Yazılım Yaşam Döngüsü

**Utku DİŞCİ - 190601068**

Yazılım ürünlerinin oluşumunun planlanmasına başlandığı günden ta ki yazılımın emekli olduğu ve artık kullanılmadığı zamana kadar geçen süreye “Yazılımın Yaşam Döngüsü” adı verilir. Bir yazılımın oluşturulduğu zamana ve kullanıcı kitlesine göre kullanılması gereken farklı yaşam döngüsü modelleri vardır. Bu modellerin genel amacı müşterinin ihtiyaçlarını aktif bir şekilde karşılamak, üretim maliyetini düşürmek, yazılımın belirlenen zamanda çıkmasını sağlamak ve yazılımda oluşabilecek hataları, bu hatalar kritik seviyelere gelmeden çözebilmektir. Yazılım yaşam döngüsü modelleri genel olarak belli aşamalardan oluşur. Farklı modellere değinmeden önce bu modelleri oluşturan aşamalardan bahsetmek isterim.

**Gereksinimler:** Projeye başlamadan önce projenin ilerleyişi ile ilgili çeşitli planlamalar bu aşamada yapılır. Müşterinin genel gereksinimleri alınır ve bunların yapılabilirliklerine karar verilir.

**Analiz:** Müşterinin gereksinimlerine bağlı olarakprojenin sahip olacağı işlevler ve yazılımın müşterinin hangi isteklerine karşılık verebileceği bu aşamada planlanır.Yapılması mümkün olmayan, başka bir şekilde yapıldığında daha verimli olabilen ya da yapıldığında bütçenin aşılmasına ve şirketlerin zarara girmesine yol açabilecek kararlar belirlenir ve müşteriye bildirilir. Bu aşamanın dikkatli bir şekilde yapılmaması projenin ilerleyen aşamalarında temelden problemler yaratacak, para ve zaman kaybına yol açacaktır.

**Tasarım:** Tasarım aşamasından önce proje belli parçalara bölünerek kişilerin uzmanlık alanlarına göre iş bölümü yapılır. Bu işlem projenin yapımını kolaylaştırır ve karmaşıklıkları önler. Ardından müşterinin özel istekleri ve kullanıcı kitlesi göz önünde bulundurularak yazılım ürününün işlevselliği ve arayüz tasarımı belirlenir. Tasarım aşaması Mimari Tasarım ve Detaylı Tasarım şeklinde iki kategoriye ayrılır. Mimari tasarım, projede önerilen sistemin yapısını yüzeysel ve açık bir şekilde anlatır. Modüller, akış şemaları ve UML kullanılır. Detaylı tasarımda ise yazılımı içeren bileşenler ve bunların ayrıntıları detaylı bir şekilde verilir.

Tasarım aşaması boyunca yapılan değişikler, verilen kararlar ve bu aşamanın geleceği için yapılan planlar düzenli ve titiz bir şekilde dokümante edilmelidir. Bu dökümanlar projenin ilerleyen aşamalarında yapılabilecek geri dönüşlerde ya da projeye sonradan katılmış kişilerin projeyi iyi anlamasında oldukça yararlı olacaktır.

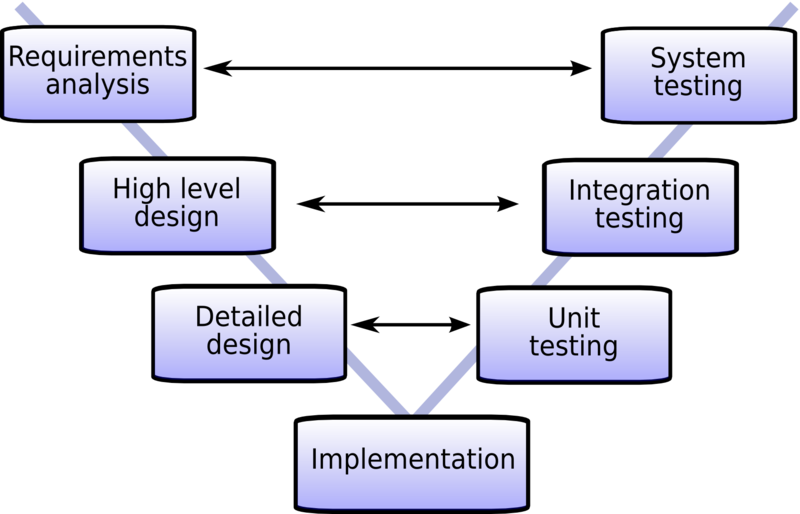
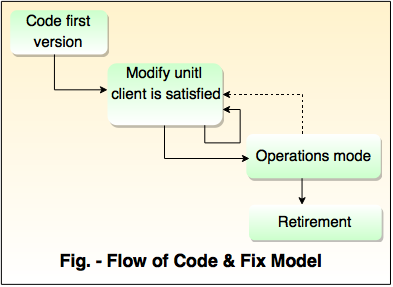
**Gerçekleştirme:** Tasarım aşamasında belirlenen modüllerin, seçilen programlama diliyle kodlanmaya başladığı aşamadır. Modüller birbirleriyle birleştirilir ve programı oluşturulan kod yazılır. Ardından program birçok çeşitli testlerden geçer. Bu testler sonucu ortaya çıkan hatalar dikkatlice belgelenir ve bunlara hata kodları verilir. Bu belgeleme ve kodlama işlemi, ileride aynı hatanın oluşması durumunda hatayı çözmeyi oldukça kolaylaştıracaktır. Kimi zaman şirketler programları beta sürümleriyle halka veya belirli bir test grubuna açık hale getirir. Bu işlem farklı kişilerin, farklı işletim sistemleri ve donanımlarda programı kullandıklarında ortaya çıkabilecek hataların kolaylıkla keşfedilmesini sağlar. Tabi ki böyle bir test için kullanıcılardan verimli bir şekilde feedback (geri bildirim) alabilmek gerekir. Bunun için kullanılan test grubuna anketler yapmak ve programa otomatik bir feedback fonksiyonu eklemek gerekir. Bu testler sadece hataları bulmaya değil, ayrıca programın farklı sistemlerdeki çalışma hızlarını ve verimini öğrenmeye de yarar. Programın çalışma günlükleri de kullanıcıların izniyle feedback içinde gelir ve bu sayede proje ekibi çeşitli performans sorunlarına çözüm bulmak için çalışabilir.

**Bakım:** Bakım aşaması, projenin teslim edilmesinden sonra başlar ve projenin emekliliğine kadar sürer. Düzenli olarak alınan geri bildirime ve kullanıcıların ya da müşterinin isteğine göre programda değişimler, düzenlemeler ve yenilikler yapılmasını kapsar. Sürekli olarak gelişen teknolojiyle birlikte eski kalan bir program verimini yitirecek ve kullanıcılar bu programı kullanmak istemeyeceklerdir. Bu yüzden programa düzenli olarak güncellemeler yapılmalıdır. Bu güncellemeler update olarak geçer ve programın versiyonuna kaydedilir.

Yeni bir programın versiyonu 1.0.0 olarak geçer. İlk sayı bize programın Major, ikinci sayı Minor ve üçüncü sayı da build değerini verir. Bu numaralandırma sistemi sayesinde kullandığımız programın hangi sürümde olduğunu ve ne zaman yenilendiğini bilebiliriz. Oldukça büyük ve köklü değişikler yapıldığı zaman genellikle Major değer değiştirilir ve 2.0.0 ile devam edilebilir. Program içinde yapılan ufak değişikliklerin gösterilmesi için minor değerler kullanılır.Örnek olarak basit bir arayüz değişikliği gibi durumlarda, ” Ergosoft 1.0.0” sürümünden, “Ergosoft 1.1.0” a geçilir. Üçüncü değer gözden geçirip düzeltme anlamına gelen revision dur.

Basi hata düzeltmeleri, bug lar ve çok küçük değişiklikler burada yer alır. Genelde haftalık ya da aylık kontrollerde bulunan bu hatalar giderilir ve yeni bir build piyasaya sürülür. Çoğu şirket programlarını bu yeni sürümler çıktığında otomatik güncellenmesi için ayarlarlar.

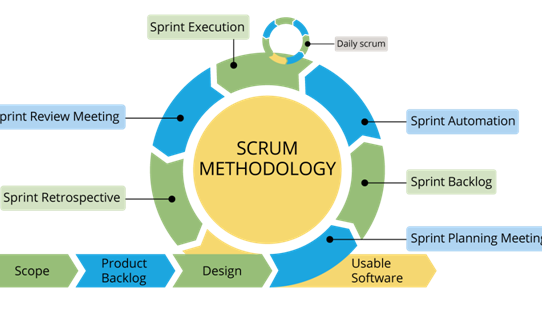
Yazılım Geliştirme Modelleri

1. **Gelişigüzel Modelleme:** Aslında bu yöntemi bir model olarak adlandırmak pek doğru olmaz çünkü bu yöntem hiçbir planlama veya belirlenmiş yöntemi kullanmaz. Proje ekibi tabiri caizse, bodoslama proje yaparlar ve bu projenin ilerleyen süreçlerinde birçok hatayla karşılaşırlar. Genellikle tek kişilik projelerde ya da acemi proje ekiplerinde görülür ve belgelendirme konusunda önemli bir eksikliğe sahip olan bu yöntemle oluşturulan projelerin izlenebilirliği ve bakım yapılabilirliği çok düşüktür. Küçük çaplı, basit projelerde, planlama ve tasarım gibi uzun süren aşamalarla uğraşılmadığı için, oldukça zaman tasarrufu sağlar ancak dokümantasyonun eksikliği yüzünden projeye bakım yapmak çok zor olur.
2. **Barok Modeli:** 70’li yıllarda ortaya çıkan bu modelde yazılım yaşam döngüsünün temel adımları doğrusal bir şekilde işlenir. Aşamalar arasında yapılan geri dönüşlerin nasıl yapılacağı tanımlı değildir çünkü dokümantasyonun en sonda yapılması öngörülür. Bakım ve geri dönüşlülüğe uygun bir model olmadığı için günümüzde kullanımı pek önerilmez.
3. **Çağlayan Modeli:** En eski, en tanınmış ve en temel modeldir. Geçmişte en popüler yazılım geliştirme modeli olarak görülmüştür ve “Geleneksel Yazılım Geliştirme Modeli” olarak da bilinir. Yaşam döngüsü temel aşamalarının en az birer kez tekrar edilmesini temel alır. Oldukça iyi ve hatta bazı ülkelerin hükümet standartlarına girmiş olmakla birlikte günümüzde kullanımı gittikçe azalmaktadır. Her aşamasında dokümantasyon yapmak esastır ve bir aşama tamamen bitmeden öbürüne geçilmez. Geri dönüşlerin nasıl yapılacağı tanımlı olduğu için bakım aşaması oldukça kolay ve verimlidir. Bu modelin kötü yanlarından bazıları ise müşterinin yapım süreciyle yeterince ilgili olmayışıdır. Bu da proje teslim edildikten sonra birçok geri dönüt gelmesine ve projenin yapımının uzamasına neden olabilir ve bu nedenle projenin maliyeti oldukça artabilir.. Ayrıca tasarım ve planlamada oldukça zaman harcanan bu modelde yazılım üretim ekipleri sabırsızlanmakta ve mutsuz olmaktadır.[](https://tr.m.wikipedia.org/wiki/Dosya:Waterfall_model.svg)
4. **V Süreç Modeli:** Bu modelde projenin üretim aşamaları iki koldan ilerler. Genellikle sol taraf üretim, sağ taraf ise test aşamalarından sorumludur. Bilgi Teknolojileri projeleri için uygun bir model olarak görülür. Ayrıca kullanıcının projeye katkısı oldukça önemsenir.V modelide üç adet temel çıktı vardır. Kullanıcı Modeli: Geliştirme sürecinde kullanıcı ile olan ilişkileri tanımlar. Mimari Model: Sistemin tasarımı ve oluşacak alt sistemle ilişkili tüm sistemin sınama işlemlerine ilişkin işlevleri tanımlar. Gerçekleştirim Modeli: Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonları tanımlar.[](https://commons.wikimedia.org/wiki/File:V-model-en.png)
5. **Helezonik Model:** Helezonik modelde program, sürekli olarak geri dönmeye, yenilenmeye ve eldeki verileri arttırmaya yönelik oluşturulur. Proje dört ana bölüme ayrılır. Bu gruplar planlama, risk analizi, yönetici ve kullanıcı değerlendirmesidir. Kullanıcı değerlendirmesi, yazılımı kullanacak kişilerin yazılımın üretimi süresince bu sürece dahil olmasına ve sürekli yorumlar yaparak yazılımın kullanıcının isteğine göre şekillenmesine yardımcı olur. Bu 4 grup bütün yazılım süreci boyunca aktiftir ancak bu aynı anda iki grubun aktif olması anlamına gelmez. Bir grubun işi bitmeden öbürüne geçilmez. Kullanıcıyı aktif olarak üretim sürecine kattığı için yapım aşamasında kullanıcının yorumları ve geribildirimleri alınabilir. Dinamik yapısı sayesinde hataların çözülmesini kolaylaştırır ve güncellemeleri kolaylaştırdığı için maliyetten tasarruf sağlar. Fakat bu modelin eksik yanları da vardır. Bir çok parçaya ayrıldığı için diğer modellere göre oldukça fazla belgeleme yapılır. Bu nedenle fazla büyük olmayan projelerde zaman kaybına yol açabilir.
6. **Artımsal Geliştirme Süreç Modeli:** Sistemi tek seferde teslim etmek yerine, geliştirme ve teslim parçalara bölünür. Teslim edilen her parça beklenen işlevselliğin bir kısmını karşılar. Son ürün yavaş yavaş, parça parça üretilir. Kullanıcının ihtiyaçları belirlenir ve öncelikli ihtiyaçlar erkenden bitirilir. Bir parçayı geliştirilmeye başlanıldığında, o parçanın gereksinimleri dondurulur ve eklenecek herhangi bir yeni gereksinim, bir sonraki teslimde yapılmak üzere beklemeye alınır. Oluşturulan her yeni sürüm, kendinden öncekileri kapsayacak şekilde üretilir ve giderek işlevselliği artar. Uzun zaman alabilecek ve kimi özelliklerinin zamanla eklenmesi uygun olan projeler bu modelle yapılabilir. İteratif bir süreçtir, her tekrarla son ürüne daha da yaklaşılır. Üretim ve Kullanım aynı anda olacağı için müşterinin geri bildirimi erken safhalarda alınabilir ve projenin işlevselliği erkenden belli olmuş olur.
7. **Kodla ve Düzelt Yaşam-Döngü Modeli:** Sadece birkaç yüz satırdan oluşan, küçük programlar için kullanılabilir. Yazılım ürünü ilk safhada üretilir ve sonradan küçük değişikliklerle istenen şekle gelene kadar devamlı geliştirilir. Bakım safhası vardır ancak oldukça zordur çünkü belgelendirme yapılmamıştır. Birçok değişken hatası çıkması olasıdır. Yazılım geliştirmenin en kolay yoludur. Oldukça kolay olduğu için küçük bütçeli firmalarda ve tecrübesiz yazılımcıların projelerinde sık sık görülür. [](https://www.tutorialride.com/software-testing/software-development-process-models.htm)

**Çevik Yazılım Modelleri**

2001’de eski yazılım modellerinin gelişen teknolojiye ayak uyduramadığını ve müşterilerin düzenli olarak artan gereksinimlerini karşılamadığını düşünen 17 yazılım mühendisinden oluşan bir grup, Utah’da toplanarak bu konu üzerinde tartışmış ve 4 temel değer ve 12 destekleyici ilkeden oluşan bir manifesto hazırlamıştır. Agile Manifesto, süreç ve araçlardan ziyade bireyler ve etkileşimlere, kapsamlı dokümantasyondan ziyade çalışan yazılımlara, sözleşme pazarlıklarından ziyade müşteri iş birliğine ve bir plana bağlı kalmaktan ziyade değişime karşılık vermeye değer verir. Bu manifestonun yayınlanmasından önce Scrum, XP ve diğer bazı çevik yazılım modellerinin, var olduğunu belirtmekte yarar var ancak sonrasında bu modeller "Çevik Yazılım Modeli" adı altında toplanmışlardır.

**SCRUM:** 90'larda oluşturulmuş ve günümüze kadar gelişme göstermiş bir modeldir. Scrum, rugby oyununda oluşturulan küçük ekiplere verilen addır. Gereksinimleri açıkça belli olmayan, değişime açık, karmaşık projelerin yönetimi için uygun bir modeldir. Esnek bir yönetim sistemidir. Scrum'ın artan popülerliğinin nedenleri arasında sürecin şeffaf olması ve aksayan noktaların açığa vurulmasıdır. Böylelikle proje ekibi ortaya çıkan aksaklıkları çözümlemek ve sürekli projeyi ilerletmek için motive olur.

[](https://www.educba.com/what-is-scrum/)

**XP:** Extreme Programming ya da kısaca XP, çevik programlama prensiplerinden birçoğunu barındırır ve temelinde yıllardır kullanılan doğru yazılım geliştirme yöntemlerinin aşırı bir şekilde kullanılması yatar. XP takım odaklı bir modeldir ve bu takıma müşteriyi de dahil eder. Müşteri, planlama toplantılarına katılır ve XP nin gerekliliklerini yerine getirecek kadar yetkin olmalıdır. Müşterinin sorumlu, istekli, kararlı ve aktif olması gerekir. Müşterinin bir diğer görevi de user story ler oluşturmaktır. User story ler, birkaç küçük cümleyle anlatılan, teknik konulara fazla girmeden oluşturulan, kullanıcı bakış açısıyla yazılan geribildirim yazılarıdır. Ardından proje ekibi aralarında bu user story lerdeki taleplerin hangilerinin bir sonraki sürümde ekleneceğini tartışırlar. Buna sürüm planlaması denir.

XP ye göre geliştiriciler çalışmaları eşli yapmalı ve eşlerini görevin gereksinimlerine göre seçmelidir. Bu sistem takım içinde iletişimi ve işbirliğini kuvvetlendirir. Müşterinin düzenli geribildirimi eksiklerin giderilmesini ve yapım anında çıkabilecek hataların ortadan kaldırılmasını kolaylaştırır ve bu da geliştiricileri motive eder.

Projenin gelişimi süresince yapılması gereken bir diğer şey de yapılan işlerin takip edilmesidir. Xp de bu işi yapmaya gönüllü bir kişi seçilir ve bu kişiye **tracker** lakabı takılır. Bu kişi projede çalışan tüm geliştiricileri haftada bir iki kez ziyaret eder ve durumlarını kontrol eder. Ne kadar iş yaptıklarını ve ne kadar işleri kaldığını not alır. Böylece, çalışan ekiplerde herhangi bir problem olursa önceden görülmüş olur ve bu problem büyümeden çözülebilir. Ayrıca her gün 15’er dakikalık **stand-up meeting** adlı toplantılar düzenlenebilir. Bu toplantılar duyuruların ve yaşanan problemlerin aktarılma yeridir. Gerek duyulursa bu problemler hakkında ayrı toplantıların düzenlenmesine karar alınabilir.

Kaynakça

1. Doç.Dr.Deniz Kılınç,(2020),”Yazılım Mühendisliğine Giriş Ders Notları”,Bakırçay Üniversitesi
2. <https://www.educba.com/what-is-scrum/>
3. Barış Aydek, <https://dhalsim.github.io/software/2015/11/14/software-development-processes#agile-model>
4. Burak Altuğ Semercioğlu, <https://medium.com/@buraks/%C3%A7evik-proje-y%C3%B6netimi-scrum-nedir-ne-de%C4%9Fildir-ab41d7b2b738>
5. <https://www.tutorialride.com/software-testing/software-development-process-models.htm>
6. Agile Manifesto, [http://agilemanifesto.org](http://agilemanifesto.org/)
7. <http://agilemanifesto.org/principles.html>
8. [http://www.agilealliance.org/](http://www.acm-software.com/)
9. <http://www.acm-software.com/>