ve yenilikçi eklentilerin yapılması zor olduğundan, bu model kullanılarak geliştirilen yazılımın yaşam süresi az olur ve emeklilik safhası kaçınılmazdır.

Bu modelde analiz aşamasından sonra diğer yazılım yaşam döngü aşamaları atlanıp direkt yazılım geliştirildiği için müşteri memnuniyetini sağlamak daha kolaydır. Dokümantasyon olmadığı için bakımı ve değişimi çok zordur, dolayısıyla pahalı bir yöntemdir.

1. **Çevik Yazılım Geliştirme Modeli (Agile Model)**

1990’larda iş gereksinimleri, müşterilerin istediği uygulamalar ve özellikler ile müşterilerin isteklerine cevap veren teknolojilerin teslimi arasında önemli ölçüde bir gecikme oluşmuş ve bu gecikme çok fazla projenin iptal edilmesine neden olmuştur. Bu gecikme süresinde, iş gereksinimleri ve müşterilerin istekleri değişmiş ve nihai ürün, güncel ihtiyaçları karşılamakta yetersiz kalmıştır. Bu dönemde, Çağlayan modelinin öncülüğünü yaptığı geleneksel yazılım geliştirme modelleri, hız talebini karşılamakta ve yazılımların hızlı bir şekilde değiştirebilmesinde yetersiz kalmıştır. Bunun üzerine 2001 yılında Kent Beck ve Robert Martin’inde içinde bulunduğu 17 kişilik bir grup, yazılım geliştirme sürecinde daha üretken ve verimli işler sunabilmek adına Agile Manifesto’yu ve Çevik Yazılım’ın 12 Prensibini yazıya geçirdiler. Bu manifestoda;



Sağdaki maddelerden çok soldaki maddelere önem verdiklerini belirtmişlerdir.

Genel olarak Çevik Yazılımın Temel Prensipleri: yüz yüze iletişime önem vermek, müşteriyi memnun etmek, değişen ihtiyaçları karşılayabilecek, esnek yazılım geliştirmek, sık aralıklarla ürün teslimi yaparak müşteri ile iş birliği kurmak, yazılım geliştirirken kolay anlaşılabilir, sade ve düzgün çalışan tasarıma odaklanmaktır.

Çevik Model, Yinelemeli Modele benzer. Çevik Modelde de bir ürün, küçük yinelemeli yapılara bölünür. Tek seferde eksiksiz bir ürün olarak geliştirilmez. Bu sayede hatalar daha kolay bir şekilde düzeltilebilir. Bu model, ürün geliştirirken gereksinimden çok esnekliğe odaklanır. Yinelemeli yapı, önceki işlevsellik üzerine inşa edilir. Her yinelemenin 2-4 hafta sürmesi planlanır. Yinelemelerin sonunda, müşteriye projenin ilerleyişi hakkında bilgi verilir ve bitmiş yineleme teslim edilir. Müşteriye en kısa sürede tamamlanan, çalışan yazılım teslim edilmesi ve projenin ilerleyişinde iş birliği kurulması verimi ve müşteri memnuniyetini artırır.

Çevik Model kullanarak yazılım geliştiren ekip genellikle küçük gruplardır. Ekibin kaliteli bilgi akışı için yüz yüze iletişim kurması önemlidir. Tüm ekip elemanları birlikte çalışırlar. Projeyi etkin bir şekilde yürütmek için ekip, düzenli aralıklarla kendi yöntemlerini gözden geçirir, iyileştirmelerini yapar. Kendini organize edebilen, motivasyonu yüksek kişilerden oluşması da ekip başarısını artıran etkenlerdir.

En yaygın kullanılan Çevik Yazılım Geliştirme Metodolojileri şunlardır:

* SCRUM
* Extreme Programming (XP)
* Özellik Odaklı Geliştirme (Feature-Driven Development-FDD)
* Test Odaklı Geliştirme (Test-Driven Development-TDD)
* Çevik Birleştirilmiş Süreç (Agile Unified Process)
* LEAN Development
* Dynamic System Development Methodology (DSDM)
* Microsoft Solution Framework (MSF)

**SCRUM**

Scrum, 1995’te Ken Schwaber ve Jeffrey Sutherland tarafından geliştirilmiştir. Scrum genellikle bir metodoloji olarak tanımlanır fakat Scrum bir süreci yönetmek için tercih edilen çerçeve olarak düşünmek daha doğru olacaktır. Yani sadece yazılım geliştirmek için kullanılmayabilir.

Scrum, proje yönetim takımlarının herhangi bir değişikliğe kolay uyum sağlamak ve bunlara cevap vermek için daha etkili ve hızlı hale getirmek için tasarlanmış yaklaşımdır. Çevik bir yaklaşım olan Scrum, takımların çalışmalarını döngüsel ve yinelemeli bir şekilde gerçekleştirmelerini gerektirir, böylece önceki döngülerde öğrendikleri her yeni fikri hızlı bir şekilde projeye entegre edebilirler. Bu özel çevik yaklaşımı, deneysel süreç kullanılması nedeniyle diğer çevik yaklaşımlardan farklıdır.

Scrum’da yazılım süreçleri küçük birimlere yani sprintlere bölünerek geliştirilir. Bu nedenle büyük ve karışık projelerin yönetimi için uygulanmaktadır.

Diğer Çevik Yaklaşımlar öngörülen verilere dayanırken Scrum’da her sprintte takımlar, ne yapmaları gerektiğini belirlemek için gerçek ve gerçek zamanlı bilgileri kullanırlar. Bu yüzden Scrum diğer birçok yaklaşımdan daha verimli kabul edilir.

**SCRUM’daki Temel Kavramlar**

1. **Roller**

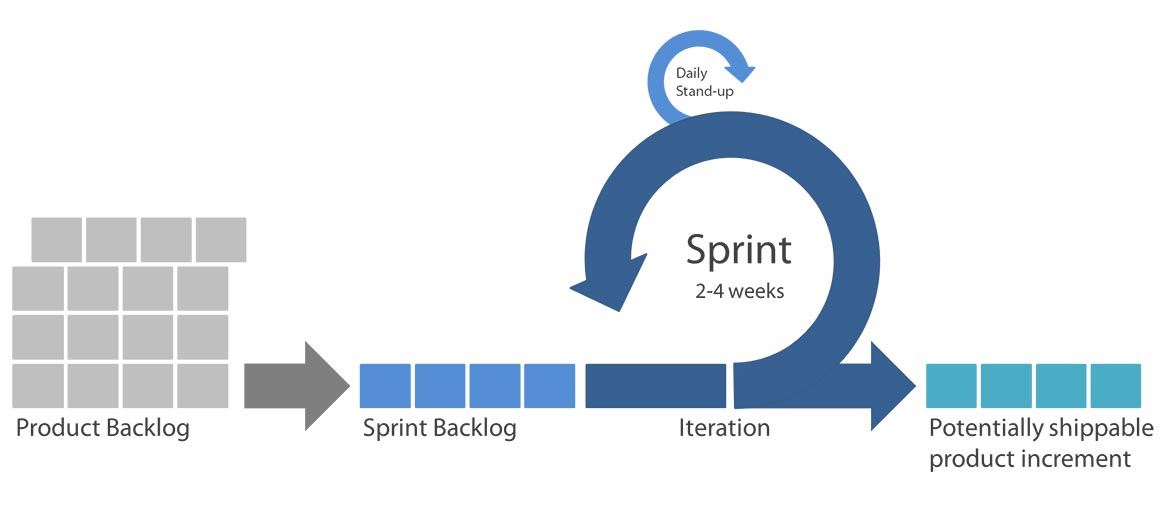
* **Ürün Sahibi (Product Owner):** Projenin gereksinimlerini belirlemekle ve geri dönüş yapmak ile sorumludur.
* **Scrum Yöneticisi (Scrum Manager):** Scrum kuralları, teori ve pratiklerini iyi bilen ve takımın genel olarak bu kuralları uygulamasından sorumludur.
* **Scrum Takımı (Scrum Team):** Yazılım geliştirirler. Genelde 5-9 kişiden oluşur. Scrum takımları, birbirleri ile daima fikir alışverişinde ve iletişimde olmalıdırlar.

1. **Toplantılar (Meetings)**

* **Sprint Planlama (Sprint Planning):** Öncelikli gereksinim listesi çıkarılır. Belirlenen gereksinimler analiz edilir ve olası gereksinim değişiklikleri belirlenir. Takımlar belirlenir. Risk değerlendirilmesi yapılır. Yazılım geliştirme araçlarına ve altyapılarına karar verilir. Fizibilite çalışmaları yapılır.
* **Sprint Gözden Geçirme (Sprint Review):** Her sprint başlangıcında planlama toplantısı yapılır. Ürün sahibi ile Scrum takımı bir sonraki tasarlanacak sprint için gereksinim listesini gözden geçirir, içeriklerini belirler. Daha sonra Scrum takımı öncelikle tasarlanması gereken gereksinimden başlayarak sprint sonunda gerçekleştirilecek gereksinimlerden Sprint Gereksinim Listesi’ni oluştururlar. Takım Sprint Gereksinim Listesi’nden hangilerini ve nasıl yapacaklarına karar verirler.
* **Günlük Scrum Toplantısı (Daily Scrum Meeting):** Sprint başladıktan sonra tüm takım genellikle sabahları yapılan bu 15 dakikalık kısa toplantılara katılırlar. Scrum takımı dün neler yaptığını, bugün neler yapacağını veya bir problemin olup olmadığı gibi projenin genel işleyişi hakkında bilgi alışverişi yapar.

1. **Bileşenler/ Araçlar (Artifacts)**

* **Ürün Gereksinim Dokümanı (Product Backlog):** Kullanıcının belirlediği gereksinimlerden oluşur. Süreç ilerledikçe yeni elemanlar eklenebilir veya artık gereksiz, anlamsız gelen elemanlar silinebilir. Projenin başında gereksinimler öncelikli ihtiyaca göre sıralanır.
* **Sprint Dokümanı (Sprint Backlog):** Ürün Gereksinim Dokümanındaki görevler sıradaki sprint için belirlenir. Bu dokümandaki işlerin amacı sprint sonunda çalışır, işlevsel bir parça elde edebilmektir. Sadece takım üyeleri tarafından değiştirilebilir.
* **Sprint Kalan Zaman Grafiği (Burn Down Chart):** Bu grafik sprint boyunca işlerin ne kadarının yapıldığı ile olması gereken ilerlemenin karşılaştırılmasını sağlar.



**Scrum Neden Popüler?**

Scrum’da koşu da denilen sprintler, değişen gereksinimler ve yapılan değişiklikler için esneklik sağlar. Takım içi iletişimin ön planda olması ve günlük Scrum toplantılarının yapılması sorunların hızlı bir şekilde çözülmesini sağlar. Böylece maliyet azalır ve kalite artar. Müşterilerin proje tasarlanırken aktif olması gereksinimlerin tam olarak karşılanmasını sağlar. Projenin geleneksel yazılım geliştirme modellerinden farklı olarak parçalı bir şekilde geliştirilip, test edilmesi hataların ileride daha fazla maliyete yol açmasını önler. Bu gibi nedenler kısa sürede verimli işler çıkarılmasını sağlayan Scrum’ın günümüzde en çok tercih edilen yazılım geliştirme modeli olmasına neden olur.

**Hangi Projede Hangi Model Kullanılmalı?**

Çağlayan Modeli: Gereksinimleri çok iyi tanımlanmış, daha sonra değişiklik yapılmayacak projelerde kullanılabilir.

Kodla ve Düzelt Modeli: Sadece kodlama aşaması olduğu için uygulaması çok kolaydır. Küçük projeler ve bireysel geliştiriciler için uygun bir modeldir.

Çevik Yazılım Geliştirme Modeli: Proje küçük parçalara bölünüp tasarlandığı için büyük, karmaşık projelerde kullanılabilir. Değişimlere açık ve esnektir. Müşteri ile süreç boyunca iş birliği yapıldığı için memnuniyet sağlanır.

V Süreç Modeli: Açık bir şekilde tanımlanan Bilgi Teknolojileri projeleri için uygun bir modeldir.

Spiral Model: Belirsiz ihtiyaçları olan büyük ve karmaşık projelerde kullanılabilir. Ar-ge faaliyetleri gibi.

Medium: <https://busracetinkaya.medium.com/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-nedir-78f43b0cf83e>

Büşra Çetinkaya 200601067

**KAYNAKÇA:**

Run Burback, (1998). “SOFTWARE ENGINEERING METHODOLOGY: THE WATERSLUICE”, Stanford Üniversitesi.

Vanshika Rastogi, (2015). “International Journal of Computer Science and Information Technologies”.

Yrd.Doç.Dr. Yunus Emre (2017). “Yazılım Mühendisliği Temelleri”.

Sadi Evren Şeker (2015). “Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü”, İstanbul Medeniyet Üniversitesi.

<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>

<https://medium.com/@melsatar/software-development-life-cycle-models-and-methodologies-297cfe616a3a>

<https://medium.com/@fatihazir/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-ve-scrum-a17aef990c4c>

<https://www.netinbag.com/tr/internet/what-is-scrum-methodology.html>

<https://www.tutorialspoint.com/sdlc/sdlc_overview.htm>

<https://caglartelef.com/yazilim-yasam-dongusu/>

<https://www.softwaretestinghelp.com/software-development-life-cycle-sdlc/>

<https://www.acmagile.com/agile-nedir/>

<https://enprobilisim.com/yazilim-gelistirme-sureci-modelleri-sdmp/>

<https://www.javatpoint.com/software-engineering-software-development-life-cycle#:~:text=A%20software%20life%20cycle%20model,through%20its%20life%20cycle%20stages>

<https://bilginc.com/blog/scrum-nedir.jpg>