**YAZILIM YAŞAM DÖNGÜ MODELLERİ**

Bu yazımda sizlere Yazılım Yaşam Döngü Modelleri hakkında bilgi vereceğim. Ayrıca en önemli çevik yazılım geliştirme ürünlerinden biri olan Scrum’a da değineceğim. Yazılım Yaşam Döngü Modellerinin ayrıntılarına girmeden önce modellerde yer alan temel adımlara bir göz atalım.

**Yazılım Yaşam Döngü Modeli’nin Temel Adımları**

Yaşam döngü modeli bir yazılım geliştirilirken takip edilecek adımlardır. Genel olarak altı safha içerir.

**Gereksinim Safhası**: Burada elimizdeki problemin ne olduğunu anlamaya çalışırız. Müşterinin ihtiyaçlarını belirlemek buradaki en önemli aşamadır.

**Analiz Safhası:** Bu aşamada bir önceki aşamada topladığımız gereksinimleri analiz ederek işe başlarız. Analiz sonucunu şartname dökümanı olarak yazarız. (Bunu yaparken müşterinin ne istediğini değil neye ihtiyacı olduğunu dikkate almak önemlidir. Müşteri yazılım ürününün yapma imkanı olmayan uçuk isteklerle gelebilir. Durumu analiz edip gerekensimleri belirlemek ve kesinleştirmek bu aşamda önemldir.) Bütün bunlardan sonra kendi çalışmamız için bir Yazılım Proje Yönetim Planı oluşturmak gerekir.

**Tasarım Safhası**: İki tasarım türü vardır: Mimari Tasarım ve Detaylı Tasarım. Mimari Tasarımda kaba taslak planlar ve modüller gerçekleştirilir. Detaylı Planda ise kullanılacak algoritmalar, diyagramlar vb. hazırlanır.

**Gerçekleştirme Aşaması:** Modüllerin her biri için kodlama yapılır. Modüller birbirinden bağımsız olarak test edilir. Sonraki aşamda modüller birleştirilir ve toplu halde test edilir. En son da gerçek veriyi kullanarak sistem test edilir. Müşteri kabul testi yapar.

**Teslim Sonrası Safhası**: Burada da iki tip bakım olduğunu görüyoruz. Düzletme Bakımında sistemde oluşan hatalar düzeltilir. Özelliklerin arttırılması ise bir nevi güncelleme yapmak demektir. Şartnamede değişikler meydana gelirse o değişiklikler bu kısımda yazılıma uygulanır. Teslim sonrası bakım aşamaları önemlidir. Şunu unutmamak gerekir iyi yazılım, yaşayabilen yazılımdır. Yaşayabilen yazılım da bakıma ihtiyaç duyar. Kötü yazılım ise bakıma ihtiyaç duymaz çünkü direk atılması gerekir. Bunun yanı sıra bu aşamada çıkacak hatalar büyük maliyetlere neden olur çünkü her şey bitmiştir ve o hatanın düzeltilmesi ona bağlı sistemlerin değiştirilmesi anlamına gelebilir. Yani yazılım içerisindeki hatayı ne kadar erken bulup düzeltirsek hatayı giderme maliyeti de o kadar düşük olur.

**Emeklilik Safhası**: Yazılım artık gelişen donanın ile birlikte işlevlerini yerine getirememeye başladığında faaliyetlerinin durdurulması olayıdır.

**Yazılım Yaşam Döngü Modeli**

Teoride yazılım geliştirme bir işe sıfırdan başlayarak, gereksinimler, analiz, tasarım ve gerçekleştirme aşamalarının doğrusal olarak tanımlanmasıyla gerçekleşir fakat pratikte böyle işlemez. Sonuçta hepimiz insanız tabiki hata yapabiliriz. Bu durumda önceki adımlara dönüp düzeltme yapmamız gerekebilir. Bir de bizden kaynaklanmayan bir sıkıntı olarak müşterinin gereksinimleri değişebilir. Bu durumda yine başa çıkıp düzeltmeler yapmak gerekir. Bu yüzden tüm bu işlemler baştan başlayıp son adıma kadar doğrusal olarak ilerleyen bir dizi gibi değil de bir döngü olarak düşünülmelidir. Şimdi bu temel fikri özümsediğimize göre Yazılım Yaşam Döngü Modellerinin ayrıntılarına girme vakti geldi.

**1 .Gelişigüzel Model**: Aslında buna tam olarak bir model diyemeyiz çünkü belirlenmiş bir yöntem bulunmaz. Kişiye bağlı olarak gerçekleştirilir. Bu yüzden izlenebilirlik ve bakım yapılabilirliği oldukça zordur. Tek kişinin yapabileceği basit programlama işlemlerinde kullanılan bir modeldir.

**2. Barok Modeli:** Bu model Yazılım Yaşam Döngüsü adımlarını doğrusal bir biçimde ele almışır. Geri dönüşlerin nasıl yapılacağı tanımlı değildir. Dökümantasyon ayrı ele alınır testlerden sonra yapılır. Günümüzde çok rağbet görmeyen bir modeldir.

**3.Çağlayan Yaşam Döngü Modeli**: Geleneksel bir modeldir. Yazılım aşamalarının en az bir kez tekrarlanması ile geliştirilir. İyi tanımlanan ve üretimi az zamanda gerçekleştirilebilen projeler için işe yarar bir modeldir. Günümüzde gördüğü rağbet giderek azalmaktadır. Burada işler aşama aşama yapılır. Biri bitmeden diğer aşamaya geçilmez. En önemli noktalardan biri de her safhada dökümantasyon yazılmasıdır. Öyle ki dökümantasyon ve test olmadan o aşama bitmemiş kabul edilir. Görüldüğü üzere Barok modelinden farklı olarak Çağlayan Yaşam Döngü Modeli’nde dökümantasyon üretimin doğal bir parçasıdır. Ayrıca bu modelde aşamalar arası dönüşlerin nasıl olacağı da tanımlıdır. Her ne kadar aşamalar arası geri dönüşler yapılsa da analiz kısmının tasarım aşamasında sıkıntı yaşamamak için müşteri ve sistem gereksinimlerinin en ince aşamasına kadar belirlenmesi gerekmektedir. Dolayısıyla proje ekipleri en fazla zamanı ilk iki adımda harcarlar.

Yazılın geliştirme sürecini inceleyecek olursak bu kesinlikle yineleme gerektiren bir işlemdir (küçük çaplı projeleri dışarıda tutarsak) ve bu süreçte yazılım ürünün kullanıcıya ulaşma süresi uzundur. Gereknsinimlerin tanımlanması belirli sınırlar içine koyulması çoğu zaman net bir biçimde yapılmadığından dolayı ileriki aşamalara meydana gelecek bir sorunda hataları düzeltme veya eksiklerin giderilme maliyeti daha pahalı olacaktır. Yazılım üretme ekipleri genellikle bir an önce kodlama, programı yazıp, çalıştırıp sonucu görme eğiliminde oldukları için bu modelin kullanıldığı projelerde ekipler kod yazma dışında kalan kısma fazla önem vermemektedirler. Modelde kullanıcı sürecin içerisinde yer almaz ve bu durum haliye yazılım oluşturulup kullanıcıya sunulduktan sonraki geri dönüşleri arttırır. Bu geri dönüşler yazılım geliştirme maliyetini önemli ölçüde arttırırlar.

**4. V Süreç Modeli:** Adı üstünde V şeklinde bir model düşünün. Sol taraf üretim sağ taraf sınama işlemleri için kullanılıyor. Adımlar üstten alta birikimli bir şekilde ilerliyor. V For Vendetta filminden ilham alan birkaç yazılım geliştiricisinin bu modeli oluşturduğu düşünülüyor. Tamam tam olarak öyle olmasa bile estetik açıdan gayet güzel duran bir model. Bu modelde üç temel çıktı var.

Kullanıcı Modeli => Geliştirme sürecinin kullanıcı ile olan ilişkisini tanımlar. Planlar ortaya çıkarılır.

Mimari Model => Sistem tanımı ve oluşacak alt sistem ile tüm sistemin sınama işlemine göre ilerler.

Gerçekleştirim Modeli => Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonları içerir.

Bu model kullanıcının projeye olan katkısını ve katılımını arttırmaktadır.

**5. Helezonik (Spiral) Model**: Dört basamak içerir : Planlama, Risk Analizi, Üretim ve Kullanıcı Değerlendirmesi. Modelde risk analizi olgusu öne çıkar. Her döngü bir fazı ifade eder. Gereksinim, analiz… gibi doğrudan belli temel aşamalar görülmez. Yinelemeli artımsal bir yaklaşım izler de diyebiliriz.

Üretim süreci boyunca ara ürün üretme ve bu ürüün kullanıcının tarafından sınanması esasına dayanır. Kullanıcıyı olaya erken dahil ederek ileride sıkıntı çıkma ihtimalini ortadan kaldırır. Yöneticiler için projeyi takip etmesi oldukça kolay bir süreçtir. Yazılım geliştiricisi gözünden bakacak olursak da yazılımın kodlanması ve sınanması daha erken başlar.

**6.Artımsal Geliştirme Süreci Modeli**: Bu model kısaca bir taraftan üretim devam ederken bir taraftan da kullanımın devam etmesi prensibine dayanır. Sistemi geliştirme ve teslim işlemleri parçalara bölünür. Her teslim bir işlemi karşılar. Kullanıcı gereksinimleri öncelendirmeye tabi tutularak öncelikli gereksinimler daha erken teslim edilir. Aynı zamanda öncelikli gereksinimler daha çok test edilir. Bir parçada meydana gelecek olan değişiklikler bir sonraki teslimde ele alınır, anında düzeltme olmaz. Tüm proje birikimli bir şekilde ilerler.

Uzun zaman alabilecek ve sistemin eksik işlevsellikle çalışabileceği türden projeler bu modelin kullanımına uygun olabilir. Bu modelde benimsenen yaklaşım tarzı eldeki projenin tümden batması riskini azaltır.

**7.Kodla ve Düzelt Modeli**: Yazılım geliştirmenin en basit fakat en pahalı yoludur. Birkaç yüz satırdan oluşan programlar için kullanılabilir. Direkt olarak yazılım ürünü gerçekleştirilir. Sistem en son istenilen şekle gelene kadar geliştirmeye devam edilir. Dökümantasyon olmadığından bakımı çok zordur. Emeklilik safhasının en sık görüldüğü modeldir. Genellikle küçük firmalarda kullanılır.

**ÇEVİK YAZILIM GELİŞTİRME**

Büyük projeler, büyük ekipler gerektirir. Büyük bir ekibin birlikte başarılı bir yazılım ürünü ortaya çıkarması demek herkesin kendi bölümünü hatasız yapması artı mükkemmel bir iletişim içinde işi gerçekleştirmiş olmaları demektir. Ayrıca şirket başarılı bir iş yaptığı için sonraki projelerde de bu istikrar ve standart beklenir. Ama işler her zaman böyle yürümez. Yazılım ürünü geliştirirken ortaya birçok sorun çıkar. Yazılımın zamanında tamamlanamaması, eksik çıkması, süreden dolayı yeterli test yapılamaması gibi birçok etmenle karşılaşılabilir. Bir de oluşturulan yazılımda gereksinimleri önceliklendirme işlemi yapılmadıysa veya yanlış yapıldıysa zaman gerçekten sıkıntı olabilir.

Mesela bir bankacılık uygulamasını ele alalım. Bu uygulama üzerinde gerçekleştirilecek olan en temel işlem şunlardır: bakiye sorgulama, para transferi, hesap hareketleri… böyle basit herkesin aklına ilk gelen işlemler bunlardır. Zaten çoğu insan da bu uygulamalarda sadece bu işlemleri yaparlar. Öyleyse koca bir ekibin geliştirdiği onlarca diğer fonksiyon boşuna mıydı? Yazılım projelerinde üretilen özellik ve fonksiyonların yaklaşık yüzde 64’ü hiç kullanılmamaktadır. Durup düşündüğümüz zaman çok doğru, geliştirici ekip tarafından baktığımız zaman ise üzücü bir durum.

Konuya dönmek gerekirse yazılım geliştirme sürecinde ortaya çıkan böyle birçok sorunu çözmek için 1990’lı yılların sonuna doğru Çevik Yazılım Geliştirme Sistemi’nin temelleri atılmıştır. Bu sistemin sahip olması istenilen temel özellikler vardır.

Bu sistem öncelikle hızlı olmayı gerektirir. Hızlı olurken dikkat etmeniz gereken birtakım öncelikler de vardır. Çevik Yazılım Geliştirme yöntemi ile onlar belirlenir ve yöntem proje sürenizi kısaltmanızda, proje maliyetinin azalmasına, iletişimin daha hızlı ve kolay bir şekilde sağlanırken aynı zamanda test kısmını de gerçekleştirmenize olanak sağlar.

Çevik Yazılım Geliştirme’de projenin ölçeğine bakılmaksızın küçük parçalara ayrılır ve her parça kendi başına bir proje gibi düşünülerek geliştirilir. Her parça sonunda da proje ekibi müşteriye, projenin durumu hakkında bilgi verir. Burada her bir parçanın 2-4 hafta kadar sürmesi planlanır. Her parça kendi içinde işlevini yerine getirebildiği için müşteriye teslim edildiğinde önemli bir memnuniyet artışı olduğu görülmüştür.

# Çevik yazılımda proje ekibinde çalışan tüm üyeler sürekli iletişim halindedir. Bir yazılım takımında bilgi alışverişinin en verimli ve etkili yöntemi yüzyüze iletişimdir. Bunun sayesinde her şey hızlı ilerler, sorunlar anında çözülür ve yola devam edilir.

# 2001 yılında Kent Beck ve arkadaşları tarafından çevik manifesto oluşturulmuştur.

***Çevik Yazılım Geliştirme Manifestosu***

*Bizler daha iyi yazılım geliştirme yollarınıuygulayarak ve başkalarının da uygulamasına yardım ederek ortaya çıkartıyoruz.  
Bu çalışmaların sonucunda:*

*Süreçler ve araçlardan ziyade bireyler ve etkileşimlere  
Kapsamlı dökümantasyondan ziyade çalışan yazılıma  
Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine  
Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye değer vermeye kanaat getirdik.*

*Özetle, sol taraftaki maddelerin değerini kabul etmekle birlikte,  
sağ taraftaki maddeleri daha değerli bulmaktayız.*

Projelerin temelinde motive olmuş bireyler yer almalıdır.Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır. Böyle bir ekip tarafından çalışan bir yazılım üretildiği zaman ilerlemek daha basit olacaktır. Basitlik demişken yazılımda basitlik çok önemlidir. Karmaşık ve ilk bakışta anlaşılmayacak öznel kodlar yazmak yerine “clean code” felsefesine uygun kodlar oluşturulmalıdır ki ekipten bir üye diğerinin çalışmasına bakarken ortada neler döndüğünü anlayabilsin.

En yaygın kullanılan çevik metodlar: SCRUM, Agile Unified Process, Feature-Driven Development (FDD), Extreme Programming (XP), Test-Driven Development (TDD), LEAN Development, Dynamic System Development Methodology (DSDM), Microsoft Solution Framework (MSF).

Ben şu an bu metodların hepsini ayrıntılı bir şekilde açıklamak yerine günümüzde en popüler olan metodu yani Scrum’u anlatıcam.

**SCRUM**

Scrum kelimesi ile ilk karşılaşmam üniversitenin ilk döneminde İzmir’de bulunan Atolye15 adlı bir kuruluşun iki haftada bir herkese açık olarak düzenlediği küçük konferanslara gitmem ile oldu. Scrum’ı kendi projelerinde ekip olarak nasıl kullandıklarını anlatan güzel bir sunumdu tabi o zamanlar bu camiaya yeni girmiş biri olarak özellikle kullandığı programlardan pek bir şey anlamadığımı itiraf etmeliyim. Fakat Scrum çerçevesinde kullanılan ana mantık çok basit ve çok mantıklıydı. Öyle ki bu sadece yazılım geliştirme projelerinde kullanılmak zorunda değildi. Aslında belli bir büyüklük derecesini geçmiş her projede bu yöntem kullanılabilirdi.

Yöntem şöyle basit bir mantığa dayanıyor bir proje geldiği zaman Product Backlog yapılıyor. Bu müşteri ile masaya ourup (illa bir masa olması gerekmez sadece müşteri ile iletişim kurmak yeterli sanalda da gerçekleşebilir) gereksinimlerin sıralaması yapılıyor. Sonra sıra sprint backlog’a geliyor. 2-4 haftalık zaman biçiliyor projeye (Atolye15’te bu prensib olarak 2 hafta, yani şirketten şirkete değişiyor) ve birkaç sprintten oluşuyor proje. Sprint demek aslında koşu demek. Bir problemi ele alıp sadece hızlı bir şekilde onu bitirmek şeklinde özetleyebilirim kendimce. Projeler küçük parçalara ayrılıyor. Divide and Conquer (böl ve fethet) yöntemi uygulanıyor.

Burada da günlük hayatımızda kullanabileceğimiz bir olayla karşılaşıyoruz. Bir problem ile karşılaştığımızda yapmamız gereken ilk şey sorunu anlamak sonrasında da onu kolaylıkla halledebileceğimiz küçük parçalara bölmek önem sırasına ayırırak o küçük parçaları tek tek bitirmek ve ta da koca bir problemi Divide and Conqure sistemi ile kolayca çözdünüz. Psikolojik olarak bakacak olursak da insan beyni çok büyük bir problemle karşılaştığı an ondan kaçmak ister, bir türlü işe başlamak istemez, onu gözünde büyütür. Bu sistem ile yapıalcak şeyler bir sorunun küçük parçaları olduğundan parçaları yapmaya daha hevesli oluyoruz. Çünkü beyin bir şeyi bitirdiğinde kendini ödüllendirmek için seratonin hormonu salgılar. Bu hormon sayesinde kendimizi mutlu hissederiz. Büyük bir iş birden bitmeyeceği için uzunca bir süre mutlu olamayız moralimiz düşer.Küçük parçaları ise kısa sürede bitirdiğimizden hem zihnen daha rahat hem de sağlıklı oluruz.

Konudan baya saptık ama günlük yaşamdan bir örnek vermek konuyu anlamak açısından iyi oldu diye düşünüyorum. Konuya dönersek parçalama işlemi sonrası yapılması gereken her bir parça sprint oluyor. Bu sprintler belirlenen sürede koşuluyor. Sprintlerin sayısı problemin büyüklüğüne göre değişiklik gösteriyor. Bu işlemler gerçekleşirken bir yandan ekip üyeleri Scrum Daily Meeting’ler düzenliyor. Bu meeting yani görüşmeler her gün yapılıyor. Öyle ciddi bir toplantı gibi değil de genel olarak ayakta 30 dakika kadar bile sürmeyecek şekilde yapılıyor. Herkes ne yaptığını ve o gün ne yapacağını anlatıyor. Bir sorun varsa birbirlerine söylüyorlar. Birinci ağızdan bundan haberdar alıyorlar. Mükkemmel bir iletişim ağı kurulmuş oluyor. Tabi bu görüşmeler genel olarak böyle şirketlerin bu konuda farklı politikaları olabilir, birtakım prosedürler uygulayabilirler bu toplantılarda. Mesela Atolye15’te bir program aracılığıyla bu Sprint’ler düzene konuluyor planlaması ordan yapılıyordu. Mesela bir uygulamayı kod ile hayata geçirecekler o kodu yazmaktan biri sorumlu, sorumlu olan kişi işini halledip o programa yüklüyor test edecek kişi o programdan alıp kodu test ediyor. İş tamamsa bitmişler kısmına yolluyor tamam değilse aynı program üzerinden kullanıcıya notlarıyla birlikte geri yolluyor böylelikle işler ilgili tüm ekibin erişebileceği bir program üzerinden çok hızlı bir şekilde ileriyor.

Günümüzde neredeyse her firma Scrum yöntemini kullanıyor Çevik Yazılım Geliştirme’de. Çünkü en kullanışlı yöntem bu. Ekip üyeleri arasında mükkemmel bir iletişim oluyor. Bunun sonucunda işler hızlı ilerliyor. İşlerin hızlı ilerlemesi hem müşteriyi hem de ekip üyelerini mutlu ediyor. Çünkü hiçkimse bir işle gereğinden fazla uğraşmak istemez, uzama olan durumalrda ilgileri kayar, verim azalır.

Size elimden geldiğince çok fazla detaya da inmeden, kendi yaşantımdan da kesitler vererek Scrum ve Yazılım Yaşam Döngüsü’nü anlatmaya çalıştım. Anlatmayı başarabildiysem ne mutlu bana. Buraya kadar okuduysanız da sizlere teşekkür ediyor, iyi çalışmalar diliyorum.

Utku Gezensoy

190601058

**YARARLANDIĞIM KAYNAKLAR**

https://blog.burakkutbay.com/cevik-agile-yazilim-gelistirme-metodu-nedir.html/

<https://agilemanifesto.org/iso/tr/manifesto.html>

<https://mmf.ders.bakircay.edu.tr/pluginfile.php/6504/mod_resource/content/1/BİL%20102%20-%201.Hafta%20-%20Yazılım%20Mühendisliği%20Tanımı%20ve%20%20Kapsamı.pptx>

<https://mmf.ders.bakircay.edu.tr/pluginfile.php/6618/mod_resource/content/1/BİL%20102%20-%202.Hafta%20-%20Yazılım%20Yaşam-Döngü%20Modelleri.pptx>

https://mmf.ders.bakircay.edu.tr/pluginfile.php/6768/mod\_resource/content/1/BİL%20102%20-%203.Hafta%20-%20Çevik%20Yazılım%20Geliştirme.pptx

<https://www.meetup.com/tr-TR/atolye15/events/265019198/>