İsim: Metehan Kaya

Bölüm: Bilgisayar Mühendisliği

Sınıf: 1. Sınıf

Numara: 210601001

Konu: Yazılım Yaşam Döngüsü Modelleri

Yazılım Yaşam Döngüsü Modelleri Nelerdir.

Bu modelleri ayrıntılı olarak incelemeden önce “Yazılım Yaşam Döngüsü” kısaca nedir onu inceleyelim.

Yazılım Yaşam Döngüsü: Bir yazılım ürünü üzerinden çalışmaya başladıktan itibaren ürün halihazırda kullanılırken bile bazı aşamalardan geçer. Bu aşamalar bütününe “Yazılım Yaşam Döngüsü “denir.

Bu aşamalar kısaca şöyledir:

1-Planlama: Müşterinin isteklerine bağlı olarak bir dokümantasyon oluşturulur. Ürün hakkında fizibilite çalışma yapılır.

2-Analiz: Ürün hakkında detaylı bilgilerin takımca kavrandığı aşamadır. Risk analizi, projenin alacağı vakit vb. detaylı bilgilerin analiz edildiği kısımdır.

3-Tasarım: Ürünün mantıksal ve fiziksel olarak tasarlandığı aşamadır.

4-Gerçekleştirim ve Test: Ürünün kodlandığı ve üzerinde testlerin yapıldığı aşamadır.

5-Bakım: Ürün çalışır bir vaziyette müşteriye teslim edildikten sonra müşterinin talebine bağlı olarak üzerinde bir takım düzenleme ve bakım işleminin gerçekleştirildiği aşamadır.

Yazılım Yaşam Döngüsü Modelleri: Yazılım yaşam döngüsü modelleri, yazılım yaşam döngüsünün aşamalarını temel alır fakat devam yolları genelde birbirinden farklıdır. Birbirinden farklı olmaları ve birden fazla olmalarının nedeni ise yazılım ürününün büyüklüğüne bağlıdır diyebiliriz.

Bu modeller iki ana başlık etrafında toplanır:

1-Geleneksel Yazılım Geliştirme : Gelişigüzel model, barok modeli, çağlayan modeli, V süreç modeli…

2-Çevik Yazılım Geliştirme: XP (Xtreme Programming), Scrum.

Gelişigüzel Model: Tam olarak bir model sayılmaz. Bir ya da iki kişilik ekipler tarafından genellikle yazılım olarak değerlendirilemeyecek küçük programlar geliştirmek için kullanılır. İlerleyen vakitlerde ürünün geliştiricisi bile nerede ne yaptığını anlamakta zorlanabilir. Ürünün gelişimi takip etmek ve üzerinden değişiklik yapmak oldukça zorludur.

Barok Modeli: Yazılım yaşam döngüsü adımları birbiri ardına takip edilerek ürün geliştirilir. Bir adımı bitirdikten sonra ilerleyen adımlardan bu adımı nasıl geri dönüş yapılacağı tanımlanmamıştır.

Yazılım geliştirme aşaması tamamlandıktan sonra ürünün dokümantasyonu yapılır. Yani yazılım geliştirme ve dokümantasyon birbirinden ayrı olarak gerçekleştirilir. Günümüzde tercih edilmiyor.

Çağlayan Modeli: En eski, en tanınmış ve en temel modelimizdir. %100 olarak ne istendiğinin dokümantasyonu çıkarılmış ve az zamanda teslim edilebilecek projelerde tercih edilebilir. Eski zamanlarda çok büyük bir popülerliği bulunsa da günümüzde bu ünü hızlı bir şekilde azalmaktadır.

Bir aşamanın bitirildikten sonra diğer aşamaya geçilmesi özelliği bakımından barok modeliyle benzerlik göstermektedir fakat bu modelde geri dönüşlerin nasıl yapılacağı tanımlıdır. Yapılacak her aşama için hazır bir dokümanın bulunması gerekmektedir.

Bu modeli kullananları dikkat etmesi gereken bir nokta bulunmaktadır: Her ne kadar geri dönüşlerimiz tanımlı olsa da analiz aşamasında mümkün olan her detayın tasarım aşamasına geçirilebilmesi için müşteri dokümantasyonunun ve sistem gereksinimlerinin en ince ayrıntısına kadar belirlenmesi gerekmektedir. Fakat bu kadar ayrıntının düşünülmesi ve incelenmesi gerektiğinden en çok zaman bu iki aşamada(Analiz ve Tasarım) harcanmaktadır.

Müşteri bu yazılım ürününün geliştirilmesinde tamamen etkisiz kalmaktadır. Bu nedenle ürün çalışır halde müşteriye teslim edildikten sonra bir eksik veya müşterinin yeni bir istekte bulunması gibi durumlar yaşandığında bu duruma adapte olmak çok zor ve maliyetli olacaktır.

V süreç modeli: V şeklinde bir yapı düşünelim. Bu yapıyı iki eşit parçaya ayıran bir çizgi de olsun. Çizginin sol tarafı üretim-geliştirme aşaması, çizginin sağ tarafı ise test aşamasıdır.

3 tane temel çıktıdan oluşur:

1-Kullanıcı Modeli: Geliştirme sürecine müşterinin ne kadar dahil edileceğini tanımlamaktadır ve sistemin nasıl müşterinin isteğine bağlı ilerleyeceğinin planları yapılır.

2-Mimari Model: Sistemin ve alt sistemlerin oluşturulması ve bu sistemlerin sınanması işlemlerine ilişkin işlevler belirlenir.

3-Gerçekleştirim Modeli: Kodlama aşamasına ait modüllerin koda dökümü ve bu modüllerin sınanmasına ilişkin fonksiyonlar oluşturulur.

Çağlayan modeline benzer şekilde analiz aşamasında mümkün olan her detayın tasarım aşamasına geçirilebilmesi için müşteri dokümantasyonunun ve sistem gereksinimlerinin en ince ayrıntısına kadar belirlenmesi gerekmektedir fakat bu modelde müşteri çağlayan modeline göre daha fazla işin içindedir. Bu açıdan iki model farklılık gösterir.

Daha çok BT(Bilgi Teknolojileri) projelerinde kullanıma uygundur.

Helezoik(Spiral) Model: İsminden de anlaşılacağı üzere bu modelin gösterimi içten dışa doğru büyüyen bir spiral şeklindedir. 4 adet temel gerçekleştirimden oluşur:

1-Planlama: Ana ürüne ulaşmadan önce prototip amacıyla geliştirilen ara ürünler bulunur. Bu aşamada üretilecek ara ürünün planlaması yapılır ve bu aşamanın ilk kez gerçekleştirilmesi hariç diğer tekrarlarda önceki ara ürünler ile yeni ara ürünün birleştirilmesi yapılır.

2-Risk Analizi: Risk kategorisine girebilecek her türlü durum araştırılıp analiz edilir ardından hangilerinin gerçek bir risk durumu oluşturup oluşturmayacağına bakılır.

3-Üretim: Planlanan ara ürünün üretildiği kısım.

4-Müsteri Değerlendirmesi: Çıkarılan ara ürün-prototipin müşteri tarafından test edilmesi ve değerlendirmesi bu aşamada gerçekleştirilir.

Risk analizi kavramı bu modelde diğer kavramlara göre daha çok öne Diğer modellerdeki gibi direkt olarak yazılım yaşam döngüsü temel adımlarını izlemez. Yukarıda açıklanan aşamalar takip edilir. Bu 4 temel aşamanın her döngüsü bir “fazı” ifade eder.

Artımsal Geliştirme Süreç Modeli: Bu modelde teslim işlemi bir seferde gerçekleşmez. Birden fazla parçaya bölünmüştür. Dokümantasyonda istenen işlevlerden biri çalışır hale getirilip teslim edilir. Her teslim önceki teslimdeki işlevleri de kapsamalıdır. İstenilen ana ürün ortaya çıkana kadar bu geliştir-teslim et döngüsü devam eder.

Bu teslim edilen işlevlerin sırası müşterinin isteğine göre belirlenir. Bir işlev üzerinde çalışılmaya başladığında o an yeni bir istek istenmez, istenirse de diğer teslimatta bu istek dahil edilerek teslim edilir.

Bu model uzun zaman alabilecek ve eksik işlevde çalışabilecek projelerde kullanılabilir. Çünkü aynı anda hem kullanılıp ve test edilip hem de geliştirmeye devam edilmesi gerekmektedir.

Kodla ve Düzelt Yaşam-Döngü Modeli: Birkaç yüz satırdan oluşan küçük çapta geliştirilen programlarda kullanılır. Hiçbir doküman yazılmadan direkt kodlama aşamasına geçilir. İlk önce temel olarak birkaç işlevi gerçekleştirebilen bir prototip hazırlanır daha sonrasında müşterinin isteklerinin tamamı karşılanana kadar program üzerinden oynanmaya devam edilir. Teslim sonrası bakım aşaması ise herhangi bir doküman bulunmadığından dolayı çok zorludur.

Herhangi bir hata veya bug oluşması durumunda bu hata veya bugun düzeltilmesi çok maliyetlidir. Buna karşılık en kolay yazılım geliştirme bu modeldir. Bu kolaylıktan dolayı genelde küçük çaptaki firmalar ve tecrübesiz ekipler ve şirketler tarafından kullanılır.

Buraya kadar olan bölüm geleneksel yazılım geliştirme bölümüydü. Buradan itibaren çevik yazılım geliştirme bölümüne geçiyoruz.

XP(Xtreme Programming): XP kullanımı kolay, takım arkadaşları arasındaki iletişime önem veren ve müşterinin geri dönüşlerinin daha fazla olması yönüyle öne çıkan bir yazılım geliştirme yöntemidir.

XP’nin 4 adet önem verdiği konu vardır:

1-İletişim: Günümüzde yazılım geliştirme haricindeki olaylarda bile iki taraf arasındaki temel sorunun iletişimsizlik olduğu görülür. Bu iki taraf arasındaki sorunun en sorunsuz ve hızlı bir biçimde çözülmesi için sağlıklı bir iletişim şarttır. Yazılım geliştirme ekiplerinde de bu böyledir. Projelerin sorunsuz ve hızlı bir şekilde teslim edilmesi için ekip arkadaşları arasında sağlıklı bir iletişim olması gerekir. XP ise tam olarak bu beklenileni sağlayarak ekip arkadaşları arasındaki bu iletişimsizliği ortadan kaldırmayı amaçlar. Ekip arkadaşları arasındaki bu ilişki aynı şekilde ekip ve müşteri arasında da olmalıdır. Bu sayede eğer bir sorun ortaya çıkmışsa bu sorunun erken safhada farkedilmiş olur.

2-Basitlik: Basitlik her ne kadar isim anlamı nedeniyle kolay görünse de bunu bir projeye entegre etmek aslında zorlu bir iştir. Buradaki basitliğin anlamı gereksinimlerin karşılandığı en temel çözüm yolunun kullanılmasıdır. Bir problemi karmaşık bir biçimde çözmek XP’nin doğasına aykırıdır. Bu nedenle XP problem çözümlerini olabildiğince esnek ve basit bir sistemle gerçekleştirmeye çalışır.

3-Geri Bildirim: Geri bildirim hem yazılım geliştiriciler için hem de müşteri için paha biçilemez bir nimettir. Bir problemin farkedilmesi ne kadar erken olursa bu hatadan geri dönüş çok daha az maliyetli ve çözümü çok daha kolay ve hızlı gerçekleşir. Bu nedenle XP müşteriyi yazılım geliştirme ekibine dahil olarak kabul eder.

2-4 hafta arasında belirli sürümler üretilerek müşterinin görüşü alınır ve gelinen nokta gözden geçirilir. Bu sayede doğabilecek anlaşmazlıklar erken safhada halledilmiş olur.

4-Cesaret: Kavranması ve benimsenmesi açısından diğer 3 maddeden çok daha zorlu bir konudur. Yapılabilecek hatalardan ve işin sonunda başarısız olmaktan korkmayıp hatanın ve başarısızlıkların nedenini kendi içinizde kavramak çok daha mantıklı bir hareket olacaktır. Eğer şu ana kadar yaptığınız iş veya geldiğiniz nokta sizi memnun etmiyorsa onları yok etmekten çekinmeyin. Başarısızlık yaşamaktan korkmak psikolojik olarak da sizi çok etkileyecek bir durum olup verimden düşmenizi ve yavaş bir şekilde hareket etmenize neden olur.

Bu 4 temel konu dışında XP daha kolay bir şekilde yazılım geliştirmek için 12 adet de pratik yönteme sahiptir.

1-Planlama Oyunu: Planlama oyunu, müşterinin de bulunduğu bir toplantıda ekip üyelerinin yazılımın parça parça tesliminin hangi aralıklarla (3gün-1ay vs.) yapılacağı ile ilgili bir tahminde bulunmasıdır. Bu şekilde hem ekip olarak ortak bir noktada buluşulacak ve müşterinin kafasında hangi aralıklarda neyin teslim edileceğini belirlemek daha kolay olacaktır.

2-Ekipte Müşteri: Müşterinin istek ve fikirlerine daha kolay ve hızlı ulaşmak için müşteriyi ekipten biri olarak kabul etmemiz gerekir. Bu sayede hem müşterinin isteklerine hem de yazılımcının ihtiyaç duyduğu bilgilere ulaşımı çok daha kolay olacaktır.

3-Önce Test: Yazılımcı kodu yazmadan önce yazılacak bu kodu test etmek amacıyla başka bir program yazılır. Bu sayede kodda ortaya çıkan bir problemi erken aşamada belirleyerek daha güvenli bir yazılım ortaya koymak amaçlanır.

4-Basit Tasarım: Bu pratik, fazla bir karışıklığa yol açmadan müşterinin isteklerini karşılayacak en temel tasarımın yapılmasını amaçlar. Bu şekilde yaptığınız iş, kafa karışıklığına yol açmadan kısa sürede anlaşılabilir ve üzerinde değişiklik yapılması daha kolay bir yazılıma dönüşür.

5-Çiftli Programlama: Ekibin her üyesi sahip olduğu bilgi birikimi ve deneyimleri sayesinde ürünün ortaya çıkmasında katkı verirler. Fakat çoğu zaman üyelerden biri diğerinden daha hızlı bir şekilde ortaya çıkan sorunu çözmektedir. Bu da programın yavaşlamasını engellemektedir.

Aynı zamanda bu pratiğin başka bir avantajı da farklı bakış açılarından bakan iki kişinin birbirlerinin eksikliklerini giderebilmesidir. Bir üyenin göremediği bir eksiği diğer üye çoğu zaman görebilmektedir.

Çok fazla iş deneyimi olmayan yeni yazılımcıların yetişmesi için de çok kullanışlı bir yoldur.

6-Sürekli Entegrasyon: Sistemde yapılan her yeni değişiklik ve her yeni bileşen anında sisteme entegre edilip gün sonundaki derlemelerle test edilir. Sürekli bir entegre edilme halinde olunması sayesinde diğer üyeler bu sistem üzerinde yapılan değişiklikleri erkenden görür ve olası bir sorun ve eksiklik halinde bu sorun ve eksikliklerin erkenden tespit edilmesini sağlar.

7-Kısa Aralıklı Sürümler: Bu pratikte ana proje belirli zaman aralıklarına (2-4 hafta) bölünür. Her bir periyot kendisine ait yapılacak bir iş ve bir son teslim tarihi bulundurur. Son teslim tarihine proje takımı elinden geldiğince uymaya çalışır ve bu sürenin sonunda ortaya çıkan prototip ürün müşteriye teslim edilir. Bu şekilde müşterinin gelişimi takip etmesi de büyük bir ölçüde kolaylaşır.

8-Yeniden Yapılandırma: Kod ve tasarım “müşteriden gelen yeni istekleri mevcut şart ve durumlara bakılarak nasıl geliştirebiliriz?” sorusuna cevap aramak amacıyla sürekli olarak bir gözden geçirilme halindedir. Buradaki ana fikir müşteri memnuniyetini sağlamaktır. Bu sebeple yeniden yapılandırılma kullanılır.

9-Ortak Kod Sahiplenme: Yazılım geliştirme tek kişilik bir iş değildir. Aksine bir ekip işidir. Bu nedenle yapılan iş ve yazılan kod bütün ekibin ortak malıdır. Bir yanlış gördüğümüzde o kişiyi direkt olarak suçlamak yerine yazılan koda erişip kodu nasıl daha iyi bir noktaya taşıyabilirim düşüncesiyle yaklaşılmalıdır.

10-Benzetim (Metafor): Yazılımın içindeki bölünen sistemlerin birbirine benzetilmeye çalışılarak yazılımın geliştirilmesidir.

11-Kodlama Standardı: Ekipte kaç tane üye varsa bu üyelerin hepsi ekip içinde önceden tanımlanmış bazı kod yazım kurallarına uymalıdır. Bu kurallara örnek vermek gerekirse isimlendirme şekli, yorum satırı ve açıklamalar, dizi kullanımı vb.

Bu kuralları koymaktaki amaç kodun anlaşılabilirliği ve okunabilirliğini kolaylaştırmak ve koddaki karmaşıklığı azaltmaktadır.

12-Haftada 40 Saat: Ekip içindeki kişilerin birer makine değil bir insan olduğunu unutmamak gerekir. Yapılan işin ve üyelerin verimliliği açısından günlük 8, haftalık ise toplamda 40 saati aşmaması uygun görülür. Eğer aşılsa bile verimlilik açısından beklenilen kadar bir artış olmaz aksine üyelerin yaşayacağı dikkat dağınıklığı ve yorgunluk nedeniyle yapılan hata sayısında artış meydana gelmesi riski de artar.

Scrum: Scrum, 1990ların ortalarında Jeff Sutjerland ve Ken Schaber tarafından geliştirilmiş bir çevik yazılım geliştirme fikrine uygun bir proje yönetimi fikridir. Bu nedenle kullanım alanı sadece yazılım geliştirme ile sınırlı değildir. Karmaşık olan ana projeyi daha küçük parçalara bölerek yani bir nevi “böl ve fethet” stratejini kullanarak yazılımı geliştirilmesini destekler. Adını ise Rugby (Amerikan Futbolu) sporundaki bir atak taktiğinden alır. Aynı zamanda bu modelde bir parti teslim 30 günü aşmaz ve günlük 15 dakikalık toplantılar düzenlenerek iş takibi sürekli olarak yapılmaktadır.

Gereksinimlerin kolay belirlenemediği ve kafa karıştırıcı durumların olması beklenen projelerde en uygun modeldir.

Scrumda 3 temel kavram bulunur:

1-Roller:

1.1-Ürün Sahibi: Ürüne yapılan geri dönüşlerden sorumlu olan kişi. Bu kişi müşterinin kendisi ya da müşteri temsilcisi olabilir.

1.2-Scrum Yöneticisi: Takım üyelerinin Scrum kurallarına bağlı kalmasını denetleyen kişidir. Aynı zamanda üyeleri Scrum ortamına adapte etmeye çalışan kişidir.

1.3-Scrum Takımı: Yukarıdaki 2 maddeyi de içine alan genel bağlamda sürekli bir şekilde iletişimde kalan ve tek bir problemi çözmeye odaklanan ekip üyelerinden oluşur.

2-Toplantılar:

2.1-Sprint Planlama: Geniş kapsamlı bir gereksinim listesi çıkarılıp ekip üyeleri alt takımlara bölündükten sonra alt takımların her birine bir iş dağıtımı gerçekleştirilir. Bu takımlar kendi içinde bazı riskleri değerlendirir ve bu risklerin kontrollerinin nasıl yapılacağını belirler. Geliştirmenin altyapısı ve araçları belirlenir ve onaylanır. Dağıtım, geliştirme ve pazarlama giderleri hesaplanır.

2.2-Sprint Gözden Geçirme: Her sprint yapılmadan önce scrum takımı kendi arasında bir toplantı gerçekleştirir. Toplantı iki kısımdan oluşur. İlk kısımda ürün sahibi ile scrum takımı gereksinim listesini tekrar gözden geçirir ve diğer toplantıya kadar olan hedefler belirlenir. Toplantının ikinci kısmında ise takım üyeleri ana gereksinim listesinin en başından başlayarak sprintin sonuna kadar gereken gereksinimler belirlenerek yeni bir sprint gereksinim listesi belirlenir.

2.3-Günlük Scrum Toplantısı: Sprinte başlandıktan sonra takım üyeleri günlük olarak 15 dakikalık bir toplantı gerçekleştirir. Bu toplantıya tüm takım üyelerinin katılması zorunludur. Tüm takım üyeleri dün ne yaptıklarını ve bugün ne yapacakları ile rapor verirler. Takım üyelerinin her birinin karşılaştığı sorun ve engeller üzerinde bu sorunların nasıl çözüleceği ile ilgili beyin fırtınaları gerçekleşir.

3-Bileşenler/Araçlar:

3.1-Ürün Gereksinim Dokümanı: Projenin başından projenin teslimine kadar olan süreçte gereken iş elemanlarının tutulduğu kapsamlı bir dokümandır. Bu doküman canlı bir dokümandır. Üzerinde eklemeler ve değişiklikler yapılabilir.

3.2-Sprint Dokümanı: Mevcut sprintin yapılacakları ile ilgili bilgilerin tutulduğu dokümandır. Buradaki bilgilerin amacı sprintin sonuna kadar bitirilmesi ve çalıştırılabilmesi gereken işlevi elde etmektir. Bu doküman üzerindeki değişikler sadece bu dokümana sahip scrum takımı tarafından gerçekleştirilebilir.

3.3-Sprint Kalan Zaman Grafiği: Bu dokümanın amacı sprintte yapılması planlanan ve asıl yapılan işlerin kıyaslanmasıdır. Aradaki farka göre takımın hızlanması ya da yavaşlanması planlanır.

Scrum Günümüzde Neden Popüler? : Karmaşık yapıları böl ve fethet stratejisini kullanmak amacıyla basit yapılara indirger. Müşteri ile yapılan toplantıların sonrasında gereksinim listesini daha anlaşılır bir hale getirir. Ekip arasındaki iletişime ve üyelerin motivasyonunu üst düzeyde tutmaya dikkat etmesi sebebiyle şirketler için güzel bir etki yaratır. Kısa aralıklarla yapılan toplantılar ile de gelişimi görmekte ve proje üzerindeki kontrolleri sağlar. Bu gibi sebepler sayesinden Scrum şirketler ve ekipler arasında popülerliğinde hızlı bir artış görülmektedir.