**YAZILIM YAŞAM DÖNGÜ MODELLERİ**

Yazılım yaşam döngüsü nedir? Yazılımı bir canlıya benzetebiliriz, çünkü yazılımın da bir yaşam süreci vardır ve bu süreç birkaç adımdan oluşur:

Gereksinim: Müşterinin ulaşmak istediği yazılımın gereksinimlerini elde edildiği ve projenin planlandığı döngünün ilk aşamasıdır.

Analiz: Yazılım projesinin tamamlanmasının ne kadar zaman alacağı ne gibi risklerinin olacağı belirlenir. Bu aşamada müşteri, yazılım mühendisi ve diğer iş grupları arasında çalışmanın çok yoğun olması gerekir.

Tasarım: Yazılım projesinin nasıl sürdürüleceğinin ve neler yapılacağının tasarlanması. Bu aşamada yazılımın özellikleri belirlenir.

Gerçekleştirme: Yazılım projesi belirli bir noktaya geldikten sonra kodlama kısmı başlar. Kodlama sürecinden sonra ise test kısmı ile devam edilir.

Bakım: Yazılım projesinin ürün olarak sunulduktan sonra güncelleme, bakım olaylarının yapılmasıdır.

**Yazılım Yaşam Döngü Modelleri**

**1. Gelişigüzel Model**

▪ Bu yöntemi bir model olarak adlandırmak doğru değildir. Çünkü bir yöntem veya model yoktur.

▪ Gelişigüzel geliştirmede belirlenmiş bir model ya da yöntem bulunmaz.

▪ Genellikle kişiye özel yapılır ve bu yüzden yazılımın izlenebilirliği, bakım yapılabilirliği oldukça zordur.

▪ 1960'lı yıllarda uygulanan bu yöntem, genellikle basit programlama içeren ve çoğunlukla tek bir kişinin üretim yaptığı yöntemdir.

Dezavantajları

▪ Projenin teslim sonrası bakım yapılması oldukça zordur.

▪ Takım çalışmasına elverişli bir model değildir.

**2. Barok Modeli**

▪ 1970'li yıllarda kullanılmıştır. Yazılım yaşam döngüsünün adımları doğrusal bir şekilde ele alınır ve geliştirilir. Döngü yoktur.

▪ Aşamalar arasında gereken geri dönüşlerin nasıl yapılacağı belirli değildir.

▪ Belgeleme günümüzde kullandığımızda farklı olarak ayrı bir süreç olarak değerlendirilir. Yazılımın geliştirme ve test aşamaları gerçekleştirildikten sonra yapılır.

▪ Günümüz uygulanan bir yazılım yaşam döngüsü modeli olmaktan çıkmıştır.

Dezavantajları

▪ Aşamaları arası geri dönüşün belli olmaması karışıklığa sebeb olabilir.

▪ Güncel olmaması.

**3. Çağlayan (Waterfall) Modeli**

▪ Bu model günümüzde kullandığımız yaşam döngü modellerinin temelidir diyebiliriz. Geleneksel bir modeldir ve geçmişte oldukça popüler olan bir döngü modeliydi fakat günümüzde popülaritesini yitirmektedir.

▪ Geleneksel yazılım geliştirme modeli olarak da bilinir.

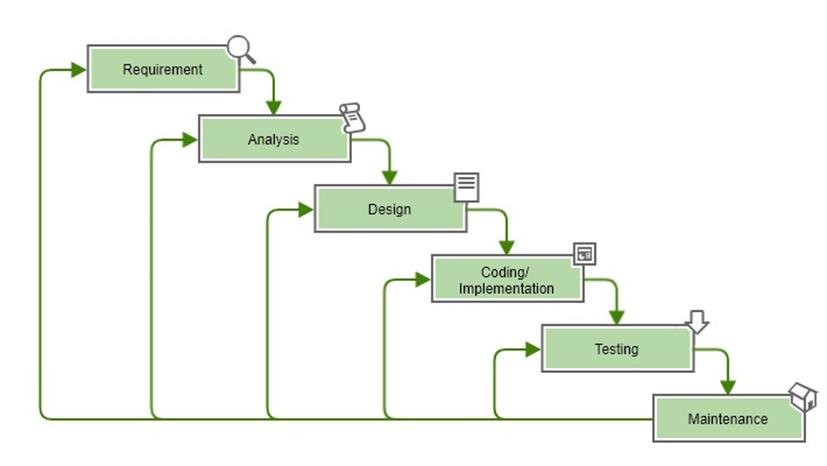
▪ Çağlayan modelinde yazılım, aşamaların en az birer kez tekrarlanması ile geliştirilir.

▪ Çağlayan modeli çok iyi tanımlanmış ve kısa sürede bitebilecek projelerde kullanılır. Bir sonraki adıma geçebilmek için bulunduğunuz adımı bitirmek zorundasınızdır.

▪ Her şeyin dokümantasyonu olması gerektir. O yüzden her adımda dokümantasyon hazırlanmalıdır. Eğer bir adımda dokümantasyon ve test gerçekleştirilmemiş ise o adımın tamamlandığı kabul edilmez.

▪ Bu modelde adımlar arasında geri dönüşler yapılabilse de analiz aşamasında mümkün olan tüm detayın tasarıma uygulanabilmesi için müşteri ve sistem gereksinimlerinin en ince ayrıntısına kadar tasarlanması gerekir.

▪ Bu modelde tasarım aşaması ek olarak ayrıntılı yapılmalıdır. Ayrıntıların çok olması sonucu gereksinim aşamasında ufak değişikler olması olağandır. Bu değişimler kodlama veya test aşamasında olursa maliyet artışı daha fazla olacaktır.



Avantajları

▪Adımlar arası geri dönüşe elverişli olması.

▪ Ayrıntılı ve sistematik bir yapıda olması müşteri memnuniyetini arttırabilir.

▪ Basit ve anlaşılması kolaydır.

Dezavantajları

▪ Gereksinim aşamasının eksik kalması sonucu değişimlerin artmasıyla maliyet artışı.

▪ Projenin bitme süresinin uzun olması.

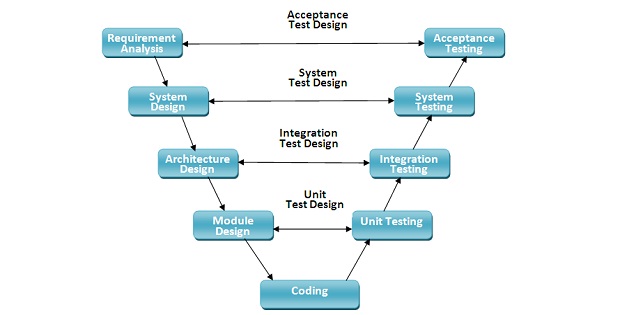
▪ Bu modelde çalışırken gerçekleştirme adımı odaklı olduğundan çalışılan ekipte mutsuzluk olasılığı artar.

▪ Müşteri veya kullanıcı projenin içerisinde yer almadığından dolayı olası bir sorun sonucu geri dönüşler teslim sonrası olur. Bu da maliyeti arttırıcı bir etkendir.

▪ Önceki aşamaya dönmek zordur.

▪ Fazla zaman gerektiren projeler için uygun değildir.

**4. V Süreç Modeli**



▪ V- model şelale modelinin gelişmiş hali olarak düşünülebilir. Bu model belirsizliklerin az olduğu, iş tanımlarının ise belirgin olduğu projelerde kullanılır.

▪ Sol taraf üretim, sağ taraf sınama (test) işlemleridir.

▪ Bu döngü temel aşamadan oluşmaktadır:

1-) Kullanıcı Modeli: Kullanıcının istekleri ve proje için tanımları kullanılır ve tamamlanmış hali teslim edilir.

2-) Mimari Model: Projenin tasarımı ve bunların denenmesi işlemlerinden oluşur.

3-) Gerçekleştirim Modeli: Kodlama ve bunların denenmesinden oluşur.

Avantajları

▪ V-modeli basit ve kullanımı kolaydır.

▪ Bu model, yazılımın güvenilirliğini ve kalitesini artırır.

▪ Gereksinimlerin kolayca anlaşıldığı küçük projeler için uygundur.

▪ BT (Bilgi Teknolojileri) projeleri için uygun bir modeldir.

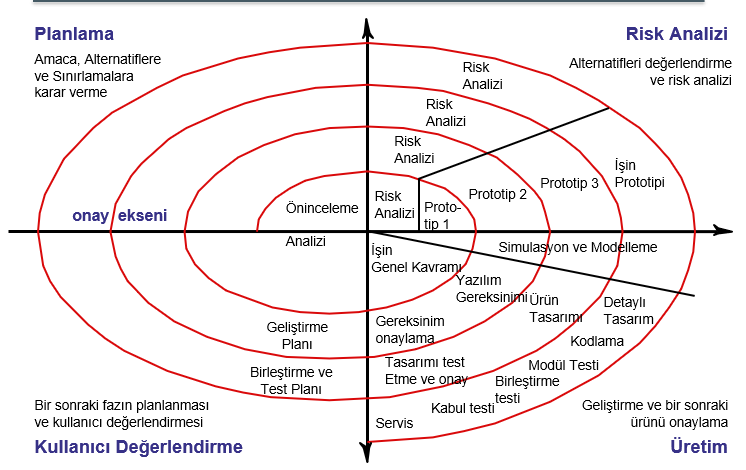
Dezavantajları

▪ Bu modelde gereksinimleri değiştirmek diğerlerine göre daha zordur.

▪ Komplike yazılımlar için iyi bir model değildir.

▪ Gereksinimleri değiştirmek maliyeti arttırır.

▪ Aşamalarda tekrar bulunmaz ve risk çözümleme için ayrılan bir yer yoktur.

**5. Helezonik (Spiral) Model**

▪ Helezonik modeli diğerlerinden ayıran özellik ise risk analizinin ön planda olması ve prototip oluşturulmasıdır.

▪ Prototip oluşturma da her aşamada olduğu için kullanıcı da her aşamada yazılım projesinin bir parçasını görme imkanına sahip olur bu da sorunların azalmasını sağlayabilir. Dört temel aşamadan oluşur:

1-) Planlama: Her aşamada olan ara ürün için bir planlama yapılır. Üretilecek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme.

▪2-) Risk Analizi: Risklerin araştırılması, belirlenmesi ve çözülmesi.

▪3-) Üretim: Ara ürünün üretilmesi.

▪4-) Kullanıcı Değerlendirmesi: Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmeler sonrası diğer aşamaya geçilmesi.

▪ Her döngü bir fazın varlığını gösterir.

▪ Doğrudan gereksinim, analiz, tasarım, vs. gibi bir faz bulunmaz.

▪ Yinelemeli artımsal ve prototip yaklaşımı vardır.

Avantajları

▪ Büyük projeler için uyumludur.

▪ Gereksinimleri değiştirmeye elverişlidir.

▪ Özellikler sistematik bir şekilde işlenir.

▪ Risk yönetimi kolaydır.

▪ Kullanıcının sürece erken dahil olmasından dolayı olası sorunlar erken tespit edilir ve çözülür.

▪ Yazılımın kodlanması ve test edilmesi daha erken başlar.

Dezavantajları

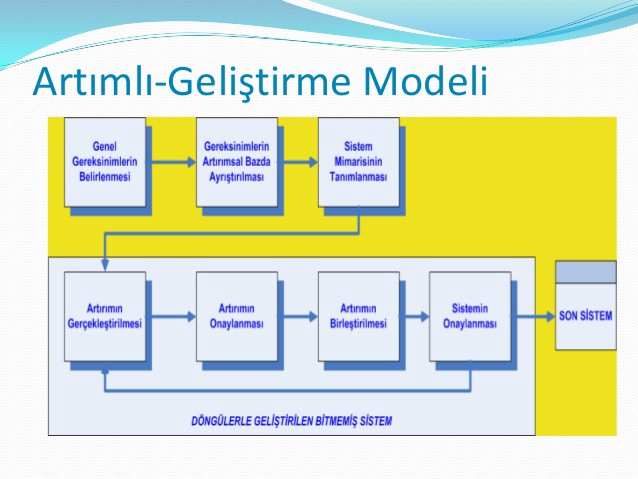
▪ Küçük projeler için uygun değildir ve ayrıca pahalıya mal olur.

▪ Diğer modellerden daha karmaşıktır.

▪ Sürecin çok zaman alması.

▪ Fazla dokümantasyondan oluşması.

**6. Artımsal Geliştirme Süreç Modeli**



▪ Geliştirme ve teslim ayrı olarak yapılır.

▪ Teslimler ayrı ayrı yapılır ve her teslim bir beklentiyi karşılar.

▪ Kullanıcının belirttiği gereksinimler kategorize edilir ve önem sırasına göre teslim önceliğine göre teslim edilir.

▪ Bir parçanın geliştirmesi başladığında, gereksinimleri durdurulur.

▪ Gerçekleşecek olan değişiklikler diğer teslimde uygulanır.

▪ Bir sonraki teslim piramit gibi diğerini kapsayacak şekilde olur.

▪ Kullanıcı son teslim modelini kullanırken diğer tarafta da üretim yapılır.

▪ Erken teslimler, sonraki teslimler için gereksinimleri çıkarmada prototip vazifesi görür. ▪ Projenin tümden batması riskini azaltır.

▪Bu model uzun zaman alabilecek ve ürünün eksik işlevsellikle çalışabileceği türdeki yazılımlar için uygundur.

▪ Bu model ile sistemin başarısız olma olasılığı azalır, ara ürünler yazılımın geliştirilmesinde önemli bir yere sahip olur.

Avantajları

▪ Yazılım gereksinimlerini değiştirmek daha uygun maliyetlidir.

▪ Yazılım hızlı bir şekilde geliştirilebilir

▪ Sürümler kolayca yönetilebilir.

▪ Riskler her sürümde tanımlanır ve bir sonraki sürümde çözülür.

▪ Sonuçlar erken ve periyodik olarak elde edilir.

Dezavantajları

▪ Daha fazla bütçe gerektirir.

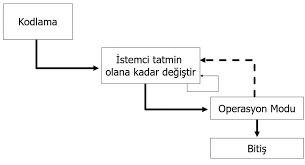
▪ Yazılım gereksinimlerini değiştirmek uygun maliyetli olsa da gereksinimleri değiştirmek için uygun değildir.

▪ Risk analizi daha zordur.

▪Küçük projeler için pek uygun değildir.

▪ Her bir parçanın kendi içinde tekrar etmesine izin verilmez bu yüzden de bir ara ürünün bitip diğeri başlayana kadar herhangi bir değişiklik yapılamaz, parçaları oluşturmak için de bu sistemin detaylı bir şekilde tanımlanması lazımdır

**7. Kodla ve Düzelt Modeli**



▪ Plansız bir şekilde ürün hazır olana kadar ya da gerekli zaman bitene kadar kodlama yapılarak devam edilir.

▪ Birkaç yüz satırdan oluşan programlar için kullanılabilir.

▪ İlk olarak yazılımın ilk sürümünü yaparız.

▪ Sistem, en son istenilen şekle gelinceye kadar devamlı değişiklikler ve eklemeler yapılır.

▪ Bakım az vardır ama zordur, ürünü hazırlar ve kullanıma sunarsınız.

▪ Ayrıca emeklilik aşaması vardır.

Avantajları

▪ Küçük programlar için kullanılabilir

▪ Yazılımı geliştirmek oldukça kolaydır.

▪ Herkes bu modeli kullanabilirler. Uzmanlık gerektirmez.

▪ Herhangi bir planlamaya ihtiyaç duyulmaz

Dezavantajları

▪ Kontrolsüzdür.

▪ Oldukça maliyetlidir.

▪ Büyük projeler için uygun değildir.

▪ Hata tespiti ve çözümü zordur.

▪ Yazılım müşterinin beğenisini kazanmayabilir.

**8. Evrimsel Geliştirme Modeli**

▪ İlk tam ölçekli modeldir.

▪ Geniş çaplı şirketler için önerilir.

▪ Her adımda üretilen ürün tam işlevselliğe sahiptir.

▪ Modelin başarısı ilk evrimin başarısına bağlıdır.

▪ Evrimsel geliştirme iki çeşittir:

1-) Keşifçi geliştirme (Exploratorydevelopment)

* Hedef: Müşterinin gereksinimlerini incelemek için müşteri ile çalışıp son sistemi teslim etmek.
* İyi anlaşılan gereksinimlerle başlanmalıdır.

2-) Atılacak prototipleme(Throw-Awayprototyping)

* Hedef: Sistem gereksinimlerini anlamak.
* Tam anlaşılmamış gereksinimlerle başlar.

Avantajları

▪ Müşterinin kendi gereksinimlerini anlamasını sağlar.

▪ Değerlendirmenin sürekli olması riski azaltır.

▪ Hatalar azalır.

Dezavantajları

▪ Düzenli olarak teslim edilmemesi sonucu sürecin nasıl ilerleyeceğini görmek zorlaşır.

▪ Yazılımın gereksinimlerinin yenilenmesi gerekebilir.

▪ Bakımı zordur.

▪ Sürekli değişiklik sonucu yazılımın yapısı zarar görür.

**ÇEVİK YAZILIM GELİŞTİRME METODOLOJİLERİ**

Çevik modeller yazılım projelerindeki başarı oranlarını ve ekip içi iletişimi arttırmak amacıyla ortaya çıkmıştır. Bazı çevik yazılım geliştirme modelleri şunlardır:

**· Extreme Programming (XP)**

▪ Dört temel maddeden oluşur: Basitlik, Cesaret, Geri Dönüş, İletişim.

1-) Basitlik: Yazılan kodun ve yapılan işin sade, anlaşılır ve karmaşık olmadan yapılmasını gerektirir.

2-) Cesaret: Yapılan işte cesur olunmalıdır. Bir kodun gerekirse tamamen silinip yeniden yazılması sağlanmalıdır.

3-) Geri Dönüş: Geri dönüşler ile oluşabilecek hatalar azaltılır/ortadan kaldırılır. Müşteri ile yazılım ekibi birbirleriyle iletişim halindedir.

4-) İletişim: İletişim, projelerde önemli sorunlardan birisidir. XP ise bunu aşmaya çalışmaktadır. Ekip içi iletişime önem verir ve artırılması için çalışır.

**· SCRUM**

SCRUM büyük projeleri parçalara ayırarak her birine “sprint” adını verir. Her bir sprinti teker teker geliştirir. SCRUM’da ekip içi iletişim çok önemlidir öyle ki her gün “SCRUM MEETINGS” denen toplantılar yapılır. SCRUM, gereksinimleri açıkça belli olmayan, değişime açık, karmaşık yazılım projelerin yönetimi için uygulanması en ideal yöntemdir. İhtiyaca yönelik, esnek, bir geliştirme sürecidir.



SCRUM’da üç temel kavram bulunmaktadır:

1-) Roller: Ürün sahibi, Scrum yöneticisi ve Scrum takımından oluşur. Takım beş ile dokuz arası kişiden oluşur. Bu insanlar birbirleriyle sürekli iletişim halindedirler.

2-) Toplantılar: Her gün SCRUM toplantıları yapılır bu günlük toplantılarda her gün yazılım geliştiricilerinin önceki gün neler yaptıkları, karşılaştıkları sorunlar ve bugün neler yapacakları hakkında bir konuşma olur. Her sprint için de bir gözden geçirme toplantısı yapılır.

3-) Bileşenler/Araçlar: Ürün Gereksinim Dokümanı oluşturulur. Bu dokümanda proje boyunca yapılması gerekenler basitçe yazılıdır. Sprint Dokümanı oluşturulur.

SCRUM’ın sık kullanılma sebeplerini şu şekilde sıralayabiliriz:

▪ Zamandan ve paradan büyük ölçekte tasarruf edilmesi.

▪ Gelişen teknolojiye kolayca uyum sağlaması.

▪ Karmaşık görülen ve gereksinimleri tam belirlenmemiş projelere uygun olması.

▪ Ekip içi iletişimin yüksek tutulması ve bununla beraber hataların erken fark edilip düzeltilmesi.

▪ Kullanıcıdan gelen sık geri bildirimler sonucu hataların kolayca tespit edilip çözülmesi.

▪ Diğer yazılım geliştirme metodolojileri gibi yinelemeli olması.

▪ Değişen gereksinimlere hızlı bir şekilde tepki vermesi.

▪ Parçalara bölünerek yapıldığı için göze daha yapılası ve kolay gelmesi.

**KAYNAKÇA**

Bakırçay Üniversitesi Prof. Dr. Öğretim Üyesi Deniz Kılınç Ders Notları

<https://medium.com/@omerharuncetin/yazılım-yaşam-döngü-modelleri-543c7879a742>

<https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696>

<https://www.researchgate.net/figure/Software-development-life-cycle-SDLC-of-waterfall-model-The-Digi-Learnik-application_fig1_336603302>

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

<https://cyberthreatportal.com/models-of-software-development-life-cycle/>

SEKER, S. E. (2015). Yazılım geliştirme modelleri ve sistem/yazılım yaşam döngüsü.

**Gökhan MİCOZKADIOĞLU 200601008**