**Yazılım Yaşam Döngü Modelleri (Sofware Life Cycle Models)**

Yazılım yaşam döngü modellerini açıklamadan önce yazılıma ve yazılımın nasıl bir gelişim geçirerek günümüze kadar geldiğine bakmamız gerekmektedir. Böyle bir öğrenme sürecinin daha sağlıklı ve “büyük resmi görme” açısından daha faydalı olduğuna inanmaktayım.

**Yazılım(Sofware) Nedir?**

Yazılımı en yalın ve kısa biçimiyle, bir sistemin (bilgisayar, çamaşır makine, uzay aracı, kahve makinası v.b) donanım kısmı dışında kalan her türlü kısmı olarak özetleyebiliriz. Sistem için verdiğim bazı örnekler size fazla abartılı gelebilir ama artık günümüzde en basit aletten tutun da en karmaşık yapılara kadar her şeyin içinde kodlanmış ve belirli bir algoritmaya sahip yazılım bulunmaktadır.

Bu yazıyı okuduktan sonra çevrenizdeki eşyaların hangilerinin içerisinde ne tür yazılımların olduklarına bakmaya ne dersiniz? Emin olun çok şaşıracaksınız.

Bir diğer açıdan baktığımızda yazılım denilince insanların akıllarına sadece bilgisayar programları gelmektedir. Evet kesinlikle bilgisayar programları birer yazılımdır ama tek başına yazılımın tanımını oluşturamamaktadır. Yazılım bilgisayar programları ile bunlarla ilişkili yapılandırma ve belgeler bütündür diyebiliriz.



**Yazılımın Tarihsel Gelişimi**

Yazılım gelişimi ile bilgisayar kavramının gelişimi de aslında birbiriyle çok fazla bağlantılı ve paraleldir. Bilgisayarların ilk icat edildiği ve buna takiben ilk 25-30 yıllık süreçte yazılım büyük oranda donanıma bağımlıydı. Bu yüzden elektrik- elektronik mühendisleri ve ilk bilgisayar mühendisleri yoğun bir şekilde donanım üzerinde çalışmış ve yazılım alanında pek bir gelişme gösterememişlerdir. Ama zamanla donanım safhasındaki gelişmeler ile yazılım etkin ve önemli bir konuma gelmiştir.

**1940, 1950 ve 1960’lar:**

Bilgisayarların bir oda büyüklüğünde ve hantal bir yapıda olması, bilgi gösteriminin çok kısıtlı olması gibi sebeplerden dolayı gelişmiş bir yazılım olgusundan söz edemiyoruz. Daha çok elle yazılan basit makine kodları yazılım işini görmektedir. Daha sonra 50’li ve 60’lı yıllarda biraz daha donanım kısmının geliştiğini görsek de halen bu yıllarda yazılımın donanıma bağlı olması günümüzdeki tam anlamıyla yazılım olgusu ile örtüşmemektedir.1968 yılında “yazılım mühendisliği ” kavramının tartışılması ve gündeme alınması ise yaşanan önemli ve olumlu bir gelişme olarak tarihe not düşünülebilir.

**1970, 1980, 1990 ve 2000’ler:**

Transistörlerin icadıyla beraber bilgisayarların yaygınlaşması ve buna binaen yazılıma olan talep de hızla artmaya başlıyor. 80 ve 90’larda ise bilgisayarların kişisel amaçlar için yaygınlaşması ile yazılım ihtiyacı maksimum seviyeye geliyor diyebiliriz. 2000’ler ve günümüzde ise globalleşen dünyada neredeyse her şeyimizi birer yazılım sayesinde gerçekleştiriyoruz. (Yemek sipariş etmekten tutun da online bankacılık hizmetlerine kadar)

Tüm bu gelişimler doğrultusunda çok hızlı bir ivme ile yazılıma talep arttı ve ortaya bir “Yazılım krizi ” çıktı. Çünkü artan talebe karşı yeterli bir yazılım üretimi gerçekleşemiyordu. Tabii bu krizi başarısız ve zamanında teslim edilemeyen yazılım projeleri ve yazılım yaşam döngülerinin olmayıp sistemli mühendislik yaklaşımının olmaması de desteklemiş oldu.

Tüm krizlerin aslında bir fırsat olduğunu göz önüne alırsak, insanoğlu bu krizden sonra yazılım olgusunun bir mühendislik disiplini altına oturtulması gerektiğini ve bir yazılımın geliştirilirken belirli yöntemlerin takip edilmesini gerektiğini anlamış oldu.

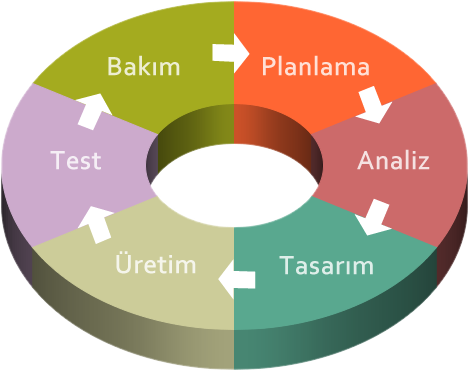
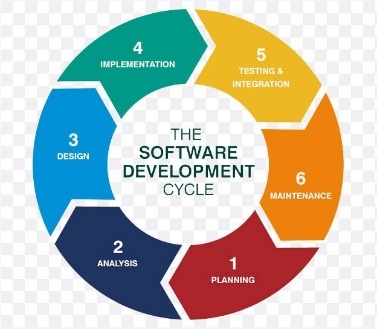
Bu çıkarılan dersler bizlerin hayatına yazılım mühendisliği (software engineering) ve yaşam-döngü modelini (life-cycle model) kattı.

**Yaşam-Döngü Modelleri (Life-Cycle Models)**

En genel anlamda yaşam-döngü modelleri, bir yazılım ürünü ihtiyacının ortaya çıkmasından kullanımdan kalkmasına kadar geçen dönemdir. Bu dönem, yazılımın oluşturulurken takip edilecek adımlarını kapsar. Bu süreç genel anlamda iki model şeklinde özetlenebilir.

* Geleneksel Modeller (Legacy Models)
* Çevik Modeller (Agile Models)

**Yaşam Döngüsü Çekirdek Süreçler (The Sofware Development Cycle)**

****

Piyasada, küçük farklılıklar ile listelense de özünde hepsi temel aynı anlamları kapsayan yaşam döngüsü süreçlerini gözükmeyiz. Biz ise aşağıdaki şekilde listeledik ve açıklamaya çalıştık.

1. Planlama (Planning)
2. Gereksinimlerin Belirlenmesi (Requirement)
3. Analiz (Analysis)
4. Tasarım (Design)
5. Gerçekleştirme (Implementation)
6. Test (Test)
7. İşletim ve Bakım (Operating and Maintenance)

**1.Planlama (Planning)**

Hangi işe bir plan oluşturmadan başlarız ki? Veya plansız bağladığımız bir işte ne kadar başarıyı yakalayabiliriz?

Bu gibi nedenlerden dolayı yazılım yaşam döngüsünün ilk ve önemli bir basamağıdır. Yazılım için temel ihtiyaçlar, toplantı tarihleri ve projenin genel anlamda planlanması ile bu basamak tamamlanmış olur. Müşteri ve yazılım ekibindeki çoğunluğun katılımı ve ortak kararlar ile planlama yazılım, yaşam döngüsünün ilerleyen safhalarının çok daha kolay ve sağlıklı geçmesini sağlar.

**2. Gereksinimlerin Belirlenmesi (Requirement)**

Karşılaşılan problem ile yüzleşildiği ve müşteri ihtiyaçlarının kararlaştırıldığı aşamadır. Müşteriden istekler ve müşteriye gereksinim önerileri verilir. Bir şeyi çözmek için o şeyi çok iyi anlamak gerekir bu yüzden her iki tarafın da **problemin ne olduğunu** çok iyi anlaması amaçlanır.

**3. Analiz (Analysis)**

Bu aşama müşteri ile yazılım ekibinin en fazla işbirliği göstermesi gerektiği adımlardan biridir. İstekler sonsuz, imkanlar ve zaman ise kısıtlıdır. Bu yüzden bu aşamada müşterinin ne istediğinden çok **neye ihtiyacı olduğunu** kararlaştırılır. Ve bu **dokümantasyona** dökülür**.** Önemli bir aşamadır çünkü artık yazılımın tam olarak ne yapması gerektiği belirlenir ve bir hüküme bağlanır. Çeşitli yazılım geliştirme modellerinde bu adımda test planları oluşturulmaya başlanır.

**4. Tasarım (Design)**

Gereksinim ve analiz aşamasından sonra teorik ve fikirsel olarak istenen yazılımın somut bir şekilde **nasıl** hayata geçirileceği kararlaştırılır. Yazılım sisteminin temel yapısı ve tasarımı oluşturulmaya başlanır.

iki tür tasarım sisteminden bahsetmek mümkündür.

* **Üst seviye ve mimari tasarım;** modüller, akış şemaları ve use caselere rastlanır.
* **Detaylı tasarım;** ekran tasarımları, veri yapıları, sınıflar ve yazılım programlama dili gibi olgular belirlenir.

**5. Gerçekleştirme (Implementation)**

Artık bu evrede önceki tüm yapılan planlar, belirlenen ihtiyaç ve istenen istekler kodlanır. Kodun “Clean Kod” felsefesine uygun kısa açık ve hedefe uygun bir şekilde oluşturulması gerekir. Bir çok alan dışı insanın aklına sadece bu kısım gelmekte ve yazılım yaşam döngüsünün sadece bu aşamadan ibaret olduğunu sanmaktadır. Oysaki bundan önce bir çok aşama tamamlanmış olmakla birlikte bu aşamadan sonra da takip edilmesi gereken bir çok adım bulunmaktadır.

Bu aşama fark yaratan ve yazılımcının kendi reklamını ve markasını oluşturduğu önemli bir aşamadır. Bu yüzden yazılımcının kendine özgü, bir başka yazılımcının kişinin kodunu gördüğü zaman “a evet bu x kişisinin kodudur.” diyebileceği bir özgün anlaşılır kodlama sitemi geliştirmelidir.

**6. Test (Test)**

Bu aşama gerçekleştirme aşamasından kesinlikle ayrı düşünülemez ve bir nevi içinde yer almaktadır. Gerçekleştirdiğimiz şeyi mutlaka sınamalı ve böylelikle sorunları baştan halletmeliyiz.

**Early testing (erken test) mantığına en baştan sahip olmalı ve analiz aşamasından itibaren bu mantık ile hareket etmeliyiz. Çünkü tabiri caiz ise yılanın başını baştan ezmek projelerde verim ve finansal anlamda çok büyük getiriler sağlamaktadır.**

**Diğer önemli husus ise gerçekleştirme aşamasında kodu oluşturan ekibin test aşamasında da tekrar kullanılmamasıdır. Çünkü hiçbir yazılımcı bilerek ve isteyerek hatalı kod yazmaz bu sebepten dolayı da kodu sıfırdan oluşturan ekip, test basamağında yer almamalıdır. Bu tür bir yanlışın yapılması halinde test aşaması başarısızlık ile sonuçlanmakta ve ileride çok daha büyük problemler ile karşılaşılmaktadır.**

1. **İşletim ve Bakım (Operating and Maintenance)**

**Yazılım oluşturulup müşterinin hizmetine sunulduğu aşamadır. Yazılım, oluşturuldu ve bitti gibi bir olgu olmadığı için ürününü piyasaya sunulduktan sonraki takibi ve iyileştirilmelerinin yapıldığı aşamadır. Bu noktada müşteri geri dönüşleri büyük önem taşımaktadır. Bu süreçte iki tarafın uzlaşması ve onayıyla yazılıma yeni özellikler eklenip çıkarılması da yapılabilir.**

**Şimdi de temel modellerden daha özel modellere geçiş yapalım.**

**Gelişigüzel Model**

**Aslında bazı uzmanlar bunu tam olarak bir model olarak ele almamaktadır. Çünkü belirli bir yöntemi ve kuralı bulunmamaktadır. Kişiden kişiye göre kodlama tarzı ve süreç değişmekte bunlar da yazılımın güncellenmesini ve bakımını çok zor hale getirmektedir.**

**Barok Modeli**

**Eski bir modeldir. 70’li yıllarda ortaya çıkmıştır. Yazılım yaşam döngüsündeki aşamalar sırayla takip edilir ve aşamalar arası geçişler net ve esnek değildir.**

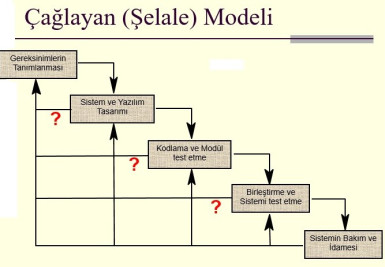
**Çok önemli olan dökümantasyon farklı bir süreç olarak ele alınıp yazılım yaşam döngüsünden koparılmış ve test aşamasından sonraya kadar itilmiştir. Bu yüzden süreç sırasında dökümantasyon açısından bir çok veri kayba uğramakta ve dökümantasyon açısından büyük sıkıntılar açmaktadır.**

**Yukarıda saydığımız bir çok olumsuz sebepten dolayı günümüzde geçerliliğini kaybetmiş ve uygulanan bir model olmaktan çıkmıştır.**

**Çağlayan Yaşam-Döngü Modeli (Waterfall)**

Günümüzde birçok modelin öncüsü ve atasıdır. Bu yüzden geçmiş yazılım dünyasının süper star bir yazılım yaşam döngü modeli denilebilir.

Bu modelde yazılım yaşam döngüsündeki her aşamaya büyük önem verilir ve çok ayrıntılı bir şekilde tanımlamaları ve içerikleri oluşturulur. Her aşamada dökümantasyon ve test yapılmadan aşama tamamlandı sayılmaz ve diğer aşamaya geçilmez. Bu çok iyi ve sağlam bir yöntem olarak görülse de projenin çok uzun sürelere sebebiyet vermesi ve tıkanmasına yol açabilmektedir.



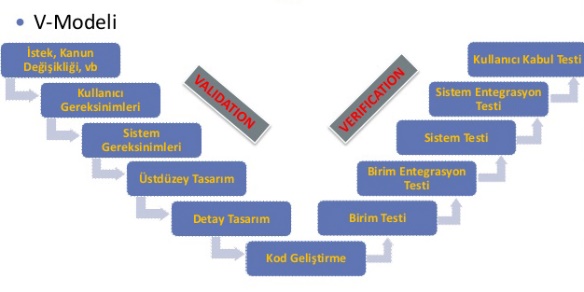
Bu modelin belki de en büyük sorunu ise yazılım geliştirme sürecinde müşteriyi işin içine neredeyse hiç katmamasıdır. Bu da yazılım geliştirme maliyetlerini artırmakta ve sonucun her iki tarafın da istediğini gibi bitmemesi gibi durumlar ile sonuçlanmasına sebebiyet vermektedir.

Bütçe sıkıntısının olmadığı, kamusal işler gibi dökümantasyonun önemli ve şart olduğu projelerde kullanımı uygun olmaktadır. Özellikle yüksek gizlilik ve adımlar içinde hata kabul etmeyen, ulusal güvenlik ve askeri projelerinde tercih edilmektedir.

**V Süreç Modeli**

V şeklinin sol tarafı üretim, sol tarafı ise test için ayrılmıştır. Bu model projenin iki takım halinde daha hızlı işlemesini sağlamaktadır. Müşteri daha fazla işin içine katılmakta ve görev üstlenmektedir. BT(Bilgi Teknoloji) gibi projeler için son derece uygundur.

Fazlar arasında tekrar yapmaması, çözüme direkt, düz bir çizgi ile gidememesi ve eş zamanlı çalışma sürecinin uyumsuzluk ve çakışma gibi etkenleri de bu modelin olumsuz yönleridir.

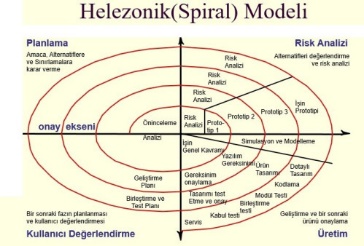


**Helezonik (Spiral) Model**

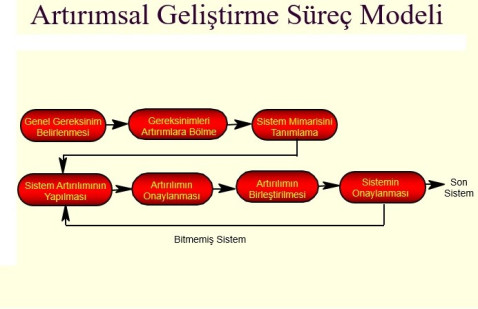
* Planlama : Üretilecek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme
* Risk Analizi : Risk seçeneklerinin araştırılması ve risklerin belirlenmesi
* Üretim :Ara ürünün üretilmesi
* Kullanıcı Değerlendirmesi : Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmeler

Risk analiz olgusu bu modelde bariz ve baskın bir şekilde görülmektedir. Yazılım sürecini küçük küçük modüllere bölerek daha kolay bir hale getirebilmektedir. Hatalar bu sayede daha erken giderilebilmektedir.

Sprial sonsuz kısır döngüye girebilmektedir. Karmaşık bir yapısı vardır. Çok büyük ölçekli proje kaynak ayırımı istemektedir.

****

**Artımsal Geliştirme Süreç Modeli**



Projeyi küçük küçük döngülere bölerek gerçekleştirme modeldir. Avantajı projenin ilerleyen dilimlerinde değişiklik yapıldığında daha düşük maliyetler ile kolayca halledilebilmesidir.

Uzun zaman alabilecek ve döngüler halinde çalışma yolunda sıkıntı yaşanmayacak projelerin tercihi olmalıdır.

Bu modelde her döngü, tasarım, kodlama, test ve entegrasyon süreçlerini içerir.

**Kodla ve Düzelt Yaşam-Döngü Modeli**

Günlük hayatta Carpe Diem diyebileceğimiz bir düşünce tarzı benzeri bir modeldir. Bu yüzden yazılım geliştirmenin en kolay yolu olmakla birlikte en pahalı yollarından biridir.

Genellikle en kolay olduğu için kurumsal olmayan, küçük firmalar tarafından sıkça kullanılmaktadır.

Bu ve bunun benzeri bahsettiğimiz modellerden dünyadaki yazılımcılar tam olarak istedikleri verimi alamadıklarından başka başka arayışlara girdiler. Ve bu aratış süreci 90’lı yıllarda meyvesini verdi ve camiaya çevik yazılım olarak ortaya atılacak olan bir yazılım geliştirme süreci çıktı. Bunlardan en popileri Scrum oldu.

**En Yaygın Uygulanan Çevik Metodolojiler**

* + **Extreme Programming (XP),**
  + **SCRUM,**
  + **Agile Unified Process,**
  + **Feature-Driven Development (FDD),**
  + **Test-Driven Development (TDD)**
  + **LEAN Development,**
  + **Dynamic System Development Methodology (DSDM),**
  + **Microsoft Solution Framework (MSF) olarak bilinen çevik metodolojiler vardır.**

Biz bunlardan en popilerleri olan SCRUM ve XP inceleyeceğiz.

**Extreme Programming (XP)**

Grıp iletişime önem veren ve yazılım geliştirme sürecinde insanların ve duyguların da var olduğunu ve bunları önemsiyen bir model olarak öne çıkmaktadır. Bir çok pratik uygulanma modeli olmakla birlikte şu an kabul gören 12 farklı uygulanma modeli bulunmaktadır.

* + Extreme Programming' in dört temel değeri vardır:
    - İletişim,
    - Basitlik,
    - Geri Bildirim,
    - Cesaret.

Süreçleri ksıa tutarak bilgi ihtiyacı gibi faktörleri hemen vakit kaybetmeden hallederek ilerlemek ister. Asıl olay sadelik ve basitliktedir. Karmaşık projeleri bu model pek sevmemektedir. Esnek yapı vazgeçilmezdir. Müşteri yazılım geliştirme sürecinin daimi bir üyesidir.

Geri bildirimler büyük önem taşır bu sayede ortaya çıkabilecek hatalar erkenden tespit edilip çözülür.

Koddan memnun olmayan yazılımcı bu modelde kodu tekrar yazmaktan çekinmez ve harekete geçmekten geri durmaz.

**XP Pratikleri**

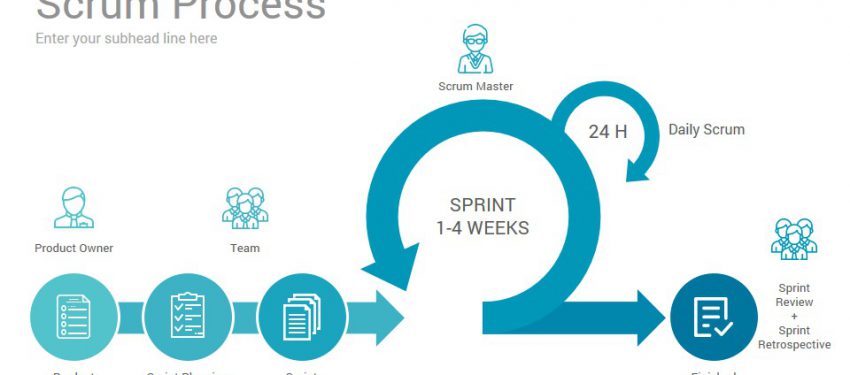
* + - Planlama Oyunu,
    - Ekipte Müşteri,
    - Önce Test,
    - Basit Tasarım,
    - Çiftli Programlama,
    - Sürekli Entegrasyon,

**SCRUM**

Bu model sadece yazılım geliştirme süreci için hayatın her evresine uyarlanabilme potansiyeline sahiptir.

Her gün yazılım ekibi ile kısa toplantılar yapmayı öğütler. Modelin asıl amacı 2 – 4 hafta arasında bir uygulama çıkarmaktır. Bunu ise hzılandırılmış bir şelala metodu yaparak yerine getirir.

Yazılım ekibi çok daha fazla çeşitlendirilmiş ve yazılım ekibinin de birer makine olmadığı insan olduğu göz önünde bulundurulmuştur.

Günümüzde ise çokça tercih edilmesinin sebebi start up kültürü ve rekabetin çok yoğun olduğu günümüz global dünyasında hızlıca uygulama çıkarma düşüncesidir. SCRUM adeta çok kısa sürelerde ne olursa olsun yazılım yaşam döngülerini tamamlayarak bir sonuç çıkarmaya odaklanmıştır. Bu da müşteri ve sermaye sahiplerini mutlu ediyor ve projede somut bir faaliyet görmüş oluyorlar. Aynı bakış açısından yazılımcı sonuca bir nebze olsun ulaştığını tadarak süreç boyunca motive bir şekilde kalabiliyor. x