**YAZILIM YAŞAM DÖNGÜSÜ**

Çağımızın getirilerinden olan ve hayatımızda büyük bir yer edinen bilgisayar, donanım ve yazılım olmak üzere iki ana unsurdan meydana gelir. Donanım, fiziki olarak işleyişi sağlayan parçaların tamamı; yazılım ise özetle, bilgisayara nasıl çalışacağını söyleyen komutlar ve veriler bütünüdür. Bilgisayar yazılımları, sistem yazılımları ve uygulama yazılımları olmak üzere iki bölümde incelenmektedir. Aygıt sürücüleri, işletim sistemleri, uygulama sunucuları gibi sistem yazılımları bilgisayarın çalışması için gerekli olan ana fonksiyonları yerine getirirken; uygulama yazılımları bir ara yüz vasıtasıyla kullanıcıdan veri alıp işleyen ve sonuçları yine kullanıcıya gönderen yazılımlardır. Diğer bir anlatımla, kullanıcıların işlerine çözüm bulmalarını sağlayan stok takip programı, personel izleme sistemi, üniversite otomasyonu, muhasebe programları gibi yazılımlardır. Günümüzde giderek artan yazılım projeleri ile yazılımın işleyiş biçimi ve kullanımı büyük önem taşımaktadır. Yazılım, her sistemde olduğu gibi kendi içinde belirli bir düzene bağlıdır. Bir yazılım projesinin geliştirilmesi sadece kodlamadan oluşmamaktadır. Proje yapım aşamasında, yazılım süreci adımlar halinde gerçekleştirilir ve bu uygulanan adımlar bir döngü oluşturur. Bu döngü Yazılım Yaşam Döngüsü adını alır. Basitçe yazılım yaşam döngüsü, bir yazılımın geliştirilmesi ve bakımı süresince icra edilen adımlar topluluğudur. Adımlar; planlama, analiz, tasarım, gerçekleştirim ve bakım şeklinde sıralanabilir. Döngü içerisinde herhangi bir aşamada geriye dönmek ve tekrar ilerlemek söz konusudur. Diğer bir deyişle yazılım yaşam dönüsünün tek yönlü ve doğrusal olduğu düşünülmemelidir. Proje tamamlandıktan sonra da müşteriden gelecek istekler ve değişen gereksinimler (Değişen Hedef Problemi), hata düzeltmeleri, projeye eklenecek yeni modüller ve veriler gibi konular için bu süreç devam etmektedir. Kısaca aşamaları inceleyecek olursak ilk olarak bir yazılım projesinin planlaması gerçekleştirilmelidir. Üretilecek yazılım ile ilgili olarak müşteri ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşama olarak tanımlanabilir. Analiz aşaması ise yazılım işlevleri ile gereksinimlerin ayrıntılı olarak çıkarıldığı bölümdür. Temel olarak sistemde var olan işler incelenir, temel sorunlar ortaya çıkarılarak, yazılımın çözümleyebilecekleri vurgulanır. Ek olarak bu aşamada temel UML diyagramları çizilebilir (Use Case, Activity, Class Diagram… vs.). Bir sonraki aşama olan tasarım, analizden sonra belirlenen gereksinimlere karşılık verecek yazılım ya da bilgi sisteminin temel yapısının oluşturulması çalışmalarıdır. Bu çalışmalar “Üst Seviye ve Mimari tasarım” ve “Detaylı tasarım” olmak üzere iki grupta incelenir. Üst seviye ve mimari tasarımda mevcut sistem değil önerilen sistemin yapısı anlatılır ve olası örgütsel değişiklikler önerilir (modüller, akış şemaları…).Detaylı tasarımda ise yazılımı içeren bileşenler ve bunların ayrıntıları bulunur (veri yapıları, ekran tasarımları…). Gerçekleştirme aşaması modüllerin kodlandığı, birleştirildiği, test edildiği ve kurulum çalışmalarının yapıldığı aşamadır. Kodlama esnasında ve sonrasında yapılan önemli bir aşamada test etmedir. Yazılımı mümkün oldukça erken test ederek zaman, para ve prestij gibi kayıplar yaşamamak temel amaçtır. Daha sonra da ürün piyasaya veya müşteriye sunulur. Son olarak teslimden sonra başlayan bakım aşaması ele alındığında işletime alınan yazılım ile ilgili olarak hata giderme ve yeni eklentiler yapma aşaması olarak bilinir ve bu aşama yazılımın tüm yaşamı boyunca süren kısımdır. Bilinmelidir ki yazılım geliştiricileri sonuç olarak insandır ve hata yapabilmeleri yüksektir. Bu da yazılım projesinde değişim demektir. Yapılan herhangi bir değişiklik sonucu potansiyel olarak bir regresyon hatasına neden olabilir. Bu nedenle değişimde çoğu zaman zor ve sıkıntılı durumlar yaşanabilir ve ürünün ortaya konulma zamanı uzatılabilir. Ancak proje tasarımı ileride değişiklik olabilecek şekilde yapılmışsa sorun en aza indirgenebilir. Bir diğer değişim müşteriden kaynaklı olarak gereksinimlerinde sürekli değişiklik olması durumunda gerçekleşir. Bu duruma değişen hedef problemi denmektedir.

Yazılım yaşam döngüsünün temel aşamaları çekirdek süreçler (core processes) olarak da adlandırılır. Yazılım süreci, bir yazılım ürününü üretmeyi sağlayan birbiriyle tutarlı aktivite grubudur. Ne yapılmak istendiğini tüm uygulama detaylarına girmeden tanımlar ve yazılım üretme yolunu belirler. Bu süreçlerin gerçekleştirilmesi amacıyla Belirtim Yöntemleri (Software Specification Methods) ve Süreç Modelleri (Software Process Models) geliştirilmiştir. Belirtim yöntemleri bir çekirdek sürece ilişkin fonksiyonları yerine getirmek için kullanılan yöntemlerdir. Bu yöntemleri üç aşamada incelemek mümkündür. Süreçler arasındaki ilişkileri ve iletişim durumunun gösterildiği yöntemler süreç akışı için kullanılan belirtim yöntemleri (veri akış şemaları, nesne-sınıf şemaları), gerçekleşen süreçlerin iç işleyiş durumlarını göstermek için kullanılan yöntemler süreç tanımlama yöntemleri (karar ağaçları, algoritma), süreçler tarafından kullanılan verilerin tanımlanmasında kullanılan yöntemler ise veri tanımlama yöntemleridir.

Süreç modelleri ise yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında hangi düzen ya da sırada nasıl uygulanacağını tanımlayan modellerdir. Süreç modelleri karmaşıklık düzenini azaltır böylece oluşacak krizleri önler. Ürünlerin belirli bir kalitede olmasına olanak sağlar ancak bu süreçlerin kontrol edilmesine bağlıdır. Yönetici ve geliştiricilerin, yazılım geliştirme sürecinin karışıklığı ile baş etmelerini sağlar. Bu modellerin temel hedefi; proje başarısı için yazılım geliştirme, yaşam döngüsü boyunca izlenmesi önerilen mühendislik süreçlerini tanımlamaktır. Modellerin ortaya çıkmasında ilgili dönemin, donanım ve yazılım teknolojileri ile sektör ihtiyaçları önemli rol oynamıştır. Bu modeller süreçlerin içsel ayrıntıları ya da süreçler arası ilişkilerle ilgilenmemektedir. Özetle yazılım üretim işinin genel yapılma düzenine ilişkin rehberler olarak kullanılabilir. Süreç modelleri temel olarak 3’e ayrılırlar. Bunlar: düzenleyici süreç modelleri, birleşik süreç modeli ve çevik yazılım süreç modelleridir.

**YAZILIM SÜREÇ MODELLERİ**

**DÜZENLEYİCİ SÜREÇ MODELLERİ**

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

Birkaç yüz satırdan oluşan programlar için kullanılabilir. Genellikle resmi olmayan bir ürün fikriyle başlar ve program ürün hazır olana kadar ya da gerekli zaman bitene kadar kodlama yapılarak devam edilir. Planlama, analiz yapma vs. kısımlar ile çok vakit kaybetmeden hemen ana problem üzerine odaklanılarak sonuca gitme hedeflenir ve direkt olarak yazılım ürünü gerçekleştirilir. Tüm gereken yeterli olacak kadar gayret ve özveridir. Emeklilik safhası bulundurur. Yazılım geliştirmenin en kolay yoludur ancak en pahalısıdır.

Avantajları; herhangi bir planlamaya ihtiyaç duymaz, çok küçük projelerde ya da kısa ömürlü prototiplerde uygulanabilir, program aşamaları çabuk geçilir, uzman görüşüne ihtiyacı düşüktür bu yüzden herkes bu modeli kullanabilir.

Dezavantajları; hataların bulunması ve doğrulanması zordur, kodlamaya başlamadan önce değişiklik tahmin edilmediğinden, birbirini izleyen değişikliklerden sonra kod karmakarışık bir hale gelir ve daha sonraki düzeltmeleri yapmak daha da zordur, kodları düzeltmek maliyetli olabilir, kontrollü değildir, kaynak planlaması yoktur, bitiş süresi belli değildir, müşterinin sürece dahil edilmemesi kullanıcı ihtiyaçlarına uygun olmamasına yol açar, bireysel geliştiriciler için uygundur, takımlar için değildir.

* Gelişigüzel Model

Geliştirme ortamında herhangi bir model ya da yöntem kullanılmaz bu nedenle bir model olarak adlandırmak doğru değildir. İlgi duyan ve geliştiren kişiye bağımlıdır ancak belli bir süre sonra geliştiren sistemi anlamayabilir ve geliştirme güçlüğü yaşar. İzlenebilirliği ve bakımı oldukça zordur. 60’lı yıllarda daha çok kişilik üretim ortamlarında kullanılan yöntemlerdir. Yani basit bir programlama yöntemidir bu nedenle günümüzde kullanılması çok önerilmemektedir.

* Barok Modeli

Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği ve aşamalar arası geri dönüşlerin nasıl yapılacağı belirsiz olabilen süreç modelidir. Bu model 70’li yılların ortalarından başlanarak kullanılmaya başlanmıştır. Belgelemeyi ayrı bir süreç olarak ele alır. Gerçekleştirim aşamasına daha fazla ağırlık veren bir modeldir. Ayrıca yazılımın, geliştirilme süreci ve test edilmesinden hemen sonra yapılmasını öngörür. Oysaki günümüzde belgeleme yapılan işin doğal bir ürünü olarak görülüp döngünün başından sonuna kadar sürekli yapılır. Bu sebeplerden dolayı günümüzde kullanılması pek önerilmemektedir.

* Çağlayan Yaşam-Döngü Modeli

Diğer bir adı Şelale (Waterfall) modelidir. Yaşam döngüsü temel adımları baştan sona en az bir kez kullanılarak gerçekleştirilir. İyi tanımlı projeler ve üretimi az zaman gerektiren yazılım projeleri için uygun bir modeldir. Barok modeline göre geri dönüşler iyi tanımlanmıştır ve barok modelinin aksine belgeleme işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür. Bir aşama bitmeden diğer aşamaya geçilemez. Her aşamanın sonucu bir ya da birden fazla onaylanan (imzalanan) belgedir. Eğer bir aşamada belgeleme ve test olmamışsa o aşamanın tamamlandığı kabul edilemez. Yazılımı tanımlamada belirsizlik yok ya da az ve aynı zamanda yazılım üretimi çok zaman almayacak şekilde tasarlanmışsa kullanıma uygun bir süreç modelidir. Gerektiğinde geliştirme aktivitelerinde iterasyonlar (tekrarlamalar) olabilir. Çağlayan yaşam-döngü modelinde dikkat edilmesi gereken önemli hususlar vardır bunlardan en öne çıkanı, her ne kadar model içerisinde aşamalar arasında geri dönüşler yapılabilse de analiz aşamasında mümkün olan tüm detayın tasarıma yansıtılabilmesi için müşteri ve sistem gereksinimlerinin en ince ayrıntısına kadar belirlenmesi gerekir. Geleneksel model olarak da bilinen bu modelin kullanımı günümüzde giderek azalmaktadır. Çağlayan modeli aşamalardan oluşmaktadır. İlk olarak Gereksinim Tanımlama gerçekleştirilir. Gerçekleştirilecek sistemin gereksinimleri belirlenir. “Müşteri ne istiyor?”, “Ürün ne yapacak, ne işlevsellik gösterecek?” gibi sorular sorulur. Sistem ve Yazılım Tasarımı aşamasında gereksinimleri belirlenmiş bir sistemin yapısal ve detay tasarımını oluşturma işidir. Tasarım aşaması, yazılımın tüm gereksinimlerini karşılayacak şekilde detaylı bir çalışma gerektirmektedir. “Ürün, müşterinin beklediği işlevselliği nasıl sağlayacak?” sorusuna cevap verir. Bir sonraki aşama Gerçekleştirme ve Birim Test aşamasıdır. Bu aşamada tasarımı yapılmış bir yazılım sisteminin kodlanarak gerçekleştirilmesidir. Daha sonrasında Birleştirme ve Sistem Testinde gerçekleştirilmiş sistemin beklenen işlevselliği gösterip göstermediğini sınama işlemi yapılır. “Ürün, müşterinin beklediği işlevselliği sağlıyor mu?” sorusuna cevap verir. Son aşaması ise İşlem ve Bakım aşamasıdır. Bu aşamada müşteriye teslim edilmiş ürünün, değişen ihtiyaçlara ve ek müşteri taleplerine göre güncellemesi yapılır. ”Ürün müşteri tarafından memnuniyetle kullanılabiliyor mu?” sorusunun cevabıdır.

Avantajları; Müşteriler veya son kullanıcılar tarafından da iyi bilinen ve anlaşılabilen adımlardan oluşur. İterasyonlar (tekrarlamalar) bir sonraki ve bir önceki adımlarla gerçekleşir, aralıklı adımlarla olması nadirdir. Değişiklik süreci yönetilebilir birimlere bölünmüştür. Gereksinim adımı tamamlandıktan sonra sağlam bir temel oluşur. Deneyimlerle erken işin miktarını arttırdığı görülmüştür. Projenin safhaları ayrı olduğundan iş bölümü ve iş planı projenin en başında net bir şekilde bellidir. Bu durum projenin yönetimi sırasında proje yöneticileri için durumu kolaylaştırır. Aşamaları iyi anlaşılabilir. Gereksinimleri iyi anlaşılabilen projelerde iyi çalışır. Kalite gereksinimlerinin bütçe ve zaman kısıtlamasına göre çok daha önemli olduğu projelerde iyi çalışır.

Dezavantajları; Bitirme kriteri olarak belgelendirmeye önem verilmektedir. Bu bazı alanlar için mümkün olsa da (derleyiciler, işletim sistemleri vb.) etkileşimli son kullanıcı uygulamaları gibi alanlar için zordur. Sistem geliştirilmesi süresince var olan gereksinimler sıklıkla değişir. Çağlayan modeli gereksinimlerin çok iyi anlaşılabildiği durumlarda kullanılmalıdır. İki ya da daha önceki fazlara girmek çok maliyetlidir. Bu durumda da gerektiğinde tüm fazı yeniden gerçekleştirmek çok büyük bir iştir. Ayrıca kullanıcının sürecin içerisinde yer almaması geri dönüşleri arttıracağından bu geri dönüşlerde yazılım geliştirme maliyetini büyük oranda yükselten bir durumdur. Bir faz tamamlanmadan diğerine geçilememesi hata riskini arttırır. Genelde yazılımın kullanıcıya ulaşma zamanı uzundur. Yazılım üretim ekipleri bir an önce program yazma, çalıştırma ve sonucu görme eğilimde olduklarından, bu model ile yapılan üretimlerde ekip mutsuzlaşmakta ve kod yazma dışında kalan kesime önem vermemektedirler. Ayrıca model aşamalardan oluştuğu için ürün son aşamada tamamlanır, ​​ gereksinimlerin iyi tanımlanmadığı müşterinin ne istediğinin anlaşılmadığı bir projede bu durum projenin bittikten sonra iptal edilmesine ve başka gerginliklere sebep olmaktadır.

* V Süreç Modeli (V-shaped Model)

V modeli, Doğrulama ve Geçerleme (Onaylama) modeli anlamına gelir Süreç adımları kodlama aşamasından sonra yukarıya doğru eğim aldığı ve V şeklini oluşturduğu için bu ismi almıştır. V-Model, geliştirme yaşam çevriminin her bir evresi arasındaki ilişkileri gösterir. Yatay ve dikey açılar zaman veya projenin tamamlanabilirliğini ve soyut seviyesini gösterir. Çağlayan yaşam-döngü modelinin biraz daha gelişmiş halidir. Bu modelde de her aşama bir sonraki aşama başlamadan önce tamamlanmalıdır. Sol taraf üretim, sağ taraf ise sınama bölümüdür. Üst ve alt seviye tasarım vardır. Üst seviye daha genel bir tasarımdır. Alt seviye de ise girdiler-çıktılar, beklentiler gibi daha detaylı bir tasarım mevcuttur. Bu yöntemde proje takibi kolaydır. Modelin kullanımı genel olarak basittir. Fakat aşamalar arası tekrarlamaları kullanmaz. V-Modelinin en temel özelliği, ürünün test edilmesi kendisine karşılık gelen geliştirme aşamasına paralel olarak planlanmaktadır. Risk çözümleme ile ilgili herhangi bir işlem yapılmaz, bu da hata olasılığını arttıran bir etkendir. V süreç modelinin temel çıktıları ise Kullanıcı Modeli, Mimari Model ve Gerçekleştirim Modeli olarak sıralanabilir. Kullanıcı modelinde geliştirme sürecinin kullanıcı ile olan ilişkilerini tamamlamakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarmaktadır. Mimari model, sistem tasarımı ve oluşacak alt sistem ile tüm sistemin sınama işlemlerine ilişkin işlevleri içermektedir. Gerçekleştirim modeli ise yazılım modüllerinin kodlanması ve sınanmasına (test edilmesine) ilişkin fonksiyonlardır. Test türleri: Unit, Integration, Regression, Smoke, Beta, System, Stress, Performance Testing olarak bilinir. Aslında bu testlerin hepsi gerçekleştirim değil diğer fazları da içeren testlerdir. Amaç her aşamanın karşısında doğrulama ve onaylama mekanizmalarını koyarak sistemin düzgün ve kalite çalıştığından emin olarak ilerlemektir. Belirsizliklerin az, iş tanımlarının belirgin olduğu bilgi teknolojileri projeleri için uygun bir modeldir. Model kullanıcının projeye katkısını arttırmaktadır. Bilgi teknolojileri projesinin iki aşamalı olarak ihale edilmesi için oldukça uygundur. İlk ihalede kullanıcı modeli hedeflenerek, iş analizi ve kabul sınamalarının tanımları yapılmakta; ikinci ihalede ise ilkinde elde edilmiş olan kullanıcı modeli tasarlanıp gerçeklenmektedir.

Avantajları; Basit ve kullanımı kolaydır. Planlama ve test tasarımı gibi test faaliyetleri kodlamadan önce gerçekleştirildiği için proje içerisinde çok zaman kazandırır. Bu nedenle çağlayan yaşam-döngü modeline göre daha yüksek başarı şansı vardır. Hataların bulunması erken aşamada bulunur. Böylece hataların bir sonraki aşamaya geçmesi önlenir. Doğrulama (Verification) ve geçerleme (onaylama-Validation) sadece son üründe değil tüm teslim edilebilir ürünlerde uygulanır. Proje yönetimi tarafında takibi kolaydır.

Dezavantajları; Uygulama şekli oldukça katı, kesin kurallara bağlıdır. Yazılım çağlayan yaşam-döngü modelinde olduğu gibi geliştirme aşamasında geliştirilir bu nedenle yazılımın erken prototipleri üretilemez. Yazılım da diğer sistemler gibi zamanla evrimleşir. Aşamalar arasında tekrarlamaları kullanmaz. Herhangi bir aşamada gereksinimler üzerinde değişiklik olursa test belgelerinin de diğer belgeler ile birlikte güncellenmelidir. Aynı zamanda gerçekleştirilebilecek olaylara kolay imkan tanımaz. Geliştirme devam ettikçe iş ve ürün gereksinimleri de değişkenlik gösterebilir. Risk çözümleme ile ilgili aktiviteleri içermez. Son ürüne ulaşma düz bir çizgi ile ifade edilemez. V süreç modeli, gereksinimlerin açıkça tanımlandığı projeler için kullanılabilir.

* Helezonik (Spiral) Model

Sarmal modeli aynı safhalara geri dönülmesinin bir zorunluluk olduğunu vurgular. Planlama, risk analizi, üretim ve kullanıcı değerlendirmesi olarak 4 ana aşamadan oluşur. Ve bir spiral oluşturacak şekilde dönerek bu aşamalar tekrar eder. Spiral üzerindeki her bir halka bir fazı gösterir. Sarmal modeli çağlayan yaşam-döngü modelinde yok sayılan riskleri göz önünde bulundurur. Risk Analizi Olgusu ön plana çıkmıştır. Risk analizi ön planda olduğu için hataları erken giderme imkanı tanıyabilir. Proje çevrimlere ayrılır ve her bir çevrimin riskleri ayrı ayrı ele alınır. Prototip kullanımı görülür. Prototip kullanımı da her aşamada olduğu için kullanıcı da her aşamada yazılım projesinin bir parçasını görme imkanına sahip olur bu da sorunların azalmasını sağlayabilir. Bazı projeler için maliyeti fazladır. Çünkü devamlı olarak prototip çıkarmak ve belgeleme yapmak gereklidir. Hedefler, alternatifler ve kısıtlamalar belirlenir. Alternatifler değerlendirilir, riskler belirlenip açık bir şekilde çözülür. Aşamanın ürünü geliştirilir ve sonraki aşama planlanır. Doğrudan tanımlama, tasarım vs. gibi kesin bir aşama yoktur. Spiraldeki halkalar neye ihtiyaç varsa onu gerçekleştirmek için seçilir. Yinelemeli artımsal bir yaklaşım bulunur.​​ Çağdaş modellere son derece yakındır. 4 ana aşaması vardır. Aşamalar; Planlama, Risk Analizi, Üretim ve Kullanıcı değerlendirmesinden oluşur. Planlama; üretilecek ana ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile birleştirmedir. Risk analizi, risk seçeneklerinin araştırılması ve risklerin belirlenmesidir. Üretim, ara ürünün üretilmesidir. Kullanıcı değerlendirmesi ise ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmelerdir.

Avantajları: Kullanıcılar sistemi erken görebilmektedir. Geliştirme pek çok küçük parçalara bölünür ve en riskli kısımlar önce gerçekleştirilir. Birçok yazılım modelini içinde bulundurur. Riske duyarlı yaklaşımı potansiyel zorlukları engellemektedir. Seçeneklere erken ve dikkatle odaklanır. Yazılım-donanım sistemi geliştirme için bir çerçeve sağlar.

Dezavantajları: Küçük ve düşük riskli projeler için pahalı bir yöntemdir. Komplekstir(karmaşıktır). Spiral sonsuza gidebilir. Ara adımların fazlalığı nedeniyle çok fazla dokümantasyon gerektirir. Spiral döngü modeli, yazılımın içten geliştirileceğini varsaydığından ve kontrat tabanlı yazılımlar adım adım anlaşma esnekliğini sağlamadığından kontrat tabanlı yazılıma uymaz. Öznel risk değerlendirme deneyimine dayanır. Yani yüksek riskli öğelere yoğunlaşmak, yüksek riskli öğelerin doğru belirlenmesini gerektirir.

* Artımsal Geliştirme Süreç Modeli

Eğer bir müşterinin ürünlerinde değişikliğe ihtiyaçları varsa, artımlı model ihtiyaç olan bu değişikliğe ayak uydurur. Artımsal model, sistemi tek parça teslim etmek yerine bir takvime bağlı olarak parça parça geliştirip teslim etmeye dayanır. Projede kullanıcının önceliğine göre ayrılan parçalar sıralanır. Yani kullanıcı gereksinimleri önceliklendirilir ve öncelikli gereksinimler erken teslimlere dahil edilir. Sıralanan bu parçalar bittiğinde sırasıyla birer ara ürün geliştirilmiş olur ve bu ara ürünler de kullanıcı tarafından kullanılır. Ara ürünler her seferinde bir öncekinin üstüne bir şeyler katarak çıkartılır. Her bir yeni parça, önceki üretilen her yazılım sürümü birbirini kapsayacak şekilde üstüne bazı ek işlevlerin eklenmesini öngörür. Örnek olarak öğrencilerin bir dönem boyunca geliştirmeleri gereken bir programlama ödevinin 2 haftada bir gelişiminin izlenmesi (bitirme tezleri) verilebilir. Artımsal model yazılım geliştirmenin kısıtlı sayıda çalışanla işin yapılmasını sağlama gibi bir üstünlüğü vardır. Ayrıca çalışanlar da her artırım geçildiğinde uygulama alanına ilişkin daha çok deneyim kazanmış olurlar. Bu modelde bir taraftan üretim bir taraftan da kullanım yapılır.​​ Önceki modellerde ürünlerdeki değişiklikler göz önünde bulundurulmaz. Bu model doğal olarak yinelemelidir.​​ Yeniden kullanılabilir bir ürün,​​ fonksiyonellik sağlamış bir şekilde tüm döngülerin sonunda ortaya çıkar. Artımlı geliştirme modeli uzun zaman alabilecek ve ürünün eksik işlevsellikle çalışabileceği türdeki yazılımlar için uygundur. Bu model ile sistemin başarısız olma olasılığı azalır, ara ürünler yazılımın geliştirilmesinde önemli bir yere sahip olur.

Avantajları; Sistem için gerekli olan gereksinimler müşterilerle belirlenir. Gereksinimlerin önemine göre teslim edilecek artımlar belirlenir. Öncelikle en önemli gereksinimleri karşılayan çekirdek bir sistem geliştirilir. Erken artımlar prototip gibi davranarak, gereksinimlerin daha iyi anlaşılmasını sağlar. Tüm projenin başarısız olma riskini azaltır. En önemli sistem özellikleri daha fazla sınanma (test edilme) imkanı bulmuş olur. Divide and Conquer (böl ve yönet) yaklaşımıdır.

Dezavantajları; Artımları tanımlamak için tüm sistemin tanımlanmasına ihtiyaç vardır. Gereksinimleri doğru boyuttaki artımlara atamak bazen zor olabilir. Deneyimli personel gerektirir. Artımların kendi içlerinde tekrarlamalara izin vermez. Yapılan işlerden edinilecek sonuçlar belli değildir. Geliştirilen yazılımlar genellikle sınırlı sayıda kullanılır ve kullanımı bittikten sonra işe yaramaz hale gelir ve atılır. Model-zaman fiyat kestirimi belli olmadığı için sabit fiyat sözleşmelerinde uygun değildir.

**BİRLEŞİK SÜREÇ (UNİFİED PROCESS)**

Nesne tabanlı (OOP) yazılım geliştirmek için var olan yöntemlerin kullanılmasıyla edinilen deneyimlerle bu süreçlerin en iyi özellikleri bir araya getirilerek bütünleştirilmiş bir yazılım geliştirme sürecidir. Yinelemeli, arttırmalı ve evrimsel aynı zamanda da risk güdümlüdür. Yinelemeli (Iterative): Problemdeki istekler bir bütün olarak görülmez. Problemdeki hedeflerin bir kısmı ele alınır ve sınanmış bir ürün ortaya konulur. Her ürün oluşumunun sonunda bir sonraki yinelemeye (iterasyon) geçilir ve yeni istekler ele alınarak geliştirilir. Her iterasyon sonunda hedeflenen ürüne yaklaşılır. Her bir iterasyona ayrı bir proje gibi bakılır ve tüm aşamalardan geçirilerek sınanmış ve çalışır bir ürün elde edilir. Artırmalı ve Evrimsel (Incremental and Evolutionary): Her yinelemenin ardından farklı ve yeni istekler ele alındığı için elde edilen ürünlerin özellikleri artarak gelişir ve istenilen ürüne daha fazla yaklaşılmış olunur. Risk Güdümlü ( Risk-Driven): İlk iterasyonda en riskli kısımlar ele alınmalıdır. Böylece proje daha ilerlemeden oluşabilecek büyük problemler görülür ve gerekli önlemler alınabilir. Örneğin bütçe, personel miktarı veya zaman planı gibi değişkenler ayarlanabilir. Birleşik süreçte yazılım geliştirme aşamaları bulunur. İlk aşama başlangıç: Vizyon kararı, fizibilite çalışması, tamam ya da devam kararı verilir. Ayrıntılandırma: Daha gerçekçi çözümleme, çekirdek yapının ve yüksek riskli kısımların yinelemeli olarak oluşturulmasıdır. Tamamlama: Daha az riskli ve düşük öncelikli kısımların yinelemeli olarak gerçeklenmesidir. Son olarak Yayım: Beta testleri, piyasaya sürme çalışmalarını kapsamaktadır.

Avantajları; Değişen isteklere uygundur. Erken geri besleme vardır. Büyük sistemlerde çözme kolaylığı sağlar. Her iterasyonda deneyim kazanılır. Riskler erken giderilir. Erken ürün elde edilebilir bu da takımda moral yükselmesine katkı sağlar.

Dezavantajları; Risk yönetimi zayıftır. Dokümantasyon yükü ağır olduğundan pahalı olabilir. Karmaşıktır.

**ÇEVİK YAZILIM GELİŞTİRME SÜRECİ (AGİLE PROGRAMMİNG)**

Yenilemeli (iterative) geliştirmeyi temel alan bazı yazılım geliştirme metodolojilerine dayanarak harmanlanmış, yenilikçi bir yazılım geliştirme süreçleri topluluğudur. “Heavyweight” olarak da bilinen eski, yavaş ve bürokratik sistemlere tepki olarak geliştirilmiştir. Burada gereksinimler ve çözümler kendinden örgütlü olan farklı grupların ortak çalışmasıyla olgunlaşır. Çevik yazılım metotları genellikle takım çalışmalarının, sık denetimlerin, düzenli ürün sunmanın ve müşteri ihtiyaçlarıyla firmanın yeteneklerinin uyumlu bir şekilde bir araya gelerek hızlı ve kaliteli bir biçimde ortaya ürün koyan süreçlerdir. “İlerlemenin en iyi göstergesi, çalışan yazılımdır.” prensibiyle devamlı surette müşteriye bir ürün sunulur. Müşteri ihtiyaçlarını ve firma amaçlarını yan yana getiren business yaklaşımı desteklemektedir.​​ Bilinmelidir ki çevik yazılım metodları bir yazılım geliştirme yaklaşımı değil, geliştirme süreçleri topluluğudur.​​ Takımların kendi kendilerine organize olmaları önemlidir. Bunun için düzenli aralıklarla, takımlar kendilerini değerlendirerek nasıl daha etkili olabileceklerine bakar, çalışmalarını buna göre düzenler ve devam ederler.

* Çevik Manifesto

2001 yılının Şubat ayında yazılım dünyasının önde gelen 17 ismi Utah’da bir araya gelerek yazılım geliştirme üretkenliğini arttırmaya yönelik 2 günlük beyin fırtınası yapmışlar ve “Çevik Yazılım Geliştirme Manifestosu ve Çevik Yazılımın Prensipleri”ni yayınlamışlardır. Yazılım geliştirme amacıyla üretilen bu modelleme biçimi, kapsadığı değerler, prensipler ve pratikler sayesinde geleneksel modelleme metotlarına göre yazılımlara daha esnek ve kullanışlı biçimde uygulanabilmektedir. Çevik Manifesto, çevik yaklaşımını yazılım geliştirmeye yönlendiren 4 temel değerden ve 12 prensipten oluşur. Bu 4 ana maddeye göre; Süreçler ve araçlardan ziyade bireyler ve etkileşimlerin, kapsamlı dokümantasyondan ziyade çalışan yazılımın, sözleşme pazarlıklarından ziyade müşteri ile işbirliğinin, bir plana bağlı kalmaktan ziyade değişime karşılık vermenin daha çok önemli ve öncelikli olduğuna kanaat getirilmiştir. Her çevik yazılım geliştirme metodolojisi dört değeri farklı şekillerde uygular, ancak hepsi yüksek kaliteli, çalışan yazılımların geliştirilmesi ve sunulması için onlara rehberlik eden unsurlardır. Çevik Manifesto’ da ifade edilen 4 değeri destekleyici 12 prensip ise şunlardır: En önemli öncelikleri değerli yazılımın erken ve devamlı teslimini sağlayarak müşterileri memnun etmektir. Değişen gereksinimler yazılım sürecinin son aşamalarında bile kabul edilmelidir. Çevik süreçler değişimi müşterinin rekabet avantajı için kullanmalıdır. Çalışan yazılım, tercihen kısa zaman aralıkları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunulmalıdır. İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdırlar. Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara ihtiyaçları olan ortam ve destek sağlanmalı, işi başaracakları konusunda güven duyulmalıdır. Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüz yüze iletişimdir. Çalışan yazılım ilerlemenin birincil ölçüsüdür. Çevik süreçler sürdürülebilir geliştirmeyi teşvik etmektedir. Sponsorlar, yazılımcılar ve kullanıcılar sabit tempoyu sürekli devam ettirebilmelidir. Teknik mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği arttırmaktadır. Sadelik, yapılmasına gerek olmayan işlerin mümkün olduğunca arttırılması sanatı, olmazsa olmazlardandır. En iyi mimariler, gereksinimler ve tasarımlar kendi kendini örgütleyen takımlardan ortaya çıkar. Takım, düzenli aralıklarla nasıl daha etkili ve verimli olabileceğinin üzerinde düşünür ve davranışlarını buna göre ayarlar ve düzenler.

Avantajları; İnsanın doğal eğilimine çok yatkındır, öğrenim gerektirmez ve adaptasyon sağlanması hızlıdır. Süreç ve araçlardan ziyade insanlar ve etkileşimler daha değerli olduğundan müşteriler, geliştiriciler ve test edenler sürekli olarak birbirleriyle etkileşim içindedirler. Kısa döngüler dolayısı ile takım elemanlarında motivasyon çok yüksektir. Verim artışı yaşanır. Sık çıktı üretip geri besleme aldığından kaynağı müşteri ihtiyaçlarına ve sonuca kanalize etmeye odaklanır. Plan aşamasında ayrıntılı plan yerine iterasyonun planı yapılır. Değişime açıklık ve esneklik en üst düzeydedir. Sürdürülebilir kalite sağlar. Proje planlama ve yürütme bir aradadır. Takım çalışmasını destekler.

Dezavantajları; Kurumsal bir yapıda uygulaması gerçekten zordur. Dokümantasyon hakkında ki taşları yerinden oynatan yaklaşım mevcuttur. Sürekli değişen ihtiyaçlar dolayısı ile aşırı çalışmalara yol açar. Ürünün başarısı ve projenin başarısı eşitliği hainde kariyer riski bulunur. Takım üzerindeki hedef baskısı fazladır.

Yazılım Projelerinde Yaygın Uygulanan Çevik Metodolojiler

* Extreme Programming (XP)
* Rational Unified Process
* Feature-Driven Development (FDD)
* Test-Driven Development (TDD)
* LEAN Development
* Dynamic System Development Methodology (DSDM)
* Microsoft Solution Framework (MSF)
* SCRUM

**Extreme Programming (XP)**

Kent Beck ve arkadaşları tarafından 1996 yılında kurulmuştur. Tüm gereksinimlerin senaryolar şeklinde oluşturulduktan sonra bu senaryoların işlere bölünerek küçük gruplara paylaştırıldığı bir yöntemdir. Yazılımcılar çiftler halinde çalışmaktadır. Takım içi görev paylaşımı yoktur. Herkes tasarım, kodlama ve test aşamalarında görev alır. Bir müşteri temsilcisi de geliştirici takımlarla beraber çalışır. Aynı iş üzerinde 2 haftadan fazla çalışılmaz. Kullanıcı gereksinimleri değişken veya tam olarak net olmadığı durumlarda kullanışlıdır. Daha çok küçük ve orta ölçekli projelerde kullanılmaktadır. 4 temel maddeden oluşur: Basitlik, Cesaret, Geri Dönüş, İletişim. Basitlik; Yazılan kodun ve yapılan işin sade, anlaşılır ve karmaşık olmadan yapılmasını gerektirir. Uzun uzun dokümantasyondan uzak durulmasıdır. Ancak basitliği sağlamak zordur; çünkü basitleştirmek isterken ayrıntı gibi görünen önemli hususları atlamak, başka problemlere sebebiyet verebilir. Cesaret; 4 temel değer arasında en zorudur çünkü başarısızlık üzerine korkmadan gitmek, başarısızlığın sonucuna değil sebebine ve telafisine yoğunlaşılması gerekir. Başarısızlık durumunda yazılımcılar hız kesmeden devam etmeli, gerekirse tamamen silip baştan başlayabilmelidirler. Geri Dönüş; Projenin başarıya ulaşması için vazgeçilmez bir ihtiyaçtır. Gerek yazılımcıların yaptıkları birim testleriyle, gerekse müşteriye sunulan sürümlerle sağlanır. Müşteri ile yazılım ekibi birbirleriyle iletişim halinde bulunur. Geri dönüşler ile oluşabilecek hatalar azaltılır veya ortadan kaldırılır. İletişim; İletişim, projelerde önemli sorunlardan birisidir. Proje sırasında oluşabilecek aksamaları önlemede önemli rol oynar. XP iletişimde oluşacak sıkıntıları aşmaya çalışmaktadır. Ekip içi iletişime önem verir ve artırılması için çalışır. Ekip içi iletişim ve yazılımcı ile kullanıcı arasındaki iletişim devamlı ve yüz yüze olmalıdır. Bu iletişimler hızlı olursa projenin de aksama ihtimali ortadan kalkmış olur.

Yazılım geliştirmede basitliği ve verimi sağlayabilmek için XP 12 pratik değeri göz önüne alır. Bunlar; planlama oyunu, ayakta toplantı, ekipte müşteri, kısa aralıklarla yeni sürüm, geriye bakış, metafor, ortak sorumluluk, sürekli entegrasyon, kod standartları, kalıcı tempo, test etme, sade tasarım, yeniden yapılandırma ve eşli programlamadır.

**SCRUM**

Scrum, kelime olarak rugby oyununda oluşturulan küçük ekiplere verilen isimdir. Bu yöntem 1990’lı yılların ortalarında Jeff Sutjerland ve Ken Schawaber tarafından oluşturulmuş ve günümüze sürekli bir gelişme halindedir ve çokça kullanılmaktadır. Bu model, gereksinimleri açıkça belli olmayan, değişime açık, karmaşık yazılım projelerin yönetimi için uygulanması en ideal yöntemdir. Scrum ile yazılım geliştirme, genellikle bir metodoloji olarak algılanır; ancak Scrum’ı bir metodoloji olarak görmek yerine, bir süreci yönetmek için bir çerçeve olarak düşünmek daha doğru olacaktır. Çoğu zaman kompleks yazılım süreçlerinin yönetilmesi için kullanılır. Bunu yaparken bütünü parçalayan, tekrara dayalı bir yöntem izler. Düzenli geri bildirim ve planlamalarla hedefe ulaşmayı sağlar. Bu anlamda ihtiyaca yönelik ve esnek ama üretken bir işleyiş sunmaktadır. Müşteri ihtiyacına göre şekillendiği için müşterinin geri bildirimine göre yapılanmayı sağlar. En önemli özelliği ise müşteri ve geliştirici doğrudan temas halinde olmadıkları için yazılım geliştirici baskı altında olmaz, yapması gereken işe yoğunlaşır. Scrum’ın sağladığı faydalardan biri de, işleyiş sürecini şeffaf hale getirerek aksaklıkları ortaya çıkarır. Böylelikle proje ekibinin hataları fark ederek sürekli olarak iyileştirme yapmalarına olanak sağlar. Bu sebeplerden iletişim ve takım çalışması çok önemlidir. Başarısı kanıtlanmış bir yazılım geliştirme yöntemi olan Microsoft, Google, Facebook gibi şirketler tarafından kullanılmaktadır.

Scrum büyük projeleri parçalara bölerek her birine “Sprint(koşma)” adını verir. Scrum içerisindeki tüm aktiviteler sprint içerisinde gerçekleşir. Her bir sprint teker teker geliştirilir. 1–2 haftalık süreçlerdir. Karşılayabileceğimiz diğer bir kavram Scrum Board’dır. Bir sprint içerisinde yapılacak olan maddeler burada yönetilir. Yapılacak olan görevler “TO DO” bölümüne alınır. Takım üyesi bu işe başladığında “IN PROGRESS” bölümüne getirilir. Bir iş, test için hazırsa “TO VERIFY” durumuna getirilir. İş, kontrol edildikten sonra “DONE” bölümüne getirilir. Scrum toplantılarında bu maddeler durumlarına göre yerleri değiştirilir. Scrum’da ekip içi uyum sandığımızdan daha çok önemlidir öyle ki her gün “Scrum Meetings” denen toplantılar yapılır. 3 temel prensip üzerine kurulmuştur. Bunlar şeffaflık, denetleme ve uyarlamadır. Şeffaflık; projenin ilerleyişinin, sorunların, gelişmelerin herkes tarafından görülebilir olmasıdır. Denetleme, projenin ilerleyişinin düzenli olarak kontrol edilmesi demektir. Uyarlama ise projenin yapılabilecek değişikliklere uyum sağlayabilmesidir.

Scrum geliştirmedeki birincil yapı elbette ürünün kendisidir. Scrum modeli, ekibin ürünü veya sistemi her scrum sprintinin sonunda potansiyel olarak sevk edilebilir bir duruma getirmesini bekler. Ürün biriktirme listesi, scrum’ın başka bir eseridir. Bu, ürüne eklenmek üzere kalan işlevselliğin tam listesidir. Ürün sahibi, iş yığınına öncelik verir, böylece ekip her zaman önce en değerli özellikler üzerinde çalışır. Scrum metodolojisini kullanarak bir ürün birikimi oluşturmanın en popüler ve başarılı yolu, onu bir kullanıcı veya müşteri perspektifinden açıklanan işlevselliğin kısa açıklamaları olan kullanıcı hikayeleri ile doldurmaktır. Scrum proje yönetiminde, bir sprintin ilk gününde ve planlama toplantısı sırasında ekip üyeleri sprint iş yığını oluşturur. Sprint iş yığını, takımın sprint için yapılacaklar listesi olarak düşünülebilirken, bir ürün iş yığını, oluşturulacak özelliklerin bir listesidir (Kullanıcı hikayeleri şeklinde yazılır). Sprint iş yığını, sprint sırasında sunmayı taahhüt ettiği işlevselliği sağlamak için takımın gerçekleştirmesi gereken görevlerin listesidir. Burndown çizelgeleri, bir sprint veya bir sürümde kalan çalışma miktarını gösterir ve bir sprint veya sürümün planlanan tüm işlerin istenen tarihte bitirilmesi için programa uygun olup olmadığını belirlemek için scrum yazılım geliştirmede etkili bir araçtır. Ayrıca scrum’da 3 temel kavram bulunur. Bunlar roller, toplantılar ve bileşenler/araçlardır.

**Scrum’da Temel Kavramlar**

* Roller

-Pig Roller; Scrum sürecine dahil olanlar yani projede asıl işi yapan kişilerdir. Bunlar Scrum Master, Product Owner, Geliştirme Takımı’dır.

1) Product Owner; Geliştirme takımı ve müşteri arasındaki iletişimi sağlar. Projenin özelliklerini tanımlar. Projenin önceliklerine göre product backlogu oluşturur. Sprint’i iptal yetkisine sahiptir. Sprint neden iptal edilmek istenebilir? Hızla değişen ortamlarda bir sprinte alınan işlerin iş birimi için önemi kalmamış olabilir ya da sprinte alınan işlerden daha önemli işler ortaya çıkabilir. İş sahibi bunu görüp sprinti iptal etmek isteyebilir.

2) Scrum Master; Scrum kurallarını, teorilerini ve pratiklerini iyi bilir ve takımın bu kurallarını uygulamasından sorumlu kişidir. Takımın yöneticisi değildir. Takımı rahatsız eden, verimli çalışmalarını engelleyen durumları ortadan kaldırır.

3) Geliştirme Takımı; Bir Sprint’e alınan bütün işleri tamamlayacak özelliklere sahip kişilerdir. sprint backlogu oluştururlar. Kendi kendini yönetir. İşin verilmesini beklemezler, işi kendileri alır ve geliştirirler. Kişilerin tek bir görevi yoktur, çapraz görev dağılımı yaparlar, herkes her şeyi yapabilir konumdadır. 5–7 kişi arasında değişir. Projenin geliştirilmesi ile ilgili sorumluluk geliştirme takımına aittir.

-Chicken Roller; Scrum’ın işleyişinde aktif olarak yer almayan kişilerdir. Müşteriler, satıcılar gibi örnekler verilebilir.

* Toplantılar

1) Sprint Planning; Product backlog ile belirtilen gereksinimler, bu toplantı ile geliştirme takımı tarafından küçük görevlere (task) ayrılır. Takımdaki her bir kişi kendi hızına göre bu taskleri kendilerine alır. Bu toplantıya product owner, geliştirme takımı ve scrum master katılır. Sprintler; her sprint sonunda product owner a sunulmak üzere yazılım geliştirmeyi hedefleyecek şekilde belirlenir.1–3 haftalık sprintler oluşturulur.

2) Daily Scrum; Her gün aynı yerde aynı saatte ayak üstü yapılan 15 dakikalık toplantılardır. Üyeler davet edilmeyi beklemezler. Bu toplantı gelecek 24 saati planlamak üzere yapılır. Takımdaki her üye dün ne yaptım, bugün ne yapacağım, işimi engelleyen herhangi bir sorun var mı sorularına cevap verir. Bu sayede herhangi bir sorunu var ise scrum master bu problemi ortadan kaldırır. Takım üyelerinden bu probleme yardımcı olabilecek biri var ise toplantı sonunda iletişime geçebilirler. Daily scrum her ne koşulda olursa olsun yapılır. Takımdaki birinin geç kalması veya gelmemesi toplantıyı etkilemez. Sadece takımdaki büyük çoğunluk yok ise toplantı yapılmaz.

3)Sprint Review; Her sprint sonunda yapılır. Yapılan sprint gözden geçirilir, ortaya çıkan ürün değerlendirilir. Amaç yazılımın ürün sahibinin gereksinimlerine uygun olarak geliştirildiğinden emin olmaktır. Eğer bir hata var ise fark edilir ve düzeltilir.

4)Sprint Retrospective; Sprint boyunca yapılan işlerin kalitesinin, doğruların ve yanlışların değerlendirildiği toplantıdır. Bu toplantı scrum takımının kendini geliştirebilmesi için bir fırsattır. “Neleri daha iyi yapabiliriz?”, “Nasıl daha iyi yapabiliriz?” sorularına cevap aranır. Bu aşamadan sonra bir sonraki sprint planning toplantısı gerçekleştirilerek yazdıklarımızın hepsi tekrardan yaşanır.

* Bileşenler/Araçlar (Artifacts)

1)Product Backlog; Proje için gerekli olan gereksinimler listesidir. Liste genellikle kullanıcı hikayelerinden oluşur ve kullanıcı bakış açısından ele alınır. Proje sonunda “Ne üretilmek isteniyor?” sorusuna cevap aranır. Product owner tarafından müşteriden gereksinimler alınır, öncelik sırasına göre sıralanır. Product owner, değişen ihtiyaçlara göre product backlog’a ekleme veya çıkarma yapabilir. Böylece değişim, projenin her aşamasında projeye kolayca entegre edilebilir olur. Ayrıca Product Backlog Item, product backlog içindeki her bir gereksinime verilen isimdir.

2)Sprint Backlog; Geliştirme takımı tarafından product backlog itemlar öncelik sırasına göre sprint içerisine alınırlar. Bir sprint boyunca yapılacak itemların listesini oluşturur. İşlerin detaylı olarak zaman çizelgesi çıkarılır.

3)Burndown Chart: Yatay ekseninde sprintin günlerini, dikey ekseninde sprintte kalan işi gösteren grafiktir. Scrum’un temel ilkelerinden olan şeffaflığı sağlar.

**Scrum Neden Bu Kadar Kullanılıyor?**

Scrum günümüzde en çok kullanılan yazılım geliştirme yöntemidir. Hatta sadece yazılım geliştirmede değil birçok sistemin geliştirilmesinde de kullanılır. Bu kadar popüler olmasının nedenleri ise; Zamandan ve paradan büyük ölçekte tasarruf edilmesi, yüksek teknolojiler ve son gelişmelere kolaylıkla uyum sağlayabilmesi, karmaşık görülen ve gereksinimleri tam belirlenmemiş projeler için ideal olması, ekip içi iletişimin yüksek tutulması ve bununla beraber hataların erken fark edilip düzeltilmesi, kullanıcıdan sürekli geri bildirim gerektirmesi ve bununla beraber sorunların azalması, diğer yazılım geliştirme metodolojileri gibi yinelemeli olması, değişen gereksinimlere hızlı bir şekilde tepki vermesi, böl ve fethet yaklaşımı içerisinde olması şeklinde sıralanabilir.

**SONUÇ**

İçinde bulunduğumuz teknoloji çağı gereği aslında yazılım her alanımızda mevcuttur. Şu an bu yazıyı okuduğunuz bilgisayar veya telefon, televizyon, buzdolabı, fırın, iş makineleri, röntgen cihazları… Aklınıza gelebilecek, elimizin altında bulunan hemen hemen her şey bir yazılımdan ibarettir. Bununla beraber insanların yazılıma olan ilgisi ve kullanımı gün geçtikçe artmaktadır. Yazılım kullanım sürecinde ise bize yardımcı olabilecek faktörlerden biri yazılım geliştirme modelleridir. Yazılım geliştirme her ne kadar kod parçası ve bunun bir uygulama olarak kullanılması gibi gözükse de aslında içerisinde bundan fazlasını barındırır. Geleneksel olan modeller veya modern modeller bize yol gösteren, yazılım projelerinin çözümlenmesini ve ürünün ortaya çıkmasını kolaylaştıran yardımcı yollardır. Bu aşamada önemli olan yazılımın doğru analiz edilmesidir. Geliştirilecek yazılımın özelliklerine ve sürecine, fazlarının hangi uzunlukta olduğuna, hangi öncelikte ve hangi tekrarla yapılacağına, oluşabilecek riskleri tolere edebilme kapasitesine, projenin büyüklüğüne ve karmaşıklığına, gerçekleştirecek kurumun yapısına, zaman ve maliyet gibi kriterlere bakılmalıdır. Bundan sonra hangi modelin yazılıma uygun olduğu tartışılmalı ve kararlaştırılan model yazılımda kullanılmalıdır. Ayrıca günümüzde en çok kullanılan​​ model Çevik Modeldir. Kısa sürede, esnek bir şekilde geliştirilerek, yüksek başarımla sonuçlanan bir model olduğundan en çok tercih edilen model olması muhtemeldir.​​

YAZILIM GELİŞTİRME MODELLERİNİN KARŞILAŞTIRMASI

Gelişigüzel ve barok modelleri artık kullanımını yitirmiştir bunun nedenleri yinelemeli olmamaları, dokümantasyonu işin içinde barındırmamaları gibi sıralanabilmektedir. Kodla ve düzelt modeli küçük programlarda kullanılabilir. Bakım safhası oldukça zordur, emeklilik safhası bulunur; büyük projeler için kullanılamaz çok maliyetlidir, genelde tek bir kişinin isteğine göre program en son halini alana kadar sadece kodlama yapılır ve teslim edilir. Dokümantasyon gibi olaylar yoktur o yüzden kullanımı sadece kişiye özel ile sınırlıdır. Çağlayan Modelinin kullanılması ve uygulanması basit, iş bölümü ve iş planlaması detaylıdır. Küçük ve özellikleri iyi belirlenmiş projelerde kullanılabilir. Eskiden yaygın olarak kullanılmıştır ancak şimdi kullanıcı ile iletişimin az olması, büyük projelerde yetersiz kalması, değişimlere kapalı olması gibi sorunlar yüzünden kullanımı çok düşüktür. V Modeli, Çağlayan modelinin gelişmiş versiyonudur. Kullanımı ve takibinin kolay olması, onaylama ve doğrulama işlemlerinin planın erken aşamasında uygulanabilir olması avantajlıdır ancak fazlar arası tekrarlamalara imkan tanımaması, risk çözümleme basamaklarının bulunmaması gibi sorunlar bulundurur. Spiral modelde kullanıcılar ara ürünleri gördükleri için sistemin içinde sayılırlar. Risk analizi çok önemli bir yerdedir bu yüzden büyük projeler için idealdir. Hatalar erken giderilir. Planlama büyük bir parçayı oluşturur ancak çok karmaşık olması, çok uzun sürmesi, dokümantasyonun çok fazla tutulması gibi nedenlerle çok tercih edilmemektedir. Artımsal Geliştirme modelinde her teslimde müşterinin geri dönütü alındığı için işlevsellik erken ortaya çıkar ve proje daha güvenli bir hal alır. Gereksinimler müşteri ile belirlenir. Özellikler daha fazla test edilmiş olur ancak her artımı tanımlayabilmek için sistemin detaylı bir şekilde tanımlanması lazımdır. Deneyim gerektirir ve artımlar kendi içlerinde tekrarlanamaz. Çevik modellerde ise müşteri ile iletişim ve ekip içi iletişim yüksek olduğu için hatalar minimuma indirilir. Alışılması basittir ayrıca değişime açık ve esnektir. Günümüzde çokça kullanılan ve kullanılmaya devam edilebilecek bir modeldir. Takım önemli bir konumdadır ancak dokümantasyon çok detaylı olmadığı için alışılagelmişin dışındadır. İhtiyaçlar sürekli değiştiğinden çalışma saatleri de fazla olabilir.

HANGİ MODEL HANGİ PROJEDE KULLANILMALIDIR?

Kişiye özel, zaman sorunu olmayan, küçük programlarda kodla ve düzelt kullanılabilir. Küçük ve özellikleri iyi tanımlanmış projelerde Çağlayan modeli kullanılabilir. Belirsizliklerin az olduğu, iş tanımlarının belirgin olduğu bilgi teknolojileri projeleri için V modeli uygun bir modeldir. Büyük, maliyetli ve uzun süren projelerde spiral model veya artımsal geliştirme modeli uygundur. Orta ve küçük büyüklükte, uzun sürmeyen projelerde çevik modeller uygundur.

**REFERANSLAR**

* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* <https://slideplayer.biz.tr/slide/12386328/>
* <https://hayririzacimen.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-s%C3%BCre%C3%A7-modelleri-70fdfb2f8f77>
* <https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742>
* <https://medium.com/@brfn.kcr26/yazilim-geli%CC%87%C5%9Fti%CC%87rme-ve-s%C3%BCre%C3%A7-modelleri%CC%87-2131ea5f09b2>
* <https://osmanozaydin.com/yazilim-yasam-dongusu-ve-agile-yazilim-gelistirme/>
* <https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8#:~:text=Scrum%3B%20Agile%20proje%20y%C3%B6netim%20metodolojilerinden,ve%20planlamalarla%20hedefe%20ula%C5%9Fmay%C4%B1%20sa%C4%9Flar>.
* <https://www.yazilimtestmerkezi.com/post/yazilim-gelistirme-modeli-nedir-cesitleri-nelerdir>
* <https://iskulubu.com/yazilim/yazilim-gelistirme-yasam-dongusu/>
* <https://medium.com/architectural-patterns/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-modelleri-62915545c51e>
* <https://www.codex.com.tr/yazilim-gelistirme-modelleri>
* <http://www.ilkimdilara.com/agile-manifesto/>
* <https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8>
* bil-102-2\_hafta-yazilim-yasam-dongu-modelleri ders notları
* biÌ\_l-102-3\_hafta-cevik-yazilim-gelistirme ders notları
* <https://www.akademikkaynak.com/makale-nasil-yazilir.html>