Hayatımızı kolaylaştıran yazılımları yaratırken birçok yazılım döngü modeli kullanılabilir. Bu yazılım modellerinden en yaygın kullanılanları açıklayıp karşılaştırma yaparak, hangi modelleri hangi projelerde kullanabiliriz inceleyelim.

Yazılım döngü modellerini açıklamadan önce bir yazılımın yaşam döngüsünün nasıl gerçekleştiğini anlamak önemli. Bir yazılımın yaşam döngüsü temel olarak şu aşamalarla şekillenir:

1. Gereksinim
2. Analiz
3. Tasarım
4. Gerçekleştirme
5. Teslim sonrası bakım
6. Emeklilik

Gereksinim aşamasında, elimizdeki problemi ve yazılımın müşterisinin ihtiyaçlarını anlamaya çalışarak fizibilite çalışmaları yaparız.. İhtiyaçları belirlemeden, problemi anlamadan yapmaya girişilen projede; ileriki aşamalarda birçok hatayla karşılaşılır, bu da yazılımın maliyetinin artmasına ve tamamlanma süresinin uzamasına sebep olur. Hatta müşterinin ihtiyacına cevap vermediğinden işlevsiz bir yazılım ortaya çıkabilir.

Analiz aşamasında gereksinimleri analiz ederek dokümante ederiz. SRS(Software/System Requirement Specification) dokümanında sistem özellikleri, sistem ara yüzleri, fonksiyonel olmayan sistem gereksinimleri tanımlanır. Bu doküman yazılım yaşam döngüsü süresince tasarlanıp geliştirilmesi gereken her şeyi içerir. Yani bu aşamada proje ürününün ne yapmasını istediğimizi belirlemiş oluruz.

Tasarım aşamasında projenin sisteminin temel yapısını oluştururuz. Bir sonraki aşama olan gerçekleştirme safhasına geçmeden önce tüm kararların verilmiş olması önemli. Çünkü gerçekleştirim aşamasında herhangi bir soru veya karar projenin tamamlanma süresini uzatabilir. Bu aşama üst düzey tasarım ve ayrıntılı tasarım olmak üzere ikiye ayrılır. Üst düzey tasarımda modüller ve akış şemaları ile kaba taslak bir plan yapılır. Ayrıntılı tasarımda ise yazılımın bileşenleri, veritabanı tasarımı, kullanılacak algoritmalar, sınıf, aktivite ve etkileşim diyagramları gibi ayrıntılar tasarlanarak bir plan oluşturulur.

Gerçekleştirme aşaması yazılım projesinin en uzun aşamasıdır.Bu aşamada tasarımı biten proje kodlanmaya başlanır. Bu aşamadan itibaren herhangi bir analiz işlemi yapılmamalı ve projeyi tasarım aşamasında planladığımız gibi ilerletmeliyiz. Kodlama yaparken derleyiciler, hata ayıklayıcılar gibi programlama araçlarını kullanırız. Yazılımın türüne göre farklı programlama dilleri kullanılır. İlk önce her bir modül için kodlar yazılır. Her modülün kodu birbirinden bağımsız olarak test edilir. Gerekli düzenlemelerden sonra bütün modüller birleştirilir. Kendi içinde hiçbir hata vermeyen modüller birleştirilirken hata verebilir. Bu hataları görmek ve düzeltmek için testler (integration testing) yaparız. Kodlama işlemi bittiğinde müşteri tarafından kabul testi (acceptance testing) yapılır, herhangi bir sorun çıkmadığı taktirde proje teslim edilir. Ama bu yazılım ürünümüzün tek başına bırakıldığı anlamına gelmez. Çünkü yazılım yaşam döngüsüne devam eder.

Teslim sonrası bakım aşamasında yapılan bir değişiklik teslim sonrası bakım(post-delivery maintenance) olarak adlandırılır. Bakım iki çeşit olabilir: Düzeltici bakım ve özelliklerin arttırılmasıyla olan bakım. Düzeltici bakımda sistemdeki hatalar düzeltilir. Özelliklerin arttırılması (yazılım güncellemesi) ise yapılan değişikliklerin mevcut yazılıma uyarlanmasıdır. Proje ürünümüze çalışma ortamındaki değişiklikler sebebiyle uyarlama yapılabilir veya müşterinin istekleri doğrultusunda uyarlanabilir.

Emeklilik ise yazılımın artık kullanım süresini doldurmasıdır.

Yazılım projemizi geliştirirken takip ettiğimiz çeşitli yazılım geliştirme yaşam modelleri vardır. Yazılım geliştirme aşamaları, hata yapabilecek olmamızdan ve müşterinin istekleri değişebileceğinden pratikte; gereksinim, analiz, tasarım, gerçekleştirim, bakım, şeklinde doğrusal değildir. En yaygın olarak kullanılan bazı modeller:

1. Şelale Modeli

2. Barok Modeli

3. V Süreç Modeli

4. Helezonik/Spiral Model

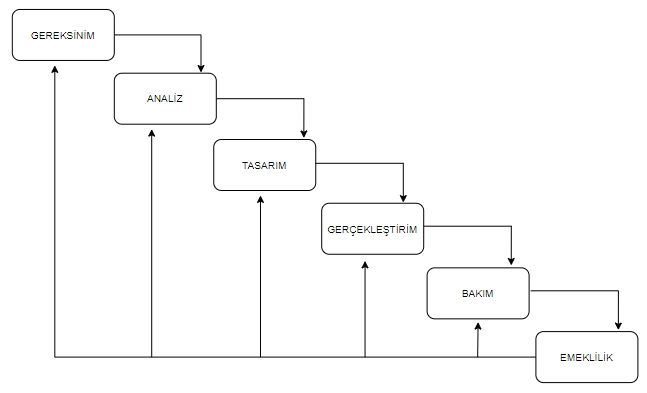
5. Arttırımsal Geliştirme Süreci Modeli

6. Kodla – Düzelt Yaşam-Döngü Modeli

7. Evrimsel Geliştirme Modeli

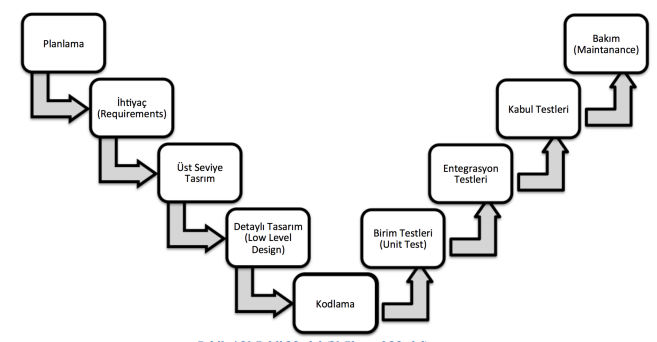
8. Gelişigüzel Model

1. Şelale Modeli (Waterfall Model) : En eski model olmakla birlikte geçmiş yıllarda en popüler modeldir. Geleneksel yazılım geliştirme modelidir. Proje aşama aşama yapılır ve bir aşama bitmeden diğer aşamaya geçilemez, böylece tüm aşamalar en az bir kere tekrarlanır. Aşamanın çıktısı bir sonraki aşamanın girdisidir. Her aşama sonunda dokümantasyon hazırlanmak zorundadır. Büyük ve karmaşık projeler için uygun değildir. Kısa sürecek projeler için uygundur çünkü model içinde geri dönüşler olabilse bile (geri dönüşler tanımlıdır.) analiz ve tasarım aşamalarında çok detaylı planlamalar gerektirdiği için proje süresinin uzaması sonucu müşterinin gereksinimleri değişebilir. Bu değişiklikler gerçekleştirim ve test aşamasına denk gelirse değişikliklerin sisteme adaptasyonunun maliyeti artar. Biz geliştiriciler bir an önce projeyi yazma ve çalıştırma odaklı olduğumuz için yazılım ekiplerinin uzun süren ve ciddi bir efor gerektiren bu modeli kullanmaları sonucu ekip isteğini kaybederek verimsizleşebilir. Ayrıca müşterinin model içerisinde yer almaması model içindeki geri dönüşleri arttırarak maliyetin yükselmesine sebep olabilir.



Şekil 1 Şelale modeli

1. Barok Modeli : Yazılım yaşam döngüsünün aşamalarını doğrusal bir şekilde işler. Model içindeki geri dönüşler tanımlı değildir. Dokümantasyon bu modelde ayrı bir süreç gibi işlenir. Gerçekleştirim aşamasına önem verilir. Günümüzde Barok modeli kullanılmamaktadır.
2. V Süreç Modeli : Şelale modelinin daha gelişmiş bir verisyonu olmasına rağmen bu model de büyük ve karışık projelere uygun değildir. Modelin sol tarafı üretim, sağ tarafı test işlemlerinden oluşur. Anlaşılması ve kullanımı kolay bir model olsa da belirsiz ve değişen gereksinimleri olan projeler için kullanılması uygun olmaz çünkü model içindeki aşamalarda geri dönüş yoktur. Bu da hataların düzeltilmesini zorlaştırarak projenin maliyetini arttırır. Diğer taraftan hata kabul etmeyen projelerde kullanmak için iyi bir modeldir. Müşteri projeyi test eder yani müşteri de işleyişe dahildir. Bu modelin üç temel çıktısı vardır:
3. Kullanıcı modeli : Projeyi geliştirme sürecinde kullanıcının rolünü tanımlar ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarır.
4. Mimari model : Projenin tasarımı ve bunları test işlerini kapsar.
5. Gerçekleştirim modeli : Kodlama ve bunları test etme işlemlerini kapsar.

**

Şekil 2 V Süreç Modeli

1. Helezonik (Spiral) Model : Arttırımlı yaklaşımla ilerler yani tasarımlar prototip oluşturma, analiz ve test etme gibi işlemlerle gerçekleştirilir.Her döngü bir fazı ifade eder ve her döngü öncesinde bulunulan bölümle ilgili risk analizi yapılır. Her döngüden sonra yeniden planlama yapılır, yeni bir hedef belirlenir. Gereksinim, tasarım gibi aşamaları yoktur;

Planlama : Üretilecek ara ürünün planı, hedef belirleme, bir önceki adımdaki ara ürünle birleştirmeyi kapsar.

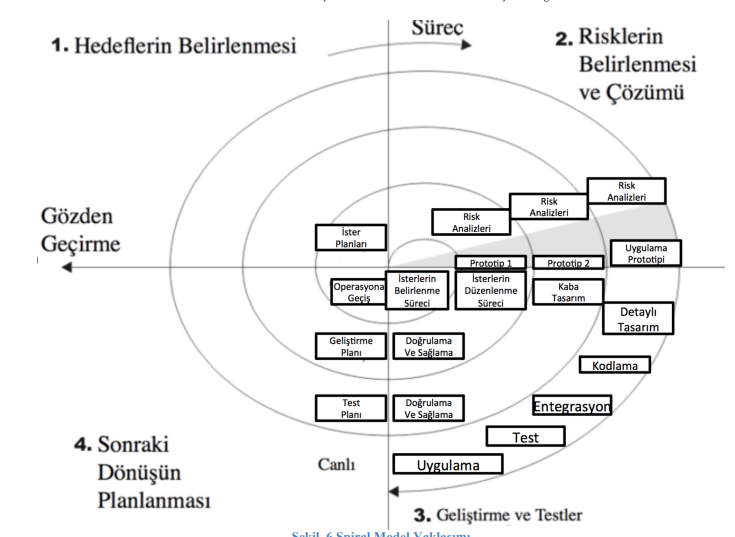
Risk Analizi : risklerin belirlenmesini ve raporlanmasını kapsar.

Üretim : Ara ürünün üretimini kapsar.

Kullanıcı Değerlendirmesi : Ara ürünle ilgili müşterinin yapmış olduğu test ve değerlendirmeleri kapsar. Müşterinin sürece bitmeden, erken katılması ileride oluşabilecek istenmeyen durumları önlemek adına bir avantajdır.

En büyük avantajı risk analizinin detaylı ele alınmasıyla projenin tamamlanma zamanı ve maliyeti kolayca hesaplanabilir. Genellikle güvenlikle ilgili yazılım projelerinde kullanılır. Yazılımın kodlanması ve test edilmesi daha erken başladığı için geliştiriciler için avantajlıdır.

Risk değeri taşımayan, küçük projeler için pahalılıyla dezavantaj oluşturur. Sıklıkla risk analizi yapılması sonucu fazla dokümantasyon oluşması da bir dezavantajdır.

**

Şekil 3 Helezonik Model

1. Artımsal Geliştirme Süreci Modeli : Projenin teslimini tek seferde yapmak yerine geliştirme ve teslim parçalara bölünür. Müşterinin gereksinimleri önceliklendirilerek öncelikli gereksinimler yakın teslime dahil edilir. Bir kısmın geliştirmesine başlandıysa o kısmın gereksinimlerinin güncellenmesi bir sonraki teslime kadar ertelenir. Yani olası değişiklikler sonraki teslimde projeye adapte edilir. Erken teslimler prototip görevi görür.

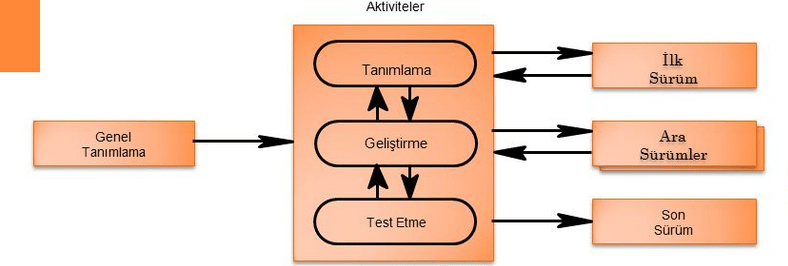
Üretilen her yazılım sürümü birbirini kapsar ve içerilen işlev sayısı giderek artar. Bu modelde müşteri ve geliştirici ilk artımla sonuçları erken gördüğünden dolayı müşteri bir şeyden hoşlanmazsa problemi çözmek daha kolay olur. Geliştiriciler öncelikli gereksinimlere odaklandığı için hatalar ortaya çıktıkça düzeltileceğinden bu model verimlidir. Özellikle; eksiksiz ve net gereksinimleri olan, eksik işlev ile çalışabilen, uzun zaman alabilecek projelere uygundur.

Bir taraftan üretim yaparken diğer taraftan kullanım yapılır.

1. Kodla – Düzelt Yaşam-Döngü Modeli : Bu modelde direkt yazılım ürünü oluşturur ve kullanıma süreriz. Proje en son istenilen hale gelene kadar geliştirilir. Küçük çaplı projelerde kullanılır. Bakım aşaması vardır, kontrollü olmadığı için dokümantasyonu yoktur, bu sebeple bakım aşaması zor bir hal alır. Yazılım geliştirmenin en kolay ama en pahalı yoludur. Bunun sebebi kodlar yazılırken sonradan değiştirmek planlanmadığı için değiştirmesi zor ve maliyetlidir.

Emeklilik aşamasına sahiptir.

1. Evrimsel Geliştirme Modeli : Ne istediğinizi bilmiyorsanız ama görünce tanırım diyorsanız bu model projenizde kullanmak için uygundur. Modelin başarısı ilk evrimin başarısına bağlıdır. Gereksinimi anlamayı kolaylaştırdığından risk ve hata daha azdır. Dezavantaj olaraksa diğer modellere kıyasla daha yavaş bir işleyişe sahiptir. Düzenli bir ürün oluşumu yoktur. Gereksinimi sürekli yenilemek gerekebileceğinden bakımı zordur.



Şekil 4 Evrimsel Geliştirme Modeli

1. Gelişigüzel Model : Bir model olmasa da bu yazılım geliştirme yöntemini de inceleyelim. Genellikle tek kişilik projelerde kullanılır. Yani projenin izlenebilirliği azdır bu sebeple projenin bakımı zordur. Basit programlama kullanılır.

Yazılım sektöründe; yazılım sürümlerinin planlanan zamanda tamamlanamaması, müşterinin değişiklik taleplerine çabuk cevap verememesi, hataların geç fark edilmesi gibi sebeplerle sorunlar yaşanmıştır. Bunun sonucunda 1990’lı yılların sonuna doğru çevik modeller ortaya çıkarak projelerdeki başarı oranını ve ekip içindeki iletişimi etkili bir şekilde artırmak, en kısa sürede bir yazılım ürününü müşteri hizmetine sunmak hedeflenmiştir. Bu modeller sayesinde verimlilik ve yaratıcılık artar. Bu yöntemler verimliliği yüksek, değişime açık ve esnek, hata oranı düşük, ucuz çözümler sunar.

2001 yılında oluşturulan çevik manifestoya göre süreçler ve araçlar yerine bireylerin ve etkileşimlerinin, plan izleme yerine değişikliğe açık olmanın daha önemli ve öncelikli olduğu ortaya kondu. İlerlemenin ilk ölçüsü ‘çalışan yazılım’dır.

Proje küçük yinelemelere bölünerek her yineleme kendi başına bir proje gibi ele alınır. Her yineleme sonunda müşteri bilgilendirilir. Her yineleme 2-4 hafta süreyle gerçekleştirilmesi planlanır. Projenin bu şekilde küçük parçalara bölünmüş olması hataların erken farkedilip düzeltilmesini kolaylaştırır. Çevik yazılım geliştirme modelleri temelde aynıdır ama pratikte bazı farklılıklar içerir. Bazı popüler çevik yazılım geliştirme modellerini inceleyelim:

1. Extreme Programming : Kent Beck tarafından 1999’da ortaya konuldu. Dört temel maddeden oluşur:
2. Basitlik: Yazılacak kodun sade ve anlaşılır olması gerekir. Basitlik zorunlu işlerin yapılmasını içerir. Uzun süren dokümantasyon işlemi olmaz.
3. Cesaret: Projeyi yaparken cesur olmalı ve eğer gerekiyorsa kod tamamen silinip baştan yazılmalıdır.
4. Geri Dönüş: Geri dönüşler sayesinde hatalar farkedilir, düzeltilir. Müşteri proje ekibinin bir üyesidir. Müşteriyle belli zamanlarda buluşarak projenin gidişatı hakkında bilgilendirilir. Bu da sonradan oluşabilecek anlaşmazlıkları önler.
5. İletişim: Projenin başarılı olabilmesi için ekibin etkili bir iletişim içerisinde olması gerekir. İletişim yüzyüze olmalıdır. Tabiki müşteri ile iletişim de projenin başarısının önemli bir kısmıdır.
6. Scrum : Jeff Sutjerland ve Ken Schawaber tarafından 1990’ların ortasında geliştirildi. Çevik yazılım geliştirme metotlarıyla uygulanabilecek bir proje yaklaşım yöntemidir ve yazılım geliştirmek dışındaki projelerde de kullanılabilir.

Projede küçük parçalara (sprint) bölünerek ilerlenir. Gereksinimlerini belirlemenin zor olduğu ve karmaşık durumların ortaya çıkması muhtemel olan projelerde kullanıma uygundur. Üç temel kavramı vardır:

1. Roller : Ürün sahibi, scrum yöneticisi ve scrum ekibinden oluşur. Proje ürün sahibinden alınan geri bildirimlerle yürütülür. Scrum yöneticisi ekibi ve projeyi kontrol eder, scruma adapte eder. Scrum takımı sürekli iletişim halindedir, tek bir hedefe beraber ulaşmaya çalışırlar. Genellikle 5-9 kişiden oluşur.
2. Toplantılar : Scrum metodunun temelini oluşturur. Üç alt başlıkta incelersek:
   1. Sprint Planlama: Kapsamlı gereksinim listesinin çıkarımı, risk değerlendirmesi, maliyetlerin hesaplanması, dağıtımlar için takımların belirlenmesi vb.
   2. Sprint Gözden Geçirme : Her sprint başlangıcında planlama toplantısı yapılır. Toplantının ilk kısmında ürün sahibi ve scrum takımı gereksinim listesini gözden geçirir. Toplantının ikinci kısmında takım üyeleri gereksinim listesinin en üstünden başlayarak sprint sonucunda gerçekleştirilecek olan Sprint Gereksinim Listesini oluştururlar. Takım, ürün sahibi tarafından önceliklendirilmiş gereksinimlerden ne kadarını yapacağını belirler.
   3. Günlük Scrum Toplantısı : Takımın ilerleyişini ve karşılaşılan engelleri görmek için her gün 15’er dakikalık toplantılar yapılır. Bu toplantılar ilerleyişi görmek ve hataları farketmek açısından çok önemlidir.
3. Araçlar/ Bileşenler : Proje boyunca yapılması gerekenlerin bir listesi olan Ürün Gereksinim Dokümanı yazılır. Sprint Dokümanı, Ürün Gereksinim Dokümanının ardından oluşturulur. Bu dokümanın yazılma amacı her sprintin sonunda son ürünün parçası olan bir işlevselliği veya çalışabilir bir parçayı elde etmektir. Sprint Kalan Zaman Grafiği, sprint boyunca işlerin ne kadarının yapıldığıyla planlanan miktarı karşılaştırmak için yazılır.

Scrum günümüzde en yaygın kullanılan yazılım geliştirme yöntemlerindendir. Başlıca sebepleri:

Zamandan ve maliyetten tasarruf etmesi,

Değişimlere kolaylıkla uyum sağlayabilmesi,

Takım içindeki ve takımın müşteriyle iletişiminin güçlü olması sebebiyle hataların ve istenmeyen durumların en aza inmesi,

Sadece yazılım alanı değil, birçok alanda kullanılabilmesidir.

Kaynaklar

[Yazılım Mühendisliği: Yazılım Süreç Modelleri – Furkan Alnıak (furkanalniak.com)](https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/)

[Gereksinim Tanımlama ve Dokümantasyon (ba-works.com)](https://ba-works.com/blog/gereksinimlerin-dokumantasyonu-hakkinda-merak-edilenler/)

[SDLC (Software Development Life Cycle) Nedir? – Zeynep AYGÜN (wordpress.com)](https://zeynepaygun.wordpress.com/2017/05/29/what-is-sdlc-sdlc-nedir/)

[SDLC: Yazılım Geliştirme Yaşam Döngüsünün Aşamaları ve Modelleri (ichi.pro)](https://ichi.pro/tr/sdlc-yazilim-gelistirme-yasam-dongusunun-asamalari-ve-modelleri-234877250586705)

[https://ybsansiklopedi.com/wp-content/uploads/2015/08/Yazılım-Geliştirme-Modelleri-Yazılım-Yaşam-DöngüsüSDLCYBS.pdf](https://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)

[Yazılım Yaşam Döngüsü | Çağlar Telef (caglartelef.com)](https://caglartelef.com/yazilim-yasam-dongusu/)

[Yazılım Süreç Yönetim Modelleri ve Karşılaştırılması – Fikir Jeneratoru](https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/)

[YAZILIM YAŞAM DÖNGÜ MODELLERİ. YAZILIM YAŞAM DÖNGÜSÜ NEDİR? | by Züleyha ÖZ | Medium](https://medium.com/@ozzuleyha/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri%CC%87-b060ee08ca76)

[Yazılım Yaşam Döngü Modelleri. Yazılım Yaşam Döngüsü Nedir? | by Ömer Harun Çetin | Medium](https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742)