**T.C**

**İZMİR BAKIRÇAY ÜNİVERSİTESİ**

**MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**LİSANS PROJESİ**

**YAZILIM YAŞAM DÖNGÜSÜ**

**MODELLERİ**

**HAZIRLAYAN**

**NİHAT KEREM BORA**

**190601030**

**DANIŞMAN**

**DOÇ. DR. DENİZ KILINÇ**

**İZMİR**

**2021**

**Yazılım Yaşam Döngüsü Nedir?**

Yazılımın hem üretim hem de müşterideki kullanım süreci boyunca geçirdiği tüm aşamalar yazılım geliştirme yaşam döngüsü olarak adlandırılır. **planlama**, **analiz**, **tasarım**, **gerçekleştirim(kodlama ve test)**, **teslim ve bakım** olmak üzere 5 temel adımdan oluşur.

* **Planlama:** Temel ihtiyaçlara göre **fizibilite** çalışması yapılır ve projenin planlanması yapılır.
* **Analiz:** Sistemin işlevleri ve kesin gereksinimleri belirli formatta **doküman** edilir.
* **Tasarım:** Gereksinimleri karşılamak üzere sistemin özellikleri, yetenekleri ve arayüzleri belirlenir. “Nasıl yapılacak?” sorusuna cevap aranır.
* **Gerçekleştirim:** Kodlama aşaması, oluşturulan tasarımın esas alınarak ürünü oluşturma aşamasıdır. Test aşaması, ürünün gereksinimleri hatasız yapıp yapamadığı kontrol edilmesidir.
* **Teslim ve bakım:** Ürünün kullanılabilir bir versiyonu çıkartılarak teslim edilir. Ürünün teslim edildiği andan kullanılamaz hale gelene kadar dahil olduğu gerekli güncellemelerin yapıldığı adıma bakım aşaması denir.

**Yazılım Yaşam Döngüsü modelleri nedir?**

Yazılım yaşam döngüsü modelleri , yazılım döngüsünün aşamalarını ve bu aşamaların yürütüldüğü sırayı tanımlar. Her aşama, yaşam döngüsünün bir sonraki aşaması için gerekli olan gereksinimleri üretir. Müşterinin Gereksinimleri tasarıma dönüştürülür. Tasarımın gereksinimlerine göre kod üretilir. Üretilen kod, ürünün gereksinimlerine göre test edilir.

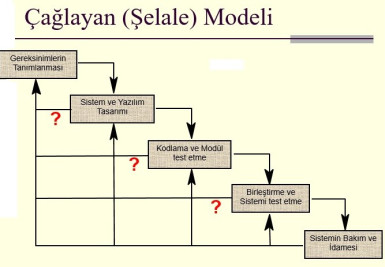
Üretilmek istenen ürüne ve üretimde izlenecek yola göre farklı yazılım yaşam döngüsü modelleri kullanılır. Belli başlı yazılım süreç modelleri aşağıdaki gibidir;

**Gelişigüzel Model:** Belli bir standart olmadan, kişinin kendi tercihleri ve gereksinimlerine göre kendi planlamasını yapmasıdır. Yalnızca geliştiren kişiye bağımlı olduğundan bu modelde üretilen ürünlerin izlenebilirliği oldukça zor bazen imkansızdır.

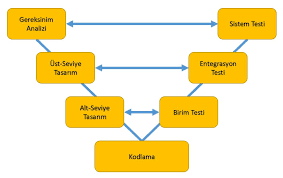
**Barok Modeli:** Yazılım yaşam döngüsü temel süreçlerini doğrusal bir biçimde gerçekleştirilmesini ister. Planlama , çözümleme, tasarım, ve gerçekleştirim aşamalarından oluşur. Her aşamadan sonra bilgilerin belgelenmesi bir süreç olarak ele alınır. Modelin zayıf tarafı aşamalar arası geri dönüşün nasıl yapılacağının tanımlı olmamasıdır.

**Kodla ve düzelt:** Genellikle kesinleşmemiş olan gereksinimlerle başlanır. Herhangi bir planlama yapılmaz. Uzmanlık gerektirmediği için herkes bu modeli kullanabilir. Ancak dokümantasyon olmadığı için kontrol edilemez veya hatalar bulunamaz. Çok küçük projeler ya da kısa ömürlü prototipler için uygundur

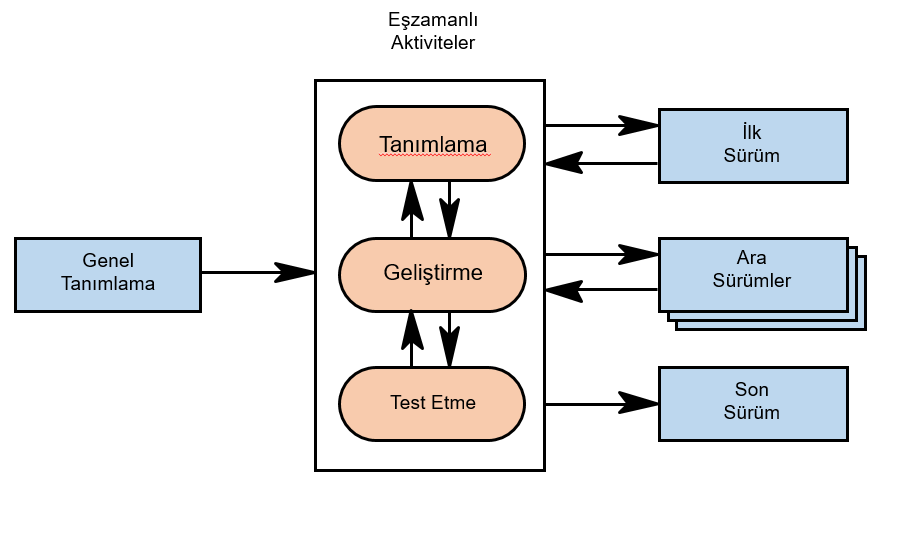
**Çağlayan Modeli:** Belirtilen yazılım yaşam döngüsü adımlarının baştan sona giderek gerçekleştirilmesinden oluşur. Her aşama tek seferde yapıldığı ve aşamalar arası geri dönüş olmadığından hataları tespit etmek zor ve maliyetlidir bu sebeple üretimi az zaman gerektiren projeler dışında kullanılmamakta **“Geleneksel Model”** olarak anılmaktadır.



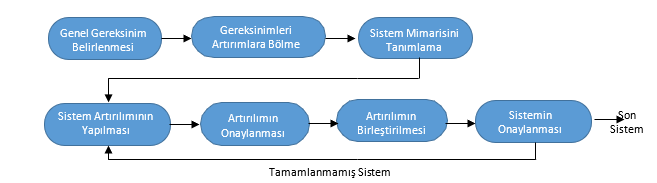
**V Modeli:** Çağlayan modelinin aşamalarını, “Üretim” ve “Sınama” olarak iki bölümde inceleyerek hangi işlevin ne zaman yapılacağı belirtilmiş halidir. Bu nedenle Çağlayan modelinin geliştirilmiş hali olarak düşünülebilir. V’nin sol tarafı üretimi, sağ tarafı ise sınama işlevlerini belirtir. Sınama işleminde hata bulunma durumunda aynı aşamadaki üretim işlevine dönülmektedir. İş tanımlarının belirgin olduğu Bilgi Teknolojileri için uygun bir modeldir.



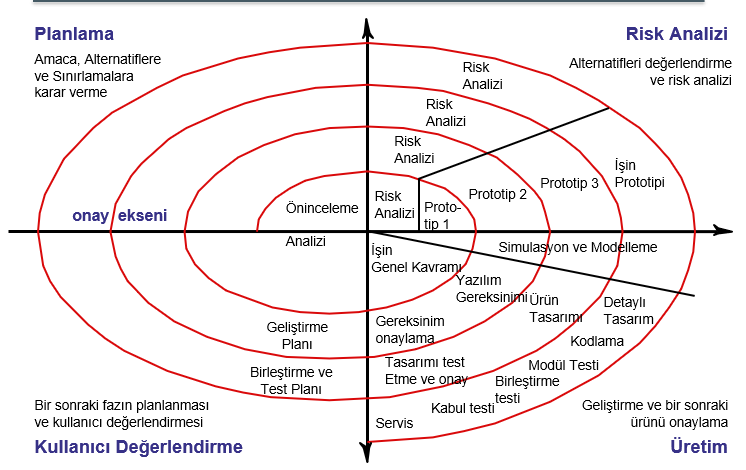
**Evrimsel Geliştirme modeli:** Sistem, müşteriyle koordine bir şekilde çalışarak kazanılan anlayışa göre gelişir. Sistem, en iyi bilinen gereksinimlerle oluşturulan prototipe yeni gereksinimlerin eklenmesiyle geliştirilir**. ”Ne istediğimi bilmiyorum ama görsem tanırım.”** mantığına dayanır. Sisteme sürekli ekleme yapıldığından gelişme süreci izlenebilir değildir. Küçük ve orta ölçekli, etkileşimli sistemler için uygulanabilir.



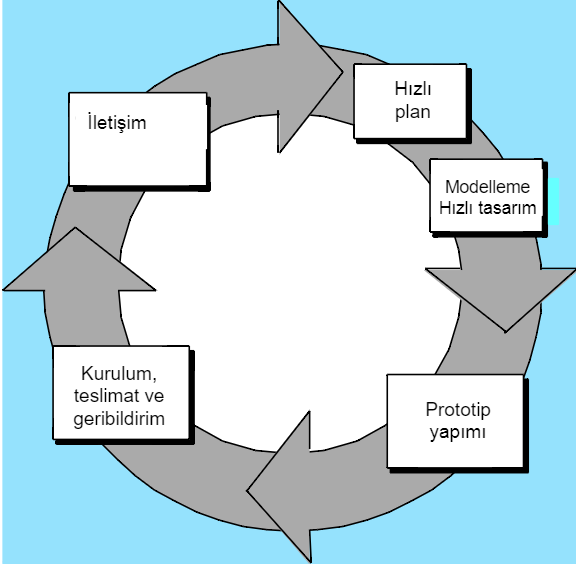
**Artırımlı Model:** Sistemi tek seferde teslim etmek yerine, geliştirme ve teslim parçalara bölünür ve bu parçalar kullanıcının önceliğine göre sıralanır. Parçalar sırasıyla geliştirilir ve teslim edilir. Her teslim edilen parça aynı zamanda bir ara ürün olduğundan kullanıcı tarafından kullanılabilir. Bu sayede bir taraftan üretim devam ederken tamamlanan parçalar ürün olarak kullanılabilir. Artırımlı model, proje adım adım üretildiğinden, geliştirmesi uzun zaman sürecek ve bu süreçte eksik işlevlerle kullanılabilecek ürünlerde tercih edilir.



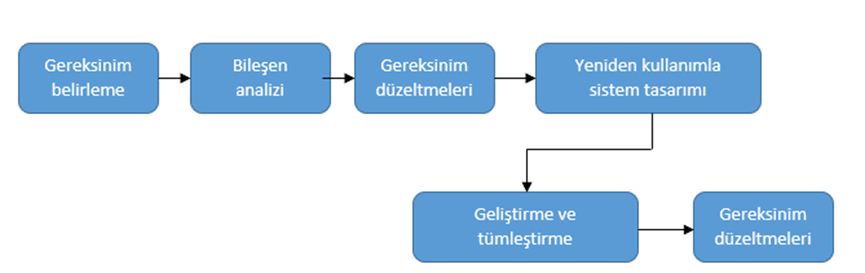
**Spiral Model:** Süreç, sıralı aşamalar yerine döngüsel olarak ele alınır. **Planlama**, **risk analizi**, **gerçekleştirim** ve **kullanıcı analizi** olmak üzere 4 bölümün spiral oluşturacak şekilde dönerek tamamlanması bir döngüyü temsil eder. Her döngü, süreçteki bir aşama olarak kabul edilir. Model, prototiplerden yararlanarak yüksek riskli ögelerin kullanıcıyla birlikte en az riskle üretilmesini hedefler. Bu yaklaşım hataları erken giderebilse de çok fazla ara adım içerdiğinden karmaşık ve maliyetlidir. Bu yüzden büyük ve yüksek riskli projelerde kullanılmalıdır.



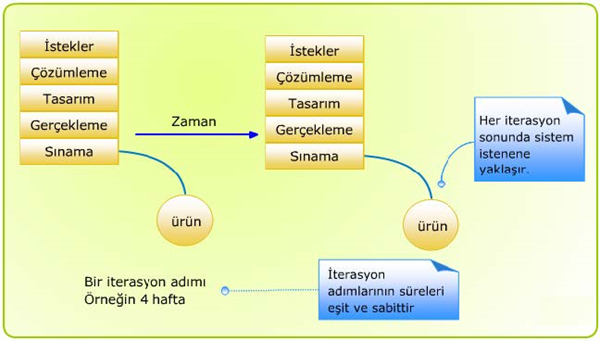
**Prototipleme:** Doğrusal modelin döngüsel versiyonudur. Gereksinimler toplanarak başlanılır ve gereksinimlere göre bir tasarım oluşturulur. Oluşturulan tasarıma göre ilk prototip üretilir ve kullanıcının kullanımına ve değerlendirmesine sunulur. Kullanıcı prototipi test eder ve düşüncelerini, ürünü müşterinin tam beklentilerine göre düzenleyen geliştiriciye iletir.​​ Bu döngü hedeflenen ürüne ulaşıncaya kadar devam eder. Önemli nokta her aşamayı hızlı bir şekilde tamamlamaktır. Kısaca prototipleme, doğrusal modeller gibi uzun süreli tek seferde ürünü oluşturmak yerine kısa süreli çok seferde ürünü oluşturmayı hedefler.



**Yeniden Kullanıma Yönelik Geliştirme:** Önceden üretilmiş yazılımların veya yazılımın bir kısmının kullanarak yeni bir ürün ortaya çıkarmayı hedefler. Önceden oluşturulduğundan, Maliyet denetimi ve kaynak kontrolü yapmak mümkündür. Önceden oluşturulan sınıflar tekrar kullanılabildiğinden kısa sürede yazılım geliştirme şansı sunar. Bunların yanında gereksinimlerini anlamak zordur ve Uzmanlık gerektirir. Gereksinimler doğru anlaşılamazsa istenilen hedefe ulaşılamaz ve proje başarısız olabilir.



**Birleşik süreç:** Nesneye dayalı yazılım geliştirmek için var olan yöntemlerin deneyimler sonucu kabul gören en iyi özellikleri bir araya getirilerek tümleştirilmiş yazılım geliştirme süreci oluşturulmuştur. Yinelemeli, arttırmalı, evrimsel ve risk güdümlüdür**. Başlangıç**, **ayrıntılandırma**, **tamamlama** ve **yayım** olmak üzere 4 aşamadan oluşur.

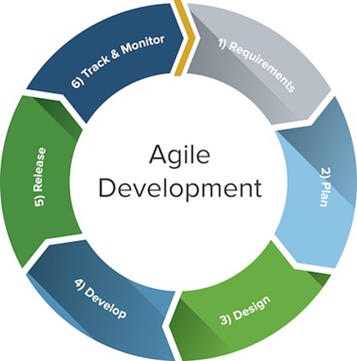


* **Başlangıç:** Vizyon kararı, fizibilite çalışması, tamam ya da devam kararı.
* **Ayrıntılandırma:** Daha gerçekçi çözümleme, çekirdek yapının ve yüksek riskli kısımların yinelemeli olarak oluşturulması.
* **Tamamlama:** Daha az riskli ve düşük öncelikli kısımların yinelemeli olarak gerçeklenmesi.
* **Yayım:** Beta testleri, piyasaya sürme çalışmaları.

En iyi özelliklerden bir araya getirilerek oluşturulduğundan süreci çok fazla yöntemle işletir. Bu sayede her yönden başarılı bir model oluşturulmak istense de bu kadar detay Dokümantasyon yükünü ağırlaştırmış, süreci karmaşık ve yönetimi zor hale getirmiştir.

**Çevik Yazılım Geliştirme Nedir?**

belirli kaidelere göre işleyen süreçtense, bireyler arasındaki iletişim ve yazılımın gereksinimlerine göre gelişen değişikliklere uyum sağlayarak çalışan yazılımı en iyi şekilde üretmeyi hedefleyen yazılım geliştirme metotları gruplarının genel adıdır. Çeviklik metotları genellikle sık denetim ve adaptasyonun teşvik edildiği disipline edilmiş bir proje yönetim sürecini, takım çalışmasının, kendinden örgütlenmenin ve izlenebilirliğin özendirildiği liderlik felsefesini, kaliteli yazılımların hızlı biçimde geliştirilmesinin hedeflendiği en iyi mühendislik uygulamalarını ve yazılım geliştirme ile müşteri ihtiyaçlarını ve firma amaçlarını yan yana getiren iş yaklaşımını destekler.​​ Çeviklik metotları bir yazılım geliştirme yaklaşımı değil, **geliştirme süreçleri topluluğudur**.​​



**Çevik Yazılım Geliştirme Manifestosu Nedir?**

2001 yılında, dünyanın önde gelen çevik modellerinin temsilcileri, ortak bir zeminde buluşabilmek adına bir araya gelerek “ çevik yazılım geliştirme manifestosu” nu yayınlamışlardır. Bu manifestoya göre;

* Bireyler ve aralarındaki etkileşim, kullanılan araç ve süreçlerden;
* Çalışan yazılım, detaylı belgelerden;
* Müşteri ile işbirliği, sözleşmedeki kesin kurallardan;
* Değişikliklere uyum sağlayabilmek, mevcut planı takip etmekten; daha önemli ve önceliklidir.

**Çevik Yazılım Geliştirme Prensipleri:**

* İlk öncelik, kaliteli yazılım teslimatıyla müşteri memnuniyetini sağlamaktır.
* Proje hangi aşamada olursa olsun değişiklikler kabul edilir.
* Kısa zaman aralıklarıyla çalışan, kaliteli yazılım teslimatı yapılır.
* Projede çalışan her ekip üyesi birbirleriyle iyi iletişim kurar.
* Proje ihtiyaçları eksiksiz biçimde karşılanmalıdır.
* Ekip içerisinde kaliteli bilgi akışı için yüz yüze iletişim önemlidir.
* Çalışan yazılım, projenin geliştiğini gösteren ilk ölçüdür.
* Projenin geliştirilme hızı sürdürülebilir olmalıdır.
* Basitlik önemlidir.
* Düzenli olarak ekip kendi işleyişini incelemeli ve verimliliği artırmak için gerekli iyileştirmeleri yapmalıdır.

**Çevik Yazılım Geliştirme Avantajları:**

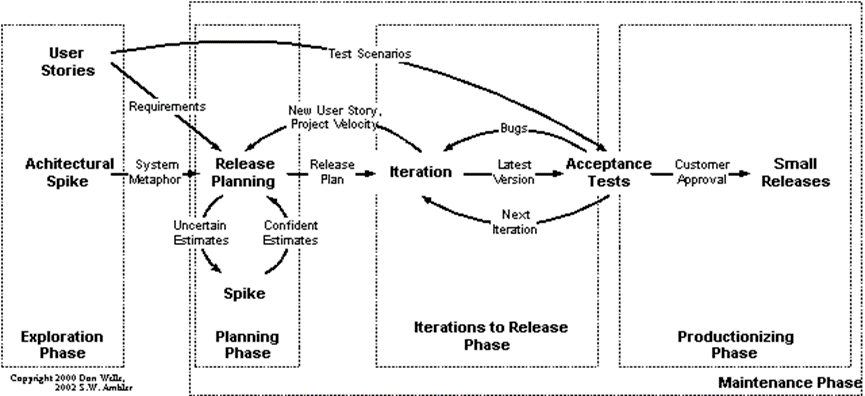
* İnsanların düşünce yapısına uygun olduğundan adaptasyon kolaydır.
* Kısa döngülerle sürekli küçük üretimler yapıldığından motivasyonu arttırır.
* Kullanıcı üretime dahil olduğundan ihtiyaçlar ve gereksinimler daha iyi oluşur.
* Ayrıntılı planlama yerine yineleme planı yapılır.
* Değişime açıklık ve esneklik en üst düzeydedir.
* Ekip içerisindeki iletişimi kolaylaştırır.
* Kaliteli proje üretiminin sürdürülebilir olmasını sağlar.

**Çevik Yazılım Geliştirme Dezavantajları:**

* Kurumsal, herkesin görevinin net bir şekilde belli olduğu bir yapıda uygulamak zordur.
* Dokümantasyon önemsenmemesi yönetimde belirli sorunlara yol açabilir.
* Sürekli değişen ihtiyaçları karşılamak için artan iş yükü.
* Yüksek maliyet.

**Uçdeğer Programlama:** 1999 yılında **Kent Beck** tarafından sunulan uçdeğer programlama, Basitlik, iletişim, **geri** bildirim, cesaret olmak üzere 4 prensibi esas alır.

* **Basitlik:** basitliği bu sorunun cevabı olarak ele alabiliriz, **“işe yarayacak en basit şey nedir? ”** bu sorunun cevabını yalnızca bildiğimiz gereksinimleri ele alarak gelecekte gerekli olabilecek olan gereksinimleri tahmin etmeden oluşturulan yazılım olarak kabul edebiliriz. Bu bize ilerde oluşabilecek bakım veya revizyonlarda kolaylık sağlayacaktır.
* **İletişim:** Yazılım geliştirme, doğası gereği bilgiyi bir takım üyesinden takımdaki diğer herkese aktarmak için iletişime dayanan bir takım sporudur. Uçdeğer programlama, uygun iletişim türünün önemini vurgular ve uygun model bir beyaz tahta veya başka bir çizim mekanizmasının yardımıyla yüz yüze görüşmedir.
* **Geri Bildirim:** Takımlar, Önceki çabaları hakkında geri bildirim yoluyla iyileştirme alanları belirleyebilir ve uygulamalarını revize edebilirler. Aynı zamanda geri bildirimle, basitlik ile birbirini çok iyi bir şekilde destekler. Tek seferde toplanılan gereksinimlerle bazı eksiklikleri tahmin ederek yazılımı oluşturmaktansa sadece istenilen gereksinimlerle basit bir kod oluşturup buna yapılan geri dönüşe göre ekleme yapmak hem daha kolaydır hem de risksizdir.
* **Cesaret:** Kent Beck, cesareti **“korku karşısında etkili eylem”** olarak tanımlamıştır. Bu tanım, ekibin etkinliği azaltan, üretimi yavaşlatan veya üretimin kalitesini düşüren örgütsel faktörlerin veya sorunların korkmadan dile getirmeyi ifade eder. İnsanların doğası gereği işe yaramasa bile bu zamana kadar kullandığı bir şeyi bırakıp başka bir şeyi deneme riskine girebilmek için cesarete ihtiyacı vardır.

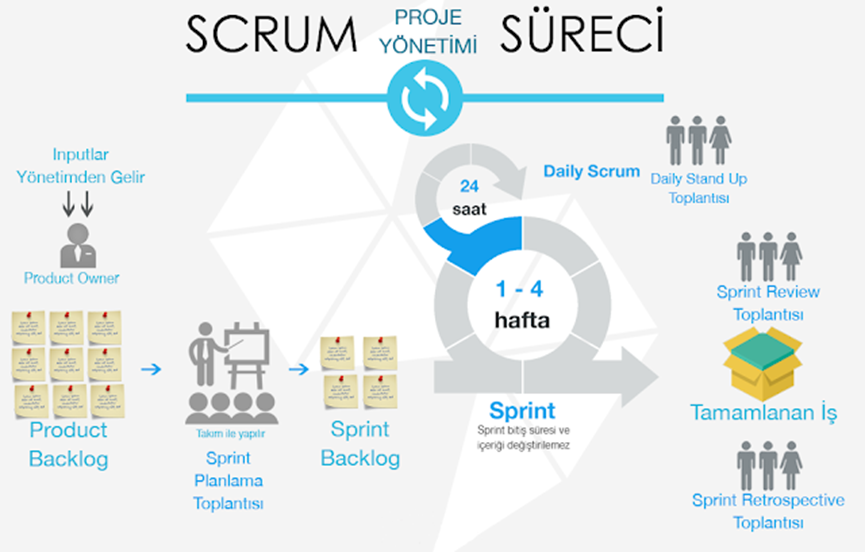


Bu model, bu dört prensibi esas alan 12 uygulamadan oluşur. uygulamalar şunlardır; Planlama oyunu, kısa aralıklı sürümler, müşteri Kalıtımı, yeniden yapılandırma, önce test, ortak kod sahiplenme, basit tasarım, metafor, eşli programlama, kodlama standardı, sürekli entegrasyon, haftada 40 saat çalışma. Aynı zamanda projenin üretimi 4 aşamadan oluşur. Bunlar:

* **Araştırma Aşaması:** projenin ilk aşamasıdır. Bu safhada kullanıcı hikayeleri oluşturulur. Ekip üyeleri teknik altyapı için gerekli araştırmayı ve hazırlığı yapar.
* **Planlama Aşaması:** Bu safhada ekip üyeleri kullanıcı yardımıyla yineleme ve sürüm planlarını oluşturur. Yineleme planlaması için oluşturulan hikayelerin uygulama süresi ekip üyeleri tarafından tahmin edilir. Kullanıcı oluşturulan hikayeleri önceliklerine göre sıraya koyar ve hangi hikayelerin öncelikli olarak uygulanması gerektiğini tespit eder.
* **Yineleme ve Teslim Aşaması:** Kullanıcı hikayelerinin uygulama geçirilmesi bu aşamada olur. Hataya geçirilen hikayelerin doğruluklarını kontrol edebilmek için kullanıcı tarafından kabul testleri belirlenir. Her yineleme sonunda kullanıcıya çalışır bir yazılım sistemi sunulur. Sunulan yazılım sistemleri hakkında kullanıcıdan geri dönüşler alınır. Bu döngü hedeflenen yazılım sistemi üretilene kadar tekrar eder. Son olarak bitmiş yazılım sistemi kullanıcıya veya müşteriye sunulur.
* **Bakım Aşaması:** Yineleme ve teslim aşaması sonucunda artık yazılım sistemimiz kullanıma açılmıştır ve birçok kullanıcı tarafından kullanılmaktadır. Bu aşamada yapmamız gereken kullanımı engelleyen veya aksatan hataları engellemek ve birçok geri dönüş değerlendirerek gerekli eklemeleri yapmaktır.

**SCRUM:** **Jeff Sutjerland** ve **Ken Schawaber** tarafından 1990’ların ortalarında geliştirilen, proje yönetimi ve planlama ile ilgili yöntemlere odaklı olan ve mühendislik detayları içermeyen bir modeldir. Scrum, bir yazılım üzerinde çalışırken yaşanılan problemler ve başarımlar hakkında düşünmeye ve bu düşünceleri deneyime çevirerek gelişmelerini sağlayacak çevre şartlarını oluşturmayı hedefler. Bu çevreyi oluşturmak için uyum içinde çalışan bir dizi **toplantı, araç** ve **rolü** tanımlar.

Scrum’ın çalışma mantığı hedeflenen gereksinimleri belirli parçalara bölerek, her parçayı 2-4 hafta süreyle gerçekleştirmektir. Tamamlanan her bir parça için harcanan süreye **”Sprint”** denir. Bu şekilde **yinelemeli** olarak son ürüne ulaşmayı hedefler. Scrum, süreçteki gereksinim, analiz, tasarım, gerçekleştirme aşamalarını sıralı takip etmek yerine eşzamanlı olarak kullanır buna **“örtüşme”** denir.



Scrum, rolleri 3 temel parça olarak kabul eder. Bunlar **ürün sahibi**, **scrum yöneticisi**, ve **scrum takımıdır**. Ürün sahibi, müşteri veya müşteriyle iletişime geçen bir ekip üyesi olabilir.

* **Ürün Sahibi’nin** sorumluluklarının bir kısmı, gereksinimleri anlamak ve scrum takımına aktarmaktır. Diğer bir yandan sistemin **pazarlama**, **ürün yönetimi**, **Pazar yeri** ve **rekabet** gibi gelecekteki eğilimler hakkında bir anlayışta oluşturması gereklidir.
* **Scrum Yöneticisi’nin** görevi çalıştığı takıma yardımcı olmak, karşılaşılan problemleri çözebilmek veya çözülebilmesi için takımı yönetmek, ürün sahibiyle takım arasındaki iletişimi sağlamak, proje verimliliğini arttırmak adına yapılabilecek iyileştirmeleri yönetmek, takımın ve organizasyonun Scruma’a adapte olmasını sağlamak.
* **Scrum Takımı,** ürün sahibinden gelen gereksinimleri çalışan bir yazılım haline getiren veya var olan bir yazılımın iyileştirmesini yapan bir **topluluktur**. Her topluluk gibi scrum takımı için de en önemli şey **iletişimdir**. İletişim, ortak bir amaç için farklı görevleri üstlenmelerini sağlayan etkendir. Genellikle 5 ila 10 kişiden oluşurlar.

Scrumda 3 farklı toplantı çeşidi vardır. Bunlar **“Sprint Planning”** , **“Daily Scrum Meeting”** ve **“Sprint Review”** dir.

* **Spring Planning,** içinde 2 farklı toplantıya ayrılır. Bunlardan ilki Ürün Sahibinin gereksinimlerini aktardığı toplantılardır. Diğeri bu gereksinimlerin takım içinde farklı görevlere ayrılıp paylaştırılmasıdır.
* **Daily Scrum Meeting,** Gün içinde 15 dakika süren ayak üstü toplantılardır. Bu toplantıların amacı **“Dün ne yaptın? ”**, **“Bugün ne yapacaksın? ”**, **“İlerlemenizi engelleyen bir şey var mı? ”** gibi sorulara cevap vererek her gün projenin ilerlemesi hakkında durum incelemesi yapmaktır. Başka bir amacıysa takım üyelerin birbiriyle iletişimde olup takım çalışmasının korunmasını sağlamaktır.
* **Sprint Review,** Sprint’in sonunda yapılan bir toplantıdır. Takımın tamamladıklarıyla ilgili geri bildirim toplamak için kullanılır. **“Demo”** olarak bilinen bu tören, takımın çalışmalarını sergilemeleri sağlar. Bu sayede motivasyonu arttırır. Aynı zamanda ürünün yol haritası incelenir.

Üçüncü temel değer olan **Araçlar,** kendi içinde **Ürün Gereksinim Dokümanı**, **Sprint Listesi**, **Sprint Kalan Zaman Grafiği** olmak üzere 3 farklı araçtan oluşmaktadır.

* **Ürün Gereksinim Dokümanı,** oluşturmak istediğiniz ürünü tanımlar: Ürünün amacını, özelliklerini, işlevlerini ve davranışını ana hatlarıyla belirtir. Yazılımı üretirken bir pusula görevi görür ve ürünün amacına yönelik yazılım üretilmesinde görev alan herkese net bir yön sağlar. Aynı zamanda piyasaya sürerken ve pazarlarken işinizi kolaylaştırır. Bu listeyi **Ürün Sahibi** yönetir.
* **Sprint Listesi,**  sprint sırasında tamamlanacak **kullanıcı hikayelerinin** tamamlanması için belirlenen görevlerin bir listesidir. Genellikle, her bir görevin kaç saat süreceği de tahmin edilerek listede belirtilir. Sprint sırasında takım üyelerinden gelen yeni bilgilerle sürekli güncel tutulur. Her gün, sprintte kalan işler tahmini olarak hesaplanır ve **Scrum Yöneticisi** tarafından grafiğe dökülür. Buna **sprint kapanma tablosu** denir. Elektronik tablo olarak tutulur, ancak scrum için tasarlanmış bir dizi yazılım ürünü de mevcuttur.
* **Sprint Kalan Zaman Grafiği,** bir sprintte yapılacak iş miktarını gösterir. Sprintte kalan toplam işi izlemek ve hedeflere ulaşma olasılığını tahmin etmek için kullanılır. Sprint boyunca kalan iş takip edilerek önceliklere yanıt verilebilir. Eğer grafik takımın sprint hedefine gerektiği zamanda ulaşamayacağını gösteriyorsa, geç kalmadan gerekli adımlar atılarak bu düzeltilebilir.

Bu kısma kadar birçok yazılım yaşam döngüsü modeli inceledik. Peki bu modellerden hangisini kullanmalıyız? Bu sorunun net bir cevabı yoktur. Bu noktada devreye **süreç mimarları** girer. Projenin büyüklüğü, karmaşıklığı, şirketin yapısı, bütçe, zaman ve projenin ne kadar riskli olduğu gibi değişkenlere göre en uygun süreci seçmek süreç mimarının görevidir. Bu noktada aralarında karşılaştırma yaparak hangisinin hangi konularda avantajlı veya dezavantajlı olduğunu belirlemek, nerede hangisini kullanmamız gerektiğini anlamamıza yardımcı olacaktır. Belirtmeliyim ki Bu karşılaştırmalar yapıldığında genellikle scrum’ın avantajları diğerlerine göre daha fazla olduğundan, scrum günümüzde en popüler kullanılan modeldir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Özellikler** | **Çağlayan Modeli** | **V Modeli** | **Evrimsel Geliştirme** |
| 1 | Gereksinim Belirleme | Başlangıç | Başlangıç | Başlangıç |
| 2 | Maliyet | Yüksek | Yüksek | Düşük |
| 3 | Başarı Garantisi | Düşük | Orta | - |
| 4 | Uzmanlık Gerekliliği | Orta | Orta | - |
| 5 | Fazların Örtüşmesi | Hayır | Hayır | Hayır |
| 6 | Maliyet Kontrolü | Evet | Evet | Evet |
| 7 | Basitlik | Basit | Orta | Karmaşık |
| 8 | Risk Duyarlılığı | Yüksek | Orta | Yüksek |
| 9 | Esneklik | Katı | Düşük​​ | Düşük |
| 10 | Bakım | Düşük bakımlı | Düşük Bakımlı | Düşük |
| 11 | Değişiklik Yağma | Zor | Zor | Zor |
| 12 | Yeniden Kullanılabilirlik | Düşük ihtimal | Düşük İhtimal | Düşük |
| 13 | Eğitim Gerekliliği | Zorunlu | Evet | Evet |
| 14 | Zaman | Çok uzun | Uzun | Uzun |
| 15 | Uygulama | Kolay | Kolay | Kolay |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Özellikler** | **Prototipleme** | **Spiral Model** | **Yeniden Kullanıma Yönelik** |
| 1 | Gereksinim Belirleme | Belirli Sıklıkla | Belirli Sıklıkla | Başlangıç |
| 2 | Maliyet | Düşük | Maliyetli | Düşük |
| 3 | Başarı Garantisi | Orta | Yüksek | Çok Yüksek |
| 4 | Uzmanlık Gerekliliği | Orta | Yüksek | Yüksek |
| 5 | Fazların Örtüşmesi | Evet | Hayır | Hayır |
| 6 | Maliyet Kontrolü | Hayır | Evet | Hayır |
| 7 | Basitlik | Orta | Karmaşık | Karmaşık |
| 8 | Risk Duyarlılığı | Düşük | Düşük | Düşük |
| 9 | Esneklik | Çok Esnek | Çok Esnek | Çok Esnek |
| 10 | Bakım | Evet | Evet | Zor |
| 11 | Değişiklik Yağma | Kolay | Kolay | Kolay |
| 12 | Yeniden Kullanılabilirlik | Mümkün | Mümkün | Evet |
| 13 | Eğitim Gerekliliği | Çok az | Çok az | Çok az |
| 14 | Zaman | Uzun | Uzun | Uzun |
| 15 | Uygulama | Kolay | Karmaşık | Karmaşık |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Özellikler** | **Artırımlı Model** | **Birleşik Modelleme** | **Çevik Modeller** |
| 1 | Gereksinim Belirleme | Belirli Sıklıkla | Başlangıç | Belirli Sıklıkla |
| 2 | Maliyet | Düşük | Yüksek | Çok Yüksek |
| 3 | Başarı Garantisi | Yüksek | Yüksek | Çok Yüksek |
| 4 | Uzmanlık Gerekliliği | Orta | Orta | Çok Yüksek |
| 5 | Fazların Örtüşmesi | Evet | Evet | Evet |
| 6 | Maliyet Kontrolü | Hayır | Hayır | Evet |
| 7 | Basitlik | Orta | Karmaşık | Karmaşık |
| 8 | Risk Duyarlılığı | Yüksek | Düşük | Yüksek |
| 9 | Esneklik | Çok Esnek | Çok Esnek | Çok Esnek |
| 10 | Bakım | Evet | Evet | Evet |
| 11 | Değişiklik Yağma | Kolay | Kolay | Kolay |
| 12 | Yeniden Kullanılabilirlik | Mümkün | Mümkün | Mümkün |
| 13 | Eğitim Gerekliliği | Çok az | Evet | Evet |
| 14 | Zaman | Uzun | Uzun | Kısa |
| 15 | Uygulama | Kolay | Zor | kolay |

**Peki Scrum’u Diğer Modellerden Daha Popüler Yapan Sebepler Nelerdir?**

Scrum, kısa zamanda yüksek kaliteli yazılım üretilebilmesi, kolay uygulanması , düşük riskli olması gibi sebeplere avantajlıdır. Bunun yanında çok yüksek maliyetli olması gibi bir dezavantajı vardır. Günümüzde bilişim sektörünün, çok yüksek kazanç sağlaması üretimdeki maliyettense kısa sürede kaliteli yazılım üretmeyi önemsemesini sağlamıştır. Bu scrum’ı popüler yapan en önemli sebeptir. Bir başka sebebi ise scrum’ın en çok yazılım geliştirme takımları tarafından kullanılsa dahi prensipleri ve uygulamalarının her türlü takım çalışmasına uygun olmasındandır.

**Kaynaklar:**

<https://www.linkedin.com/pulse/what-software-development-life-cycle-sdlc-phases-private-limited>

<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://www.youtube.com/watch?v=SV6CfXWCKSo>

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

<https://tr.wikipedia.org/wiki/Atik_yaz%C4%B1l%C4%B1m_geli%C5%9Ftirme>

<https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1)>

<http://www.agile-process.org/heartbeat.html>

<https://www.atlassian.com/agile/scrum>

<https://hayririzacimen.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-s%C3%BCre%C3%A7-modelleri-70fdfb2f8f77>

<https://www.atlassian.com/agile/product-management/requirements#:~:text=A%20product%20requirements%20document%20defines,launch%20or%20market%20your%20product>.