**YAZILIM YAŞAM DÖNGÜSÜ**

Bir yazılım geliştirirken takip edilmesi gereken bir yol vardır. Bu yol değişebilse de adımlar genel olarak aynıdır. Bütün bu adımlar ve bu adımlardan oluşan döngüye yazılım geliştirme yaşam döngüsü adı verilir. Planlama, analiz, tasarım, gerçekleştirim, dağıtım ve bakım yazılım geliştirme yaşam döngüsünün aşamalarıdır.

**1)Planlama**

Yazılım yaşam döngüsünün ilk adımı planlamadır. Yazılım ürününün özellikleri, kullanıcılar belirlenir. Fizibilite çalışmaları yapılır. Proje planlanır.

**2)Analiz**

Gereksinim analizi kullanıcının isteklerinin anlaşıldığı önemli bir adımdır. Yazılım geliştirme yaşam döngüsünün başlarında olmalıdır. Kullanıcının istekleri başta iyi anlaşılmazsa ilerleyen zamanlarda tekrar başa dönmek gerekebilir ve bu da zaman ve maliyet konusunda sorunlar yaşanmasına sebep olabilir. Bu yüzden kullanıcı gereksinimleri doğru analiz edilmeli ve kullanıcılar ile sürekli etkileşim halinde olunmalıdır. Bu aşamada gereksinimler belgelenir.

**3)Tasarım**

Bu aşamada yazılımın modeli oluşturulur. Tasarım dokümanı oluşturulur. Tasarım dokümanında proje bilgileri, sistem tasarım bilgileri, tasarım detayları, arayüz tasarımları vardır. UML şeması gibi yöntemler kullanılabilir.

**4)Gerçekleştirim**

Kodlamanın yapıldığı aşamadır. Veri tabanı ve kullanıcı arayüzleri oluşturulur. Raporlar yazılır. Test yapılır. Yazılımın önceden belirlendiği özelliklere uygun olup olmadığına bakılır. Performans testi, yük testi gibi testler yapılır ve yazılımın doğru çalışıp çalışmadığına bakılır.

**5)Dağıtım**

Yazılım hedef ortamda kurulur. Çalışır hale getirilir. Kullanıcılara yazılımla ilgili eğitim verilir.

**6)Bakım**

Yazılımda sorunlar varsa düzeltilir. Yazılım ürününü daha iyi hale getirmek için değişiklikler yapılır.

**YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ**

Bu adımların sırası değişerek yazılım yaşam döngüsü modelleri oluşur. Birçok yazılım yaşam döngüsü modeli vardır. Bu modellerin dezavantajları ve avantajları vardır.

**1)Barok Modeli**

1970’li yıllarda ortaya çıkmıştır. Günümüzde kullanılmamaktadır.

Yazılım yaşam döngüsü adımları doğrusal şekilde izlenir. Döngü sırasında hata olursa veya bir şey değiştirmek istenilirse geri dönüş yapmak, adımlar doğrusal şekilde izlendiği için zor olur. Bu yüzden maliyet çok artabilir. Günümüzdeki diğer modellerden farklı olarak bu modelde dokümantasyon diğer aşamalar bittikten sonra yapılır.

**2)Şelale Modeli**

Bu model en eski, en temel ve en tanınmış modeldir. Şelale modeli geçmişte en popüler yazılım geliştirme modeli olarak görülmüştür. Geleneksel bir yazılım geliştirme modelidir. Bu modelde işler aşama aşama yapılır. Bir aşama bitmeden diğerine geçilmez. Aşamalar arasındaki geri dönüşlerin nasıl olacağı tanımlıdır. İleride fark edilebilecek hataları düzeltmek veya değişiklik yapmak zordur ve maliyetin artmasına sebep olur. Her aşamada dokümantasyon yapılmalıdır. Bu, projenin ilerleyişinin daha doğru bir şekilde izlenmesini sağlarken aynı zamanda daha fazla zaman harcanmasına neden olur. Bir aşamadaki dokümantasyon yapılmamışsa o aşama tamamlanmış kabul edilmez.

Bu modelde en fazla zaman, analiz ve tasarım aşamasında harcanır. Detayları doğru bir şekilde tasarıma yansıtabilmek için analiz aşamasında müşteri ve sistem gereksinimleri en ince ayrıntısına kadar belirlenmelidir. Tasarım aşamasında ise tüm bu gereksinimleri karşılayacak şekilde detaylı bir çalışma yapılmalıdır.

Genelde yazılımın tamamlanıp kullanıcıya ulaşması uzun zaman alır. Gereksinim tanımlamaları çoğu zaman net bir şekilde yapılamadığından oluşabilecek sorunları düzeltmek maliyetin yükselmesine neden olur.

Kullanıcı yazılım geliştirme aşamalarında sürecin içinde genellikle yer almadığı için geri dönüşleri yazılım tamamlandıktan sonra yapar. Yazılım tamamlandıktan sonra bir şeyleri değiştirmek daha zor ve tamamlanmadan önce değiştirilenlere göre çok daha fazla maliyetlidir.

**3)V Süreç Modeli**

V süreç modeli, şelale modelinden farklı olarak üretim ve testin ne zaman yapılacağını vurgular. “V” şeklinde bir yol izlenir. V modelinde sol taraf üretim sağ taraf test işlemleridir. V süreç modeli, bir yandan ürün geliştirilirken diğer yandan bu geliştirilen kısmın hemen test edildiği bir yazılım yaşam döngüsü modelidir.

Test kısmında bir hata bulunduğunda nereye dönülmesi gerektiği belirtilmiştir. Sağ tarafta yapılan testler sonucunda bir hata bulunursa onun solunda kalan aşamaya dönülecektir. V süreç modelinde müşteri, proje yapılırken aktif olarak projede bulunmaktadır.

V modeli belirsizliklerin az olduğu ve iş tanımlarının belirgin olduğu bilgi teknolojileri projeleri için uygun bir modeldir. V modelin kullanımı ürün tanımı değişiklik göstermeyen, kısa süreli olan projeler için daha uygundur. Karmaşık projeler için iyi bir model değildir. Nesne yönelimli projeler için iyi bir model değildir. Devam eden projeler için zayıftır.

**V Süreç Modelinin Çıktıları**

V süreç modeli 3 alt modelden oluşur. Bunlar:

1-Kullanıcı modeli: Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanır. Kullanıcının istekleri doğrultusunda sınama belirtimleri ve yazılım ürününün planı çıkarılır.

2-Mimari model: Yazılım ürününde kullanılacak ve test edilecek mimari işlevler mimari modelde tasarlanır ve belirlenir.

3-Gerçekleştirim modeli: Bu modelde, tasarlanan mimari işlevler kullanılarak kodlama yapılır. Ürün test edilir ve burada ortaya çıkmış olur.

**4)Helezonik (Spiral) Model**

Helezonik model 1988 yılında Barry Boehm tarafından tanımlanmıştır. Şelale modelinden farklı olarak düz bir şekilde değil de yinelemeli bir şekilde akışı vardır. Risk analizi ön plandadır. Her döngü öncesi içinde bulunulan fazın risk analizi yapılır ve o faz için planlanan prototip geliştirilir. Her döngüde yazılım ürünü bir önceki döngüdeki sonuçlardan yararlanılarak geliştirilir.

Helezonik model küçük projeler için uygun değildir. Büyük projelerde kullanılabilen bir modeldir. Döngülerin tekrarlanması şeklinde oluştuğu için gereksinimlerin belirlenmesi daha kolaydır. Geliştirilerek gidildiği için projenin sonu başlangıçta belli olmayabilir. Karmaşıktır. Ara aşamalar çok olacağından aşırı dokümantasyon gerekebilir.

Helezonik model 4 ana bölümden oluşur. Bu bölümler:

1-Planlama: Üretilecek olan ara ürün için işin planlanması, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme yapılmasından oluşur.

2-Risk Analizi: Bu bölümde risk seçenekleri araştırılır ve riskler belirlenir.

3-Üretim: Ara ürün üretilir.

4-Kullanıcı Değerlendirmesi: Üretilen ara ürün hakkında kullanıcılar tarafından yapılan testleri ve değerlendirmeleri içerir.

**5) Artımsal Geliştirme Süreç Modeli**

Bu modelde üretilen her ürün sürümü birbirini içerecek şekilde giderek artan sayıda işlev içerecek şekilde geliştirilmektedir. Öncelikle bu yazılım ürünü için çekirdek bir kısım geliştirilir. Daha sonra bu çekirdeğe işlevler eklenerek yeni sürümler elde edilir. Artımsal geliştirme süreç modelinde üretilen her yazılım sürümü birbirini kapsar ve işlevselliği giderek artar. Yinelemeli bir modeldir. Artımsal geliştirme süreç modelinde bir yandan üretim yapılırken diğer yandan kullanım yapılmaktadır. Her döngü sonunda kullanılabilir bir ürün ortaya çıkmış olur.

Gereksinimler önemlerine ve birbirlerine bağımlılıklarına göre sıralanır. Her yinelemede bunların bir kısmı tamamlanır.

Bir parça geliştirilmeye başlandığında gereksinimleri dondurulur, o döngünün içinde değiştirilmez. Yapılacak değişiklikler sonraki teslimlere ele alınır.

Ürünlerde değişikliğe ihtiyaç varsa bu model kullanılabilir.

**7) Kodla ve Düzelt Yaşam Döngü Modeli**

Yazılım ürününün ilk sürümü geliştirilerek başlanır. Direkt olarak yazılım ürünü gerçekleştirilir. Yazılım en son istenen şekle gelene kadar devamlı geliştirilir. Dokümantasyon yapılmaz. Bu nedenle bakım aşaması zor olur. Kodları düzeltmek maliyetin artmasına sebep olabilir. Herhangi bir plana ihtiyacı yoktur. Kontrollü değildir. Hataların bulunması belirli bir plan ve testler olmadığı için zor olur. Kodlar sonradan değiştirilmek üzere planlanmadığından dolayı değiştirilmesi zor olacaktır.

Küçük projelerde kullanılmaya uygun bir yazılım yaşam döngüsü modelidir. Herkes bu modeli kullanabilir.

**8)Agile (Çevik) Model**

Çevik modelleme, yazılım sistemlerini etkili ve verimli bir şekilde modellemeye ve dokümantasyonunu yapmaya yönelik pratiğe dayalı yöntemlere verilen genel addır.

Süreç ve belgeleme yerine yazılıma yoğunlaşır. Dokümanlar çok olmadığı için yeni ekip üyelerinin projeye dahil edilmesi zor olur. Yinelemeli bir modeldir. Yazılım düzenli olarak müşteriye sunulur. Müşterinin geri dönüşleriyle değiştirmeler yapılabilir.

Müşteriden gelebilecek tahmin edilemez talep değişikliklerinin olabileceği durumlarda bu model kullanılır. Çevik modelin amaçları değişen isteklere hızlı bir şekilde yanıt verme, yazılım ürününü en kısa sürede müşteriye sunmaktır.

Çevik model verimliliği yüksek, hata oranı düşük, hızlı, ucuz ve esnektir.

Küçük ve orta çaplı projeler için uygun bir modeldir.

Proje küçük parçalardan oluşur. Bu yüzden hataların düzeltilmesi daha kolaydır. Yazılım ekibinin sürekli etkileşim halinde olması gereken bir modeldir. Düzenli aralıklarla ekip kendi yöntemlerini gözden geçirerek verimliliği artırmak için gerekli iyileştirmeleri yapar.

**SCRUM**

Scrum, 1990’ların ortalarında Jeff Sutherland ve Ken Schawaber tarafından geliştirilen bir çevik yazılım geliştirme metodolojileriyle uygulanabilecek bir proje yönetim yaklaşımıdır. Yani sadece yazılım değil her yere uygulanabilir.

Scrum, tekrar eden ve aşamalı bir çevik yazılım geliştirme yöntemidir. Karmaşık olan yazılım işleri küçük birimlere (sprint) bölünerek yazılım geliştirilir. Günlük toplantılarla iş takibi yapılır. Bir yinelemenin tamamlanması 30 günden fazla sürmez.

Bu yöntem müşterilerin değişen isteklerine hızlı cevap verebilir.

Karmaşık ve gereksinimlerin kolay tanımlanamadığı projeler için en uygun yöntemdir.

**Sprint**

Belirli bir işin tamamlandığı ve incelemeye hazır hale geldiği yaklaşık 2-4 hafta uzunluğundaki bir dönemdir.

**Scrum Rolleri**

Scrum rolleri ürün sahibi, scrum yöneticisi ve scrum takımından oluşur.

Ürün Sahibi (Product Owner): Çalışma sonunda ortaya çıkacak ürünün nasıl olacağına, bu ürünün özelliklerine ve ürünün nasıl görüneceğine karar veren ürün sahibidir. Ürünün teslim tarihi ile ilgili kararlar verir. Sprintlerin önceliği ve özellikleri ile ilgili değişiklikler yapabilir.

Scrum Yöneticisi (Scrum Master): Ekiptekileri bir arada tutan, ekibin verimliliğini ve motivasyonunu artırmaya çalışarak ekipteki ilişkileri organize eden kişidir. Lider özelliklerine sahip olmalıdır. Sorunların üstesinden gelir. İşleri takip eder. Ekiptekilerin scrum toplantılarına, sprint planlama ve sprint inceleme toplantılarına katılmasını scrum yöneticisi sağlar.

Scrum Takımı (Scrum Team): Sprintteki işleri tamamlayabilecek özelliklere sahip kişilerdir. Ekipteki herkes her şeyi yapabilir. 5-9 kişiden oluşur.

**Toplantılar**

Sprint Planlama (Sprint Planning): Product backlog ile belirtilen gereksinimler, sprint planlama ile geliştirme takımı tarafından küçük görevlere ayrılır. Geniş kapsamlı gereksinim listesi çıkarılır. Sprintin amacı ve iş listesi belirlenir. Her geliştirme üyesi ve ürün sahibinin bu toplantıya katılması gerekir.

Sprint Gözden Geçirme (Sprint Review): Her sprintin sonunda yapılan, sprintin gözden geçirildiği, ortaya çıkan ürünün değerlendirildiği toplantılardır.

Günlük Scrum Toplantısı (Daily Scrum Meeting): Her iş gününde belirlenen saatte gerçekleştirilen ve tüm ekibin katıldığı 15 dakikalık toplantılardır. Takımın ilerleyişine bakılır. Karşılaşılan engeller görülür. Üyeler bir önceki gün ne yaptıklarını, o gün ne yapacaklarını ve karşılaştıkları sorunları söyler. Bir sorun varsa scrum master bu sorunu ortadan kaldırır.

**Bileşenler/Araçlar (Artifacts)**

Ürün Gereksinim Dokümanı (Product Backlog): Proje boyunca yapılacak olan işlerin listesidir. Ürün gereksinim dokümanının geçerli ve kullanışlı olması için devamlı bakım gerekir. Yeni elemanlar eklenebilir veya daha küçük parçalara ayrıştırılabilir. Bazı gereksinimler gereksiz ise silinebilirler. Sıralamalar değişebilir.

Sprint Dokümanı (Sprint Backlog): Sprint dokümanı günlük scrum toplantılarında ilerlemenin anlaşılabilmesi için yeterli ayrıntıyı içeren bir plandır. Geliştirme ekibi sprint boyunca sprint dokümanını değiştirir. Gerekli olan işler daha iyi anlaşıldıkça sprint dokümanı belirginlik kazanır. Yeni iş gerektikçe sprint dokümanına eklenir. Geliştirme ekibine aittir.

KAYNAKÇA

<https://fikirjeneratoru.com/>

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

<https://www.argenova.com.tr/scrum-nedir>

<https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8>