**Yazılım Yaşam Döngü Modelleri**

Bir yazılımın üretim aşamasından başlayıp kullanım süresini de kapsayan tüm süreçte yazılımın geçirdiği tüm aşamalar, yazılım yaşam döngüsü olarak adlandırılır . Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için , söz konusu aşamalar sürekli bir döngü biçiminde ele alınır. Yazılım yaşam döngüsü sarmal bir döngüdür. Döngü içerisinde herhangi bir aşamaya geri dönülebilir ve tekrar aynı aşamadan devam edilmesi mümkündür. Bu döngü bazı kaynaklara göre 5 ana bölümden oluşurken bazı kaynaklara göre bu sayı 10’a kadar yükselmektedir. Biz bu yazımızda 5 ana bölümden bahsetmiş olacağız. Yazılım yaşam döngüsünün ana bölümleri aşağıdaki gibidir;

**1-Planlama:** Müşterinin istediği gereksinimler öğrenilir , temel ihtiyaçlar belirlenir. Proje için fizibilite çalışmaları yapılır. Tüm aşamaların başlangıç aşamasıdır. Tüm projenin planlanması bu aşamada yapılır. **2-Analiz:** Yazılımın işlevlerini ve gereksinimlerini ayrıntılı bir şekilde incelendiği ve ortaya çıkarıldığı aşamadır. Mevcut sistem incelemesi yapılır. Bu çalışmada yazılım mühendisi, sistem analisti ve müşterinin bulunması projenin başlangıcında ortaya çıkabilecek sorunları en aza indirmektedir. Çeşitli yazılım geliştirme metodolojilerinde bu aşamada kullanım dokümanları ve test plan dokümanları da oluşturulabilir. **3-Tasarım:** Tasarım aşaması bir tür süreç şeklindedir. Aşamalar halinde sürdürülen bu süreç sonunda ortaya çıkan tasarım , kodlamanın ve testin temelini oluşturduğu için  mutlaka yeterli dikkat ve zaman ayrılmalıdır. Gereksinimlerin tamamlanmasıyla yazılımın ve sistemin tasarımı yapılır. Planlamaya göre proje çizimleri yapılır. Mimari tasarım dokümanları oluşturulur. Bu dokümantasyona göre projede hangi arayüzlerin kullanılacağı, yazılım ürününün özellikleri ve yetenekleri ortaya konulur. Tasarımın birinci amacı basit ve anlaşılır olmalıdır. Bu hem kodlamada hem de sonradan yaptığımız değişikliklerde bize kolaylık sağlar. Bu aşama ikiye ayrılır. Bunlar; **3.1)** Mimari Tasarım: Yazılım modüllerinin genel yapıları ve organizasyon içerisindeki etkileşimleri ile ilgilenir. Sonucunda mimari tasarım dokümanları oluşturulur. **3.2)**Detaylı Tasarım: Nasıl ve Ne sorularına cevap aranır. Tasarıma bu sorulara cevap vererek devam edilir. Bu aşamada kullanılan en önemli yöntem soyutlama yöntemidir. Soyutlama kavramı veri, işlev ve yapısal açılar için geçerlidir. Örneğin bir kapı nesne olarak ele alındığında onun kulpu, rengi menteşeleri, malzemesi gibi detayları düşünmeden kapıyı bir ev mimarisi içinde değerlendirebiliriz. Aksi taktirde diğer detaylara yoğunlaşan bir tasarımcı ‘oda’ düzeyinde görsel canlandırmalara hakim olamaz. **4-Gerçekleştirme(kodlama):** Bu aşama , çözümlemesi ve tasarımı bitmiş olan sistemin kodlanması aşamasıdır. Müşteriye teslim edilecek ürünün yapımının gerçekleştiği aşamadır. Daha önceden tasarım aşamasında karar verilen ortama uygun olarak, yine tasarım aşamasında verilen kararlar doğrultusunda projenin gerçekleştirilmesine başlanır. Kodlamamızı yaparken kodlama kalite standartlarına uymamız gerekmektedir. Kodlama kalite standartlarına uymamız hem bize hem de ürünümüzü inceleyen başka yazılımcılara kolaylık sağlar. Kodlama yaparken bu standartlara uyarsak sonradan karşımıza çıkan herhangi bir problemde veya kod yenileme işlemlerinde işimiz daha çok kolaylaşır. Bu aşamada sistem artık hayata geçer. Oluşturduğumuz sistemi test etmemiz çok önemlidir. Analiz aşamasından itibaren test bakış açısıyla ürünümüze yaklaşırsak hata yapma oranımız ve projenin maliyeti azalmaktadır. Her duruma göre uygulanabilecek çok farklı kategoride ve derinlikte test türü bulunmaktadır. **5-Teslim ve Bakım:** Tüm test aşamaları tamamlandıktan sonra yazılım ürünün sahaya teslim edilebilir bir versiyonu çıkartılır ve teslim aşaması gerçekleştirilir. Test etme ve kurulum aşamasından sonra gerçek ortama kurulan yazılımın, hata giderme, güncelleme gibi işlemlerin yapıldığı aşamadır. Bu aşama sürekli devam eden bir aşamadır. Ürün üretilir ve kalıcı olması beklenir. Bu kalıcılık belirli sürelerde yapılacak bakımlar ve onarımlar ile yapılır ve bu süreç ürün yaşadığı sürece devam eder. **\***Yukarıda 5 adımlı döngünün yanında Kendall and Kendall tarafından sunulan 7 adımlı yaklaşımdan söz etmekte mümkündür. Bu yaklaşım ise aşağıdaki gibi oluşmaktadır; a-) Problem, fırsatlar ve hedeflerin belirlenmesi b-) İnsan seviyesi enformasyon ihtiyaçlarının belirlenmesi c-) Sistem ihtiyaçlarının belirlenmesi d-) Şimdiye kadar olan adımlar ışında elde edilen isteklere cevap veren bir sistemin tasarlanması e-) Yazılımın geliştirilmesi ve dokümantasyonu. f-) Sistemin testleri ve bakım adımlarının belirlenme g-) Sistemin gerçeğe alınması ve değerlendirilmesi.

Yazılım yaşam döngüsünü takip etmemiz bize üretim aşamasında bir çok fayda gösterir. Bunlardan en önemlisi aşamalarda geri dönebilme yetkisine sahip olmamızdır. Tasarım aşamasında gözden kaçırdığımız en ufak bir hata kodlama aşamasında bize daha büyük hatalar oluşturabilir. Yazılım yaşam döngüsünü kullanarak bir aşamadaki hatayı başka aşamalara aktarmadan çözmüş oluruz. Yazılım yaşam döngüsü büyük projelerde kullanılması gerekmektedir. Çünkü küçük projelerde bu kadar ayrıntılı çalışma zaman kaybına sebep olacaktır. Bu da bizlere maliyet açısından zarar vermiş olacaktır. Yazılım yaşam döngüsünde belirtilen aşamaların hangi sırayla ve nasıl uygulanacağı ile ilgili birçok yazılım süreç modelleri vardır. Yazımızın devamında bu modellerden bahsedeceğiz. Bu modeller aşağıdaki gibidir;

**\*Gelişigüzel Model:** Bir model olarak adlandırılması doğru değildir. Herhangi bir modeli ve yöntemi yoktur. Yazılımcıya bağımlı bir proje olur. Yazılımcı belirli bir süre sonra kendisi bile anlmayabilir.1960’lı senelerde kullanılmıştır. Projenin anlaşılırlığı az olduğu için bakımı zor yapılmaktadır. Bakımı zor olduğundan bazı ürünlerin yaşamları kısa sürmektedir. Bu yüzden gelişigüzel model fazla tercih edilen bir model olmamıştır. Genellikle tek bir kişinin üretim yaptığı basit programlamalardır. **\*Barok Modeli:** Barok modelinde yazılım yaşam döngüsü doğrusal olarak uygulanmaktadır. Sırasıyla inceleme, analiz, tasarım, kodlama, testler, belgeleme ve kurulum olarak devam etmektedir. Diğer yazılım süreç modellerinden farkı, belgelemeyi ayrı olarak ele almasıdır. Bu modelin kullanımı 70’li yıllarda bırakılmıştır. Ancak günümüzde belgeleme işlemi, işin doğası olarak görülmektedir. Tüm aşamalarda belgeleme yapıldığı için ayrı bir aşama olarak göz önünde bulundurulmaz. Diğer modellerden farklı olarak aşamalar arasında geçişlerin nasıl yapılacağı tarif edilmemiştir. Barok modelinin en belirgin özelliklerinden biri ise gerçekleştirme aşamasına daha fazla ağırlık veren bir model olmasıdır. **\*Çağlayan (Şelale) Modeli:** Barok modelinden farklı olarak belgelemeyi işin doğası olarak görmektedir. Bir başka farkı ise aşamalar arasında geri dönüşlerin nasıl yapılacağı açıkça ifade edilmiştir. Günümüzde bazen projelerin yeniden oluşturulması istenildiğinden veya değişiklikler istenmesinden dolayı projeler yeniden gözden geçirilir fakat çağlayan modelinde bunu yapmak mümkün değildir. Fakat diğer modellere öncülük yapması bizim için önemlidir. Çağlayan yaşam döngüsü modelinde dikkat edilmesi gereken birçok konu vardır. Bunlardan en önemlisi aşamalar arası geri dönüşlerin nasıl yapılacağı bize açıkça ifade edilmiş olsa da analiz aşamasında mümkün olan tüm detayların ve isteklerin en ince ayrıntısına kadar belirlenip tasarım aşamasına aktarılması gerekmektedir. Analiz aşamasında “Ne istiyoruz?” sorusuna en ince ayrıntısıyla cevap verilerek tasarım aşamasına geçilir. Tasarım aşaması projenin en ayrıntılı aşamasıdır. Bu yüzden analiz ve tasarım aşamaları en fazla zaman alan aşamalardır. Bu kadar fazla zaman sürmesinin faydaları ve zararları bulunmaktadır. Uzun zamana yayılan projelerde isteklerin ve gereksinimlerin değişmesi kaçınılmaz bir sonuçtur. Bu değişikliklerin kodlama veya test aşamalarında uygulanması maliyet olarak çok yüksektir. Faydası ise bu kadar zaman alan bir çalışmayla kodlama aşamasına geçerken oluşabilecek tüm hataları en az indirgemiş olmaktayız. Yazılım üretim ekipleri programı yazıp sonucu hemen görmek istediklerinden ekip genel olarak mutsuz olmaktadır. Bu da işin işleyiş süresini etkilemektedir. Ayrıca kullanıcı üretim süresince işin içinde bulunmadığından dolayı proje bitiminde geri dönüşler artar ve bunlar maliyeti arttıran sebeplerden olmaktadır. Sonuç olarak çağlayan yaşam döngüsünde projenin süresini ve maliyetini etkileyecek birçok sebep vardır. Bu sebeplerin yanında bu modelin dökümantasyon hazırlamaya zorlaması, hataların hangi aşamalarda ve kim tarafından yapıldığı, bir sonraki aşamaya geçmeden bir önceki aşamanın tam anlamıyla tamamlanması gibi faydaları bulunmaktadır. **\*Fıskiye Modeli:** Fıskiye modeli şelale modelinden öykünerek oluşturulmuş bir modeldir. Henderson-Sellers and Edwards tarafından geliştirilmiş bir modeldir. Şelale modelinden farkı, döngüleri vardır. Döngüler şeklinde her bir aşamada belli döngüler elde edilir. Tasarım aşamasında tekrar koddan tasarıma dönülmesi, tasarımdan istek analizlerine geri dönülmesi ve operasyona geçtikten sonra testlere geri dönülmesi gibi döngülere sahiptir. Bakım ve kriterler belirlenir. Bu yaklaşımda şelale modelindeki adımları görülebilir. **\*V Süreç Modeli:** Aşamalar “V” şeklinde olduğundan V süreç modeli olarak adlandırılmaktadır. V’ nin sol tarafı üretim sağ tarafı ise sınama bölümüdür. V süreç modelinin 3 temel çıktısı bulunmaktadır. Bunlar; **-**Kullanıcı Modeli: Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceği hakkında planlamalar ortaya çıkarılmalıdır. **-**Mimari Model: İstenilen tasarımın analizlerini yaptıktan sonra tasarımın yapılması ve sınanmasıdır. -Gerçekleştirim Modeli: İstenilen yazılımın kodlanması ve sınanmasıdır. Bu modelin en iyi şekilde uygulanabilmesi için iş tanımında belirsizliklerin en az seviyede olması gerekmektedir. Bu yüzden belirsizliklerin az, iş tanımlarının belirgin olduğu bilgi teknolojileri projeleri için uygun bir modeldir. Bu süreç modelinde kullanıcının sistem gelişimine katkısı bulunmaktadır. Aşamalar genel olarak çağlayan yaşam döngüsüne benzemekte olmasına rağmen ikisini birbirinden ayıran önemli noktalar bulunmaktadır. Çağlayan yaşam döngüsünde olduğu gibi planlama, üst seviye ve alt seviye tasarımlar bulunmaktadır. Üst seviye tasarımda daha genel bakış açısıyla tasarım yapılır. Sonrasında ise daha detaylı alt seviye tasarımları yapılır. Aşağıya doğru akan bir aşama sistemi söz konusudur. Kodlama aşamasından sonra yukarı doğru çıkılmaya başlanır. Bu noktadan sonra çağlayan yaşam döngüsünden farkı ortaya konulur. V şeklinde olan bu modelde sol tarafında yapılan işlemlerin sağ tarafta aynı hizada testleri yapılır. Detaylı tasarım yapıldıktan sonra bunun hemen karşısında kodlama bittikten sonra birim testleri yapılır. Yapmış olduğumuz tasarıma ait testler yapılır. Üst seviye tasarıma ait entegrasyon testleri yapılır. İhtiyaç analizleri müşteriden istenir ve nelere ihtiyaç olunduğu çıkarılır. İhtiyaçların ise kabul testleri yapılır. Planlamanın da karşılığı bakım aşamasıdır. V şeklindeki bu modelin yaptığımız tüm aşamaları projeyi bitirmeden bize tek tek test etmemizi sağlar. Böylelikle kullanıcı ve yazılımcı arasında yapılan tüm kontroller ve değişiklikler sayesinde hata en aza indirgenir. **\*Helezonik (Spiral) Model:** 4 ana aşamadan oluşmaktadır;

- Planlama: Her döngüde üretilecek ara ürün için planlama yapılmasıdır.Bir önceki ara ürün ile bütünleştirme işlemi yapılır. -Risk Analizi: Yazılımda oluşabilecek tüm risklerin araştırılıp incelenmesi ve herhangi bir riskte nasıl çözüm bulunacağının belirtilmesi. -Üretim: Ara ürünlerin üretilmesidir. -Kullanıcı Değerlendirmesi: Oluşturulan tüm ara ürünlerin kullanıcı tarafından sınanması ve değerlendirilmesi. Bu aşamaları spiral şekilde dönerek ilerlenmektedir. Her döngü bir fazı ifade eder. Bundan dolayı adı helezonik modeldir. Bu dönme sırasında prototipler ortaya çıkmaktadır. Prototipler maket uygulamalardır. Her çıkarılan prototip müşteriye gösterilir ve yeni eklemeler veya yazılımdan çıkarmalar yapılır. Müşteri böylelikle yazılımın her noktasından haberdar olmaktadır. Öncelikle proje başlanırken istekler belirleniyor. İhtiyaçlar belirlendikten sonra risk analizleri yapılıp prototip ortaya çıkarılıyor. Bu prototip doğrulanıp onaylandıktan sonra tekrar yazılım sürecine devam edilir. Bu yazılımdan sonra bir prototip daha çıkartılıp tekrar risk analizi ve kontroller yapılır. Bu adımların sayısı 2-3 ü bulurken bazı projelerde bu sayı arttırılabilir. Uygulama prototibi yapıldıktan sonraki aşamalar şelale modeline benzer şekilde devam etmektedir. Sonucunda ürün ortaya çıkmaktadır. Her döngüde olduğu gibi Spiral döngününde avantajları ve dezavantajları bulunmaktadır. Müşterinin üretim anında ara ürünlerin sınanmasında bulunması, yazılımı kullanacak personelin sürece erken katılması ileride oluşabilecek istenmeyen durumların engellenmesi, hataların erken giderilme şansı bu döngünün avantajlarındandır. Fakat küçük projeler için maliyetli olması ve fazla dökümantasyon bulundurması dezavantajdır. **\*Prototipleme :** Prototipleme yazılım süreç modelinde amaç isminden de anlaşılacağı gibi prototip üretip geliştirmektir. Gereksinimlerin doğru anlaşılması bu süreç için çok önemlidir. Bu modelde hızlı plan, hızlı tasarım ve hızlı kodlama ön plandadır. Karmaşa ve yanlış anlaşılmayı engeller. Yeni gereksinimlere açık olması ve her aşamada risk kontrolünün yapılması başlıca avantajlarındandır. Fakat belgeleme olmayan hızlı ve kirli prototiplerin oluşturulması ve düzeltme adımı atlanırsa düşük performansa neden olması dezavantajlarıdır. **\*Evrimsel Geliştirme Süreç Modeli:** İlk tam ölçekli yaşam döngüsü modelidir. Genelde banka uygulamarı gibi geniş coğrafyaya yayılmış birimlerde kullanılması öngörülmektedir. İlk aşama olarak pilot uygulama üretimi yapılır. Bu pilot uygulama bir birimde uygulanır. Kullanılırken uygulama tümüyle test edilir. Test sonucunda eksiklikler giderilir, güncellemeler yapılır ve diğer birimlere taşınır. Bu yaşam döngüsü modelinin başarısı ilk evrimin başarısına bağlıdır. **\*Artımlı Geliştirme Modeli:** Modelde bir döngü söz konusudur. Planlamayla başlanır ve döngüye girilir. Analiz, tasarım ve uygulama aşamalarından sonra projeyi canlıya geçirme söz konusudur. Bu aşamadan sonra da problemler meydana gelebilir. Fakat bu modelde bu problemler bir test aşaması olarak görünüp döngüye bu problemlere dikkat edilerek devam edilir. Artımlı modelde kullanıcı istekleri aşamalı olarak karşılanacak şekilde birkaç yazılım ortaya konularak gerçekleştirilebilir. Öncelikler kullanıcının isteklerine göre belirlenir. Teslimler arasında oluşabilecek istekler sonraki teslimlere aktarılır. Projelerin uzun sürdüğü ve belirli eksikliklere kullanıma devam edilebilecek uygulamalarda tercih edilir. Erken teslimler , sonraki teslimler için prototip görevi görür. Bu modelde üretim varken kullanım da söz konusudur. Bu da projenin tümden batma olasılığını ortadan kaldırır. Bu modelde yazılımımızı yaparken açık mimaride ya da herkesin anlayabileceği yapıda geliştirilmelidir. Yazılımın birbiriyle ilgisiz olan parçaları bağımsız geliştirildiğinden daha hızlı yazılım olur. Yazılım kısmi olarak yapıldığı için maliyeti az olmaktadır. Bu özellikleri bu modelin avantajlı yönleriyken dikkat edilmesi gereken önemli dezavantajları bulunmaktadır. Bunlar ; Yeni yapılan yazılımın, bir önceki yazılımın karşıladığı tüm işlevleri yerine getirmek zorundadır ve yazılımın açık mimaride oluşturulması bakımın yapılmasını zorlaştırmaktadır. **\*Araştırma Tabanlı Model :**  Yap-at prototibi olarak ta bilinir. Geliştirilen yazılımlar genellikle sınırlı sayıda kullanılır ve kullanım bittikten sonra işe yaramaz hale gelir ve atılır. Sözleşmelerde fiyat anlaşması yapılması zor bir durumdur. Çünkü zaman ve fiyat kestirimi olmamaktadır. Örnek olarak en büyük asal sayıyı bulan program ve satranç programı örnek olarak verilebilir. **\*Kodla ve Düzelt:** Birkaç yüz satırdan oluşan kısa programlarda kullanılır. İlk aşama olarak yazılım ürününün ilk sürümü üretilir ve en son istenilen şekle gelinceye kadar devamlı geliştirilir. Bakım aşaması bu döngüde çok zordur çünkü belirli bir dökümantasyon aşaması yoktur. Yazılım geliştirmenin en kolay yoluyken aynı zamanda en pahalısıdır. Bunun sebebi ise kodlamadan sonra uygulamada yapılacak değişikliklerin maliyetli olmasıdır.

**\*Bileşen Tabanlı Model:** Sistemin COTS(commercial of the shelf) adı verilen hazır bileşenler birleştirilerek uygulama elde edilir. Bu yaklaşım , bileşen standartlarındaki gelişmelerden dolayı daha yaygın olarak kullanılmaya başlanmıştır. -COTS: Piyasadan hazır temin edilebilen malzeme ifadesi savunma programlarında kullanılabilen ve üretildikten sonra piyasadan temin edilebilen ürünleri tarif etmektedir.

**Çevik Yazılım Geliştirme Metotları**

Yazılım geliştirme süreci sıkıntılı, yorucu ve uzun süren bir dönem olduğundan ve yönetimsel eksikliklerden dolayı projelerin kısmi başarılı olmasından sebeple 1970’lerde başlanıp 1990’larda hız kazanan metotlar geliştirilmiştir . Bunlar çevik yazılım geliştirme metotları olarak adlandırılmaktadır. Çevik yazılım geliştirme metodu, tekrarlanan yazılım geliştirme metodu taban alınarak geliştirilmiş, sık aralıklarla parça parça yazılım teslimatını ve parçalarda değişikliğe gidilmesini teşvik eden bir yazılım geliştirme metodudur. Bu metotların kullanıldığı projelerin bitirilme hızı , proje ekibinde çalışan tüm ekip üyelerinin sürekli iletişim halinde olmasına bağlıdır. Projenin küçük parçalara bölünüp üretilmesi geriye dönük hataların daha kolay giderilmesini sağlamaktadır. Böylelikle daha az hatalı , verimliliği yüksek , hızlı ve ucuz çözümler üretilmektedir. 2001 yılında Kent Beck ve beraberinde dünyanın önde gelen çevik yazılım geliştiricileri bir araya gelerek “çevik yazılım manifestosun” nu ve “çevik yazılım prensipleri” ni yayınlamışlardır. Bu manifestoda ; Bireyler ve aralarındaki etkileşimlerin, kullanılan araç ve süreçlerden; Çalışan yazılımın, detaylı dokümantasyondan; Müşteri ile işbirliğinin, sözleşmedeki kesin kurallardan; Değişikliklere uyum sağlayabilmenin, mevcut planı takip etmekten; daha önemli ve öncelikli olduğu belirtilmektedir. Çevik yazılım prensipleri ise aşağıdaki gibidir; -Yazılımcının birinci önceliği hızlı bir şekilde kaliteli ürünü müşteriye teslim edebilmektir. Bu da projenin ilk aşamasından itibaren devamlı olarak projeyi parça parça teslim ederek müşterinin yazılımı çok önceden kullanmasıyla değer kazanmasıyla sağlanır. -Müşterinin yazılımda istediği değişiklikler projenin her aşamasında kabul edilir çünkü yazılımın amacı müşterinin ihtiyaçlarını karşılamak ve sorunlara çözüm üretmektir. -2-4 hafta aralıklarıyla projenin parça teslimleri yapılmalıdır. Bu sayede müşteri ile sürekli iletişim halinde kalınır bu da proje hakkında devamlı geri dönüş almamızı sağlar. -Proje ekibindeki tüm bireyler birlikte çalışırlar. Sürekli iletişim halinde bulunulması bilgi paylaşımını sağlar. -Proje ekibi motivasyonu yüksek, istekli kişilerle kurulur. Ekibe tüm desteğin sağlanılması gerekmektedir. -Proje ekibinin iyi bir şekilde organize edilmesi gerekmektedir. En iyi mimariler ve tasarımlar bu şekilde ortaya çıkmaktadır. -Proje ekibi kısa aralıklarda toplanmalı ve çalışma yöntemlerini tartışmalılardır. Bu tartışmalar sonucunda gerekli iyileştirmeler yapılmalıdır.

Çevik yazılım geliştirme metotlarını kullanmak yazılım ekibinin motivasyonunu yükselterek yazılım kalitemizi arttırmamızı sağlar. Kısa sürede müşteri memnuniyeti sağlanır ve üretkenliğin artması sağlanır bu da maliyetin azalmasını sağlar. En yaygın kullanılan çevik metodojileri aşağıdaki gibidir;

**EXTREME PROGRAMING (XP)**

XP Kent Beck tarafından 1999 yılında ortaya çıkarılmıştır. XP kendisine kadar süregelen metotların aşamalarının daha ayrıntılı uygulandığı bir metottur. Örneğin bir projenin iyi bir proje olması için başlangıçta müşteriden istekleri en iyi şekilde almamız gerekmektedir. Bu sebepten dolayı XP’ de müşteri tüm aşamaların içinde vardır. Müşteri de ekibin bir parçası olarak testler yapılır ve yazılım geliştirilir. Yazılımcılar birim testleri oluşturur ve tüm mantıksal noktalar test edilir. XP karmaşık problemlere basit çözümler bulmayı öngörür. Bu basit sistemi oluşturabilmek için günümüzün ihtiyaçları hedef alınır. Bu metotta 2-4 hafta aralıklarıyla sürümler ortaya konulur. Müşteri , yönetici ve yazılımcı ekip sistemin durumunu belirler ve değerlendirme yapılır. XP’nin temel noktalarından en önemlisi ise cesaretli olmaktır. Önümüze ne kadar büyük projeler gelse de başarısızlıktan korkmadan başarısızlığı oluşturabilecek nedenler üzerine gidilip çözümler üretilmelidir. Başarısızlıktan korkmak projenin yapım hızının düşmesini sağlar. XP yazılım geliştirmede kolaylığı ve esnekliği oluşturabilmek için 12 pratik ön görmektedir. Bunları madde madde inceleyeceğiz ; +Planlama Oyunu : Müşterinin yeni sürümlerde istediği yinelemer , toplantılarda öğrenilir ve yazılımcı o işin ne kadar zamanda yapılacağıyla ilgili tahmin yürütür. Böylelikle yazılım ekibi bu süreye tabi kalarak müşterinin isteklerini yerine getirir. +Ekipte Müşteri: Müşteri ekibin bir parçası olarak görünür. Böylelikle yazılımcının projeyle ilgili bilgileri kısa zamanda elde etmesi sağlanır. +Önce Test: Kodlama aşamasına geçilmeden testler oluşturulur. Böylelikle hatalar daha önceden tespit edilir. +Basit Tasarım : Müşterinin projedeki isteklerine göre en basit tasarım oluşturulur. Bu basit tasarım sayesinde yönetilmesi ve değiştirilmesi daha kolay bir yazılım oluşturulmaktadır. +Çiftli Programlama: Programın geliştirilmesine en çok katkı veren pratiktir. Her yazılımcının bilgi birikimi farklı olduğundan yazılımcılardan bir diğerine göre probleme daha çabuk çözüm üretebilir . Bu da projenin daha hızlı üretilmesini sağlar. Özellikle işe yeni başlayan yazılımcılar için önerilir. +Sürekli Entegrasyon: Sistemde yapılan değişiklikler anında sisteme entegre edilerek yazılımcıların sistemdeki değişiklikleri ve oluşabilecek hataları erkenden görmelerini sağlar. +Kısa Aralıklı Sürümler: Proje, 2-4 hafta aralıklarla bitecek birimlere bölünür ve bu birimler belirlenen sürenin bitimine kadar teslim edilmelidir. Böylelikle müşteri her aşamayı daha yakından takip eder ve devamlı yaşayan bir uygulama kullanır. +Yeniden Yapılandırma: Müşteriyi memnun edebilmeyi amaçlayan yazılımcı kod ve tasarımı devamlı gözden geçirir. Böylelikle müşteriden gelen yeni istekler basit tasarımlarla proje eklenir. +Ortak Kod Sahiplenme : Üretilen proje tüm ekibin ortak malıdır. Böylelikle ekipteki herkes bir başka ekip arkadaşının yazdığı koda erişebilir ve birbirlerini uyararak yazılımı daha iyi bir noktaya getirilmesini sağlarlar. +Benzetim : Projede parçalara bölünen her birim birbirine benzetilerek yazılım geliştirilmeye çalışılır. Böylelikle her parça birleştirilerek kaliteli bir proje elde edilir. +Kodlama Standartları : Belirlenen kodlama standartlarına uyularak yazılım geliştirilir. Böylelikle kodlar karmaşıklığı az , ekip arkadaşları tarafından daha okunabilir kodlar olmaktadır. +Haftada 40 Saat : Günde 8 saat çalışarak haftada en fazla bir kere mesai yaparak beş gün çalışılmalıdır. Daha fazla çalışıldığı durumda verimlilik azalır ve hata oranı artar.

**SCRUM**

Günümüzde en çok kullanılan metottur. Bu yazımızda Scrum’ın projelerde neden bu kadar fazla tercih edildiğinden bahsedeceğiz. Öncelikle , Scrum 1990’ların ortasında Jeff Sutjerland ve Ken Schawaber tarafından bir proje yaklaşımı olarak ortaya atılmıştır. Yani sadece yazılım alanında değil her projede kullanılabilecek bir metot ortaya koymuşlardır. Bu metodolojide, karmaşık uygulamalar parçalara ayırarak ekip halinde çözüm üretilmektedir. Scrum’da her bir yineleme 30 günden fazla sürmektedir ve günlük 15 dakika toplantılarla ufak durum değerlendirilmeleri yapılmalıdır. Peki Scrum günümüzde neden daha çok kullanılmaktadır? Çünkü rekabetçi piyasa koşullarında şirketlerin kar oranını arttırabilen bir metottur. Bunun birçok nedeni vardır. Yenilenen değişen teknolojiye uyum sağlayabilecek esnekliğe sahip olması bunun en büyük sebeplerinden biridir. Günlük toplantılar sayesinde yüz yüze iletişim olduğundan kaliteli bilgi akışı sağlanıp kaliteli ürünler üretilmektedir. Scrum metodunda müşterinin değişen gereksinimleri her yinelemede eklenir ve döngüye dahil edilir. Ürün yaşam döngüsü boyunca güncellenir. Bu da müşteri memnuniyetini arttıran başlıca nedenlerden biridir. Scrum’da görev dağılımı yapılmaktadır. Herkesin yapması gereken görevler belirlenmiştir. Ürünün sahibi projenin iş değeri açısından geri dönüşü ile sorumludur. Scrum’da en önemli rol Scrum takımıdır. Scrum takımı projenin en iyi halini alması için devamlı iletişim halinde olan , içinde 5-9 kişi bulunduran bir takımdır. Bu takımın başında sürecin anlaşılmasını ve organizyonunu sağlayan Scrum Yönetici bulunmaktadır. Scrum takımının önündeki engelleri aşmak adına takıma hizmet eder. Scrum yönetici talimat vermemeli ve anlaşma sağlamaya yönelik olup önerilerde bulunmalıdır. Aksayan her işi hatırlatmakla sorumludur. Günümüzde Scrum metodunun kullanılmasının en büyük sebebi ise kısa zamanda en kaliteli , maliyeti düşük projeler ortaya konmasıdır. Bunu sağlamanın en güzel yolu ise proje ekibinin devamlı iletişim halinde olmasıdır. Scrum metodu devamlı toplantıyı gerektiren bir metottur. Ekip proje ilk geldiğinde Sprint(Koşu) Planlama toplantıları yapmaktadırlar. Bu toplantıda geniş kapsamlı gereksinim listesi yapılmaktadır. İş bölümü yapılır , projenin parçalarından hangisini kimin yapacağı belirlenir. Projenin uygulanabilmesi için geliştirme araçlarının ve altyapının değerlendirilmesi yapılır. Ayrıca projenin maliyeti tüm riskler göz önünde bulundurularak hesaplanır. Sonrasında projeye başlanır ve her sprint başlangıcında Sprint gözden geçirme toplantıları yapılır. Bu toplantılarda gereksinim listesi incelemesi yapılarak hedeflere ne kadar ulaşıldığı belirlenir. Liste bu toplantıda yenilenir, eklemeler yapılabilir. Bu toplantıların dışında günlük olarak ayakta ekip elemanları birbirleriyle ne yaptıkları, ne yapacakları ve hangi problemlerle karşılaştıklarını yaklaşık olarak 15 dakikalık süreyle paylaşırlar. Bütün bu sebeplerden dolayı günümüzde dünyanın en büyük şirketleri de dahil büyük bir çoğunluk Scrum metodunu projelerini yaparken kullanmayı tercih etmektedir.

Sonuç olarak bütün yazımızda yazılımımızı yaparken hangi metotları kullanabileceğimiz hakkında genel bilgiler verdik. Projelerimizi hayata geçirirken projeyi iyi tanıyıp gereksinimleri iyi bir şekilde anlayıp hangi metodu kullanmamız gerektiğine karar vermemiz gerekmektedir. Hangi projede hangi metodu kullanmayı bilirsek kısa zamanda maliyeti düşük, kaliteli , hata oranı az ve verimli projeler ortaya koymuş oluruz.