**Yazılım Yaşam Döngüsü Nedir?**

Yazılım yaşam döngüsü, yazılım ürünlerinin belirli bir kalite eşiğinde üretilmesini sağlayan süreçtir. Döngü aşamalardan oluşur ve bu aşamaların bazıları kalıplaşarak standart haline gelmiştir.

**Yazılım Yaşam Döngüsü Aşamaları**

1-) *Gereksinimlerin Toplanması ve Planlama*: Müşteri talepleri alınır ve bu doğrultuda projenin maliyeti gibi konular işlenir.

2-)*Analiz*: Müşteri taleplerinin bir uzman tarafından doğru analiz edilmesi ve proje ekibine iletilmesi kısmını kapsar.

3*-) Sistem Tasarımı*: İlk iki aşamanın sonucunda belirlenen gereksinimler üzerine sistemin tasarlanmasına başlanır. Sonraki adıma geçmeden önce bütün tasarımsal kararlar net bir şekilde alınır.

4-) *Gerçekleştirim:* Kodlamanın olduğu kısımdır. Bu aşamada tasarım kısmında alınan kararlar artık hayata geçirilir ve sistemin ilk örnekleri ortaya çıkmaya başlar. Kodlama belirli bir aşamaya geldikten sonra testleryapılmaya başlanır.

5-) *Bakım:* Ürünü anladık, tasarladık, kodladık ve teslim ettik. Yeter mi? Yetmez. Herkes aldığı üründe garanti olsun ister. Bu aşamada da yazılım ürünümüzün hatalarını giderir, altyapısını güçlendirir, yeni özellikler ekleriz.

**Yazılım Yaşam Döngüsü Modelleri**

Yazılım tarihinin başından günümüze dek birçok farklı yazılım yaşam döngüsü modeli ortaya atılmıştır. Bazıları küçük projelerin, bazıları büyük projelerin, bazıları tek kişilik projelerin, bazıları ekip projelerinin başarı oranını yukarı çekmiştir. Şimdi bu modellerden bazılarına bir göz atalım.

**Şelale Modeli (Waterfall Model):** Bir zamanların en çok kullanılan döngü modelidir. Çoğu modelin temeli olarak gösterilir. Fakat bazı eksiklerinden dolayı günümüzde pek kullanılmamaktadır. Bu modelde her aşamanın üzerinden geçilmelidir ve aşamalarda belgeleme çıkarmak önemlidir, eğer çıkarılmamışsa o aşama tamamlanmamış kabul edilir.

*+Artıları:* Doğru bir gereksinim analiziyle proje gerçekleştirim aşamasını kısa tutabilirsiniz.

*-Eksileri:*Her ne kadar aşamalar arası geri dönüşler yapabilseniz de değişiklikler projede ciddi maliyet ve zaman problemlerine yol açar, proje ekibi zamanının çoğunu analiz ve tasarım aşamalarına ayıracağından çalışanların morali düşer, ne kadar doğru analizle süreyi kısaltabileceğimizden bahsetsek de gerçekte öyle olmadığından teslim tarihi uzar.

**V Modeli (V-Shaped Model):** Aslında bir kısma kadar şelale modeliyle benzerlikler gösterir. Farklı olarak bu modelde her aşamanın bir de test karşılığı bulunmaktadır. Üç ana alt modelden oluşmaktadır.

*Kullanıcı Modeli:* Kullanıcı ile iş birliği içerisinde gereksinimlerin analiz edildiği ve en son isteklerin sınandığı kısmı kapsar.

*Mimari Model:*Tasarım aşamasının yapıldığı ve bu aşamanın test edildiği kısımdır.

*Gerçekleştirim Modeli:*Kodlama kısmının ve hemen sonrasında testlerinin yapıldığı bölümdür.

*+Artıları:*Kullanıcıyı projeye katarak müşteri memnuniyetini arttırır.

*-Eksileri:*Test aşamasında karşılaşılan hatayı geriye dönüp düzeltmek zordur.

**Spiral Model:** Bu modelde aşamaların üzerinden tekrarlarca geçilir, üstüne koya koya (yinelemeli artımsal) bir ilerleme söz konusudur. Diğer modellere nazaran risk analizi ve kullanıcı değerlendirmesi gibi farklı aşamaları vardır. Bu aşamalardan bahsetmek gerekirse;

*Planlama:*Müşteri taleplerinin alınması ve planlanmanın yapılması, ayrıca bu modelde bir döngü söz konusu olduğundan bu aşamada bir önceki adımda oluşturulan ara ürün için yeni kararlar alınır.

*Risk Analizi:* Her planlama sonrası toplanılıp risklerin belirlendiği ve çözüldüğü aşama.

*Üretim:*Bir nevi gerçekleştirim aşaması. Her döngü sonunda bir ara ürün üretilmiş olur.

*Kullanıcı Değerlendirmesi:*Üretim aşamasında ortaya çıkan ara ürünün kullanıcı beğenisine sunulması. Kullanıcı onayı alınmadan sonraki döngüye geçilmez.

*+Artıları:*Olası değişiklikler çok daha rahat yapılabilir, her döngü sonucu bir prototip üretilmesi, risk analiziyle daha sorunlar ortaya çıkmadan çözülebilir, kullanıcı işin içerisinde olduğundan müşteri memnuniyeti yüksektir.

*-Eksileri:* Karmaşık bir modeldir, küçük projeler için maliyeti arttırır, ara aşama sayısı fazla olduğundan belgeleme artar.

**Arttırımlı Model (Incremental Model):** Gereksinimler alınır, parçalara bölünür ve bu parçalar önemlerine göre sıralanır. Her sefer de bir parça tamamlanarak önceki parçalarla birleştirilir ve böylece sona yaklaşılmış olunur. Ortaya çıkan her ara ürün kullanıcı tarafından kullanılabilecek seviyede olur. Aynı anda hem geliştirilir hem de kullanılır. Olası değişiklikler geliştirme başlandıktan sonra yapılmaz, bir sonraki döngüde uygulanır.

*+Artıları:* Olası değişikliklerin maliyeti azdır, sonuçlar erken zamanda elde edilir, test sonucu ortaya çıkan hataları ayıklamak kolaydır, parçaları zordan kolaya sıralayarak riski düşürür, her prototip müşterinin ürünü değerlendirmesini kolaylaştırır.

-*Eksileri:* Yönetimi zordur, bazı gereksinimler sonradan alındığından tasarımsal sıkıntılar baş gösterebilir.

**Kodla ve Düzelt Modeli:** Gereksinimler alınır, kodlamaya başlanır ve teslim edilir. Müşteri memnun olmadıysa geriye dönülür ve düzenlenir. Müşteri memnuniyeti sağlanana dek döngüye devam edilir. Diğer modellerden farklıca emeklilik aşaması da vardır.

*+Artıları:* Küçük projeler için uygun bir modeldir.

*-Eksileri:* Değişiklerin maliyeti yüksektir, belgeleme yoktur.

**Çevik Modeller:** Çevik geliştirme modelleri, süreçleri adlandırmak ve belgeleme yapmaya teşvik etmek yerine direkt yazılıma odaklanan ve arttırımsal geliştirmeye dayanan modellerdir. Bu modellerde gereksinimler parçalara bölünür ve her parça bir proje olarak kabul edilir. Belgeleme yerine ekip içerisinde iletişim çok daha önemli bir yere sahiptir. Müşteri projenin başından sonuna dek sistemin içerisinde olduğundan memnuniyet oranı yüksektir. Olabildiğince çok, çalışan yazılım teslim edilir. Ayrıca günümüzde çevik model tercihi oldukça yaygındır.

*+Artıları:* Olası değişiklikler kolaylıkla yazılım içine entegre edilebilir, belgeleme kolaydır, yönetilmesi kolay bir modeldir, hata ayıklaması kolaydır, ekip motivasyonu daima yüksektir.

*-Eksileri:* Belgelemenin az olmasından kaynaklı projeye yeni katılan birinin projeyi anlaması zorlaşabilir.

Bu bilgilerden sonra şimdi bazı çevik model metodolojilerine göz atalım;

**Uç Programlama (Extreme Programming – XP):** Kent Beck tarafından 1999 yılında ortaya atılan bu modelde bazı değerler öne çıkmaktadır. Bu değerler;

*1-) İletişim:* Çevik modellerin temeli olan iletişimi burada da görürüz. Uç programlama iletişimi bir değer olarak kabul eder ve iletişimsizlikten ortaya çıkan bazı problemleri ortadan kaldırmayı amaçlar. Bu iletişim altyapısı yalnızca ekip içerisinde değil, aynı zamanda kullanıcılarla da kurulmalıdır. Ayrıca farklı rol ve deneyimlerde bulunan ekip üyelerinin bir arada çalışmasına da yardımcı olur.

*2-) Cesaret:* Başarısızlığın önemli olmadığını, önemli olanın en kısa zamanda yeniden ayağa kalkabilmeyi savunan temeldir. Beğenilmeyen bir durum karşısında korkmadan tüm kod yeniden yazılmaya başlanır. Kısa vadede zaman kaybı gibi gözükse de uzun vadede büyük zamanlar kazandırır.

*3-) Basitlik:* Komplike bir sistemin tasarlanmasındansa o an ne istendiyse basitçe uygulanmasıdır. İstenilen değişikliklere göre üzerine ekleyerek bir geliştirme söz konusudur.

*4-) Geri Bildirim:* Sistemin doğruluğu kontrolünde müşteri geri bildirimi çok önemlidir. Kısa aralıklı teslimler sayesinde müşterinin istekleri daha hızlı cevap bulur, müşteri memnuniyeti artar.

Uç programlama modelinde bazı tekniklerden söz edebiliriz. Bunlar:

*Planlama Oyunu (Planning Game):* Müşterinin ve proje ekibinin bulunduğu bir ortamda daha öncesinde alınan istekler önceliklerine göre sıralanır. Her istek için bir zaman tahmini ortaya atılır.

*İkili Programlama (Pair Programming):* Projede aynı kodlama iki kişi tarafından yapılır. Hata oranı düşer, kalite yükselir. Ayrıca iki programcıda kısa zamanda aynı seviyeye gelir.

*Sade Tasarım (Simple Design):* Her proje parçasını olabildiğince sade üretmeyi amaçlar. Böylece ilerde değişiklikler çok daha kolay bir şekilde sisteme dahil edilebilir.

*Ekipte Müşteri (On-site Costumer):* Müşteri hep işin içinde tutularak olası problemler çok daha rahat çözülür.

*Test (Testing):* Her program parçasının test edilmesi gerekir. Daha sonra programlar birleştirilirken tekrar test aşamaları oluşturulur.

*Sürekli Entegrasyon (Continuous Integration):* Olası değişiklikler ve yeni komponentler anında sisteme entegre edilir. Bu sayede olası hataların erken tespit edilip ortadan kaldırılmasını kolaylaştırır.

*Kısa Aralıklı Sürümler (Short Releases):* Projenin gelişimi 2-4 haftalık aralıklarla müşteri beğenisine sunulur. Böylece hem sürekli çalışan bir sistem elde edilir hem de müşteri memnuniyeti ölçülebilir.

*Yeniden Yapılandırma (Refactoring)):* Bazı hatalar ilerde çok büyük sorunlara yol açabilir. Buna sebebiyet vermemek amacıyla sistem devamlı olarak gözden geçirilir.

*Ortak Sorumluluk (Collective Ownership):* Tüm proje için ekip çalışanları eşit sorumluluk alır. Bir ekip üyesi başka bir üyenin yazdığı kodu değiştirebilir. Böylece herhangi biri işe gelmediğinde işler aksamaz.

*Benzetim (Metaphor):* Proje parçasının işlevinde ekibin anlamasını kolaylaştıracak şekilde bazı benzetmeler kullanılır. Böylece sistemin anlaşılması kolaylaşır.

*Kodlama Standartları (Coding Standarts):* Proje parçalara bölündüğünden birleştirme aşamalarında sıkıntı çıkmaması açısından ekip içerisinde bazı kurallar belirlenir. Ayrıca bu şekilde bir kodun tüm ekip tarafından anlaşılırlığı arttırılmış olur.

*Haftada 40 Saat:* Günümüzde maalesef fazla mesainin verimi arttırdığı düşünülüyor. Fakat gerçekte hem verim azalır hem de hata yapma oranı da yükselir. Bu teknik bunu önleyerek çalışanlardan maksimum verimin alınmasını sağlar. Hem yöneticinin hem çalışanın mutlu olduğu bir çalışma şeklidir.

*+Artıları:* Ürün küçük parçalar halinde geliştirildiğinden test edilmesi ve hata ayıklaması kolaydır dolayısıyla ortaya çıkan ürünün sağlamlığı yüksektir ayrıca riski de düşürür*,* her döngü başında alınan geri bildirimlerle değişikliklerin uygulanması esnekliği arttırır ve maliyeti düşürür, yöneticinin, çalışanın, müşterinin mutlu olduğu genel bir memnuniyet ortamı oluşturur.

*-Eksileri:* Her müşteri işin içinde bu kadar olmaktan hoşlanmayabilir, sonradan bir üyenin projeye dahil edilmesi zordur.

**Scrum:** Avantajlarıyla büyük firmaların gözdesi olan bu modele sadece bir yazılım modeli demektense “proje yönetim biçimi” demek daha doğru olur. Kısaca anlatmak gerekirse; bir toplantı eşliğinde müşteriden gereksinimler alınır ve bir ölçüye göre bu gereksinimler sıralanır. Bir çevik model klasiği olarak proje parçalara bölünür (sprint) ve oluşturulan takımlara dağıtılır. Parçalar oluşturulurken her gün küçük toplantılarla iletişim sağlanmış olur. Takımlar her ay oluşturdukları parçaları birleştirerek çalışan bir yazılımı müşteriye teslim eder ve en sonunda ürün ortaya çıkmış olur. Daha detaya inmek için bazı temel kavramlara bakmamız gerekmektedir:

1-) Roller:

*1.1 Scrum Uzmanı (Scrum Master):* Takımın lider özelliklere sahip üyesidir.Ekip üyelerinin motivasyonunu yüksek tutarak verimliliği arttırır. Scrum modelinin uygulandığından emin olur, sorunları çözer, iş takibi yapar.

*1.2 Ürün Sahibi (Product Owner):* Bir nevi pazarlamacı da denebilir. Ürünün değerini yükseltmeye çalışır, ürün teslim tarihini belirler, gereksinim önceliklerini değiştirebilir.

*1.3 Geliştirme Ekibi (Development Team):* Ürünün geliştiriminden sorumlu üyelerdir. Sayıları 5-9 arası değişebilir.

2-) Toplantılar:

*2.1 Günlük Toplantı (Daily Scrum):* Ekip üyelerinin her gün yaptığı, 15 dakikayla sınırlı, ilerleyişi görmek amaçlı toplantılardır.

*2.2 Sprint Planlama Toplantısı (Sprint Planning):* Gereksinimlerin alındığı, belirli bir ölçüye göre sıralanıp takımlara dağıtıldığı toplantıdır.

*2.3 Sprint Gözden Geçirme Toplantısı (Sprint Review):* Sprint gereksinim listesinin oluşturulduğu toplantıdır.

3-) Bileşenler:

3.1 *Ürün Gereksinim Dokümanı (Product Backlog):* Yapılacaklar listesi de denebilir. Listede zamanla değişiklikler olabilir.

*3.2 Sprint Dokümanı (Sprint Backlog):* Her bir takıma aldıkları sprinti açıklar. Sadece takım üyeleri değişiklik yapabilir.

*3.3 Sprint Kalan Zaman Grafiği (Burn-Down Chart):* Sprint için tanımlanan görevlerin gözlenmesini sağlar. Günlük olarak güncellenir.

**Hangi Modeli Kullanalım?**

Bütün bu açıklamalardan sonra bu başlığın altında “bu şartlar için hangisi daha uygun” sorusuna bir cevap arayalım.

**V Modeli:** Gerekliliklerin çok iyi anlatıldığı ve anlaşıldığı küçük projeler için uygun bir modeldir.

**Arttırımlı Model:** Büyük ve kritik önem taşıyan projeler için kullanılabilir.

**Spiral Model:** Risk ihtimalinin yüksek olduğu büyük projeler için uygundur.

**Kodla ve Düzelt Modeli:** Ekip çalışanının az olduğu küçük projelerde denenebilir, yine de önerilmez.

**Çevik Modeller:** Küçük ve orta çaplı projeler için yönetilmesi kolay bir ortam sağlar, büyük firmaların gözdesidir.

**Yararlanılan Kaynaklar:**

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

<http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yaz%C4%B1l%C4%B1m-Geli%C5%9Ftirme-Modelleri-Yaz%C4%B1l%C4%B1m-Ya%C5%9Fam-D%C3%B6ng%C3%BCs%C3%BCSDLCYBS.pdf>

<https://medium.com/@denizkilinc/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-temel-a%C5%9Famalar%C4%B1-software-development-life-cycle-core-processes-197a4b503696>

<https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742>

<https://medium.com/@ahmetuyar/extreme-programming-xp-nedir-ddc003a515c>

Doç. Dr. Deniz Kılınç – Yazılım Mühendisliği Temelleri Ders Notları Bölüm 2-3.