**Yazılım Yaşam Döngüleri ve Kullanım Alanları**

Yazılım yaşam döngüsü; tıpkı insan yaşamındaki döngü gibi belirli bir varlık olan yazılımın geliştirme aşamasından kullanımı son bulana kadar olan tüm zaman dilimlerini kapsar. Bu süreç kaynaktan kaynağa sayısı değişmek üzere belli adımlardan oluşur : Planlama, Analiz, Tasarım, Geliştirme, Entegrasyon ve Testler, Uygulama ve son olarak Bakım aşaması. Bu aşamalar her modelde bulunan aşamalar olup geriye dönülemez bir şekilde doğrusal olarak düşünülmemelidir. Her model bu aşamaları farklı sıra ve esneklikle kullanmaktadır. Kimi modellerde bakımdan sonra emeklilik aşaması da bulunmaktadır. Planlama aşaması yazılım fikrinin ortaya çıktığı, temel gereksinimlerin tartışıldığı ve karar verildiği aşamadır. Analiz aşaması ise planlamada belirlenen gereksinimlerin ayrıntılı olarak belirlendiği, süreçte kullanılacak kavramların tanımlarının yapıldığı ve yazılımın işlevlerinin incelendiği aşamadır. Tasarım aşamasında akış şemaları çizilir, gereksinimleri karşılayacak şekilde modüller belirlenir ve arayüz tasarımları yapılır. Tüm bunlar yapıldıktan sonra geliştirme aşamasında kod yazılır. Neredeyse tüm yazılım yaşam döngü modellerinde bu aşamalar aynı şekilde ilerlerken -geriye dönmeli şekilde de olsa- geliştirme aşamasından sonra gelen entegrasyon ve test aşaması her modelde tek seferde yapılan bir adım olarak karşımıza çıkmamaktadır. Örneğin Spiral yaşam döngüsü modelinde prototip mantığıyla test ve entegrasyon döngünün birçok yerinde yapılmaktadır. Genel olarak bu aşamada yazılımın gerekli donanımlarla ve birimlerle entegrasyonu sağlanır, veri tabanı bağlantıları yapılır ve test edilir. Test edilme test yazılımlarıyla gerçekleştirilebileceği gibi bireyler tarafından da gerçekleştirilebilmektedir. Örneğin ilerleyen bölümlerde bahsedeceğimiz Scrum proje yönetim yaklaşımında Scrum ekibinde testçi kişiler bulunmaktadır. Testler de başarıyla sonuçlandıktan sonra uygulama aşamasında proje hayata geçirilir. Uygulama kullanılmaya başlandıktan sonra doğal olarak bazı sorunlar oluşabilmektedir. Ya da zamanla programdan beklenen gereksinimler değişiklik gösterebilmektedir. Bakım süreçlerinde programa yeni gereksinimler eklenebilir, mevcut hatalar giderilebilir veya kullanıcı memnuniyetini arttırmak için çalışmalar yapılabilir. Yazılım yaşam döngüsü kavramına temel hatlarına değinildiğine göre bazı döngü modellerini örneklerle açıklayabiliriz. **1- Çağlayan Modeli :** Waterfall Modeli olarak da bilinen bu model basit ve yönetiminin kolay olması sebebiyle uzun süreli olmayan projelerde tercih edilebilecek bir modeldir. Ancak projenin planlama ve tanımlama aşamalarının çok iyi yapılmış olması gerekmektedir. Her aşama bir önceki aşamadaki gelişmelerden beslenerek gerçekleştirilir. Barok modelinden farklı olarak bu anlayışta dokümantasyon ve test uygulamaları sürecin her aşamasında vardır. Bu modelin avantajlarından bazılarını saymak gerekirse basit bir model olması sebebiyle sürecin müşteri de dahil olacak şekilde herkes tarafından anlaşılabilir olması verilebilir. Scrum gibi modellerin aksine aşırı profesyonel bir ekip gerektirmez. Aşamaların tekrarlamalı bir şekilde yapılması hataların erken tespitini sağlar. Ancak yine de günümüzde kullanımı giderek azalmakta olan bir modeldir. Bunun sebepleri arasında yukarıda bahsettiğimiz gibi planlama ve analiz aşamasının detaylı ve net bir şekilde olması gerekliliğidir. Profesyonel ekipler olmadığı sürece bu aşamaların eksik yapılması sonucunda bu model iş görmemektedir. Kullanıcı geliştirme sürecinde yer almadığından dolayı geliştirme başladıktan sonra değişen kullanıcı istekleri sürece dahil edilmez. Program teslim edildikten sonra geri dönüşlere göre bakımlarla değişiklikler yapılabilir.

**2- Artımsal Geliştirme Süreç Modeli :** Kullanıcı gereksinimlerinin önem sırasına göre programa işlevsellik kazandırarak ilerleyen bir modeldir. Her bir işlev programa kazandırıldıktan sonra teslim edilir ve sıradaki gereksinim için çalışmaya başlanır. Teslim edilen erken programlar prototip gibi davranarak hataların erkenden tespit edilmesini kolaylaştırabilir. En başta temel gereksinimler programa kazandırıldığı için sonraki her aşamada bu işlevler tekrar tekrar test edilmiş olur. Acil karşılanması gereken işlevlerin ve temel şekilde çalışmasında sorun olmayacak projelerin geliştirilmesinde kullanılabilir. Avantajlarının yanında eksi yanları olarak gereksinimlerin doğru bir şekilde sıralanması önemlidir. Tecrübeli bir ekibe ihtiyaç duyulur. **3-Spiral (Helezonik) Model :** Risk analizinin ön planda olduğu bir modeldir. Prototip üretme mantığıyla ilerler. Böylece testler her aşamada gerçekleştirilmiş olur. Prototipleme mantığı ilk aşamadan itibaren prototipler üretip kullanıcının değerlendirmesine sunulur. Prototip sayısı sürecin işleyişine göre artıp azalabilir. Her prototipten sonra müşterinin istekleri ve memnuniyetleri göz önüne alınarak yeni bir prototip yapılır. Müşterinin tüm istekleri karşılandığında yazılım teslim edilir. Her ne kadar tanım olarak sınırlı sayıdaki tekrar içerdiği yazılsa da dezavantajlarından biri döngünün sonsuza girebileceğidir. Yazılım prototipleri genellikle şu özelliklere sahiptir : -Azaltılmış fonksiyonel özellikler -Azaltılmış kullanıcı arayüzü -Azaltılmış performans Artılarının yanında prototipleme mantığının da eksi yönler bulunmaktadır. En önemli sorunlardan biri bir prototipten asıl uygulamaya geçiş yapmaktır. **Çevik Yazılım Geliştirme :** Çevik yazılım geliştirme belli bir yazılım geliştirme modeli olmamakla birlikte 1990’larda yazılım sektöründeki müşteri isteklerinin hızlı şekilde değişmesi ve bu değişime hızlı yanıtlar verilememesi, bazı kritik hataların ancak süreç sonunda fark edilmesi gibi sorunlar nedeniyle bir bakış açısı olarak ortaya çıkmıştır. Süreci esnek ve daha verimli hale getirmeyi amaçlayan bu yaklaşım müşterilerle sürekli etkileşimi ve onların isteklerini ön planda tutar. Sözleşme kuralları gibi katı kalıplara değer vermez. Katı bir süreç planı yoktur. Geliştirme aşamasında oluşabilecek her türlü probleme uyum sağlayarak sürecin devam etmesini amaçlar. Bu anlayışta ekip üyeleri sürekli olarak iletişim halindedirler. Düzenli olarak süreci değerlendirip daha verimli nasıl devam edileceğine karar verirler. En çok tercih edilen çevik yazılım geliştirme metotları şu şekildedir : 1-Extreme Programing — XP 2-SCRUM 3-Agile Unified Process -AUP 4-Feature-Driven Development — FDD Çevik yazılım geliştirme modelleri günümüzde sıklıkla tercih edilen bir modeldir ancak kurumsal projelerde uygulanması zor bir anlayış olabilmektedir. Takım üyeleri her ne kadar esnek çalışsalar da sürekli değişebilen kullanıcı gereksinimleri üzerlerinde büyük bir iş yükü oluşturmaktadır. Ürünün kalitesi tüm takım üyelerinin kariyerini riske atabilecek de etkiye sahiptir.

**SCRUM:** Scrum sadece yazılım geliştirme süreçlerine uygulanan bir model değil her türlü projede kullanılabilecek bir proje geliştirme metodudur. Büyük bir projeyi küçük parçalara bölerek yapmayı öngören bir metottur. Çevik yazılım geliştirme manifestosuna uyan bir yapısı vardır. Proje genellikle 30 günlük “Sprint” ‘lere bölünerek yapılır. Scrum yapsını anlamak için öncelikle temel scrum ifadelerinin ne anlam ifade ettiğini öğrenmemiz gerekmektedir. **Product Backlog :** Yapılan projenin tüm gereksinimlerinin detaylı bir şekilde ele alınıp listelenmesidir. **Sprint Backlog :** Product Backlog’dan seçilen bir kesim gereksinimlerdir. 30 günlük süre içerisinde bu gereksinimler projeye kazandırılıp teslim edilir. Sprint Backlog’larının hangi sıra ile seçileceği ise müşteri isteği doğrultusunda ya da yazılım ekibinin yapacağı gereksinim sıralamasına göre belirlenir. **Burndown Chart:** Her sprint için başta yapılması planlanan iş ve zamanla ne kadarının yapıldığı, hedefe ne kadar yaklaşıldığının düzenli olarak tutulmasıdır. Yapılan ilerlemenin görülmesi açısından önemlidir**.** Scrumda diğer dillerin aksine aşamalar aynı anda çalışılıyor olabilir. Örneğin gereksinimler hala belirlenirken ilk belirlenen gereksinimlerin tasarım ve kod aşaması başlamış olabilir. Scrum modelinde temel olarak belirli roller, toplantı çeşitleri ve project backlog gibi ortaya çıkan eserler vardır. Bu rollerden biri olan ürün sahibi rolü müşterinin kendisi olabileceği gibi genellikle scrum takımı içinden müşteriyi temsil eden biri olur. Projenin geliştirme sürecince bulunarak müşterinin gözünden gereksinimleri ve süreci gözden geçirir. Ayrıca Scrum takımı da bir rol olarak alınabilir. Rollerden sonuncusu olan Scrum yöneticisi ise adındaki gibi bir yönetici konumundan daha çok ekibin sorunlarını çözme, önlerini açma görevlerini üstlenen takım üyesidir. Scrum takımında takımdakiler birbiri ile eşittir. Scrum modelini daha iyi anlamamıza yardımcı olan ikinci unsur ise Scrum toplantılarıdır. Sprint planlama toplantıları her sprint başında yapılan sprint’in ne kadar süreceği, hangi takım üyesinin ne yapacağı konuşulur. Günlük scrum toplantılarında ise her takım üyesi kısa bir şekilde ne durumda olduğunu herhangi bir sorunu olup olmadığını ifade eder. Sprint gözden geçirme toplantıları ise sprintte ne aşamada olunduğu, hangi gereksinimlerin tamamlandığı gibi şeyler tartışılır. **SCRUM’un Popüler Olmasındaki Temel Sebepler :** - Kullanıcı odaklı bir geliştirme süreci -Esnek çalışma koşulları nedeniyle yüksek motivasyon - Karmaşık problemlerin çözümünü basite indirger, sürece adaptasyon kolaydır -Katı kurallara bağlılık olmadığından her türlü soruna hızlı çözümler bulunabilmesi -Parça parça işin teslim edilmesi sebebiyle müşterinin temel gereksinimlerine hızlı çözüm -Takım üyeleri arasındaki güçlü iletişim sebebiyle yaratıcılık ortamının bulunması Bu özellikleri nedeniyle SCRUM Google gibi birçok sektör devi firmada kullanılmasının yanı sıra 5-10 kişilk start-up firmalarda da kullanılmaktadır. SCRUM birçok proje için uygulanabilir ve çok tercih edilen bir model olsa da her projenin istek ve ihtiyaçları farklıdır. Örneğin çoğunlukla demode olarak adlandırılan Çağlayan Yaşam Döngü Modeli gereksinimlerin proje başında çok net ve değişmeyecek şekilde belirlenebildiği askeri yazılımlar veya sağlık teknolojilerinde SCRUM’dan daha iyi bir tercih olabilmektedir. Bununla birlikte iş dünyasında genel olarak şirketler kendi çalışma tarzlarına uygun olacak şekilde her modelden parçalar alarak kendi modellerini de oluşturabilmektedirler.

**Yazılım Yaşam Döngüsü Modelleri ve Kullanım Alanlarıyla İlgili Bilgiler**

**SCRUM – Extrem Porgramming :** SCRUM VE Extreme Porgramming (XP) çevik yazılım geliştirme felsefesinin önde gelen modelleri olarak sürekli gereksinim değişikliği oluşan veya yeni gereksinimler doğan büyük ve karmaşık projeler için uygundur. Buna bir alışveriş firmasının web uygulaması örnek olarak verilebilir. Böyle bir projede ilk başlarda müşteriye sadece ürünleri ve firmayı tanıtan bir web uygulaması verilebilir. Zaman içerisinde firma siparişlerini site üzerinden alıp ödeme seçeneği olarak sadece kapıda ödeme istediğinde yeni bir gereksinim ortaya çıkacak ve web uygulamaya böyle bir özellik eklenebilecektir. Bu gereksinim de karşılandıktan sonra müşteri kartla ödeme, çift adımlı doğrulama gibi isteklerde de bulunabilecektir. Böyle bir örnekte olduğu gibi müşteri isteklerinin sürekli genişlediği ve değiştiği büyük projelerde bu modellerin kullanımı uygundur.

**Prorotipleme Modelleri :** Prototipleme mantığı bir model olarak ele alındığında, prototip geliştirmeye dayalı modeller büyük bir projeyi yapmak için doğru bir model olmayacaktır. Yukarıda da bahsedildiği üzere prototiplerden asıl uygulamaya geçiş zor bir işlemdir. Projenin tamamında kullanılması yerine kullanıcı arayüzü gibi belirli bölümlerini geliştirmek ve testler yaparak en verimli sonucu almak için kullanılabilir.

**Spiral (Helezonik) Model :** Büyük ve karmaşık projeler için uygundur. Risk analizinin ön planda olduğu bir model olduğu için oluşabilecek hataların kritik sonuçlara sebep olabileceği uygulamaların geliştirilmesinde kullanılabilir. Örnek olarak bir işletim sistemine ait temel yazılımlar bu modelle geliştirilebilir. Birçok teste tabi tutulacağı için hataların tespit edilmesi daha kolay olacaktır.

**Evrimsel Yaşam Döngüsü Modeli :** Büyük ölçekli ve geniş coğrafyalara yayılmış işletmeler için uygun bir modeldir. Geri bildirimin ön planda olduğu, kullanıcı dönüşlerine göre şekillenen bir kullanımı olduğu için birçok alanda kullanılabilir. Bunlara örnek olarak finans sektörü verilebilir. Finans sektöründe kullanıcıya yönelik kısımlarda kullanılması doğru olacaktır. Bankaların altyapıları gibi daha temel yazılımlar farklı modeller kullanılarak ya da şirketin kendi ihtiyaçlarına göre geliştirdiği modellerle ilerlemesi daha doğru olacaktır.