**T.C**

**İZMİR BAKIRÇAY ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ FAKÜLTESİ**



**YAZILIM MÜHENDİSLİĞİ TEMELLERİ DERSİ**

**YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ MAKALE ÖDEVİ**

**Hasan Özden Sabuncu**

**210601692**

**Mühendislik Fakültesi**

**Doç.Dr.Deniz KILINÇ**

**MART, 2021**

**İÇİNDEKİLER**

1- Yazılım yaşam döngüsü modellerinin açıklaması

2-Çevik yazılım geliştirme ve açıklamaları

**YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ**

Herhangi bir yazılım projesini geliştirmek demek maalesef sadece kod yazarak halledilebilecek bir iş demek değildir. Yazılım geliştirme sürecinde, kod yazmak dışında projeyi geliştiren yazılımcı ürünü ortaya koymadan önce bir plan takip etmesi gereklidir. Takip etmesi gereken bu plan belli aşamalardan oluşmalı ve yeri geldiğinde tekrar tekrar uygulanmalıdır. Planın tekrar tekrar uygulanması demek ise yazılımın işlevlerinin ve ihtiyaçlarının sürekli değiştiği ve geliştiği yani bir döngü içinde olduğu anlamına gelmektedir. Döngü dereken her bir canlı nasıl doğar, büyür, gelişir ve ölürse, yazılımlar da canlılar gibi üretilir, geliştirilir, kullanılır ve en sonunda ne yazık ki kullanımı sonra erer. Yani canlıların nasıl bir yaşam döngüsü var ise yazılımlar da aynen öyle bir döngü içerisindedir. Bu duruma da “Yazılım Yaşam Döngüsü” denir. Yazılım yaşam döngüleri, bir yazılım ürününü ortaya çıkarırken izlenmesi gereken aşamalar ve adımların bütünüdür. Peki yazılım yaşam döngüsündeki temel aşamalar nelermiş, kaç aşamadan olurmuş gelin beraber bu sorulara yanıt verelim. Yazılım yaşam döngüsü genel olarak 5 aşamadan oluşmaktadır. Bunlar sırasıyla gereksinim, analiz, tasarım, gerçekleştirme ve bakım adımlarından oluşmaktadır. Gereksinim: Yazılım yaşam döngüsün ilk aşamasıdır. Bu aşamada müşteri ile iletişime geçilerek gereksinimlerin yani müşterinin yazılımdan ne istediğini öğreniriz ve projenin fizibilitesini belirleriz. Analiz: Bu aşamada geresinim aşamasındaki belirlenen istekler ve işlevler detaylı olarak incelenir. Yazılımın projesinin ne kadar sürede tamamlanacağı belirlenir ve bunlar analiz edilerek temel sorunlar ortaya çıkarılır. Tasarım: Analiz kısmının tamamlanması sonucunda tasarım aşamasına geçilir. Bu aşamada yazılımın veya kurulacak sistemin ayrıntılı tasarımı yapılarak projenin hangi ara yüzde nasıl görüneceğinin tasarımları yapılır. Kısacası genel bir çerçeve çizilir ve tüm bileşenler ayrıntılı olarak modellenir. Gerçekleştirme: Bu aşama projenin kodlandığı ve test edildiği kısımdır. Ayrıca kurulum çalışmalarının da başladığı aşamadır. Bakım: Son aşamadır. Projenin tüm aşamalarının bitirilip müşteriye teslim edildikten sonra test sırasındaki tespit edilemeyen hataları giderme ve yeni eklentiler yapma aşamasıdır. Bu aşama ürünün tüm yaşamı boyunca sürer. Bir yazılım projesinin temel aşamalarını bu şekilde öğrenmiş olduk. Şimdide bu aşamalardan oluşan birden çok yazılım yaşam döngü modellerine bakma zamanımız geldi. Bunlar günümüze kadar gelen en yaygın yazılım yaşam döngü modellerinden birkaçı şunlardır: Gelişigüzel Model, Barok Modeli, Çağlayan Yaşam-Döngü Modeli, V süreç modeli, Helezonik (spiral) Model, Artımsal Geliştirme Süreç Modeli, Kodla ve Düzelt Yaşam-Döngü modelleridir. İlk olarak gelişigüzel modeli açıklamayla başlayalım. Gelişigüzel modelde herhangi bir model veya yöntem yoktur. Tamamen projeyi geliştiren kişiye bağlı bir yazılımdır. Bu yüzdende takibi, bakımı ve yapılabilirliği oldukça zordur. 1960’lı yıllarda uygulanan bu model çoğunlukla tek kişilik, basit programlama gereken yazılımlarda kullanılmıştır. Günümüzde geçerliliğini yitirmiştir. Barok Modeli: 1970’li yıllarda kullanılan Barok modelinde yazılım yaşam döngüsünün temel adımları doğrusal şekilde uygulanır. Aşamalar arasındaki yapılması gereken geri dönüşlerin nasıl yapılacağı ifade edilmemiştir. Belgeleme günümüz modellerinden farklı olarak ayrı bir adım olarak işlenir ve diğer tüm adımlar bitirdikten sonra yazılır. Kısacası analiz, tasarım, gerçekleştirme ve test faaliyetleri bitirildikten sonra dokümantasyon safhası yazıya dökülür. Günümüz projelerinde artık kullanılmayan bir yazılım yaşam döngü modelidir. Çağlayan Yaşam-Döngü Modeli: Çağlayan modeli yazılım dünyasındaki en eski ve en popüler yazılım geliştirme modeli olarak bilinir. Aynı zamanda geleneksel yazılım geliştirme modeli olarak da bilinir. Çağlayan modelinde işler aşama aşama yapılır ve birisi bitmeden diğerine aşamaya kesinlikle geçilmez. Her adımda dokümantasyon yazılması gerekir. Herhangi bir aşamada dokümantasyon yazılmamışsa, o aşamanın tamamlandığı kabul edilmez. Bu modelde safhalar arasında geri dönüş yapılabilse bile analiz aşamasında mümkün olan tüm detayın tasarıma yansıtılabilmesi için müşteri ve sistem gereksinimlerinin en ince ayrıntısına kadar belirlenmesi gerekir. V süreç Modeli: Bu modelin ana fikri, modeldeki her bir geliştirme aşamasına karşılık bir test aşamasının olmasıdır. Model iki sıralı süreçten meydana gelir. Modelin şeklindeki sol taraf üretim aşamalarını ve sağ taraf ise test aşamalarını temsil eder. Model bir yandan ürünün geliştirirken bir yandan da geliştirilen kısmın testini yaparak müşterinin projede aktif olarak rol almasını sağlar. Kod yazma safhası, bu iki kısmı birleştiren aşamadır. Şimdi gelin V süreç modelinin temel çıktılarına göz atalım. Kullanıcı Modeli: Bu kısımda müşterinin istekleri doğrultusunda sınama belirtimleri ve projenin planı ortaya konulmalıdır. Mimari model: Bu kısımda ise sistemde kullanılacak ve sınama edilecek mimari işlevler belirlenir ve tasarlanır. Gerçekleştirim: Bu kısımda ise yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar yapılır. Helezonik (spiral) Model: Bu model risk kavramının ilk kez ortaya konulduğu modeldir. Helezonik modelde aşamalar spiral şekilde tekrar ederek ilerler. Her tekrarda risk analizleri yapılarak bir prototip elde edilir. Elde edilen prototipler müşteriye sunularak, müşterinin en başından itibaren yazılımı takip etmesi amaçlanır. Bunun sonucunda sonraki aşamalarda karşılaşabilme olasılığı yüksek olan sorunlara önceden çözüm üretmek mümkün olur. Spiral modeli, 4 ana başlıkta incelenir: Planlama: Bu kısımda öncelikle üretilecek ürünün amacı ve bunun doğrultusunda ürünün üretim planı ortaya konur. Risk Analizi: Olası risklerin araştırılması, belirlenmesi ve bu konularda kapsamlı analizlerin yapıldığı aşamadır. Üretim: Prototipin yani ara ürünün üretildiği aşamadır. Kullanıcı Değerlendirme: Kullanıcı tarafından prototipin test edilerek, yorumlarının alındığı aşamadır. Artımsal Geliştirme Süreç Modeli: Bu modelde öncelik olarak kullanıcının ihtiyaçları göze alınır. Öncelikle en temelden başlayarak bir yazılım sürecini temel versiyonunu ortaya çıkarıp ardından oradaki eksik bölgeleri tamamlayan ve hataları gidererek daha gelişmiş ve daha iyi bir sürümünü ortaya çıkaran bir yaşam döngü modelidir. Bu demektir ki üretilen her bir ara ürün, bir önceki ara ürünün geliştirilerek üstüne konulmuş halidir. Böylece bu döngüde bir taraftan üretim kısmı sürerken bir taraftan da kullanım başlamış olur. Her prototipte sonuca daha fazla yaklaşılır. Kodla ve Düzelt Yaşam-Döngü Modeli: Bu modelde projeye ait dokümantasyon ve planlar yoktur. Direkt olarak yazılım ürünü gerçekleştirme aşaması ile başlanır. Ürünün kod kısmı bittiği an yazılımın ilk hali yayınlanır. Hata bulunursa eğer kod üstünde düzenleme yapılarak, yeni versiyon yayınlanır. Bakım aşaması vardır ama maalesef ürünün dokümantasyon aşaması olmadığı için bir hayli zordur. Yazılım geliştirme modelleri arasında en kolayıdır. Ancak ne yazık ki en pahalı olanıdır da. Ayrıca modelde emeklilik safhası da bulunur. Makalemin önceki kısımlarında değindiğimiz yaşam döngü modellerine ek olarak 20.yüzyıl bitimlerinde önemi ortaya çıkan ekip içi iletişim ve takım çalışmasını öne çıkaran, zaman kaybını azaltan, stabil olamayan, maliyeti azaltan ve değişikliklere kolayca uyum olabilen Çevik Modeller geliştirildi. Çevik yazılım geliştirme olarak makalemizden iki ana başlık olarak Scrum ve Extreme Programming (XP) den bahsedeceğiz. Extreme Programming: Model, 1999 yılında Kent Beck tarafından ortaya çıkarılmıştır. Model basit, grup içi iletişime önem veren ve müşteriyi de projeye dahil ederek geri dönüşlerin daha fazla olmasına imkân sağlayan bir yazılım geliştirme metodudur. Extreme Programming dört ana değer ve 12 alt temel prensibi vardır. Ana değerler; İletişim, Basitlik, Geri Bildirim ve Cesarettir. İletişim: Birkaç başarısız projelere baktığımızda başarısız olma nedenlerinin iletişim eksikliği olduğunu görürüz. Projenin başarılı olabilmesi için ekipteki insanların birbiriyle sağlıklı bir iletişim içinde olması gerekir. Extreme Programming iletişim eksikliği ortadan kaldırmayı amaçlar. Herhangi bir noktada bilgi alınması gerekiyor veya bir sorun ortaya çıkarsa, müşteri ile yazılım ekibi iletişime geçerek yazılımın gelişme hızı kesilmeden projeye devam edilmelidir. Ve her zaman yazılım ekibi birbiriyle yüz yüze şekilde iletişim halinde olmalıdır. Basitlik: Yazılan yazılım ve geliştirilen proje, sade, anlaşılır ve karmaşık bir durum içinde olmadan çözülmelidir. Karmaşıklık işi uzatacak ve zorlaştıracaktır. Bu yüzden Extreme Programming karmaşıklığı sevmez. Geri Bildirim: İletişim kısmında da bahsettiğimiz gibi ekip üyelerinin birbiriyle her daim iletişim halinde olması gerektiği gibi ekibin müşteriyle de iletişim halinde olması gerekir. Çünkü müşteriden aldığı geri bildirimler sayesinde projedeki hatalar hemen fark edilip düzeltilebilir. Özetle müşterilere de aynen proje grubunun bir üyesi gibi davranılmalıdır. Cesaret: Extreme Programming in dört ana değeri arasından en zoru cesarettir. Yazılım projelerini yılmadan, pes etmeden geliştirmek projeler için çok önemlidir. Başarısızlıktan korkmak yerine, yeri geldiğine yaptığımız şeyden hoşlanmayıp cesur davranarak projemizi çöpe atıp yeniden yazabilmeliyiz. Elimizi korkak alıştırmamalıyız. Çünkü korku sadece projemizin hızının düşmesini sağlar. Şimdide dört ana değer dışındaki, bize kolaylık ve esneklik sağlayan bazı alt temel prensipleri inceleyelim. Planlama Oyunu: Proje ekibinin ve müşterinin de içinde bulunduğu bir toplantıda, ürünün ne kadar zamanda yapılacağını tahmin eden bir oyundur. Planlama oyununda yeni yazılımcıların tahminleri pek tutmaz. Basit Tasarım: Amaç müşterinin isteklerini karşılayabilecek en basit tasarımı gerçekleştirmektir. Tasarım basit olması, projenin anlaşılmasını, yönetilmesini ve sonradan değiştirilmesini kolay kılar. Haftada 40 Saat: Fazla mesai yazılımcılar için iyi değildir. Yazılımcıların da her çalışan gibi zaman ayırmaları gereken sosyal hayatları olmalıdır. Haftada 40 saat çalışmak bir yazılımcı için ideal bir süredir. Üstüne çalıştığı her saate verim artacağına, yapılabilecek hata sayılarında artış meydana gelir. Ekipte Müşteri: Makalenin önceki kısımlarında belirtiğimiz gibi, müşteriyi de ekibin bir parçası olarak kabul etmeliyiz. Kısa Aralıklı Sürümler: Bu prensipte proje birbirinden farklı zaman dilimlerine bölünür. Her bir zaman dilimde yapılacak işin kendisine ait bir teslim tarihi vardır. Böylece müşteri, projeyi zamanında teslim alır. Ortak Kod: Geliştirilen yazılım, sadece bir kişinin kodu değildir. Bütün ekip üyelerinin kodudur. Çiftli Programlama: İşe ya da projeye yeni katılan yazılımcılar için önerilir. Usta-çırak ilişkisine benzetebiliriz. Yazılımcıların arasındaki farklı yetenek ve bilgiler sayesinde birbirilerine yardımcı olurlar. Bu tarz programlamada yazılım projesinin geliştirilme hızını arttırır. Şimdide çevik yazılım geliştirmedeki ikinci maddemiz olan Scrum’ı inceleyelim. Scrum, Jeff Sutjerland ve Ken Schawaber tarafından 1995 yılında geliştirilen bir çevik yazılım geliştirme modelidir. Scrum’ da yazılım sprint adı verilen küçük parçalara bölünür. Sprint ‘in bir tanesi en fazla 1 ay sürer. Günlük 15-30 dakika süren toplantılarla sürekli iş takibi yapılır. Toplantılar sayesinde projenin genel durumunun takip edilir. Model, “Böl ve Fethet” stratejisine benzerlik gösterir. Amaç büyük bir projeyi en fazla 30 gün sürecek süreçler şeklinde ve ortaya çıkacak ürünlere ayırarak gerçekleşmesini sağlamaktır. Ve bu model sadece yazılım geliştirmede değil, her alanda uygulanabilir bir modeldir de. Scrum üç ana kavram barındırır. Bunlar: Roller, Toplantılar ve Bileşenlerdir. Üç rol bulunur. Ürün Sahibi: Proje takımı ile müşterinin arasındaki iletişimi sağlarlar. Projenin gidişatını takip etmekle sorumlu, olup, geri dönüşü sağlarlar. Scrum Yöneticisi: Scrum’ın kurallarını ve pratiklerini iyi bilip, ekibin bu kurallara uymasını sağlarlar. Ayrıca ekibi ve organizasyonu Scrum’a adapte ederek çalışma ortamını iyileştirirler. Scrum Takımı: 5 ila 9 kişiden oluşan, devamlı iletişim halinde olan ve bir sprint boyunca yapılacak olan işleri yapmaya çalışan takımdır. Takımdaki herkesin tek bir görevi yoktur, çapraz şekilde görev dağılımı halindedirler. Üç çeşit toplantı bulunur. Sprint Planlama: Bu toplantıda, product backlog’daki gereksinimler belirlenerek küçük görevlere ayrılır ve scrum ekibindeki kişilere dağıtılır. Maliyet hesaplamaları ve risk değerlendirmesi yapılır. Sprint Gözden Geçirme: Her bir sprintin sonunda yapılan toplantıdır. Koşu boyunca oluşturulan yazılımlar ve yapılması gereken işlemler incelenir. Günlük Scrum Toplantısı: Sprint boyunca her gün yapılan 15–30 dakika süren, tüm takımın katıldığı ayak üstü bir toplantıdır. Ekip üyeleri teker teker; “Dün ne yaptın?”, “Bugün ne yapacağım?”, “Karşılaştığım sorunlar neler?”, gibi sorulara cevap verir. Eğer sorun yaşayan biri olursa scrum mastera danışır, scrum masterda ona yardımcı olur. Son olarak bir de bileşenlere göz atalım. Ürün Gereksinim Dokümanı: Müşteri ihtiyaçlarının önceliklerine göre sıralandığı listedir. Canlı bir dokümandır, değişen gereksinimlere göre değişiklik yapılabilir. Genellikle kullanıcı hikayelerinden oluşur ve kullanıcı bakış açısından bakılır. Sprint Dokümanı: Mevcut koşu için product backlog’dan elde edilmiş iş ve görevleri kapsar. Sprint backlog’daki işlerin amacı, sprint sonunda son ürünün bir parçası olan bir işlevselliği ya da çalışabilir bir parçayı elde etmektir. Sprint backlog sadece ekip tarafından değiştirilmektedir. Sprint Kalan Zaman Grafiği: Grafikte, sprint boyunca işlerin kaç tanesinin yapıldığı ile normalde ne kadar yapılması gerektiğini karşılaştırabilmesini sağlar. Scrum ekibinin performansını takip etmek için de kullanılabilir.

Yararlanılan Kaynaklar

<https://atakavuncu.medium.com/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-modelleri-f4738b5cfcc1>

<https://emrehalac2015.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCleri-modelleri-ve-kar%C5%9F%C4%B1la%C5%9Ft%C4%B1r%C4%B1lmas%C4%B1-301ed59d01dd>

İzmir Bakırçay Üniversitesi, Doç.Dr.Deniz KILINÇ BİL102-Yazılım Mühendisliği Temelleri, 3. ve 4. Hafta Ders Notları

<https://osmanozaydin.com/yazilim-yasam-dongusu-ve-agile-yazilim-gelistirme/>