Yazılım Modelleri;

Yazılım modelleri denince, konu hakkında hiçbir bilgisi olmayan bir insanın aklına ilk gelecek soru çoğunlukla “yazlım modelleri mi ? Ne gerek var ki bunlara” demek gelir. Ama işin içine girdikçe yazılım geliştirme sürecinde en büyük rolü oynadığını anlamış oluruz. Ufak bir örnek vermek gerekirse, bir şirkette yazılım geliştirme ekibinin bir parçası olarak bulunuyorsunuz. Yazılım modelleri ekibinizin aldığı işi gerçekleştirirken; kimin hangi işi yapacağını, iş yapılırken nasıl ilerleneceğini, ekibin birbirinden bağımsız davranmadan bir ekip gibi çalışması ve işin olabildiğince hızlı ve etkin bir biçimde tamamlanmasına yardımcı olur. Yani yazılım modelleri sizin yazılım geliştirme hayatınızda, siz farkında olmasanız bile rol oynar. Günümüzde nereyse bütün küçük büyük şirket ve kuruluşlar yazılım modellerini etkin bir biçimde kullanırlar. Hatta şu an aktif olarak kullanılan birkaç yazılım modellerinden çağlayan nam-ı değer waterfall,, barok modeli, extreme programming, artımsal geliştirme süreç modeli, helezonik model, kodla ve düzelt modeli , gelişigüzel kodlama v modeli ve scrumdan bahsedelim.

1.Gelişigüzel Kodlama:

Aslında tam bir model olarak ele almamız uygun olmaz. Çünkü diğer yazılım geliştirme modellerine bakıldığı zaman bir ilerleme yolu ve bir yöntemi vardır. Fakat gelişigüzel kodlama da bunlara rastlayamayız. Geliştirici ekip ya da kişi ürünü tamamlarken hiçbir ilkeye bağlı değildir. Tamamen kendisi nasıl isterse öyle yapar. Bu yüzden bu kodlama biçimine tam olarak yazılım modeli diyemeyiz ama bazı yerlerde kullanıldığı için bu makalemizde ele almış olalım. Bu kodlama türü, daha yazılım geliştirme modelleri ortaya çıkmadan önce kullanılmaya başlamış. Genellikle küçük çaplı projeler bu yöntemle tamamlanmış. Hatta örnek olarak şu an yazılım veya bilgisayar mühendisliği birinci sınıf öğrencileri bile zaman zaman bu yöntemi kendileri bile farkında olmadan kullanıyorlar. Bu yöntem ekip çalışmasına uygun olmadığı ve izlenebilirlik açısından neredeyse imkansız olduğu için fazla tercih edilmez. Son olarak bu yazılım geliştirme biçimde bir dökümantasyon bulunmaz.

2.Barok Modeli:

Barok modeli 1970lerde ortaya çıkmıştır ve gelişigüzel kodlama yapan insanların artık işleri daha da profesyonelleştirme çabasıyla bir nevi başarılı olmuştur. Bu yöntemde yazılım yaşam döngüsü adımları ele alınarak geliştirme yapılır. Buna bağlı olarak da bir dokümantasyon evresi bulunur. Dokümantasyon evresi de günümüzde yapılan biçimden farklı olarak ayrıca yapılır. Burada anlatmak istediğim, dokümantasyon yazılım tamamlandıktan yani yazılım geliştirilip test aşamaları yapıldıktan sonra yapılır. Günümüzde dokümantasyon yazılımın her anında yapılır. Bu model zamanında kullanışlıydı fakat sonrasında ortaya çıkan waterfall yani şelale modeli gelişigüzel kodlama ve barok modelini unutturdu. Gelişigüzel kodlama ve Barok modeli günümüzde yazılım modeli olarak ele alınmazlar.

Waterfall Modeli:

Bu model kendinden önceki yazılım geliştirme modeli olarak sayılabilme olasılığı olabilecek modelleri tarihe gömmüştür. Aslında geleneksel yazılım geliştirme modeli olarak nam salmıştır fakat hala günümüzde en fazla kullanılan yazılım geliştirme modellerinden bir tanesidir. Bu yazılım geliştirme modelinin günümüzde hala çok tercih edilmesinin sebebi ise yazılım geliştirme aşamalarının en az bir kez tekrar edilmesinden kaynaklanır. Her Yazılım geliştirme aşamasında olduğu gibi bu yazılım geliştirme modelini kullanacak insanların yapacağı işler sırasıyla; gereksinimleri belirlemek, analiz etmek, tasarımı kararlaştırmak, uygulamak ve bakımını yapmaktır. Yazılım geliştiricileri bu modeli kullanırken her aşamadan bir önceki aşamalara gidip değişiklik yapabilirler. Bu da bu yazılım geliştirme modelinin favori özelliklerindendir. Tabi yazılım geliştirilirken dokümantasyon da es geçilmez. Her bir aşama da dokümantasyon ara vermeden ya da sonra bırakılmadan yapılır. Kendinden önceki yazılım geliştirme modellerine bakarsak kendisine en yakın olan barok modeli ile arasındaki farkın büyük ölçüde dokümantasyondan kaynaklı olduğunu görürüz. Barok modelinde dokümantasyon bütün yazılım geliştirme aşamaları bitirildikten sonra dokümantasyon yapılırdı fakat çağlayan modelinde dokümantasyon yazılımın geliştirme aşamalarının her birinde rol oynar. Diğer önemli bir konu ise analizdir. Eğer analiz aşamasında kullanıcının istedikleri çok iyi bir biçimde anlaşılmamış ise o yazılım büyük bir ihtimalle başarısız kabul edilecektir. Yazılım geliştirme ekiplerini analiz aşamasında yazılımın tüm gereksinimlerini karşılayacak bir şekilde detaylı bir araştırma ve çalışma yapmasını gerektirir. Analiz aşaması kullanıcın isteklerine uygun bir şekilde anlaşmış ise başarılı ve işlevli bir ürün yapmışsınız demektir. Tabi Bu yazılım geliştirme modelinde bulunduğunuz aşamada geçirdiğiniz her vakit, çalışma saati sizin aleyhinize bir etkendir çünkü size hem efor hem para hem de zaman kaybettirir. Sizin de kaybetmek istemeyeceğiniz en önemli üç etken bunlardır.

Tabi her yazılım geliştirme sürecinde olduğu gibi bu modelde de yazılım teslim edildikten sonra siz yazılımın geliştiricileri olduğunuz için yazılım bakım işlerini de o yazılım emekliye ayrılana kadar yapacaksınız demektir ki bu da sizin için yazılımı geliştirmekten daha uzun ve yorucu bir iştir.

4.V Süreç Modeli:

Bu model kendinden önceki waterfall yazılım geliştirme modelinin gelişmiş hali olarak düşünülebilir. Bu model genellikle yapılacak işin net ve açık olduğu projelerde kullanılır. Örnek olarak ise bilgi teknolojileri için geliştirilen projeler örnek verilebilir. Bu modeli anlatmak gerekirse; aklımızda bir V harfi canlandıralım. Bu V harfinin sol tarafından sağ tarafına doğru bir çizgi çekelim. Bu işlemi aşağıya doğru biraz boşluk bırakarak beş kere yapalım ve bu çizgilere yukardan aşağıya doğru bir den beşe kadar isim verelim. Sol taraftan birinci alanımızda yazılımın gereksinimleri ve onun karşısında eş zamanlı olarak gerçekleştirilecek olan sistemin gereksinimleri vardır. Bu iki işlem birbiriyle bağlantılı olarak gerçekleştirilir. İki numara da sistemin tanımları ve karşısında tamamlanmış sistemin tanılar vardır. Burayı açıklamak istiyorum. Sistemde bulunması gereken özellikler ve sistem tamamlandıktan sonra bulunan özellikler demektir. Üçüncü satırda sistem karşısında ise tamamlanmış sistem vardır. Dördüncü satırda alt sistem karşısında ise tamamlanmış alt sistem vardır. Beşinci ve son satırda ise modül ve sınanmış modül vardır. Bu V yapısı Mimarı modelleri birden beşe kadar olan alanı kapsar. Bu beş durum da tamamlandıktan sonra üretim gerçekleştirilir ve sınama yani test yapılır. V harfimizin sol tarafı kısaca üretimi yaparken sağ tarafı ise test durumunu yapar. Bu aşamalara baktığımızda zaten bu yazılım geliştirme modelinin son derece kesin ve hata istenmeyen durumlarda kullanıldığını anlıyoruz. Bu yazılım geliştirme modelini kullanarak çalışan işletmeler de her yazılım geliştirme ekibi gibi önce gereksinimleri toplarlar sonrasında ise ürünü teslim ederler. Bu modelde, waterfall yazılım geliştirme modelinde olduğu gibi aşamalar arasında geri dönüş yoktur. Bu yüzden yazılım geliştiricileri bu modeli kullanırken dikkati olmalıdırlar. Aksi halde büyük sorunlara veya maddi kayıplara yol açabilirler.

5.Helezonik ( Spiral ) Model :

Helezonik model oldukça önemli, değeri yüksek projeler için kullanılır. Mesela devletlerin özel yazılımları buna örnek gösterilebilir. Bu yazılım geliştirme modeli kendinden önceki yazılım geliştirme modellerinden oldukça karmaşık ve farklı görünmektedir. Çünkü yazılım geliştirme aşamaları bir spiral içinde x ve y eksenleri yardımıyla bölünmüştür. Bu durum bu yazılım geliştirme modelini kullanacak ekiplere sırasyıla bütün aşamaları kontrol etmeleri ve bir aşama bitmeden diğerine geçemeyeceğini geçse bile o aşamanın bir sonraki turda karşısına çıkacağını bilir. Bu yöntem neredeyse hata istemeyen bir yöntemdir. Biraz bu yazılım geliştirme modelinin aşamalarından bahsedelim. Daha önce de dediğim gibi spirali dörde böldüğümü düşünelim. Bu bölmelerin isimleri ; planlama, risk analizi, kullanıcı değerlendirme ve üretimdir. Projeye başlanırken önce planlama aşamasında öninceleme yapılır. Sonrasında risk analizi yapılır ve ilk prototip üretilir. İşin genel kavramı incelenir ve tekrar risk analizi yapılır. Bu şekilde en az 3 prototip üretilir ve üretim aşamasında gelinceye kadar simülasyon ve modelleme, yazılım gereksinimleri gereksinim onaylama ürün tasarımı ve tasarımı test etme ve onay aşamasından geçilirse eğer birleştirme ve test planı aşamasına geçilir. Ve tekrardan risk analizi tapılır ve işin son bir prototipi yapılır. Ürün kullanıcıya sunulmadan önce detaylı tasarım kodlama, modül testi, birleştirme testi, kabul testi ve servis yapılır ürün teslim edilir. Tabi bu işlemler bittikten, ürün teslim edildikten sonra bunun zorlu bir bakım aşaması var, Eğer Devletin kurumlarına örneğin, ASELSAN, HAVELSEN, ROKETSAN gibi kuruluşlara çalışıyorsanız yaptığınız işin de sağlamlığı, güvenilirliği, verimliliği ve en önemlisi istenilen özellikleri karşılıyor olması sizin en büyük amacınız olmalıdır. Bu yazılım geliştirme modeli de göründüğü üzere oldukça fazla risk analizi bölümü barındırdığından hata neredeyse olmamasına özen gösterilmelidir. Zaten günümüzde bu modeli kullanacak şirketler istedikleri yazılımı kısa sürede teslim alamayacaklardır çünkü helezonik model kısa süreli işler için uygun bir model değildir. Genellikle 5-10 yıl sonrası için yapılır ve oluşturulan bu ürünü kullanacak şirkette ürünü uzunca bir süre kullanmak ister. Kazancı büyük olduğu kadar zorlu ve yorucu bir yazılım geliştirme modelidir. Hata olmaz denilemez ama diğer yazılım geliştirme modellerinden oldukça az bir hata payı vardır. Bu da önemli konular için çalışacak insanların veya yazılım geliştirme kuruluşlarının bu modeli genellikle tercih ettiği görülür. Tabi şimdiye kadar hep helezonik modelin nasıl işlediğini konuştuk. Sırada bunun bir de dokümantasyon evresi var. Böylesi zor ve uzun bir yazılım geliştirme sürecinin dokümantasyon oluşturma süreci zaten başlı başına büyük bir vakit alır. Dokümantasyon bu yazılım geliştirme modelinde en baştan iş bitene kadar en ince detaya kadar büyük bir itinayla yapılır çünkü üzerinde çalıştığınız problem eğer sağlam temellerle atılmamış ise hem sizin onca süre harcadığınız vakit hem de emeğinizin yerle bir olmasına yol açmış olursunuz. Küçük projeler için tavsiye edilmez. Sağlam güçlü ve ne yaptığını bilen tecrübeli bir ekibiniz olduğu takdirde bu yazılım geliştirme modelini kullanmanızı tavsiye ederim.

6. Artımsal Geliştirme Süreç Modeli:

Bu yazılım geliştirme modeli diğer 5 modele göre sistemi farklı ele alır. Önceki modeller sistemi bir bütün olarak ele alırken artımsal geliştirme süreç modeli sistemi bölerek parça parça teslim eder. Her parça sistemin bir özelliğini kendinde saklar. Bu yazılım geliştirme modelinde gereksinimler önceliklerine göre sınıflandırılır. En öncelikli parça ilk teslim edilir ya da erken teslim edilecekler listesine dahil edilir. Gereksinimler sadece önceliklerine göre değil parçaların birbirine bağlılıklarına göre de sıralanır. Bir parçanın geliştirilmesi başlandığında o parçanın gereksinimleri dondurulur. Yani o parçada istenilen gereksinim olmamışsa ise sonraki parçaya eklenir. O an üzerinde bir değişiklik yapılamaz yapılmamalıdır. Yapıldığı takdirde işin işleyişi ve anlaşmazlıklar olacağı için sorun teşkil eder. Her parça tamamlandıktan sonra birleştirilerek bir bütünü oluşturur ve ürünün işlevselliği arttırılmış olur. Bu model helezonik modelleme gereken projelerde kullanılamaz çünkü bu yazılım geliştirme modeli için gerekli ortam helezonik modelde karşılanamaz. Eğer istenilen sistem eksik parçalarla çalışabiliyor, bir sorun teşkil etmiyorsa bu yöntem çok iyi olacaktır çünkü hem kullanıcının istediğini kısa sürede gerçekleştiriyor olacaksınız hem de sizi zihinsel yorgunluk ve baskından bir nevi kurtarmış olacaksınız. Buna örnek vermek gerekirse, üniversitelerdeki dönem ödevleri tam olarak nokta alışı olabilir. Biliyorsunuz dönem ödevleri kısa periyotlarla test edilerek daha doğrusu danışman hocaya gidilip gelişimin izlenmesi ile tamamlanır ve vaktinde teslim edilmeye çalışılır. Yine her yazılım geliştirme aşamasında olduğu gibi gereksinimler belirlenir fakat diğerlerinden farklı olarak bu yazılım geliştirme modelinde gereksinimleri öncelik sırasına göre sıralama veya bölme işlemi yapılır. Daha sonra sistem mimarisi tanımlanır, sistemin parçaları birleştirilir ve onaylanır ve en son sistem son halini alır. Eğer bunu süreçlere bölecek olursak, örneğin elimizde dört haftalık bir iş olsun. İlk iki haftamızda gereksinim iş akışı sabit bir seviyede devam ederken ikinci haftada analiz iş akışı yapılmaya başlar. Üçüncü haftada tasarım iş akışı ve gerçekleştirim iş akışı tavan yapar diğer ikisi yavaşlar. Son haftada ise test iş akışı yapılarak ürün teslim edilir. Bu yazılım geliştirme modelinde her parça ile sonuca bir adım daha yaklaşılır. Sistemin işlevselliği ilk tamamlanan parçalarla oluşmaya başlar çünkü ilk gönderilen parçalar sistemin en öncelikli parçalarıdır ve ilk parçalara genellikle iş dünyasında prototip adı verilir ya da prototip olarak adlandırılır. Dokümantasyon evresini es geçmeyelim ki bu yazılım geliştirme modelinde dokümantasyon her parçadan sonra tekrar elden geçilir ve olası değişikliklere karşı ya da değişen gereksinimlere karşı yeni parçalar sunulur. Bu durum projenin batması ya da başarısız olması olasılığını azaltır.

7.Kodla ve Düzel Yaşam-Döngü Modeli:

Kolda ve düzelt yazılım geliştirme modeli çok büyük çaplı projeler için kullanılamaz. Adı üstünde kodla ve düzelt. Ufak ufak satırlardan oluşan kod parçalarını birleştirerek oluşturulur. İlk safhada ilk sürüm yazılıp teslim edilir. Sistem son hali alana kadar ufak ufak o kod parçalarından yüzlerce yazılır. Oldukça zahmetli, zaman ve dikkat isteyen bir iştir. Genellikle programlamayı, yazılım yazmayı yeni öğrenen kişiler kullanır. İşin temellerini öğretmede oldukça iyi bir yazılım geliştirme modelidir. Profesyonel arenada pek bir kullanıcısı yoktur ama yerine göre iş görür. Bu Yazılım geliştirme modeli çok sayıda kod satırından oluştuğu için bakım safhası oldukça sınırları zorlar. Bakım safhasında zorlanılmasının sebebi ise sisteme ait herhangi bir dokümantasyon olmayışıdır. Dokümantasyon olmadığı için de kullanıcının istekleri başınızı ağrıtabilir. Amatörce bir yazılım geliştirme modelidir. Hatta bazı kişilere göre yazılım geliştirme modeli bile sayılmaz. İstenilen gereksinimlere göre emeklilik safhası diğer yazılım geliştirme modellerinden çıkan ürünlere göre daha erkendir çünkü bakım aşaması oldukça zordur. Ürün geliştirilirken önce ilk bir prototip çıkar. Kullanıcı, yazılımı istediklerini karşılayana kadar değiştirir ya da düzenler. Bu durum öncelikle bir yazılım mühendisi için etiğe aykırıdır. Eğer iş yaptığınız insanla aranızda bir anlaşma yok ise bu işi yapmayınız. Dokümantasyon olmadığından kullanıcı sizi istediği kadar kullanabilir. En sonunda yazılımı teslim edersiniz ve bakım aşamasına geçersiniz. Birkaç aylık süre sonrasında yazılım emekliye ayrılır. Bu yazılım geliştirme modeli yazılım geliştirme açısından oldukça kolay bir yoldur çünkü kısa ve istenilene yönelik çalışma yapılır fakat bunun karşılığında yüklü bir miktar para ödersiniz. Yani anlayacağınız bu yazılım geliştirme yolu hızlıdır fakat cebinizi yakar. Pahalı olmasının sebebi ise anlayacağınız yazılımı geliştiren kişiye istediğinizi yaptırana kadar onunla iletişim halinde oluşunuzdur. Genellikle bu Kodla ve Düzelt yaşam döngüsü modelini küçük şirketler ya da bireyler çalışan tecrübesiz yazılım geliştiricileri kullanırlar. Eğer kendinizi geliştirmek için bu yazılım geliştirme modelini kullanıyorsanız işine oldukça yarayacaktır. Büyüme aşamasındaki bir şirketseniz sizi belli bir noktaya kadar ilerletecektir fakat işler büyüdüğünde sizin başınızı oldukça ağrıtan müşteriler çıkacaktır. Uzun lafın kısası yazılım geliştirmeye yeni başladıysanız bu yazılım geliştirme modeli sizi bu arenada belli bir noktaya kadar götürecektir

8.Extreme Programming:

Bu programlama modeli Kent Beck tarafından 1999 yılında yazılım geliştirme disiplini olarak ortaya atılmıştır. Temel ilkesi ise grup içi iletişime önem veren, geri dönüşlerin fazla olmasına imkan sağlayan bir yöntem olmasıdır. Yazılım geliştirmedeki kolaylığı ve esnekliği bu programlama modelini diğer programlama dillerinden ayıran en önemli özelliğidir. Extreme proggraming’in ön kendisini diğerlerinden ayıran dört özelliğinden bahsedelim. Temel değer olarak iletişim, basitlik, geri bildirim ve cesaret hem özelliği hem de olmazsa olmazlarındandır. Günümde proje ekiplerine yakından baktığınız zaman ortaya çıkan büyük problemlerden biri ekipler arasındaki anlaşamamazlık. Extreme programming modelinde bu durum ortadan kaldırılmaya çalışmıştır hatta ön plana alınmıştır. Projenin başarıya ulaşabilmesi için ekibin düzgün ve sağlıklı bir iletişim ağına sahip olması gerekmektedir. Bu iletişimin yüz yüze yapılması önemlidir. Kısaca XP’nin aradığı basitlik sağlanması oldukça zor bir konudur çünkü XP’de basitlik zorunlu işlerin yapılması anlamını taşır. O an gereksinimlerin karşılanmasını sağlayan en kolay ulaşılabilecek çözüme denir. XP’de geri bildirim oldukça büyük bir şanstır çünkü kullanıcıdan aldığınız geri bildirim sizin gereksinimleri daha doğru bir şekilde tamamlamanızı sağlar. XP’de müşteri proje grubunda bulunur ve sürekli geliştirme aşamasının içinde bulunur. 2-4 hafta aralıklarla sürümler sistemin durumunu belirtir. Geliştirici ekip zaman zaman müşteri ile buluşup yazılımın şimdiye kadar geldiği noktaya bakıp değerlendirme yaparlar.

9.Scrum:

Scrum yazılım geliştirme modeli aslında adını rugby sporundaki hücum taktiğinden alır. Bu taktikte tüm oyuncular durdurulamayacak bir şekilde karşı sahaya koşarlar bir nevi sprint yaparlar. Scrumdaki “sprint” tabiri de buna benzerdir. Scrum modelinde uzun soluklu işler olmaz. Sistem sprintler halinde 2-4 haftada tamamlanıp teslim edilir. Scrum şu an günümüzde en çok kullanılan yazılım geliştirme modellerinden biridir. Scrumu biraz daha tanıyalım. Çalışma mantığına baktığımız zaman günlük toplantılar ayaküstü 5-10 dakika olacak şekilde yapılır. Bu toplantılarda o gün ne yapacağınız ve dün ne yaptığınız ya da yapamadığınız konuşulur. Sabah toplantısından sonra sprint başlanır. Scrumlar gruplar halinde yapılır. Gruplar 5-10 kişilik olmalıdır. Her scrum ekibinin başında bir scrum master vardır ve bu kişi scrum takımının çalışma ortamını çalışılabilir hale getirmekle sorumludur. Bir scrum master başka bir grupta scrum takımı içinde yazılım geliştirici olarak bulunabilir. Scrum mantığındaki bu sprintler 2-4 hafta olacak şekilde sprint-1- sprint-2- olacak şekilde ürün teslim edilene kadar yapılır. Scrum takımındaki kişiler başka takımlardaki kişilerle yer değiştirebilir bu bir sorun teşkil etmez. Günümüzde scrum yazılım modeliyle çalışan bilinen şirketlerden bazıları Google, Microsoft, Siemens, IBM örnek verilebilir. Genel olarak scrum yazılım geliştirme modeline baktığımızda kendinden öncekilere oldukça fark attığını görüyoruz. Bunun asıl sebebinin ise hem ekibe hem de kullanıcıya sağladığı yarardan olduğunu söyleyebiliriz.

Sonuç olarak hangi yazılım geliştirme modelini kullanacağınıza karar verme işlemini yaparken sizden istenilen gereksinimleri en uygun yazılım geliştirme modelini seçmeniz halinde bu size hem vakit hem kazanç hem bu mecrada nam kazandırır.

Kaynakça:

Doç.Dr.Deniz KILINÇ Çevik Yazılım Geliştirme Ders Pdf

Doç.Dr.Deniz KILINÇ Yazılım Yaşam-Döngü Modelleri ders pdf

<https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742>

<https://medium.com/@HayriRizaCimen/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-ve-s%C3%BCre%C3%A7-modelleri-70fdfb2f8f77>

<http://www.mehmetduran.com/Blog/Makale.html/Yazilim-Gelistirme-Surec-Modelleri/299>

Mert Şahin 180601048