Yazılım Yaşam Döngüsü Nedir?

Yazılım yaşam döngüsü, adından da anlaşılacağı gibi bir yazılımın en başından en sonuna kadar olan süreci ifade eder. Bu süreç 5 aşamada gerçekleşir. İlk olarak Gereksinim (Requirement Phase) aşaması gelir. Bu aşamada yazılımın amacı ve kullanıcı ihtiyaçları belirlenir. İkinci olarak Analiz (Analysis Phase) aşaması gelir. Bu aşamada elde edilen amaç ve ihtiyaçlar analiz edilir. Ve bu amaç ile ihtiyaçların doğruluğundan emin olunur. Ayrıca, Yazılım Proje Yönetim Planı da bu aşamada hazırlanır. Üçüncü olarak Tasarım (Design Phase) aşaması gelir. Burada ”İstediğimizi nasıl elde ederiz” sorusuna cevap aranır. Bu aşamada gerekli teknolojiler, araçlar ve kaynaklar belirlenir. Tasarım aşamasında 2 adet tasarım yapılır. İlk olarak Üst Seviye ve Mimari Tasarım (Architectural Design) vardır. Bu tasarım genel olarak kaba taslak planları ve modülleri kapsar. İkinci olarak Ayrıntılı Tasarım (Detailed Design) gelir. Bu tasarım ise veri tabanı tasarımları, kullanılacak algoritmalar, sınıf diyagramları, aktivite diyagramları ve etkileşim diyagramlarını kapsar. Dördüncü olarak Gerçekleştirme (Implementation Phase) aşaması gelir. Bu aşamada; daha önceki aşamaların çıktıları kullanılarak, Programın gerçek kodu oluşturulur. Ayrıca bu aşamada her bir modül birbirinden bağımsız olarak test edilir. Daha sonra yazılımın bütün olarak testi için tümü birleştirilir. Modül birleştirme sırasında sorun ve yanlışlıklar çıkabilir. Bu gibi durumlar test edilir. Bu aşama, yazılım geliştirme sürecindeki en önemli aşamalardan biridir. Yazılımın başarıya ulaşması için bu aşamanın çok dikkatli ve doğru yönetilmesi gerekir. Beşinci olarak Teslim-sonrası (Post-delivery Maintenance Phase) aşaması gelir. Bu aşama yazılımın teslim edildikten sonra gerçekleşir. Bu aşama, yazılımın kullanımı sırasında ortaya çıkabilecek hataların tespit edilmesi, bu hataların giderilmesi ve yazılımın güncellenmesi gibi önemli çalışmaları kapsar. Teslim sonrası bakımın iki çeşidi vardır. İlki Düzeltici Bakım (Corrective Maintenance). Bu bakım türü, kullanım sırasında ortaya çıkan hataları düzeltmek için yapılan bakımdır. Düzeltici bakım türünde; hatalar tanımlanır, raporlanır ve düzeltilir. İkinci bakım türü ise Özelliklerin Arttırılması ( Enhancement) türüdür. Burada da Mükemmelleştirici Bakım ( Perfective Maintenance) ve Uyarlanabilir Bakım(Adaptive Maintenance) bulunur. Mükemmelleştirici bakım; yazılımın performansını ve işlevselliğini arttırmak için yapılan bakımdır. Bu bakım türü, yazılımın daha iyi hale getirilmesi için yapılan düzenlemeleri kapsar. Ayrıca bu bakımda mevcut özellikler geliştirilip yeni özellikler eklenebilir. Yazılımın kalitesi arttırılır. Uyarlanabilir Bakım ise yazılımın çevresinde meydana gelen değişikliklere uyum sağlamak için yapılan bakımdır. Bu değişiklikler; donanım, işletim sistemi, veri tabanı veya diğer yazılımlar gibi dış etkenlerden dolayı olabilir. Genel olarak uyumsuzluk sorunlarına çözüm getirmek için yapılan bakımdır. Teslim sonrası Bakım aşaması yazılım için ciddi bir önem taşır. Her zaman bir bakım söz konusudur. Çünkü Yazılım, sürekli değişen bir gerçekliğin modelidir yani yaşamaya devam eder. İyi bir yazılımın bakımdan geçmesi doğaldır. Altıncı ve son aşama ise Emeklilik (Retirement Phase) Aşamasıdır. Bu aşamada vadesini dolduran yazılımlar emekliye ayrılır, yani yazılımın kullanımı durdurulur. Yazılım yaşam döngüsü bu aşamalardan oluşur. Ve her bir adım ayrı bir önem taşır. Ve bu aşamalarda hataların giderilmesinin maliyeti her geçen adım daha da artar. Bu yüzden hataların erkenden fark edilip bir çözüm getirilmesi çok önemlidir. Genel olarak en maliyetli aşama ise bakım aşamasıdır. Ayrıca bakım aşamasında fark edilen bir hatanın giderilme maliyeti de bir o kadar yüksek olacaktır. Günümüzde kullanılan birçok Yazılım yaşam döngü modeli bulunmaktadır. Bunlardan 4 tanesi şu şekildedir.

Barok Modeli, 1970’li yıllarda kullanılan bir modeldir. Diğer modellere kıyasla biraz daha ilkel kalmaktadır ve günümüzde kullanılmamaktadır. Günümüzde kullanılmamasının temel nedenlerinden biri; bu modelde, “Belgeleme” ayrı bir süreç gibi işlenir. Ayrıca bu model, “Gerçekleştirme” adımına çok ciddi önem veren bir modeldir. Bu modelde Yaşam döngüsünün temel adımları doğrusal bir biçimde geliştirilir. Ve bu adımlar arasında gereken geri dönüşlerin nasıl yapılacağı tanımlı değildir.

Çağlayan / Şelale Modeli, diğer Yaşam Döngü modellerinin temelidir ve geçmişte en popüler döngü modelidir. Bu modelde Yaşam döngüsünün temel adımları en baştan en sona kadar en az birer kez tekrarlanarak gerçekleştirilir. Çağlayan modeli, iyi tanımlanmış ve kısa sürede bitebilecek projeler için uygun bir modeldir. Her adımda belgeleme işlemi yapılır. Barok modelinde ayrı bir süreç olan “Belgeleme” sürecini; Çağlayan model, her aşamada yapar ve eğer bir safhada belgeleme ve test olmamışsa, o aşamanın tamamlandığı kabul edilemez. Ayrıca bir sıradaki adım önceki adım tamamlanmadan başlamaz ve gerektiğinde geliştirme aktivitelerinde tekrarlamalar olabilir. Ve en önemlisi analizdeki tüm detayın tasarıma yansıtılabilmesi için müşteri ve sistem gereksinimleri ayrıntıları olarak belirlenmelidir. Yani tasarım aşaması detaylı bir çalışma gerektirmektedir. Bu da uzun bir zaman gerektirdiğinden gereksinimlerin değişecek olması kaçınılmazdır. Avantajlarından söz edecek olursak. Belgelendirme işleminin her adımda yapması avantajlarından biridir. Ayrıca Gereksinim adımı tamamlandıktan sonra çok sağlam bir temel oluşturur. Bu da gereksinimleri iyi anlaşılabilen projeler için avantaj sağlar. Bunların yanında Proje yöneticileri için işin dağılımını yapması kolaydır. Müşteriler ve son kullanıcılar tarafından anlaşılır adımlardan oluşur. Dezavantajlarından bahsedecek olursak. Bir adım tamamlanmadan diğerine geçilememesi riski arttırır. Bunun haricinde genelde yazılımın son kullanıcıya ulaşma zamanı uzundur. Ve sistem geliştirme süresince gereksinimler süreden kaynaklı sıklıkla değişir. Bu durumda gereksinimlerin iyi anlaşıldığı durumlarda kullanılması daha doğru olacaktır. Örneğin: Askeri ürün üretimi, inşaat sektörü… Bir diğer dezavantajı ise Gereksinim tanımlamaları çoğu kez net bir şekilde yapılmadığı için yanlışların düzeltilme ve eksiklerin giderilme maliyetleri yüksektir. Ve son olarak kullanıcı, sürecin içerisinde yer almaz, yazılım tamamlandıktan sonra geri dönüş verebilir.

V Süreç Modeli, adından da anlaşılacağı gibi “V” şeklinde bir yol izler ve sol tarafta üretim, sağ tarafta test işlemlerini barındırır. Bu model şelale modelinin gelişmiş hali gibi düşünülebilir. Bu model daha az belirsizlik barındıran ve iş tanımlarının daha belirgin olduğu projeler için kullanılır. Bu tür projelere örnek olarak Bilgi Teknolojileri için geliştirilen projeler örnek verilebilir. Bununla birlikte bir Bilişim projesinin iki aşamalı olarak sunulması için oldukça uygundur. İlk aşamada kullanıcı modeli hedeflenerek, iş analizi ve kabul sınamalarının tanımları yapılmakta. İkinci kısımda ise elde edilmiş olan kullanıcı modeli tasarlanıp gerçekleşmektedir. Ayrıca bu model, kullanıcının projeye katkısını arttırmaktadır. V süreç modelinin temel çıktıları üç tanedir: Kullanıcı Modeli, Mimari Model, Gerçekleştirim modeli. Kullanıcı modeli, yazılımın kullanıcılarına yönelik gereksinimlerin belirlendiği ve kullanıcıların beklentilerinin göz önünde bulundurulduğu aşamadır. Bu model “V” şeklinin solunda bulunur ve sağ taraf için bir ön hazırlık niteliğindedir. Mimari Model, yazılımın sistem düzeyinde tasarım yapıldığı aşamadır. Bu aşamada, yazılımın parçalara ayrılması ve bu parçaların nasıl bir arada çalışacağı belirlenir. Bu belirlemeler; yazılımın güvenilirlik, sürdürülebilirlik ve performans artışı amaçlanarak yapılır. Bu model de aynı şekilde “V” ‘nin solunda kalır ve sağ taraf için temel oluşturur. Gerçekleştirim modele gelirsek; Bu model, yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonları içerir. Bu aşama diğer aşamalarla koordineli bir şekilde çalışır ve yazılımın kalitesini artırmaya yardımcı olur. Ve bu aşama yaşam döngüsü içinde oldukça önemli bir aşamadır. V Süreç Modelinin avantajlarından bahsedecek olursak; ilk olarak bu model, kullanıcının projeye katkısını arttırmaktadır. Proje yönetiminin takip etmesi oldukça kolaydır. Ve ürünler üretilirken sınanır. Ek olarak sınama müşteriye sunularak onaylatılır. Ayrıca Verification ve validation planları erken aşamalarda vurgulanır ve tüm teslim edilebilir ürünlerde uygulanır. Dezavantajları ise şunlardır. İlk olarak fazlar arası tekrarlama ve Risk çözümleme aktiviteleri yoktur. Fazlar arası tekrarın olmaması bize, bir önceki aşamada yapılan hataların tekrarlanmaması için bir fırsatın olmaması anlamına gelir. Bu durum hataların daha ileriki aşamalarda bulunmasına sebep olup, projenin teslimini geciktirip maliyetlerin artmasına yol açabilir. Bunların haricinde geliştirme devam ettikçe iş ve ürün gereksinimleri değişkenlik gösterebilir.

Spiral (Helezonik) Model, Amerikalı matematikçi ve yazılım mühendisliği profesörü Barry Boehm tarafından tanımlandı. Boehm bu modeli, Çağlayan modeline olası bir alternatif olarak açıkladı. Ancak bu model diğer modellerden farklı olarak Yinelemeli artımsal ve prototip yaklaşımlarına dayanır ve projenin geliştirilmesi sırasında birçok kez geri dönülmesine olanak tanır. Bunlarla birlikte bu modelde Risk analizi olgusu ön plana çıkmıştır. Ve buna ek olarak her bir spiralin riskleri ayrı ayrı ele alınır. Süreci oluşturan spiralin her dönüşü, modelin dört aşamasını temsil eden dört parçanın her zaman geçtiği tam bir döngüyü temsil eder. Bu dört parça şu şekildedir. İlk olarak planlama gelir. Planlama aşamasında amaca, alternatiflere ve sınırlamalara karar verilir. Bunlarla birlikte performans, sistem gereksinimleri, donanım ve yazılım arayüzleri gibi önemli konular ayrıntılı olarak tanımlanır. Planlama aşamasından sonra Risk değerlendirmesi aşamasına geçilir. Bu aşamada alternatifleri değerlendirme ve risk analizleri yapılır. Bu aşamada söz konusu tüm alternatifler değerlendirilir. Risklerin olasılıkları ve etkileri incelenir. Risk içeren faktörler (Deneyimsizlik, değişen teknolojiler, sıkışık programlar vb.) tespit edilip daha düşük riskli stratejiler izlenir. Son olarak; genel olarak prototipleme yöntemi kullanılır. Üçüncü olarak Üretim aşaması gelmektedir. Bu aşama yazılımın tasarımı ve kodlanması aşamasıdır. Üretim aşaması; önceki aşamalardan elde edilen bilgi ve verilerin kullanımıyla, yazılımın gereksinimlerini karşılamak için planlanan çözümlerin kodlanması sürecidir. Ayrıca bir sonraki ürünü onaylama da bu aşamada yapılır. Ek olarak içerisinde Modül testi, Birleştirme testi ve Kabul testi gibi süreçler de barındırır. Son aşama olan Kullanıcı Değerlendirme aşaması ise, bir sonraki döngünün planlanması ve kullanıcı değerlendirmesinden oluşur. Ve bu aşamada, bir önceki döngüde yapılan işlerin değerlendirmesi yapılır. Hatalar ve geliştirme önerileri kaydedilir. Avantajlarından bahsedersek. İlk olarak bu modelde, kullanıcılar sistemi erken görebilirler. Bu özellik sayesinde; projenin bir sonraki versiyonu, farklı geri bildirimler sayesinde daha iyileştirilebilir. Bir diğer avantajlarından biri; Riske duyarlı yaklaşımı sayesinde potansiyel zorlukları engeller. Geliştirmeyi parçalara böler ve en riskli kısımları önce gerçekleştirilir. Bu modeldeki risk analizinin bu denli önemli olması diğer süreç modellerine göre en büyük avantajlarından biridir. Yazılımın kodlanmasının ve sınanmasının daha erken başlaması da bu modeli avantajlı hale getirir. Ayrıca bu model, pek çok yazılım modelini içerisinde bulundurmasıyla oldukça avantajlıdır. Dezavantajları ise şöyledir. İlk olarak bu model oldukça karmaşıktır ve spiralin sonsuza gitme olasılığı vardır. Bu modelde her biri farklı aşamalardan geçen birçok döngü vardır ve bu nedenle dokümantasyon süreci işleri zorlaştırır. Kontrat tabanlı yazılıma uymaması da bu modeli dezavantajlı duruma düşürür. Son olarak Öznel risk değerlendirme deneyimine dayanır. Öznel risk değerlendirme deneyimi; kişisel yargılar, önyargılar, tahmin ve öngörüler içerebilir. Bu da bizim riskleri objektif bir şekilde değerlendirmemizde zorluklar yaratabilir ve Daha güvenilir sistematik bir yaklaşıma kıyasla daha az doğru olabilir.

Yazılım yaşam döngüsü; daha önce de bahsettiğimiz gibi, yazılımın planlanması, geliştirilmesi, test edilmesi, uygulaması ve bakımı gibi farklı aşamalardan oluşan döngüleri ifade eder. Bir de Yazılım yaşam döngüsünü tamamlayan, Çevik Yazılım Geliştirme dediğimiz bir yaklaşım bulunmaktadır. Bu yaklaşım, geleneksel yazılım geliştirme metotlarına kıyasla daha esnek ve çevik bir yaklaşımı benimser. Ayrıca bu yöntemde müşteri ile sürekli bir iş birliği ve iletişim hakimdir. Bu iş birliği ve iletişim sayesinde müşteri ihtiyaçlarına hızlı bir şekilde dönüş sağlanır. Bununla birlikte proje küçük parçalara ayrılır ve sık sık test edilir. Buradan gelen geri bildirimler, hızlı bir şekilde programın iyileştirilmesine ve geriye dönük hataların düzeltilmesine kolaylık sağlar. Çevik Yazılım Geliştirmede basitlik çok önemlidir. Bu da adaptasyonu hızlı hale getirir. Ancak bazı dezavantajları da bulunmaktadır. Esnek yapısı yüzünden sürekli değişen ihtiyaçlar dolayısı ile aşırı çalışma gerektirir. Ürünün başarısı, projenin başarısına eş tutulduğu için kariyer riskini de barındırabilir. Bununla birlikte Çevik Yazılım Geliştirme; müşteri odaklı bir yaklaşım olduğu için, takım üzerinde bir hedef baskısı olabilir. Ancak, bu baskının yönetilemeyecek boyutlara ulaşması durumunda kurumsal yapıya zarar verme riski bulunabilir. Çevik Yazılımın en yaygın ve en çok kullanılan metodolojilerinden biri olan SCRUM şu şekildedir. SCRUM, Çevik yazılım geliştirme metodolojileriyle uygulanabilecek bir proje yönetim yaklaşımıdır. Yani sadece yazılım geliştirmeye değil, her projeye uygulanabilir. SCRUM, karmaşık ortamlarda aşama aşama yazılım geliştiren ekipler için uygundur. Bu karmaşıklığı Şeffaflık, Denetleme ve Uyarlama denilen üç ilke ile azaltmaya çalışır. Ayrıca Scrum, sprint adı verilen kısa çalışma dönemleri ile çalışır. Bu sprintler, takım üyelerinin iş birliği ve ekip çalışması ile gerçekleştirilir ve genellikle 2-4 haftalık periyotlar halinde olur. Her sprint öncesi, ekip üyeleri planlama için toplantı düzenler ve yapılacak işleri belirler. Bunlarla birlikte bir de her gün kısa toplantılar yapılır. Bu toplantılarda takım üyeleri birbirlerine yapacakları işler hakkında bilgi verir. Sprint sonunda, yazılımın işlevsel bir kısmı müşteriye sunulur ve alınan geri bildirimlerle yeniden planlamaya gidilir. SCRUM, bazı temel roller barındırır. Bunlardan ilki Ürün sahibi (Product Owner) rolüdür. Ürün sahibi, projenin iş değeri açısından geri dönüşü ile sorumludur. Müşteri tarafından görevlendirilmiştir. İkinci olarak Scrum yöneticisi (Scrum Master) gelir. Scrum yöneticisi, Takımın scrum’ın temel değerlerine, pratiklerine ve kurallarına bağlı kalmasını garanti altına alır. Ayrıca takımı ve organizasyonu Scrum’a adapte eder. Üçüncü olarak Scrum takımı (Scrum Team) gelir. Scrum takımı, devamlı iletişim halinde olan ve tek bir hedefe ulaşmak için mücadele eden kişilerden oluşur. Genelde 5 – 9 arasında kişiden oluşur. Scrumda proje yöneticisi rolü bulunmaz. Bunun sebeplerinden ilki takımın özerkliğini korumaktır. Bir diğer sebep ise deneyimsiz bir Scrum ekibinde sorun oluşturup hedeften saptırma ihtimalidir. SCRUM, 3 adet yan rol barındırır. Bunlar; işletme, müşteri ve kullanıcıdır. Bunların yanında Takım da kendi içinde üç ayrı yetenek barındırır. Sırasıyla bunlar yazılımcı, testçi ve tasarımcıdır. SCRUM, ayrıca kendi içinde bazı Bileşenler/Araçlar (Artifacts) barındırır. İlk olarak Ürün Gereksinim Dokümanı (Product Backlog) gelir. Bu kısaca proje boyunca yapılması gereken iş elemanlarının basit bir listesidir. Devamlı bakım istenir ve genellikle kullanıcı hikayelerinden oluşur ve kullanıcı bakış açısından bakılır. İkinci sırada Sprint Dokümanı (Sprint Backlog) gelir. Bu da kısaca mevcut sprint için ürün gereksinim dokümanından elde edilmiş iş ve görevleri kapsar. Buradaki işlerin amacı, sprint sonunda son ürünün bir parçası olan bir işlevselliği ya da çalışabilir bir parçayı elde etmektir. Son olarak Sprint Kalan Zaman Grafiği (Burndown Chart) gelir. Bu grafik, Sprint boyunca işlerin ne kadarının yapıldığı ile normalde ne kadar yapılması gerektiğini karşılaştırılabilmesini sağlar. Özetle SCRUM, Çevik Süreç Değişimini baz alır ve onunla yaşamayı kolaylaştırmak için yeni yazılım metotları sunar. Kısa vadeli planlar ve küçük parçalar halinde yazılımın geliştirilmesini ön görür.

Ulaş Uçan

220601076

<https://medium.com/@uls.ucn/yazılım-yaşam-döngüleri-ve-çevik-yazılım-geliştirmede-scrum-6bf162fbad65>