**Yazılım nedir?**

Elektronik parçaların kendi içinde ve başka elektronik parçalarla haberleşmesini sağlayarak kullanıcısının iş süreçlerini kısaltıp kolaylaştıran, iletişimi sağlayan ve geliştiren, verimliliği arttıran çeşitli görevler için uzun süreçlerde hazırlanmış programlar bütünüdür

.

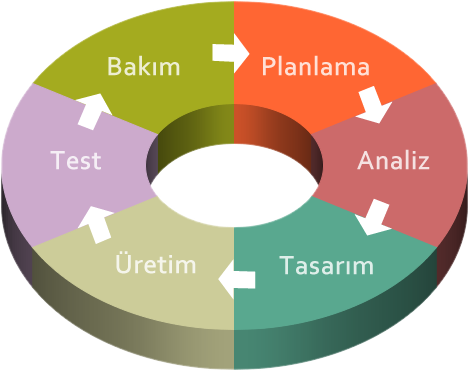
**Yaşam-Döngü modeli nedir?**

Yaşam-Döngü modeli veya modelleri aslında bir süreçler bütünüdür.Birçok Yaşam-Döngü modeli vardır ve gereksinim aşamasından en son kısım olan bakım aşamasına kadar sürer. İşte bahsettiğimiz bu döngü aslında zahmetli gibi görünse de aslında çok yararlı ve ekonomiktir. Yazılım döngüsünde kaydettiğiniz her aşama belgelenir, ürün için yazılan kodlar kolayca birleştirilir ve test esnasında daha az para harcanır. Çoğu yazılım geliştiricinin başına büyük dertler açan başka yazılım geliştiricilerin yazdığı kodlara uyum sağlayamama sorunu da kolaylıkla aşılabilir.

Yazılım geliştirme süreçlerinde buz dağının görünmeyen tarafı vardır. Müşteri tarafından düşünüldüğünde kolay gözükebilir fakat bu süreçte birçok problem boy gösterebilir. Ürünü isteyen müşteri olabildiğince en kısa zamanda ve olabildiğince en ekonomik olarak ürünü yazılım geliştiriciden ister. Fakat bu tarz istemler hem yazılım geliştirici hem de müşteriyi zarara uğratır. Yazılım geliştiriciler ise eğer bu işin para ve zaman bakımından tabiri caizse emeğine değmeyeceğini düşünürse tipik bir yazılımcı söylemi olan “ Sebep-Sonuç hatasına mal olabilir” der. Ne kadar etik değilmiş gibi görünse de bu söylem doğrudur çünkü genelde bir yerinde değiştirilen proje başka bir yerinde patlak verebilir.

**Yazılım Geliştirme Yaşam-Döngüsü Temel Adımları**

Yazılım geliştirme yaşam-döngüsü temel adımları şunlardır: Planlama, çözümleme, tasarım, gerçekleştirme, test ve bakım.



Yukarıdaki temsili resimde görüldüğü gibi bu aşamalar aslında bir döngüdür. Anlatılmak istenen işin başlayıp belirli bir yerde bitme gibi bir durumu yoktur. Döngü sürekli devam eder. Aslında yazılım da insan gibi doğar, yaşar ve ölür. İşte bu yazılımda ömrünü tamamladığı zaman gerekli düzenlemeler ve yeniliklere muhtaçtır çünkü piyasada her zaman yenilik vardır ve yazılımda bu yenilikleri karşılamaya mecburdur.

**Planlama:** Müşteriyle yazılım geliştirici bir araya gelir. Müşteri istediği ürünündeki temel ihtiyaçları yazılım geliştiriciye aktarır. Projenin yapılabilirliği ölçülür ve bu esnada yazılım geliştiricinin kafasında bir şeyler belirlenir. Bütün projelerin başlangıç aşaması olduğu bu adımdır

**Analiz:** Yazılım işlev ve ihtiyaçları ayrıntısıyla incelenir ve ortaya çıkarılır. İncelemeler mevcut sistemler üzerinde yapılır ve test edilir. Bu aşamada sadece yazılım geliştiricinin tek başına olması yetmez, yazılım geliştiriciyle beraber yanında sistem analisti ve en önemlisi müşterinin de olması gerekmektedir. Bu kısımda bütün diyagramları tek tek ve ayrıntılı bir şekilde çizmemiz gerekir ve uzmanlar tarafından raporların yazılışının da eksiksiz olması lazımdır. Projenin yapılabilirliğinin belirlenmesinde beş tane etken vardır:

**1)Ekonomik**: Projemize bütçemiz tamamlamaya yetecek mi yetmeyecek mi?

**2)Yasal**: Bu proje siber yasa ve diğer düzenleyici çevrelere uygun mu?

**3)Operasyon Yapılabilirliği**: Müşterinin bizden beklediği ürünü tam anlamıyla yapabilir miyiz?

**4)Teknik:** Şu anda projeyi yapacağımız bilgisayar sistemi ürünün yazılımını destekliyor mu?

**5)Program:** Projeyi söz verdiğimiz tarihte yapabilecek miyiz?

**Tasarım:** Bu üçüncü aşamadır. Bu aşamada şartnameler ve yazılım dokümanları şartname dokümanına göre yazılır. Bu genel sistem mimarisini tanımlamak için bize yardımcı olur. Tasarım belgeleri ikiye ayrılır: Düşük Seviye Tasarım ve Yüksek Seviye Tasarım. Yüksek Seviye tasarımda Düşük Seviye tasarıma göre daha fazla teknolojik detay ve daha eksiksiz mimari diyagramlar bulunur.

Genel olarak ise Mantıksal Tasarım ve Fiziksel Tasarım olarak ikiye ayırabiliriz.

**Gerçekleştirme:** Bu aşama dördüncü aşamadır. Bu aşamada üç tane yapılması gereken şey var diyebiliriz.

Bunlar: “Kodlama, Test, Kurulum” olarak söylenebilir. Hepsi planlı ve adım adımdır. Her yazılım geliştirici “clean code” olarak adlandırılan temiz ve başkaları tarafından anlaşılabilir kodlar yazmaya özen göstermelidir çünkü aksi takdirde bu yazılım geliştiricinin aleyhine sonuçlar doğurabilir.

**1)Kodlama:** Tasarım esnasında sistem tasarımı aşamasının hemen ardından kodlama işlemi başlar. Yazılım geliştiriciler projeye uygun programlama dilini kullanarak sistemi oluşturmaya başlarlar. İşte bu aşama Yazılım Geliştirme Yaşam Döngüsünün en zahmetli ve uzun süren aşamasıdır. Yazılım geliştirici burada önceden belirlediği kodlama yönergelerine uymakla yükümlüdür. Ayrıca yazılım geliştirici yazdığı kodun eksiksiz ve müşterinin istediği şekilde çalışması için hata ayıklayıcılar ve derleyiciler kullanmayı da ihmal etmemelidir.

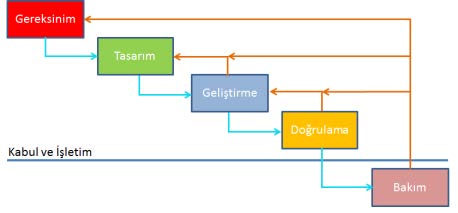
**2)Test:** Yazılım geliştirici yazdığı kod tamamlandığında işin içine test aşaması girer ve test ekibi sistemi bütün işlevselliğiyle kontrol amaçlı test etmeye başlar. Bu aşama müşteri memnuniyeti için esastır. Ayrıca bu aşamada KG ekibi koddaki bir hatayı bulursa yazılım değiştiriciye gönderir ve yazılım geliştirici de düzeltip tekrardan KG ekibine gönderir. Bu aşama hata bulmayı bırakıncaya kadar devam etmesi gerekir.

**3)Kurulum/Dağıtım:** Bu aşama ise test aşamasından sonra gelir. Test aşaması bitip sistemde herhangi bir hata olmadığından emin olunduğunda son dağıtıma başlanır. Son olarak yazılım bitmiş haliyle yayınlanır. Bu aşamada hala dağıtım sorunları olabilir kontrol edilerek işleme devam edilir

**Bakım:** Bu aşama en son aşamadır. Müşteriler geliştirilen sistemi kullanmaya başlar ve artık senaryoda hiç rastlanmayan hatalar için yazılım geliştiricilerle iletişim halinde olur. Diğer iki seçenek ise sürüm yükseltme yazılıma yeni şeyler ekleme olan geliştirme aşamasıdır. Yazılım geliştirici müşteriyle anlaştığı süre boyunca geliştirdiği projeyle ilgilenir.

**YAZILIM YAŞAM DÖNGÜ MODELLERİ**

**Çağlayan (Şelale) Modeli:** Bu model geleneksel yazılım geliştirme modeli olarak bilinir. En temel ve en eski en popüler yazılım yaşam geliştirme modelidir. Bu modelde her işlem aşamalar olarak ele alınır ve tekrar esastır. Yani aşamalar en az bir kere tekrardan geçmelidir, eğer ki bir aşama tamamlanamazsa sonraki aşamaya geçilemez. Güvenli bir yaşam-döngü modeli olarak görülür çünkü gereksinimler çok iyi tanımlanır ve tüm gereksinimleri karşılamaya çalışır. Fakat zaman bakımından çok da avantajlı olduğu söylenemez çünkü aşamalar çok detaylı incelendiği için analiz tasarım safhası beklenenden çok sürebilir. Ayrıca bu süre içinde yazılım ihtiyaçları değişebileceği için kullanıcının isteklerine kısa zamanda dönüş yapılamayabilir. Bunun sonucu olarak müşteri ihtiyaçlarını karşılamak amacıyla maliyetler beklenenden fazla olabilir.

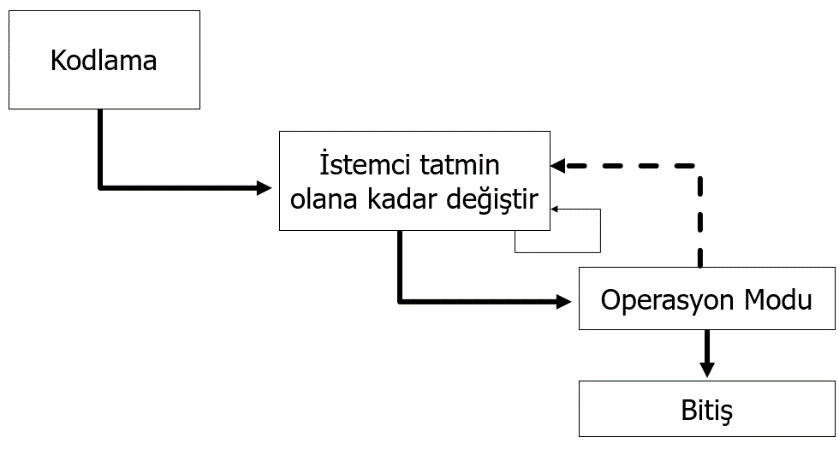


Avantajları ve Dezavantajları söyle sıralanabilir

**Avantajları:** Çağlayan (Şelale) Modelini anlaması ve yönetimi kolaydır. Diğer projelere göre gayet açıklayıcıdır çünkü projenin en başından her şey açıkça belirtilmiştir. Bu modelin uygun olacağı projeler ise iyi anlaşılmış projelerdir.

**Dezavantajları:** Uzun ve devam eden projeler için çok da iyi bir seçim olarak nitelendirilmez. Çünkü yapılacak bir hatada projeyi değiştirmek maliyeti çok büyük oranda artırıp hem müşterinin hem de yazılım geliştiricinin ekonomik olarak aleyhine olur. Müşterilerin memnuniyeti için çok fazla efor sarf edilmesi gerekir çünkü kalıplara dayalı bir modeldir. Model olarak projenin en son aşamada tamamlandığı için bu durum projenin iptaline kadar gidip hem yazılım geliştirici hem de müşteri bakımından çok kötü sonuçlar doğurabilir.

**Kodla ve Düzelt ( Code And Fix) Modeli:** Bu model çok büyük olmayan küçük çaplı projelerde kullanışlıdır ve yazılım geliştiriciler bu modeli küçük projelerde kullanırlar. Bu modelin kod satırı 300-400 satırı çok geçtiği görülmemiştir. İlk sürümden itibaren sürekli geliştirilir ve son aşamaya kadar geliştirme devam eder. Aslında çok kolay gibi görülmesine rağmen bakım aşaması en zor olan modellerden biridir. Çünkü bakım aşaması dokümantasyon olmadığı için yazılım geliştiricileri çıkmaza sokabilir.



Avantajları ve dezavantajları şöyle sıralanabilir

**Avantajları:** Herhangi bir plana veya sıralamaya gereksinim duymadığı için küçük projelerde gayet uygun bir model olarak nitelendirilir. Aşamalar çabuk geçildiği için Çağlayan Modeli gibi aşamalarda çok takılı kalınmaz. Herkesin kullanabileceği bir model olduğu için yaygın bir model olarak bilinir.

**Dezavantajları:** Planlama yoktur. Buna bağlı olarak ne zaman biteceği yazılım geliştiriciler tarafından kestirilemez. Plansız olması sonradan değişikliğe gerek olduğunda kodun ona uygun yazılmadığı için değiştirilmesi zor olabilir.

**V Süreç Modeli:** Bu modelin ismi aşamalarının “V” harfine benzemesi dolayısıyla V Süreç Modelidir. V harfinin sol ve sağ tarafı farklı anlamlar ifade eder. V harfinin sol tarafı üretim bölümü sağ tarafı ise sınama bölümüdür. Bu modelin 3 tane temel çıktısı vardır.

**1)Gerçekleştirim Modeli:** Projenin yazılımının kodlanması ve kontrol edilmesidir.

**2)Mimari Model:** Burada tasarım analizi ve sınanması yapılır.

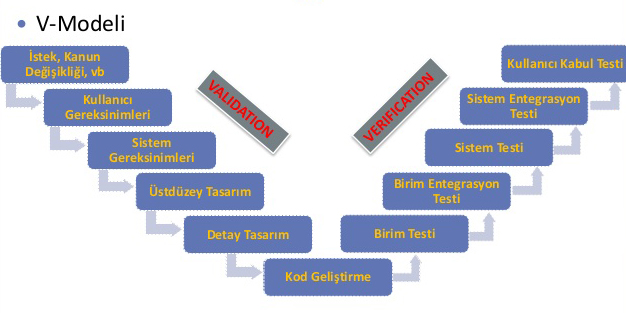
**3)Kullanıcı Modeli:** İlişkileri tanımlar sistemin kabul edilebilirliğini sorgular.

Belirsizlik bu modelde en aza indirgenmiştir. Bu modelde kullanıcılarda iş geliştiricidir. Aşağıya doğru akan bir aşama sistemi söz konusudur. Kodlama aşamasından sonra yukarı doğru çıkılır ondan sonra başlanır. İşte tam bu noktadan sonra çağlayan yaşam döngüsünden farkı ortaya konulur. “V” şeklinde olan bu modelde sol tarafında yapılan işlemlerin sağ tarafta aynı hizada testleri yapılır. Detaylı tasarım yapıldıktan sonra bunun hemen karşısında kodlama bittikten sonra birim testleri de yapılır. Bu modelde diğerlerinin aksine tüm aşamalar teker teker test edilmesini sağlar. Böylelikle kullanıcı ve yazılım geliştirici arasında iletişim en üst seviyeye çıkarılarak sorunlar da en aza indirilir.

Avantajları ve dezavantajları şöyle sıralanabilir

**Avantajları:** Kullanımı kolay olduğu için proje yönetimi ve kullanıcı tarafından takibi daha kolaylık sağlayıcı bir şekilde olur. Planlar erken aşamada vurgulandığı için takip ve esneklik açısından rahattır. Verificatıon ve validation tüm teslim edilebilir ürünlerde uygulanabilir.

**Dezavantajları:** Fazlar arası tekrar olmadığı için aynı zamanda gerçekleşecek olaylar kısıtlıdır. Son ürün olmadığı için projenin sonuna ulaşmadan tam anlamıyla ürünü kestirmek mümkün değildir. Evrimleşebilir.



**Helezonik (Spiral) Model:** Bu model bazı temel konular üzerine ağırlık verir. Bunlar risk analizi ve prototip üretme konusudur. Her döngüde risk analizi yapılarak döngüyü planını ve hedeflerini belirler. Bu model 4 ana aşamadan oluşur.

**1)Planlama**: Her döngüde üretilecek ara ürün için plan yapar ve bir önceki ara ürün ile bütünleştirir.

**2)Risk Analizi:** Kullanılacak bütün riskleri araştırıp inceler ve bu risklerle karşılaşılması anında çözüm yöntemini belirlemede rol oynar.

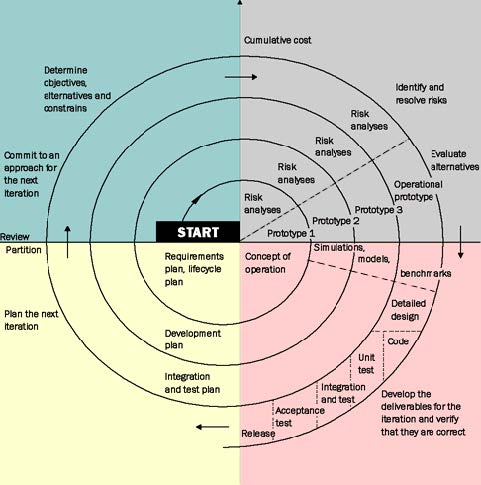
**3)Üretim:** Ara ürünleri üretir.

**4) Kullanıcı değerlendirmesi:** Oluşturulan ara ürünleri kullanıcılar değerlendirir.

Avantajları ve dezavantajları şöyle sıralanabilir

**Avantajları**: Kullanıcılar sistemi erken görebilir. Geliştirmeyi en küçük parçalara böldüğü için riskli adımları atlatmak kolaydır. Buna bağlı olarak hatalar erken görülüp düzeltilebilir. Yazılım ve donanım geliştirmesi için ön ayak olur.

**Dezavantajları:** Genelde büyük projeler için kullandığı için küçük projeler için mantıksızdır. Çünkü maliyeti küçük projeler için değerinden fazla olabilir. Karmaşıktır. Adından da anlaşılabileceği gibi spiraldir yani sonsuza kadar gidebilir. Son olarak kontrat tabanlı yazılıma uymaz diyebiliriz.



**Barok Dönemi:** Bu modelde yazılım yaşam döngüleri doğrusal olarak uygulanır. Her şey sırayla olduğu için doğrusal denmiştir. Sırasıyla inceleme, analiz, tasarım, kodlama, testler, belgeleme ve kurulum olarak devam eder. Bu model 70’li yıllarda bırakıldığı için adını da buradan alır. Diğer modellerden farklıdır çünkü aşamalar arasındaki geçişleri tarif etmemiştir.

**Fıskiye Modeli:** Aslında bu model Çağlayan Modelinden esinlenilmiştir. Henderson- Sellers ve Edwards tarafından geliştirilmiş modeldir. Şelaleden farklı olarak döngülere sahip olmasıdır. Tasarım aşamasında koddan tekrar tasarıma dönebilir ve operasyona geçtikten sonra testlere geri dönebilme gibi döngülere sahiptir. Bakım ve kriterlere sahip olup, şelale modelinden izler fazlasıyla taşır.

**Extreme Programing (XP):** Adını XP Kent Beck isminden alır. 1999 yılında ortaya çıkmıştır ve kendisinden önce gelen metotların daha ayrıntılı bir türüdür. Bu modelin amacı iyi bir proje için müşteriyle iyi iletişim kurup müşterinin istediği şeyler eksiksiz bir şekilde anlamayı amaç eder. Müşteriyi de ekibin bir parçası olarak görmemiz mümkündür. Bu modeli yani XP modelini diğerlerinden ayıran önemli şeylerden bir tanesi de cesarettir. Proje ne kadar büyük olursa olsun başarısızlıktan korkmaz aksine onların üstüne gidip başarısızlık nedenlerini araştırır. Başarısızlıktan korkmak projeyi yavaşlatır. XP yazılım geliştirmede kolaylık sağlamak için 12 pratik gösterir:

**1)Önce Test**: Kodlama aşamasından önce test yapılır ki hatalar önceden görülür ve çözümü sağlanır. Hem müşteri istediği tasarımı önceden görür hem de yazılım geliştiriciler daha esnek olabilir.

**2)Ekipte Müşteri**: Müşteri ekipten biridir. Böylelikle çok önemli olan iletişim problemi çözülmüş olur.

**3)Planlama Oyunu**: Müşterinin istediği sürümlerdeki yenilikler toplantılarla tahmin yürütülerek sağlanır. Bu da müşterinin isteklerinin eksiksiz yapılmasını sağlar.

**4)Çiftli Programlama**: Özellikle yeni başlayan yazılımcılara önerilen bir metottur. Böylelikle hem yeni yazılımcılar tecrübelenir hem de tecrübeli yazılımcılar bilgilerini tazeler.

**5)Sürekli Entegrasyon**: Sistemde yapılan değişiklikler anında sisteme girilir. Bu hataları erkenden görmeyi sağlarken müşteriyi de adım adım bilgilendirir.

**6)Yeniden yapılandırma**: Müşteri memnuniyeti adına yazılım geliştiricileri kod ve tasarımlarını sürekli kontrol ederek gözden geçirir.

**7) Ortak kod sahiplenme**: Proje ekibin ortak ürünüdür. Herkes o ürüne katkı sağlamıştır. Böylelikle herkes birbirinin koduna erişebilir ve istediği sorunun çözümünü bulabilir.

**8)Benzetim:** Projede bölümler parçalara bölünerek benzetim tarzıyla yazılım geliştirilir. Parçaların doğru birleşmesiyle iyi bir proje kaçınılmazdır.

**9)Kodlama standartları**: Kodlar standartlarına uygun olarak geliştirilir. Karmaşık olmayan okunabilir kodlar olması lazımdır.

**10)Haftada 40 saat:** Günde en az 8 saat ve haftada en fazla 1 fazladan mesai verilerek 5 gün çalışılması gerekir. Mesailer artarsa verimlilik düşer.

**Scrum:** Scrum bilindiği gibi bir çevik yazılım yöntemidir. Scrum ismi ise rugby sporundan gelir ve 90’lı yıllarda bulunmuştur. Aslında scrum sadece yazılım alanında kullanılmaz. Scrum şirket içindeki çalışanların ayaküstü 15 dakikalık toplantılar düzenlemesidir. Eğer bu toplantılar 15 dakikadan uzun sürerse bir sorun olduğu anlaşılır. Projede görev dağılımı yapılmaktadır. Aynı zamanda yönetici sorunları aşmak için takıma hizmet eder. Sürekli toplantı halinde olma durumu vardır. Çünkü bilgilerin sürekli yenilenmesi lazımdır. Ne kadar basit bir yöntem gibi görünse de büyük şirketler bunu sıklıkla kullanır çünkü verime maksimuma çıkarmayı amaçlayan bir metottur.

**Scrum günümüzde neden bu kadar popüler?**

Çünkü scrum yazılım geliştirmeyi kolaylaştırıp düzenledi. Profesyonel şirketler bunu anladığı görülüyor. İstatistikler gösteriyor ki tüm olası Scrum tüm olası çevik çerçevelerin yüzde 66’sına sahip. Scrum her sektörde kullanılabilir. Scrumun esas amacı insanlar arasında iletişimi geliştirerek yapılması gerekenleri yüksek verimle gün yüzüne çıkartıyor. Üniversiteler, hastaneler, askeriye, otomotiv sanayi gibi çok farklı sektörler bile scrumu kullanmaktan çekinmiyor. Çünkü insan ilişkilerini artırıyor ve organize davranışa özendiriyor.

Kaynakçalar

https://medium.com/@halilyesil3506/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri%CC%87-efc28dea3bf0

http://bilgisayarkavramlari.sadievrenseker.com/2013/01/10/sdlc-yazilim-gelistirme-yasam-dairesi/

https://medium.com/@furkaneren\_81038/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-203fe380b5e7

https://www.guru99.com/software-development-life-cycle-tutorial.html

https://www.quora.com/Why-is-the-Scrum-process-so-popular-in-the-software-industry