**YAZILIM YAŞAM DÖNGÜSÜ**

Yazılım yaşam döngü modellerinden önce birkaç terim bilgisi bilmek gerekmektedir. Kısaca onlara değinecek olursak:

**“Yazılım Nedir?”**

Kısaca yazılımı açıklayacak olursak, bir olay, durum veya sorunu en kısa yoldan algoritmalar eşliğinde çözüp bir programlama dili ile bilgisayar ortamında yazmaktır.

**“Yaşam-Döngü Modeli Nedir?”**

Bir yazılımda, yazılımın gereksinimleri aşamasından uzun yıllar sürecek yazılımın bakım aşamasına kadar olan sürecin tamamına “yaşam-döngü modeli” denir. Yazılım döngüsünde her aşama belgelenir bu sayede yazılan kodlar kolayca birleştirilebilir, test aşaması daha az maliyete mal olur, çıkan sorunların çözümü daha kolay kontrol edilir ve bakım aşamasında başka bir yazılım geliştirici tarafından kodlar ve yapılan değişiklikler daha kolay anlaşılabilir.

Yazılım geliştirme müşteri bazlı düşünüldüğünde kolay gözükse de bu süreçte birçok problemle karşılaşılıyor. Mesela müşteri ne istediğini pek çok zaman bilmeyebilir ve istediği birçok şeyi kısa zamanda yazılım geliştiricilerinden tedarik etmek ister fakat işin arka yüzü yani yazılım geliştiriciler için istenilen bu şeyin bu kadar kısa zamanda halledilmesi imkansızı oynar. Bu gibi durumlarda yazılım geliştiriciler taktikli bir savunma yapmaları gerekmektedir. “İş çok uzar.” bu taktiklerden birisidir. Kimi zamanda müşterinin gereksinimi değişir oysaki satırlarca yazılan koddan o istenilen şeye ulaşmak imkânsız gibidir. Burada ise “Çalıştığım saat başına 20 tl alırım ve bu iş için 600 saat çalışmam gerekiyor.” denilmesi hem müşteri için fazla maliyetli olacağı için geri kaçmasını hem de kabul etse bile projeden kazanılan para yazılımcıyı memnun edecektir ne de olsa 600 saat sürmeyeceğini en iyi kendisi bilmektedir. Eğer ki bu iş ciddi bir zaman kaybına yol açacağını düşünüyorsanız “Bu değişim regresyon hatasına mal olur.” diyebilirsiniz aslında doğru bir cümledir çünkü sonradan yapılan değişiklikler başka yerlerde sorun çıkmasına sebep olduğu birçok örneği ile iş hayatınızda karşılaştığınızdan şüphem yok.

**“Modeller”**

**1.Gelişigüzel Model**

Modelin ismi aslında her şeyi açıklamaktadır. Gelişigüzel, belirli bir sistemi olmayan ve belirli bir yapıya girmemiş bir modeldir. Bu model en sıkıntılı modellerden birisidir çünkü bir kodun ömrü boyunca tek bir kişi ilgilenmez, bakımı birçok kişi tarafından yapılır fakat tek düze bir sistemi belirli çizgileri olmadığı için bir başkası için anlayabilmek, yazılım son aşaması olan bakım aşamasının bir başkası tarafından yapılması adeta zulümdür. Hatta çoğu zaman yazılım geliştiriciler bu sistemin tekrar yazılmasından yanadır çünkü bu zaman olarak ve maliyet olarak daha ekonomik olmaktadır. Bu tip sorunlardan ötürü genellikle yazılımın yaşam döngüsü boyunca sadece bir kişi çalışmaktadır buda sorunun geç giderilmesi ve maliyet olarak pazar maliyetinin dışında kişinin kendi belirlediği ücret (bu kimi zaman çok uçuk rakamları görmektedir) ile bakım ve onarımı gerçekleşir .Gelişigüzel model aslında bir çok yerde model olarak bile kabul edilmemektedir.60’lı yıllarda kullanılan ve yazılım geliştiriciler tarafından oluşturulan bu modelde bu tip sorunlarla karşılaşılması gayet normaldir , bu tip sorunlar algılanıp bir sonraki oluşturulacak yeni model ile giderilmesi ilerleme kat edildiğini göstermektedir.

**2.Barok Modeli**

70’lerden ortaya çıkan bu model, yazılım geliştiricileri belgeleme konusunda dikkatleri üstüne çekmiştir. Bu modelde belgeleme anlayışı projenin bütünü yerine yazılımın tasarımı, bakımı gibi bir parçası olarak görülmüştür. Bu durum ilk bakışta belgeleme yapılmadan ,yani teori evresi ile uğraşmadan direk uygulama evresine zaman ayırdığımızı düşündürebilir fakat işin gerçeğine bakıldığında durum hiçte o kadar iç açıcı değildir .Belgeleme aşaması projenin her aşamasında danışman olabilecek , yardım edebilecek, değişen bölümlerin değişmesinin nedenini belirtebilecek ve uzun süren yazılım geliştirme sürecinde yazılım geliştiricilere projenin başında neler yaptıklarını unuttukları bir gereksinim veya tasarım olup olmadığını hatırlatıcı görevi üstlenecek bir niteliğe sahiptir. Önemli bir konuda bu belgeleme işleminin en sonda yapılması demek uzun süren bu yazılım geliştirme sürecinden elbette bazı konuların kaçabileceğini akla getiriyor, bu da ileriki süreçte yani bakım sürecine girdiğinde sürekli bir tehdit oluşturmaktadır çünkü yazılım bakım süreci ile ilgilenen bireyin eksik bilgiler ile bakım yapması ve bu da hem onarımı zorlaştırması hem de kaza ile başka kötü sonuçlara da yol açabileceği anlamına gelmektedir. Günümüzde bu model çok az rastlanılmaktadır ve patronlar genel olarak bu modelden kaçmak istemektedir çünkü bu model ile geliştirilip müşteriye sunulan bazı projeler öyle bir safhaya gelmiştir ki proje maliyeti projeden kazanılan ücreti geçmektedir.

**3.Çağlayan Yaşam Modeli**

Çağlayan yaşam modeli, yazılım geliştiriciler tarafından geleneksek yazılım geliştirme modeli olarak da bilinir. Bunun sebebi aşamaların sırası ile; yazılımın gereksinimler, yazılımın tasarımı, kodlama, yazılımı test etme ve son olarak da yazılımın bakım aşaması gerçekleştirilir. Bu sıra günümüzde uygulanan modellere bir ışık niteliğindedir. Çağlayan yaşam modeli bilinen en eski, geçmişte en popüler ve tıpkı program yazım dillerinde C dili gibi yaşam döngü modellerinde bir temel niteliği oluşturmuştur. Bu modelde her aşama bir kez tekrar edilir, bir aşama bitmeden bir sonrakine geçiş kesinlikle yapılmaz, bu model belgeleme üzerine kurulmuştur eğer bir sayfanın dokümantasyonu eksikse o aşama sayılmaz tekrar o aşamaya dönülüp sıra ile alt aşamalar takip edilir. Buradan da anlaşılacağı üzere belgeleme ayrı bir parça olarak görülmemekte aksine tüm projeyi kapsamaktadır. Bu model genellikle kısa zamanlı projelerde uygun görülmektedir fakat bu projelerde gereğinden fazla açıklama yapılmalıdır örnek verecek olursak sistem gereksinimlerinden müşterinin tam olarak hangi işlemleri bu projeden beklediklerine kadar en ince ayrıntısına kadar değinilmesi gerekmektedir. Bu ayrıntı ve detayların sebebi ise projenin ileriki safhası olan kodlama ve yazılımın test aşamasında oluşabilecek bir maliyetin çok büyük rakamlara tekabül etmesi, geri dönmenin de samanlıkta iğne aramaktan farksız olduğundan geri dönmek düşünülemez. Tabi bunlara ek olarak yazılım süreci uzun bir süreci kapsadığı için müşterinin gereksinimleri değişebileceği ve müşterinin gerek istek gerekse sorunlarını düzgün dile getiremeyebileceği de öngörülerek hareket edilmektedir. Çağlayan yaşam modeli ne kadar temel niteliği taşıyan bir model olsa da günümüzde bu modelin popülerliği eskisi kadar değildir ve gittikçe de azalmaktadır. Yazılım geliştirici grubun belgeleme gibi kâğıt üstünde uğraşlara harcanan sürecin boş ve sıkıcı olduğunu düşünmeleri, bir an önce kodlama aşamasına geçilerek eğlenceli süreçte vakit harcamak istemeleri bu modelin prensiplerine ters düşmektedir, buda psikolojik olarak bu modelden uzaklaşma isteğini doğurmaktadır. Ayrıca müşteri proje boyunca ortada değildir bu da o dönemin en büyük eksikleri arasında yer alır çünkü müşteri derdini tam anlamıyla ifade edemediği için sonradan maliyet ve zaman kayıplarına yol açmaktadır. Bunlar gibi birçok nedenden ötürü günümüzde popülerliğini azaltmıştır.

**4.V Süreç Modeli**

Bu model türünde 3 katman vardır. Bunlar:

**a -Kullanıcı modeli:** Sistemin görücüsü ile nasıl buluşacağını düzenler. .

**b -Mimari model:** Sistemin tasarlandığı aşamadır.

**c -Gerçekleştirim modeli:** Yazılım geliştiriciler tarafından en sevilen ve en sıkıcı olmayan modeldir, bu modelde yazılım geliştiriciler kodlama işlemini yerine getirirler.

Çağlayan modelde olduğu gibi yazılımın gereksinim ve tasarım aşaması büyük önem arz etmektedir. Bu modelde de uzun ve detaylı bir şekilde proje hakkında ayrıntı verilmesi gerekmektedir. Eğer yazılım geliştirici ekipler bu modelden kaçıyorsa bu çalışanlarını sıkıcı bir modelle yormak istememeleri olabilir .IT (Information Technology) yani Türkçe karşılığı bilgisayar teknolojileri projelerinde sıklıkla tercih edilen bir model olmasıyla ön plana çıkmıştır.

**5.Helezonik Model**

Bu model tek bir aşamalı veya tekrar eden adımlarla oluşmuş bir model değildir. Model sanki bir “For” döngüsü açılmış gibi davranır. Her döngüde bir ara ürün çıkar ve bu döngü tekrar tekrar devam eder. Bir döngü içerisinde 4 aşama vardır. Bunlar; planlama, risk analiz, üretim ve kullanıcı değerlendirmesidir. Bu modelde farklı olarak risk analizi bulunmaktadır. Risk analizi bu modelin ön plana çıkmasını sağlanmıştır. Elbette riski analiz edip öyle devam etmek bu süreçte hataları en aza indirmemize sebep olurken öteki yandan ise her aşamada tekrar tekrar riski analiz edip öyle devam ettiğimiz için bir zaman kaybına yol açıyor. Aslında bu zaman kaybı uzun vadeye bakıldığında yani yazılımın iler ki dönemlerinde yazılım geliştiricilere büyük yardımlar etmektedir. Bu nedenle patronlar eğer stratejik bir şekilde düşünürler ve projenin en önemli kısmı olan yazılım geliştiren personellerini önemser ve onların proje sonu ve uzun soluklu dönemde rahat etmesini düşünürlerse bu model tam olarak en uygun modellerden biridir. Bu modelin aşamalarını biraz açacak olursak. Planlama bölümünde isminden de anlaşılacağı üzere daha çok projenin nasıl yapılacağının planlanması ara ürünün için işin nasıl paylaşılacağı ve projedeki amaç ve kısıtların belirtilmesi gerekmektedir. Aynı şekilde risk analizi aşamasında da projenin nasıl bir risklerle karşılaştığı ve eğer birden fazla risk var ve bunlar arasında seçim yapılması gerekiyorsa proje için hangisi daha avantajlı konumda ise ona yönelinmesinin belirtildiği aşamadır. Ayrıca risk analizi ile maliyet hesaplaması daha kolay ve daha gerçekçi bir şekilde daha projenin başındayken hesaplanması sağlanabilir .Üretim bu aşamada projenin ara ürünün başka bir deyişle prototipinin üretildiği aşamadır .Ve en önemli aşamalardan biri olan kullanıcı değerlendirme aşaması, bu aşamada prototip görücüye çıkmıştır ve kullanıcı sisteme dahil olur ve daha en başlarda eğer istediği farklı şeyler veya farkı gereksinimler var ise bu aşamada belirtir ve diğer aşamalarda bu istekleri dahil etmek daha kolaydır .İşi biraz çakallık boyutuna indirgersek de projenin sonradan anlaşılan bir maliyet konusunda sorunlar yaşanıyorsa bu aşamada istenilen farklı ve yeni özellikler için maliyet çizelgesi çizerken bunları da içine dahil etme şansı bulabilirsiniz. Bu projeler de hataları en aza indirgedikleri için biraz pahalıya mal olabilirler. Buda zaman gibi uzun süreçte kârlı durumlara geçilebilecek bir durumdur.

**6.Artımsal Geliştirme Süreç Modeli**

Artımsal geliştirme süreç modeli helezonik model gibi bir anda teslim edilebilecek bir model değildir. Bu modeli de for döngüsü içerisinde düşünmek yanlış olur. Süreç parçalara bölünür ve bu parçalar müşteriye bitince teslim edilir. Her teslimde müşterinin bir isteği gerçekleştirilir. Kullanıcı aldığı her parçada memnun olmadığı konuları iletir ve bu konular yazılım geliştiriciler tarafından kaynaklanan bir hata ise istenilen eklentiler projeye tekrar eklenir fakat bu istenilen özellikler ilk başta söylenilmemiş veya sonradan söyleniyorsa burada yazılım geliştiriciler saat başı ücret alarak bu yeni özellikleri projeye kazandırabilir. Müşteriler için basit gibi gözükebilen bazı işlemler aslında yazılım geliştiriciler için zulüm olabiliyor hatta alınan maddi istek tatmin edici nitelik taşımayabiliyor. Patronlar bu gibi durumlarda bazen kurnazlık yapıp “Süreç çok uzar.”, “Sistem performansı yavaşlar.” diyerek müşterileri ikna etmeye çalışır ikna olmayan müşteriler içinse uçuk bir rakam biçip bu konudan uzaklaşabilirsiniz. Bazen müşteriler biraz inatçı olup bütün projeyi en kısa zamanda ve bir bütün olarak istemektedir. Bu gibi durumlarda müşteriye en kısa zamanda en gerekli olan projeyi temin etmek teklif edilir. Bu modelde en çok dikkat edilecek konu ise istenilenlerin en başta alınması ve kontrat imzalanmasıdır çünkü piyasada çok çeşit müşteri bulunmaktadır ve bu müşteriler için yaptığımız iş çok basite alınmaktadır. Bu yüzden teklif ettiğimiz ücretler çok fazla karşılanabiliyor ve teslim ettiğimiz projelerde eksiklik bulabiliyorlar. Elbette eksik giderilebilir ama ekstra uğraştıracak ve zaman alacaktır. Bu ekstralar için ücret teklif edebilmek için istekleri her iki tarafın imzaladığı bir kontrata aktarılmalıdır. Bu modelin en büyük katkısı ise proje bir taraftan kullanılırken bir taraftan kalan kısmı yazılmaktadır. Bu yüzden müşteri 3 ay beklemek yerine her ay bir parçasını alarak kendi gereksinimler ve ihtiyaçlarını giderebilmiştir. Modelin yaşam döngüsü şu şekildedir; Genel gereksinim belirlenmesi, gereksinimleri artımlara bölme, sistem mimarisini tanımlama, sistem artırılmasının yapılması, artırılımın onaylanması, artırılımın birleşmesi, sistemin onaylanması ve bu aşamadan sonra eğer onaylanmazsa sistem artırılmasının yapılması aşamasına geri dönülür eğer onaylanmışsa sistem tamamlanmış demektir. Şöyle bir açıklama ile isminin geldiği yer daha iyi anlaşılacaktır, proje 5 aşamada değerlendirilir bunlar gereksinim, analiz, tasarım, gerçekleştirme ve test aşamasıdır. İlk aşamadan son aşamaya doğru aktarımla devam eden bir sistem vardır. Gereksinim azalırken analiz başlar, analiz bitmeye başlarken onu takip eden tasarım başlar, daha sonra tasarım biterken gerçekleştirme aşaması başlar ve son olarak gerçekleştirme aşaması biterken test aşaması başlar ve proje sonlanır. Sürekli bir aşamanın işlevi azalırken peşindeki aşamanın işlevi artar ve böyle devam eder.

**7.Kodla ve Düzelt Modeli**

İlk önce yazılım geliştirilir ve geliştirilen bu yazılım müşteri isteklerini karşılayana kadar sürekli bir şeyler eklenip çıkarılarak tamamlanmaya çalışılır. Müşterinin tüm isteklerini karşılamak çok zor olabilir çünkü müşteri ne istediğini tam bilmiyor olabilir. Bu gibi durumlarda teslim süresinin uzayacağı belirtilmelidir. Bu modelde çok uzun kodlar yazılması tercih edilmez. Bu modelde çok fazla para harcanır bunun nedeni sürekli bir düzeltme sürekli kodların bir tarafının değiştirilme zorunluluğundandır. Kodla ve düzelt modeli daha çok tecrübesiz ve yeni kurulmuş küçük çaplı şirketler tarafından tercih edilmektedir. Bu modelin bir diğer sıkıntısı ise belgeleme işlemini ne tüm aşamalarda ne de ayrı bir bölüm olarak bulunmasıdır. Bu nedenle bakım aşaması çok zorludur, ileriye yönelik uzun süre kullanımda insan ürünü olan kodlarda bir bozulma yaşandığında kodlarda yapılan değişiklikler, daha önce hata alınan değişiklikler bilinmeyecektir bu da ileri de daha çok maliyete sebep olacaktır. Bu modelde diğerlerinden ayrı olarak bide emeklilik aşaması bulunmaktadır. Emeklilik aşamasının isminden de anlaşılacağı üzere kod çok çok uzun yıllık değildir. Bu modeli kullandığınız takdirde ilerleyen süreçlerde farklı bir sisteme gerek duyulabilir.

Birçok farklı çeşitte yazılım geliştirme modelleri bulunmaktadır. Bir şirket bir yazılıma başlamadan önce bu modülleri iyice tanımalıdır avantaj ve dezavantajlarını göz önüne alıp hangisi çıkarlarına uygun düşüyorsa onu seçmelidir. Örneğin bir şirket eğer yeni kurulmuş ise kodla ve düzelt modeli mantıklı olabilir çünkü hızlı bir şekilde üretip müşteri memnuniyeti ile networkünü genişletmesi gerekmektedir. Fakat bu modeli yıllarca köklü bir kuruluşa sahip olan bir şirketin yapmasını bekleyemezsin çünkü o şirket bir network e sahiptir ve bu şirket için önemli olan etmen daha az maliyetle daha kaliteli iş çıkarmak olmalıdır. Köklü bir şirketin ise Artımsal geliştirme modelini kullanması daha mantıklıdır çünkü bu metoda başarısızlık çok azdır ve dezavantaj olarak profesyonel kişilere ihtiyaç duymaktadır ve bu da köklü şirket olmanın getirdiği avantaj ile yok olur. Büyük ve uzun süren modellerde helezonik modeli kullanmak gayet avantaj sağlar çünkü bu model prototip halinde olduğu için hem müşterinin gereksinimini tamamlayarak sürekli irtibat sağlayacak hem de riski en az seviyeye indirmiş olacaktır. Uzun projeler için sorunları sonrasında halletmek gayet zor bir meseledir. Çağlayan modeli kullanmak isteyen şirketin yazılım geliştirici personelleri çok üst düzey olmasına gerek duymamaktadır çünkü her aşama iki kez kontrol edilir ve hata düzeyi alt seviyelere çekilir.

Bazı şirketler ise çevik yazılım geliştirme metotlarını kullanmaktadırlar. Bunlar şirketlerin çok büyük ihtiyaçlarını yerine getirebilme özelliklerine sahiptirler. Öncelikle verim yüksektir, bu verimin yüksek olması diğer modellerdeki gibi negatif bir etkiye yol açmamaktadır. Örneğin verimli olması ile birlikte aynı zamanda ucuzdurlar. Yazılan kodlar daha hızlı çalışma kapasitesine sahip olurlar. Tüm bu iyi özellikler beraber aynı zamanda da hata oranları çok düşüktürler. Çevik yazılımın amaçlarından birisi ise piyasaya kısa zamanda ürün sürüp bu ürününse kaliteli olması sağlamaktır. İleriye yönelikte sağladığı birçok avantaj bulunmaktadır. Bakım aşamasında da çevik yazılım metotları en kolay düzeltilebilen yazılımları meydana getirmeyi amaçlar.

Çevik yazılım metotları kodla düzelt modelinden de bir şeyler çalmıştır. Müşteri ile olan sürekli iletişimi farklı bir boyutta kendi özelliklerine katmışlardır. Sürekli olarak müşteriye danışıp işlem yapmaktansa kısa kısa aralıklarla müşteriyi projenin akışı hakkında bilgi vermeyi eksik etmemektedir. Buda ne sürekli bir değişime ne de müşteriye teslim süresinin uzayıp müşteri ile kötü ilişkilere girmeye sebep olur. Ayrıca müşteriye parça parça teslim gerçekleşir, bu da müşteri memnuniyetinin kazanılması ve işten alınan hazzın artmasına sebep olur. Çevik yazılım metotlarının bu denli hızlı olmasının sebebi de ekibin sürekli birbirinden hem yaşadığı sorunlar hakkında yardım alması hem de birbirlerini bugün yapacağı işler hakkında bilgilendirmesi ile gerçekleşmektedir. Bu metottun en büyük avantajları arasında ekibi oluşturan insanlar her zaman yüksek motivasyonlar doludur. Projelerin başarısı kat kat artarken müşteri ile iletişim kopmaz, müşteri memnun kalırsa networklerine de bu şirketi tavsiye eder ve daha fazla iş almaya sebebiyet verir. Bu kadar motive ve sürekli üreten müşteri ile sorun yaşamayan bir kimse yaptığı mesleğe her geçen gün daha fazla bir aşk ile dolacaktır bu da kendi özel hayatına yansıyacaktır.

**Scrum**

Bu bir çevik yazılım yöntemidir. İsmini rugby sporundan almaktadır. Bu yöntem 90’larda bulunmuş ve bu metot sadece yazılım proje yönetimlerinde uygulanmamaktadır. Scrum aslında fazla mesai yapan şirket personellerinin iç sesidir. Bu metot da projeler 30 günlük 4 hafta sürede tamamlanmalıdır aksi takdirde bu Scrum olmaktan çıkacaktır. Her gün şirket içinde ayak üstü çalışanlarla 15 er dakikalık toplantılar düzenlenmektedir. Bu küçük toplantılarda şirket personelleri dün yaptıklarını hangi sorunlarla karşılaştıklarını ve bugün neler yapacaklarını belirtirler eğer bu küçük 15 dakikalık toplantı uzarsa bir sorun var demektir bu yüzden bunun için ayrıca başka bir zaman da büyük bir toplantı planlanır. Scrumın bu küçük toplantıları sayesinde proje her an kontrol altında yürütülür. Scrum takımı 5 – 9 kişi arasında oluşur, her takımın ayrı görevleri bulunur. Bir adet proje yöneticisi vardır, takımın sorumluluğundan bu personel sorumludur. Sprintler planlanır bu planlarda gereksinim listesi çıkarılmalıdır ve projelerde kendi inisiyatif alıp alınacak riskler gözden geçirilir. Sprint aslında her projenin kendi başına adlandırılmasıdır. Sprintlerin olmazsa olmazı 15 dakikalık toplantılardır.

Günümüzde büyük şirketlere bakıldığında büyük çoğunluğun scrumı seçtiği söylenebilir. Buda sonuç olarak scrumın başarılı bir yönetim biçimi olduğu anlamına gelir.

**Xp (Exteme programming)**

Bu yöntemde önemli olan grup içi iletişim ve ekibin sürekli birbiri ile kontak içerisinde olmasıdır. XP için iletişim, basitlik, geri bildirim ve cesaret önemli 4 etmendir. Xp projeleri ekibin üstün iletişim yeteneği ile başarıya ulaşmaktadır. İletişimin bu denli önemsenmesi olayların sorunların hızlı bir şekilde üstesinden gelinmesine sebebiyet verir. Xp de çözümler sıralanır ve aralarından en basiti seçilir karmaşık ve daha zorlu yöntemler ile uğraşmaktan kaçınılır.2 ile 4 haftalık bir süreçten oluşur. Kodla ve düzelt modeli gibi müşteri de bu yazılım geliştirme metoduna dahil olur. Belirlenen aralıklarda müşteriye gelinen son noktaya kadar gösterilir. Bunun temel amacı ise bu metot ile yapılan projelerin sonunda şirket ve müşteri arasındaki sorunları en aza indirgemektir. Cesaret kelimesini bir projede tehdit unsuru olarak görenler olabilir fakat Xp metodunda bu durum hiç de o gözle bakılmıyor. Xp de yazılımı gerçekleştiren kişinin cesaretli olması beklenmektedir çünkü yanlış yapsa bile o kodu çöpe atıp yeni baştan yapma imkanını Xp yöntemi sağlamaktadır. Bu durum projelerin daha hızlı yönetilmesine sebep olur belki yırtıp atılan projeler için büyük zaman kaybı denilebilir fakat korkmadan yazılan bir kod çoğu zaman doğrudur ve bu da gereken hızı ekibe kazandıracaktır.

Belirlenen aralıklarda müşteriye projeler gösterileceği için en başında ve belirli aralıklarla müşterinin hangi yazılımı ilk istediği öğrenilir. Bunun üstüne yoğunlaşılır. Bu metodun bir diğer özelliği ise müşteri temsilcisi ile belirli süreler beraber çalışılır ve gereken bilgiler müşteri temsilcisinden kısa sürede alınır ve zaman kaybına hiç uğramadan yazılımın geliştirilmesi devam edilir. Projenin tasarımı gayet basit olması ve herkes tarafından anlaşılabilir olması beklenir ileride istenilen değişiklik proje üstünden yapılabildiği ve müşterilerin genel itibari ile memnun olmama sorunları olduğu için bu tasarım aşamasından zaman kaybetmekten çekinilir. Bu metoda özel bir şey de bir program için çifter kişi çalışılmasıdır. Bu durum teoride fazla insan gücü ve gereksiz maliyet gibi gözükse de bu durum bir çırağın usta tarafından ustalaşması anlamına gelir. Böylece zaman geçtikçe 2 usta personele sahip olunur. Ayrıca sorunlar iki kişi tarafından daha kolay fark edilip yine aynı şekilde iki kişi tarafından daha kolay çözüme ulaşılması sağlanır. Sistemin entegrasyonundan sürekli kaçınılmaktadır çünkü bu entegrasyon sistemde oluşursa sorunun nedenini bulmak çok zorlaşır ve çöp bir projeye sebebiyet verir bu denli önemli bir konuda ancak sürekli olarak sık bir şekilde programın entegre olmadığını kontrol ederek anlaşılabilir. Unutulmayacak bir diğer unsur ise güvendir. Güven bir şirketin büyümesi veya küçülmesine sebebiyet verir. Bir kişiye verilen söz, örnek olarak teslim tarihi ele alınırsa; teslim tarihine kadar proje tamamlanıp verilmelidir, ayriyeten bu tarihe kadar küçük parçalara ayrılan parçaların da teslim edilmesi şirket müşteri ilişkisini dışarılara taşıyacaktır. Eğer şirket zamanında müşterinin sorununu çözerse başına daha çok iş alacaktır ki bunu da her şirket ister. Xp metodunda farklılık oluşturan bir diğer özellik ise sürekli olarak sistemin yapılandırılmasıdır. Bu yapılandırma müşterinin yeni istekleri ve eski istediklerinin daha verimli şekilde sunulmasını kapsamaktadır. Bu metodu çok çalışan sürekli patronlarından şikâyet eden insanlar seveceklerdir çünkü bu metotta fazla mesai kavramı yoktur. Günlük 8 saat ve haftalık 40 saat hesaba alınarak proje süreleri belirlenir. Bu aslından hem şirket personeli hem de patron için güzeldir çünkü fazla çalışacağını bilen insan günün ilk saatlerinde kendisini işe vermez diğer yandan da yaptığı mesailer için de patron fazla ücret ödemesi yapar lakin fazla çalışmamıştır. Xp yönteminde kodların yazılımı için belirlenen bir standart vardır ve bu standart sayesinde bir kodu birden fazla insan tarafından okunup düzeltilebilme şansı vermektedir.

Sonuç aşamasına gelecek olursak patronlar kendi kriterlerine göre birer çalışma model veya metodu seçerler buna göre de personel organize olur.

**UNUTULMAMALIDIR Kİ!!!**

**Bir yazılım geliştirilirken en önemli husus müşteri memnuniyeti değildir, en önemli husus bir patron için çalışan çok değerli insanlardır. Lakin bir patron olarak çalışanlarınızı önemserseniz onlarda sizi ve sizin işinizi önemseyeceklerdir bu da sizin için en önemli husus olan MÜŞTERİ MEMNUNİYETİ anlamına gelmektedir…**

**HALİL YEŞİL**

**180601003**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**1.SINIF**