**1.Giriş**

Her ürün gibi yazılımlar da geliştirilirken belli aşamalardan geçmektedir. Yazılım geliştirmekte sadece kodlamadan ibaret olmayıp geliştirilirken belli aşamalardan geçmektedir. Geliştirilen yazılımın geçtiği bu aşamalara “yazılım geliştirme yaşam-döngüsü (software development life-cycle (SDLC))” denir.

Yazlılm geliştirme yaşam-döngüsü metodları kullanılarak geliştirilen yazılımın daha başaralı olması, maliyetlerin düşürülmesi, müşterinin ihtiyaçlarının daha iyi karşılanması gibi birçok avantajı vardır. Ayrıca bir ürün olan yazılımların da belli bir ömrü vardır ve yazılım geliştirme yaşam-döngüsü içerisinde bu ömrün tamamlanması ve ya yazılım geliştirilmesi gibi aşamalar vardır. Bu sayede bir yaılımın ömrünün uzamasının ve ihtiyaçları karşılamasının devamlılığı sağlanır. Burada da yzılım ürünlerinin diğerlerinden bir fark olarak, diğerlerinde bakım aşaması onarım olarak yapılırken, yazılımlarda düzeltme ve ya güncelleme içerir.

**2.Yazılım Geliştirme Yaşam Döngüsü Temel Adımları**

**2.1.Gereksinim (Requirements):** Müşterinin gereksinimlerini anlamak ve ürünün neleri yapabilmesi gerektiğinin sorusuna yanıt aranmaktadır. Bu işlem birkaç şekilde olabilir. Müşteriyle ve yazılmı kullancak kişilerle görüşmek, iş sahasını araştırmak , yazılı kaynaklardan faydalanmak…

**2.2.Analiz (Analysis):** Toplanılan gereksinimlerin analiz edildiği safhadır. Burada dikkat edilmesi gereken müşterinin ne istediği değil neye ihtiyaç duyduğudur. Çünkü müşteri bir yazılımcı bakış açısıyla bakamayabilir ve ihtiyaçlarını tam olarak anlatamayabilir. Burada önemli olan gereksinim safhasında ihtiyaçların doğru olarak toplanmasına bağlıdır.

**2.3.Tasarım (Design):** Toplanılan gereksinimlere yanıt verecek sistemin temel yapısının hazırlandığı safhadır. İlk olarak mimari tasarım (architectural design) yapılır ve kabataslak olarak geliştirilecek sistemin mimarisini gösteren planlama yapılır. Daha sonra ayrıntılı tasarım (detailed design) yapılarak kullanılacak algoritmlar , veri tabanı mimarisi, sınıflar ve sınıflar arasındaki ilişkiler planlanır.

**2.4.Gerçekleştirme (Implementation):** Bu aşama kodlamanın (coding) ve test (testing) işlemlerinin yapıldığı aşamadır. İlk olarak modüller ayrı ayrı test edilir (unit testing), daha sonra bu modüllerin bir bütün olarak çalışması gerektiğinden entegrasyon testleri (integration testing) yapılır ve daha sonra müşteriye teslim edilecek şekilde ürün testi (product testing) yapılır. Son olarak da müşterinin kabul testi (acceptance testing) yapabilmesi için hazırlanan yazılım sistemi müşteri için kurulumu yapılır. Burada müşteri geçek verileri kullanarak ürünü test eder.

**2.5.Bakım (Maintenance):** Yazılım ürünün müşteriye tesliminden sonra ürün üzerinde yapılan herhangi bir değişikliğe denir. Bu değişikler hata giderilmesi ve ya yazılım ürününde değişiklikler yapılarak geliştirme şeklinde olabilir. Bakım safhasının iki türü vardır. Bunlar düzeltici bakım (corrective maintenance) ve özelliklerin artırılmasıdır (enahancement) . Düzeltici bakım geliştirilen yazılım ürününde ortaya çıkan bir hatanın giderilmesidir. Özellkiklerin artırılması da iki şekilde olur . İlk olarak yazılım ürünün yararlılığını artırmak için yapılan değişiklikleri kapsayan mükemmelleştirici bakım ( perfective maintenance), diğeri ise çalışma ortamlarında oluşan değişikliklere karşı yapılan uyarlanabilir bakım (adaptive maintenance) dır.

Yazılım ürününün geliştirilmesinde bu beş adım bir döngü içerisinde devam eder. Bu yazılım geliştirilmesi ve bakım safhalarını içeren dööngüye yazılım geliştirme yaşam-döngüsü (software development life-cycle) denir. Bu aşamalaradan geçen yazılım ürünü daha başarılı olacaktır. Ayrıca bu safhalaratan geçen üründe oluşabilecek hatalar daha erken farkedeilir ve giderilmesinde maaliyeler düşer ve bu şekilde başarılı yazılım gerekliliklerinden biri olan maliyetlerin aşılmamaıda gerçekleşmiş olur.

3.**Yazılım Geliştirme Yaşam-Döngü Modelleri**

* Gelişi Güzel Model
* Barok Modeli
* Çağlayan Yaşam-Döngü Modeli
* V Süreç Modeli
* Helezonik (Spiral) Model
* Artımsal Geliştirme Yaşam-Döngü Modeli
* Kodla ve Düzelt Yaşam-Döngü Modeli
* Çevik Yazılım Geliştirme Metodları (Agile Development)

**3.1.Gelişi Güzel Model**

Belirli bir yöntem veya model bulunmadığı için buna model demek tam olarak oğru sayılmaz. Genelde kiişye bağlı yazılım geliştirmede kullanılır. İzlenebilirliği ve bakımı zordur. Tek kişininyaptığı kolay programlamada kullanılır.

**3.2.Barok Model**

Yazılım geliştirme yaşam-döngüsü adımlarını bir döngü olarak değil doğrusal olarak kullanan bir modeldir. Günümüzde geçerliliği kalmamıştır. Gereken geri dönüşlerin nasıl yapılacağı tam olarak belli değildir. Dökümantaston kodlamadan sonra yapılır.

**3.3.Çağlayan Yaşam-Döngü Modeli (Waterfall Life-Cycle Model)**

Çağlayan yaşam döngü modeli (waterfal life-cycle model) en bilinen, en eski, ve en temel yaşam döngü modelidir.

Çağlayan yaşam-döngü modelinde, aşamalar en bir kez tekrar edilerek yazılım ürünü geliştirilir. Gereksinim, analiz, tasrım, gerçekleştirme, bakım aşamalrı sırasıyla yapılır ve bir aşama bitmeden diğer aşamayaa geçilmez. Her aşama sonrasında dökümantasyon hazırlanır. Yapılan her işin bir dökümanı olması gerekir. Eğer herhangi bir aşama sonrasında doküman hazırlanmadıysa o safha bimiş sayılmaz. Barok modelden farklı olarakdökümantasyonu ayrı bir süreç olarak değil, üretimin doğal bir parçası olarak kabul eder. Ayrıca çağlayan yaşam-döngü modeli emeklilik safhasıyla biter.

Kullanımı ve anlaması kolay olan çağlayan yaşam-döngü modelin yönetimi de basittir. Gereksinimleri iyi anlaşılmış projelerde iyi çalışır.

Karmaşık projelerde iyi çalışmaz.kullanıcı süreç içerisnde yer almaz ve bu durum gereksinimlerin doğru anlaşılmasını önler ve yazılımın tesliminden sonra geri dönüşleri çoğaltır. Bu geri dönüşler maliyeti büyük oranda artırır.

**3.4.V Süreç Modeli**

Adından da anlaşılacağı gibi “V” harfi yapısında bir yol izleyerek adımları gerçekleştirir. “V” harfinin bir kolu üretim işlemini , diğer kolu sınama işlemini ifade eder. Üretim ve sınama etkinlikleri paralel olarak devam eder.

V süreç modelindeki işlemleri üçbaşlık altında toplayabiliriz. Bunlar “Kullanıcı Modeli”, “Mimari Model”, “Gerçeklerştirim Modeli”dir.Kullanıcı modelinde kulanıcı ile olan ilişkiler tanımlanmaktadır. Ayrıca sistemin kullanıcı tarafından nasıl kabul edileceği ilgili planlar ortaya çıkmaktadır. Mimari modelde sistem tasarımı ve tüm sistemin sınma işlemleri dikkate alınmaktadır. Gerçekleştirim modelde ise yazılım modüllerinin kodlanması ve sınanması ile ilgili işlevler yer almaktadır.

V süreç modelinde kullanıcı yazılı projesi içierisinde yer almaktadır. Üretim ve sınamaplanları erken aşamalarda uygulanmaktadır. Ayrıca sadece son ürünlerde değil tüm teslim edilebilir ürünlerde uygulanabilir. Proje ekibi tarafından takibi kolaydır. Kullanıımı da kolaydır.

**3.5.Helezonik (Spiral) Model**

Helezonik (spiral) model risk odaklı bir yazılım geliştirme sürecidir. Bu süreç değişikliklerin proje risklerinden biri olduğunu öne sürer. Projenin her aşaması belirli süreçlerden geçer. Her döngüi çin planlama, risk analizi, üretim, kullanıcı değerlendirmesi yapılır.

* **Planlama :**Üretilecek prototip için planlama,amaç belirleme ve bir önceki adımda üretilen prtotiple entegrasyonu işlemlerigerçekleşir.
* **Risk analizi:** Risklerin belirlenmesini sağlar.
* **Üretim:** Prptotip oluşturulması ve geliştirilmesi sağlanır. Her aşamada üretilen prtotipler geliştirilerek son ürün ortaya çıkarılır.
* **Kullanıcı değerlendirmesi:** Üretilen prtotip ile ilgili kullanıcı tarafından sınamave değerlendirme yapılır.

Spiral modelde risk analizi öne çıkmıştır. Her döngü bir faz olarak kabul edilir. Doğrudan analiz, gereksinim tasarım gibi safhalar yoktur. Yinelemeli artımsal bir yaklaşım vardır. Prototip yaklaşım vardır.

Kullanıcılar bu modelde sistemi erken görebilir. Bu model geliştirmeyi küçük paröalra böler, en riskli kısımları önce gerçekleştirir. Riske duyarlı bir yaklaşım vardır. Hataları erken çözmeyi amaçlar.

Spiral model küçük projeler için pahaı bir metoddur. Karmaşık olabilir ve spiral sonsuza gidebiilir.adımların fazlalığı nedeniyle çok fazla dökümantasyon gerekebilir.

**3.6.Artımsal Geliştirme Yaşam-Döngü Modeli**

Bir yazılım ürünü geliştirilirken müşterinin değişen ihtiyaçlarından dolayı farklılaşan gereksinimleri karşılayan ve ona ayak uyduran bir yaklaşımdır. Bu modelde ürün tek seferde teslim eilmez ve geliştirme ve teslim parçalara bölünerek ilerlenir. Artımsal modelde bir takvime bağlı kalınarak yazılım ürünükısım kısım geliştirilip teslim edilir. Bir parçanın geliştirilmeye başlanıldığı zaaman gereksinimler beklilir ve olası değişiklikler sonraki teslimlerde ele alınır.

Her artırım sonrası müşteriye çalışan bir ürün teslim edilir. Ürün teslimatıyla müşteriye bir sistem verilir ve geri dönüşler alınır. Böylece bu erken teslimler bir sonraki teslimler için ara ürün görevi görür ve geliştirmeye yardımcı olur.

Sitem gereksinimleri müşteri ile belirlenir. Öneme göre teslim edilecek artırımlar belirlenir. En önemli gereksinime göre çekirdek sistem geliştirilir. Erken teslimler prtotoip olarak kullanılır ve gereksinimlerin daha iyi anaşılması sağlanır. Projenin başarısız olma ihtimali ni azaltır. Daha fazla sınınma olur. Böl ve yönet mantığını kullanır.

Artımları tanımlamak için tüm sistemin tanımlanmasına ihtiyaç duyulur. Gereksinimleri doğru boyuttaki artımlara atamak zaman zaman zorluk çıkarabilir. Deneyimli personel gerektirir. Artımların kendi içlerinde tekrarlamalara izin vermez.

**3.7.Kodla ve Düzelt Yaşam-Döngü Modeli (Code and Fix)**

Genelde resmi olmayan ürünler için kullanılır ve ürün hazır olana kadar kodlama yapılarak devam eder. Bu model öncesinde bir planlamaya ihtiyaç duymaz. Küçük çaplı pojelerde kullanılır. Kısa ömürlü prtotiplere uygun bir modeldir. Uzmanlık gerekmez herkes bu modeli kullanabilir.

Kodla ve düzelt modeli kontrollü değildir. Herhangi bir kaynak planlaması yoktur. Herhangi bir takvime bağlı değildir. Büyük projeler için uygunsuzdur. Hataları bulunması çok zordur. Maliyetler çok yükese olabilir. Müşteri ihtyaçlarını karşılayamayabilir. Kodların esnekliği yoktur sonradan değiştirlmeye imkan tanımaz.

**3.8.Çevik Yazılım Geliştime Metodları**

Kent BECK ve çevik yazılım geliştirme temsilcisi olan 16 arkadaşı tarafından 2001 yılında bir manifesto yayınlandı. Bu manifestoda ;

* Süreçler ve araçlar yerine **Bireyler ve Etkileşimle**r: Ekip içi ve aynı zamanda müşteri ile olan etkileşimlerin önemini vurgular. Başarılı projelerin yüksek motivasyonlu ekpiler tarafından yapıldığını söyler.
* Kapsamlı belgeler yerine **Çalışan Yazılım:** Müşteri çalışan yazılımdan daha iyi bir kontrol sağlar.
* Sözleşme görüşmeleri yerine **Müşteri İlişkileri :** Müşteriyi de süreç içine alır.
* Plan izleme yerine **Değişikliğe Açıklık :** Değişikli vardır vebunu iyi yönetmek gerekir.

Çevik yazılım presipleri içierisinde ilk olarak müşteriye hızlı bir şekilde sürekli olarak bir ürün teslimatı vardır. Esneklik vardır ve gereksinimdeki değişiklikler kabul edilir.tüm ekip elemanları sürekli iletişim halindedir. İlerlemenin en önemli ölçüsü çalışan bir yazılımdır. En zor olanı basitlikdir. Bu böl ve yönet düşüncesiyele kolayca aşılır.

Çevik yazılımda yazılım ekibinin motivasyonu sürekli yüksek tutulur. Bu durum kaliteyi de artırır. Kısa sürede müşteri memun edilir ve maliyet düşer.

**3.8.1.Extreme Programming (XP)**

Kent BECK tarafından ortaya atılan, grup içi iletişime önem veren ve geri dönüşlerin fazla olmasına imkan sağlayan bir çevik geliştirmee metodudur.

XP’nin dört temel değeri vardır.

1. **İletişim:** İlk temel taşı iletişimdir. Bazen ekip içerisindeki iletişimsizlik yazılım geliştirme sürecini olumsuz etkiler. Bunu aşılmasının tek yolu da ekip içerisindeki iletişim çok iyi olmasıyla çözülür. Ayrıca ekip ile yazılımı kulancaklar arasında da iyi bir iletişim ağı olması gerekir.
2. **Basitlik:** Basitliğin sağlanması zor bir konudur. Basitlik zorunlu işlerin yapılmasıdır. Zaman ve maliyeti göz önünde bulunduran senek bir yapı oluştutrulması gerekir. O anki ihtiyaçalrı karşılayan en basit yöntemler kullanılır.günün ihtiyaçlarını karşılayan esnek ve basit sistem gerçekleştirilmeye çaılşılır.
3. **Geri Bildirim:** geri bildirimler sayesinde ortaya çıkacak hatalar erkenden çözüme kavuşturulur. Bu sayede ekip ve müşteri arasında daha sonraları çıkacak anlaaşmazlıklar önceden çöülmüş olur.
4. **Cesaret:** En zor konu budur. Projenin geliştirilmesinde yılmadan çalışılması gerekir. Ortaya çıkan bir sorunda vazgeçmek yerine üstüne gidilmesi gerekir.

XP 12 farklı pratiği ön görür.

1. **Planlama oyunu**: Zaman planlaması ve hangi işlerin öncelikli olduğunu belirtmek için yapılır.
2. **Ekipte müşteri:** Yazılımcınınihtiyaç duydupu bilgilere daha hızlı elde eder.
3. **Önce test:** Hataların önceden belirlenmesi için önce bir test yazılımı yazılır.
4. **Basit tasarım:** Anlaşılması, değiştirilebilmesi ve yönetilmesi kolay bir proje olur.
5. **Çiftli programlama:** Usta-çırak ilişkisi
6. **Sürekli entegrasyon:** Sistem üzerindeki değişikliklerin takibi ve hataların erken belirlenmesi.
7. **Kısa aralıklı sürümler:** Müşteri yazılımın gelişimini daha kolay takip eder.
8. **Yeniden yapılandırma:** Kod ve tasrım sürekli gözden geçirilir.
9. **Ortak kod sahiplenme:** Geliştirilen yazılım tüm ekibin ortak malıdır.
10. **Benzetim:** Geliştirilen yazılımda sistemler birbirine benzetilerek yazılım geliştirilmeye çalışılır.
11. **Kodlama standartı:** Karmaşıklık azaltılarak tüm ekip tarafından yazılım anlaşılabilir hale geir.
12. **Haftada kırk saat: Ç**alışma verimiliğini artırır.

**3.8.2.Scrum**

Scrum, yinelemeli ve artırımlı bir çevik yazılım geliştirme metodudur. Bu model karmaşık yazılımprojelerinin yönetimi için uygundur. Karmaşık yazılım işleri küçük parçalara (sprint) bölünerek geliştirmeyi ön görür.karmaşık ortamlarda adım adım yazılım geliştirilir.

Müşteriden gereksinimler öğrenilerek bir tablo oluşturlur. Gereksinimlerin onem sırası belirlenir ve ayrı ayrı sprintlere bölünür. Belii periyotlar halinde bu sprinler koşulur vebelirtilen gereksinimi karşılayan bir prototip müşteriye sunulur.

Müşteri sunulan prtotipi inceler gereksinimler tekra belirlenir ve yine önem sırasına göre sıralma yapılır. Bu süreç müşterinin ihtiyaçları tam olarak karşılanana kdar devam eder.

Scrum’da üç temel kavram vardır.

1. **Roller:**
2. Ürün sahibi, geri dönüşten sorumludur. Müşteri tarafından belirlenir. Detayları takip eder.
3. Scrum yöneticisi, takım ve organizasyonu scruma adapte eder. Takımı dış etkilerden korur.
4. Scrum takımı, devamlı iletişim halinde olan kendi kendine organize eden ve bir bütün olrak hareket eden ekiptir. Geliştirlen prototibi belli aralıklarla müşteriye sunar.
5. **Toplantılar**
6. Koşu planlama, sprint başlamdan planlanlamaların yapıldığıgeniş kapsamlı toplantıdır.
7. Koşu gözden geçirme,
8. Günlük scrum toplantısı, günlük yapılan , “bugün ne yaptım?”, “yarın ne yapacağım?”gibi soruların sorulduğu toplantıdır.
9. **Bileşenler/Araçlar**
10. Ürün gereksinim dökümanı, gereksinimlerin belirlendiği dökkümandır.
11. Koşu dökümanı, mevcut ürün gereksinim dökümanında n elde edilmiş iş ve görevleri kapsar.
12. Sprint kalan zaman grafiği, bu grafik sprint içerisinde işlerin ne kadar zamanda yapılması gerektiğini ve ne kadar zamanda yapıldığnı karşılaştırmak için kullanılır.

**4.Metodların Karşılaştırılması**

**Gereksinşm belirleme:** Kodla ve düzelt, çağlayan modeli, V modeli metodlarında başlangıçta belirlenir. Spiral model, artımsal geliştirme ve çevik metodlarda belirli sıklıklarla belirlenir.

**Maliyet:**  Artımsal geliştirme ve kodla düzelt metodlarında düşük, çağlayan modeli, V modeli, spiral model ve çevik metodlarda yüksektir.

**Başarı garantisi:** Çağlayan modeli ve kodla düzelt metodlarında düşük, artımsal geliştime, V modeli, spiral model ve çevik metodlarda yüksektir.

**Basitlik :** V modeli, artımsal geliştirme, Çağlayan modeli ve kodla düzelt metodları basit, diğerleri daha karmaşıktır.

**Esneklik:** Kodla ve düzelt ile çağlayan modelleri katı, diğerleri çok esnektir.

**Uygulama :** Spiral model karmaşık diğerleri kolaydır.

**5.Metodlar hangi projelerde kullanılır?**

Şelale Modeli çok küçük ve gereksinimleri çok iyi anlaşılmış projelerde iyi çalışır. V modeli bilişim teknolojileri (BT) projeleri için uygundur. Spiral Yazılım-donanım sistemi geliştirme için bir çerçeve sağlar​​. Kodla ve düzelt küçük projeler için uygundur. Çevik metodlar daha karmaşık projeler için uygundur.

**6.Scrum Günümüzde Neden Popüler?**

Zaman ve maliyet konusunda tasarruf sağlar. Gereksinimlerin tam olarak belirlenemediği ve karmaşık bir ortam bulunduğu zamanlarda en uygun modeldir. Kullanılması kolaydır. Sıklıkla yapılan toplantılarla sürecin kontrol edilmesi ve değişikliklerin takip edilmesi çok kolaydır. Müşteriden sürekli geri bildirimler alındığı için hatalar erken fark edilerek hızlıca düzeltilir. Günlük toplantılar sayesinde ekip içinde bireysel verimlilikte kolayca takip edilir. Aynı zamanda ekip üyelerinin yaşadığı sıkıntılar hızlıca tespit edilip giderilebilir. Müşterinin kullanımına sürekli bir prtotip çıkarılarak gereksinimleri daha iyi karşılanır. Herhangi yazılım geliştirme projesi için ygulanabilir. Ayrıca scrum sadece yazılım geliştirmeler için değil her türlü proje için uygulanabilir bir yöntem.

**7.KAYNAKÇA**

<https://www.quora.com/Why-is-the-Scrum-process-so-popular-in-the-software-industry>

<https://www.techwell.com/2013/02/why-scrum-so-popular>

<https://www.guru99.com/software-development-life-cycle-tutorial.html#11>

<https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>

BROOKSHEAR J.G. Bilgisayar Bilimine Giriş, Nobel Akademik Yayıncılık Eğitim Danışmanlık Tic. Ltd. Şti., 2016

YÜCALAR, F. ve BORANDAĞ, E, (2018), Yazılım Mühendisliğine Modern Yaklaşımlar,Ankara: Nobel Akademik Yayıncılık Eğitim Danışmanlık Tic. Ltd. Şti.