**YAZILIM GELİŞTİRME SÜRECİNDE KULLANILAN MODELLER VE İŞLEVLERİ**

NUSRET ÖZTÜRK

210601018

(BİLGİSAYAR MÜHENDİSLİĞİ, BAKIRÇAY ÜNİVERSİTESİ, İZMİR)

**1.YAZILIM YAŞAM DÖNGÜSÜ NEDİR?**

Herhangi bir ürünün, yatırım veya servisinin canlı organizmalar gibi varlıklarını sürdürmelerinden hareketle yaşam döngüsü kavramı ortaya çıkmıştır. Maliyetlerin takibi ve kontrolünün yapılabilmesi için döngünün yönetilmesi gerekmektedir. Yazılımlara gelince, kullanılmakta ve geliştirilmekte olan her sistem için bir yazılım yaşam döngüsü söz konusudur ve bu döngünün de iyi yönetilmesi şarttır.

Yaşam döngüsünde genel kabul görmüş altı aşama vardır. Bunlar; ihtiyaçlar, tanımlama, tasarım, gerçekleştirme, uygulama ve bakım aşamalarıdır. Bu genel çerçeve içinde, bilgi sistemlerinin yaşam döngüsü modellerinden de bahsedilmektedir. Bu yaklaşımların ilki ve en yaygın biçimde kullanılanı, yaşam döngüsündeki aşamaların birbiri ardına sıralandığı, birinin çıktılarının bir sonraki adımın girdilerini oluşturduğu anlayışına dayanan Çağlayan Modelidir. Bu modelin gerçek yaşamda ortaya çıkardığı birtakım problemleri ortadan kaldırmak için öne sürülen Ön-Örnek Geliştirme Modeli, çağlayan modelinin bir türevi olan Artışlı Model ve yazılım mühendisliği sürecini her türlü uygulamayı içerecek şekilde açıklayan Sarmal Model, Hızlı Uygulama Geliştirme ve Yap ve Düzelt Modeli yaklaşımlardan belli başlılarıdır. Tüm modellerin kendi içinde zayıf ve kuvvetli yanları bulunmaktadır.

**1.1 PLANLAMA**

Yazılım yaşam döngüsünün ilk aşamasıdır. Temel ihtiyaçlar belirlenir, proje için fizibilite çalışmaları yapılır (maliyetlerin ve sistemin yararlarının tanımlanması) ve proje planlaması gerçekleştirilir.

**1.2 ANALİZ**

Bu aşamanın amacı sistemin işlevlerini ve kesin gereksinimleri açıklığa kavuşturmak ve sonucunda bunları belirli bir formatta belgelemektir. Bu çalışma müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi vb. rollerin bir araya geldiği gruplar tarafından yapılabilir. İhtiyaçların net olmadığı durumlarda yazılım mühendisi ve müşteri arasında iletişim ve birlikte çalışmanın çok daha fazla olması gerekir. Çeşitli yazılım metodolojilerinde bu aşamada kullanım dokümanları ve test plan dokümanları da oluşturulabilir.

**1.3 TASARIM**

Gereksinimlerin tamamlanmasıyla beraber sistem tasarım aşamasına başlanır. Yazılım ürün tasarımı, müşterinin gereksinim ve isteklerini karşılamak üzere yazılım ürününün özellikleri, yetenekleri ve ara yüzlerinin belirlenmesi etkinliğidir.

**1.4 GERÇEKLEŞTİRİM(KODLAMA ve TEST)**

Tasarım aşamasının belirli bir olgunluğa ulaşmasıyla birlikte kodlama aşaması başlar. Müşteriye teslim edilecek ürünü programlama aşamasıdır. Kodlama süresince ve kodlama sonrasında yapılan diğer önemli aşama test aşamasıdır. Erken test et yaklaşımı ile hareket edip, analiz aşamasından itibaren test bakış açısına sahip olmamız hata yapma oranımızı ve maliyetleri (zaman, para, prestij vb.) düşürecektir. Birim testleri, duman testleri, yanlış değer testleri, kabul testleri, yük testleri, kullanıcı kabul testi gibi sürece ve duruma göre uygulanabilecek çok farklı kategoride ve derinlikte test türü bulunmaktadır.

**1.5 TESLİM ve BAKIM**

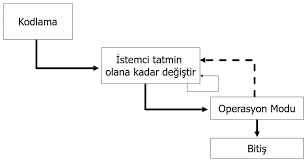
Tüm test aşamaları tamamlandıktan sonra yazılım ürününün sahaya teslim edilebilir bir versiyonu çıkartılır ve teslim aşaması gerçekleştirilir. Teslim çıktısı olarak ürün tek başına yeterli değildir. Mutlaka son kullanıcı için kullanım kılavuzu ve versiyon fark dokümanı oluşturulmalıdır. Teslim ile birlikte bakım aşaması da başlar. Hata giderici, önleyici, altyapıyı iyileştirici, ürüne yeni özellikler ekletici gibi farklı bakım faaliyetleri mevcuttur.

**2.YAZILIM YAŞAM MODELLERİ**

Literatürde birçok Yazılım Yaşam Döngüsü modeli vardır. Her modelin avantajları olduğu gibi dezavantajları da vardır. Fakat bir modelin her proje için en iyi olduğu konusunda belirli bir kural yoktur.

**2.1 YAP ve DÜZELT MODELİ**

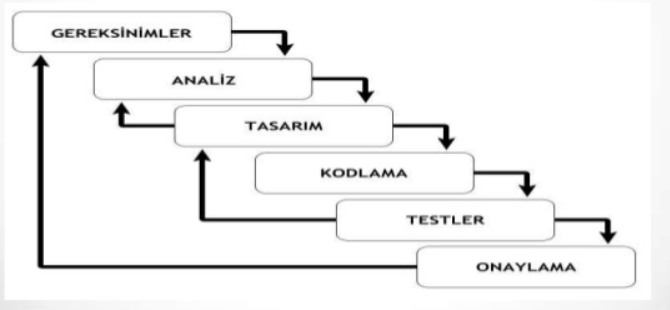
Bu model daha önce de belirtildiği gibi modelsizlik olarak tanımlanabilir. Yazılım, ihtiyaçlar belirtimi doğrultusunda hazırlanır ve müşteri tatmin oluncaya kadar geliştirilir. Aşağıda sistemin grafik gösterimi görülebilir.



Şekilden de görüleceği üzere bu modelde hiçbir analiz ve dizayn aşaması yoktur. Müşteri tatmin oluncaya kadar gerçekleşecek çok uzun bir geliştirme döngüsü söz konusudur. Döngünün çok uzun olmasını ve beraberinde getirdiği yüksek maliyet nedeniyle “yap ve düzelt” modeli genellikle tercih edilmez. Çok küçük çaplı yazılım isteklerinde ise uygulama kolaylığı nedeniyle tercih edilebilir.

**2.2 ÇAĞLAYAN YAŞAM-DÖNGÜ MODELİ**

1970’li yıllarda yaygınlaşan çağlayan modeli, basit ve tek boyutlu oluşuyla, günümüzde de bilgi sistemleri geliştirmede proje sözleşmelerine temel oluşturmakta yaygın olarak kullanılmaktadır. Projenin her aşaması, çağlayanın bir düzeyi olarak görülür, süreç, ilk düzeyden son düzeye doğru geriye dönüş olmadan ilerler. Aşamalar ve düzeyler, farklı uygulamalarda farklı biçimlerde bölünebilir, ancak bu modelin belirgin özelliği, geriye dönüşlerin öngörülmemesidir. Şekilde tipik bir çağlayan modeli örneği görülmektedir.



Çağlayan modelinin, bilgi sistemi geliştirme projelerinde nasıl sözleşme çerçevesi olarak kullanılabileceği açıktır. İşin sahibi ile projeyi gerçekleştiren kuruluş arasında, hangi aşamanın sonunda hangi ara ürünlerin teslim edileceği, bunların nasıl denetleneceği, vb. hükme bağlanır, hangi aşamalarda ne kadar ödeme yapılacağı kararlaştırılır ve proje başlar.

**Avantajları**:

**1.**Proje ilerlemesinin daha doğru bir şekilde izlenmesini, muhtemel kaymaların erken tespitini ve ölçülebilir yazılım geliştirmeyi sağlar.

**2.**Bütçeyi tahmin etme kolaylığı sağlar.

**3.**Teknik açıdan zayıf veya deneyimsiz personel için iyi çalışır.

**4.**Projeler daha yönetilebilir ve maliyet aşımı olmadan zamanında teslim edilir.

**5.**Sistemi test etmek ve korumak için kullanılan belgeleri üretir.

**Dezavantajları:**

**1.**Esnek değildir.

**2.**Her türlü ihtiyacı önceden tahmin etmek zordur.

**3.**Projenin sona ermesinden önce hiçbir gösterge yoktur.

**4.**Müşterililer, gereklilikleri tamamen, doğru ve net bir şekilde ifade etmelidir.

**5.**Planlama ve belgeleme için çok fazla zaman harcanmaktadır.

**6.**Gereksinimlerdeki değişiklikler ve hataları düzeltmek zor ve masraflıdır.

**2.3 ÖN ÖRNEK GELİŞTİRME MODELİ**

Sıklıkla, müşteri yazılımın genel hatlarını belirler, fakat girdi, işlemler ve çıktı konusunda ayrıntılı olarak bilgi sağlamaz. Bunun yanı sıra, geliştirici algoritmanın kesinliğinden, işletim sisteminin uygunluğundan emin olamaz. Bu tür durumlarda ön örnek modeli en iyi yaklaşımdır.

Diğer yaklaşımlar gibi, bu modelde gereksinimlerin belirlenmesi ile başlar. Bundan sonra hızlı tasarım aşamaları başlar, bundan sonra ön örnek yaratma gelir. Bu örnek müşteri tarafından değerlendirilir ve iyileştirmeler başlar. Bu öteleme, müşterinin gereksinimlerinin tam olarak karşılandığı anda son bulur. İdeal olarak ön örnek yaratma, yazılımın gereksinimlerini belirleme de kullanılır. Çalışan bir ön örnek yaratılırsa, geliştirici çalışan programı kullanılır hale hızlı bir şekilde getirmeye çalışır. Çağlayan modeli gibi, bu modelde bazı durumlarda problemli olmaktadır. Bunlar:

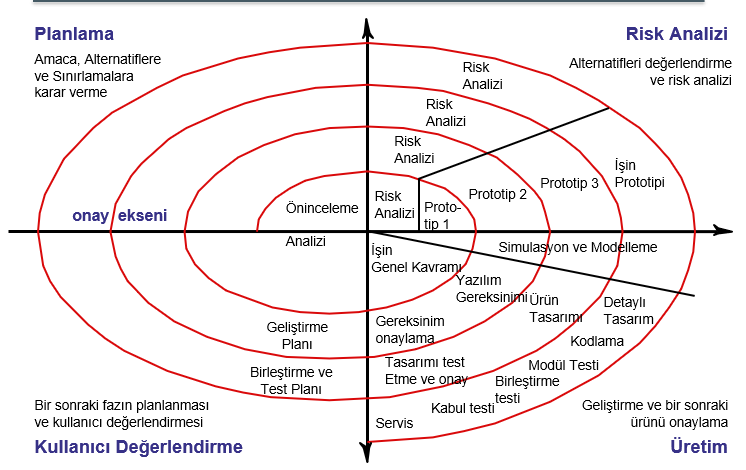
**1.**Müşteri çalışan modeli gördüğünde, bu ön örneğin ne işe yarayacağını bilmez. Bu yaklaşımın gereklerini bilmediği için problemler ile karşılaşılır.

**2**.Geliştirici çalışan bir ön örneğe sahip olmak için, bazı uzlaşmalarda bulunabilir. Uygun olmayan bir işletim sistemi veya algoritma, iyi bilindiği için kullanılabilir. Zaman geçtikçe, geliştirici bu seçimlerle uzlaşıp, neden diğer seçimlerin yapıldığını unutabilir.

Problemlerle karşılaşıldığı halde prototip modeli yazılım mühendisliği için etkin bir yaklaşımdır. Yöntemin iyi uygulanmasının anahtarı, oyunun kurallarının baştan belirlenmesidir. Müşteri ile baştan, bu modeli gereksinimlerin belirlenmesine yardımcı olduğu belirlenmelidir.

**2.4 HELEZONİK (SPİRAL) MODEL**

Yazılım geliştirme yaklaşımlarından olan bu model, klasik yaşam döngüsü ve prototip modelinin en iyi özelliklerinin birleştirilmiş halidir. Ek olarak, bu birleşime risk analizi eklenmiştir.



**1.Planlama:** hedeflerin, alternatiflerin ve sınırlamaların belirlenmesi.

**2.Risk analizi:** alternatiflerin ve risklerin analizi.

**3.Üretim(mühendislik):** bir sonraki düzey ürünün geliştirilmesi.

**4.Kullanıcı değerlendirmesi:** üretimin sonuçlarının değerlendirilmesi.

Spiral modelin numarası şekilde de görüldüğü gibi merkezden çevreye doğru olan yapıda saklıdır. Spiralin çevresindeki ilk devre üzerinde, hedefler, alternatifler ve sınırlamalar tanımlanır, riskler tanımlanıp analiz edilir. Eğer risk analizinde, bir problem varsa, prototip yaratma mühendislik kısmında yaratılır ve bu prototip geliştirici ile müşteriye yardımcı olur. Benzeşim veya diğer modeller, daha sonraki problem tanımlama ve gereksinimlerin iyileştirilmesinde kullanılır. Müşteri, mühendislik işini değerlendirir ve önerilerde bulunur. Bu verilere göre tekrar planlama ve risk analizi aşamaları gerçekleştirilir. Risk analizinin sonucuna göre projeye devam edilir ya da durdurulur.

Spiral model yaklaşımı, çok büyük hacimli işlerin geliştirilmesinde kullanılan en gerçekçi yaklaşımdır. Spiral model, projenin her aşamasındaki teknik risklerin üzerinde durur. Eğer iyi uygulanırsa, problem haline gelmeden riskleri indirger.

Bu modelde diğer modeller gibi her derde deva değildir. Her müşteriyi bu modelin kontrollü olduğuna ikna edebilmek zordur. Modelin uygulanması, risk kestirim deneyimi gerektirir. Eğer büyük bir risk fark edilmez ise problemler şüphesiz olarak oluşur.

**Avantajları:**

**1.**Kulanıcı sistemi başlangıçta görür.

**2.**Değiştirme gereksinimleri karşılanabilir.

**3.**Prototiplerin gereksinimleri karşılanabilir.

**4.**Geliştirme daha küçük parçalara bölünebilir ve riskli kısımları daha önce geliştirilebilir ve bu da daha iyi risk yönetimine yardımcı olur.

**Dezavantajları:**

**1.**Projenin sonu başlangıçta bilinmeyebilir.

**2.**Yönetimi daha karmaşıktır.

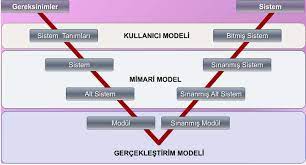
**3.**Süreç karmaşıktır.

**4.**Küçük veya düşük riskli projeler için uygun değildir ve küçük projeler için pahalı olabilir.

**2.5 ARTIMSAL MODEL**

Pratikteki ihtiyaç ve uygulamalardan doğan bu model bir varsayım hariç çağlayan modelinin benzeridir. Çağlayan modelinde tüm uygulama süreci tüm ihtiyaçların karşılandığı bir yazılım uygulaması şeklinde iken artışlı modelde ise yazılım yapı adı verilen fonksiyonel parçalara bölünerek uygulanır. Yapıların belirlenmesi ise sistem gereksinimlerinin belirlenmesi aşamasında müşterinin ihtiyaç duyduğu fonksiyonları kritiklik açısından değerlendirmesiyle olur. En kritik yapılar ilk olarak geliştirilip uygulamaya konulur. Ardından tüm gereksinimler karşılanıncaya kadar yeni yapıların geliştirilmesi ile geliştirme uygulama süreçleri arasında bir döngü oluşur. Bu esnada uygulamaya alınmış yapılar ise geliştirme-uygulama döngüsüne paralel olarak bakım sürecine geçerler. Artışlı model müşteri tatmini açısından oldukça başarılı bir modeldir. Müşteri birçok kritik olmayan fonksiyon için beklemek zorunda kalmaz. Bu modelin eksik yönü ise uygulanabileceği durumların modüler olarak modellenebilir durumlarla sınırlı olmasıdır. Pratikte neredeyse her fonksiyon diğer fonksiyonlarla etkileşim içerisinde olduğundan modüler yapılara rastlanma sıklığı enderdir.

**2.6 V SÜREÇ MODELİ**

****

Bu modeli çağlayan modelin uyarlanması olarak düşünebiliriz. Model V harfini andırırken sol tarafı üretimi, sağ tarafı ise sınama işlemleri ile ilgili aşamalardan oluşmaktadır. Her bir üretim aşamasının karşısında bulunan sınama aşamaları sayesinde hata kaynaklarına dönüş daha kolaydır.

Modelde temel çıktılar olarak: kullanıcı modeli, mimari model ve gerçekleştirim model olarak isimlendirilmiş üç alt model vardır.

**1.Kullanıcı modeli:** geliştirim aşamasında kullanıcı ile gerçekleşecek temasları tanımlar. Bu temaslar genel olarak gerçekleştirim veya test hakkında kullanıcı beklentileri hakkında gerçekleşir. Sistemin nasıl test edileceği bu aşamada planlanır.

**2. Mimari model:** sistem tasarımı ve oluşacak alt sistemlere ve tüm sisteme ilişkin oluşacak bütün test detaylarını içerir.

**3.Gerçekleştirim modeli:** yazılım modüllerinin kodlanması ve test edilmesi işlemlerinin gerçekleştiği bölümdür.

**Avantajları:**

**1.**Gerekliliklerin çok iyi anlaşıldığı küçük projeler için iyi çalışır.

**2.**Basit ve kolay anlaşılır ve kullanılır.

**3.**Her aşamadaki modelin sağlamlığı nedeniyle yönetilmesi kolaydır, belirli aşamalar ve inceleme süreci vardır.

**4.**Bu model oldukça disiplinli bir modeldir ve aşamalar birer birer tamamlanmıştır.

**Dezavantajları:**

**1.**Yaşam döngüsü sonuna kadar çalışma yazılımları üretilmez.

**2.**Uzun ve devam eden projeler için zayıf bir modeldir.

**3.**Karmaşık ve nesne yönelimli projeler için iyi bir model değildir.

**4.**Bir uygulama test aşamasında iken geriye dönüp bir işlevselliği değiştirmek zordur.

**5.**İhtiyaçların orta ve yüksek risk altında olduğu projeler için uygun değildir.

**2.7 ÇEVİK (AGİLE) YÖNTEMLER**

Çevik yöntemler yazılım geliştirme aşamasındaki verimi, başarı oranını ve ekip içi iletişimi arttırmayı amaçlayan bir yazılım geliştirme süreç modelidir.

**Avantajları:**

**1.**Kaynak gereksinimleri minimumdur.

**2.**Erken kısmi çalışma çözümleri sunar.

**3.**Değişen ortamlar için iyi bir modeldir.

**4.**İşlevsellik hızla geliştirilebilir ve gösterilebilir.

**5.**Değişim ve kolayca uyum sağlama esasına dayandığı için küçük ve orta çaplı projeler için uygundur.

**6.**Yönetmesi kolaydır ve geliştiricilere esneklik sağlar.

**7.**Genel olarak planlanmış bağlamda eşzamanlı geliştirme ve teslimat yapılmasını sağlar.

**8.**Minimal kurallar olduğu için dokümantasyon kolayca uygulanır.

**Dezavantajları:**

**1.**Sürdürülebilirlik için fazla risk vardır.

**2.**Karmaşık bağımlılıkların kullanımı için uygun değildir.

**3.**Dokümantasyon minimum seviyede oluşturulduğundan bireysel bağımlılık çok yüksektir.

**4.**Müşteri ile etkileşimine bağlı olarak, müşteri açık değilse proje ekibi yanlış yönde projeye devam edebilir.

Başlıca en çok kullanılan çevik yöntemler: SCRUM, XP yöntemleridir.

**SCRUM:** ismini Amerikan futbolundaki bir oyun olan tüm takımın birlikte hareket ettiği bir yöntemden alır. SCRUM da büyük projeler sprint adı verilen küçük parçalar halinde ayrı ayrı geliştirilir. SCRUM da asıl önemli olan ekip içi uyumdur. Bu bağlamda SCRUM toplantıları tarzı etkinlikler ile ekip bağları geliştirilir. SCRUM üç temel kavramdan temel alır. Bunlar: roller, toplantılar, bileşenlerdir.

**1.Roller:** Proje ekibi, ürün sahibi, SCRUM yöneticisi ve SCRUM takımından oluşur. Ürün sahibi projeye genel hatları ile hakim olan projenin beynidir. SCRUM yöneticisi projenin gidişatını ve SCRUM ekibini denetler. SCRUM ekibi yazılım geliştiricilerinden oluşmuş birbirleri ile iletişim halindeki gruptur.

**2.Toplantılar:** Toplantılar SCRUM için olmazsa olmazdır. Her gün yapılan bu toplantılarda projenin gidişatı, önceki gün ne yapıldığı ve bugün ne yapılacağı, projeden çıkan hatalar üzerine konuşulur ve içinde bulunulan sprint için değerlendirme yapılır.

**3.Bileşenler:** Proje için gereksinim dokümanı oluşturulur ve projede yapılacak olanlar bu dokümana basitçe yazılır. Gereksinim dokümanına bağlı olarak sprint dokümanları oluşturulur ve amaç her sprint dokümanının gereksinim dokümanına uygun hazırlanmasıdır.

Paradan ve zamandan tasarruf sağlaması, karmaşık görünen detaylandırılmamış projelerdeki kullanım kolaylığı, değişen gereksinimlere hızlı tepki vermesinin yanında teknolojilere de kolay adapte olması gibi yönleri ile günümüzde en çok kullanılan yazılım hatta bazı zamanlarda sistem geliştirme biçimi olmuştur. Kullanıcı ile sürekli bir geri bildirim ağı içerisinde olması projede gerçekleşebilecek hataları minimuma indirir. Ayrıca diğer metodolojiler gibi yinelemeli olması ve böl-fethet mantığı üzerine kurulu yapısı da yine kendisini alternatiflerinden öne çıkartıyor.

**XP(EXTREME PROGRAMMİNG):** 1996 YILINDA Kent Beck ve ekibi tarafından oluşturulmuş bu model, dört ana maddeden oluşur.

**1.Basitlik:** Yazılan kodun yalın, karmaşıklıktan uzak ve anlaşılır olması gerekir. Dokümantasyonunun uzun olmasından kaçınılır.

**2.Cesaret:** Projede önyargılar bir kenara bırakılıp korkusuzca kod üzerine işlem yapılmalıdır. Bazen yazılan kodun üzerinde değişiklik yapmak yeniden kod yazmaktan daha uzun olabilir. Bu bizim için hem daha maliyetli hem de daha fazla zaman kaybı demektir. Bu gibi bir durumda koda manevi bir şekilde bağlanmak yerine yeri geldiğinde kodu tamamen çöpe atmaktan çekinmemeliyiz.

**3.Geri Dönüş:** Yazılımın geliştirilmesi aşamasında proje üzerinde geri dönüş ve tekrar kontrollerin yapılması ile kodda bulunan hatalar azaltılır ve kaliteli bir ürün ortaya çıkar. Bunun için de müşteri ve geliştiriciler iletişim içinde olmalıdır.

**4.İletişim:** İletişim yukarıda da bahsedildiği gibi yazılım geliştirme süreci için çok önemli bir rol oynamaktadır. Kaliteli içeriklerin üretilmesi için kullanıcı yorumları ve test sonuçları geliştiriciler için yol haritası oluşturabilir. Bunun yanında sadece müşteri-geliştirici iletişimi değil geliştirici-geliştirici iletişimi de son derece önemlidir.

**Kaynakça:**

<https://webdosya.csb.gov.tr/db/cbs/icerikler/salihsoylu_tez_v10-20180925134450.pdf>

<http://ceur-ws.org/Vol-1221/6_Bildiri.pdf>

<https://polen.itu.edu.tr/bitstream/11527/5973/1/2137.pdf>

Deniz Kılınç Yazılım Sistem Temelleri ders notları

Okan Öztürkmenoğlu Yazılım Sistem Temelleri ders notları

**Linkedin:** https://www.linkedin.com/in/nusret-%C3%B6zt%C3%BCrk-4b335a228/

**Medium:** https://medium.com/@nusret.111oz

**Github:**  https://github.com/nusretozturk