**Ad, Soyad: Suna BUĞDAY**

**Numara: 210601027**

İlk kısımda, yazılım yaşam döngüsünü iyice açıklayabilmek için “Yazılım nedir?” ve “ Hangi alt başlıklar altında toplayabiliriz?” in kısa bir açıklamasına yer verilecektir. Yazılım; bilgisayara, gerçekleşecek işlemler dizisinin bilgisayarın nasıl yapması gerektiğini anlatıp yönlendiren komutları içerir. Yazılım türleri, sistem yazılımları ve uygulama yazılımları olarak ikiye ayrılır. Sistem yazılımı, bir bilgisayarın donanım ve uygulama programlarını çalıştırmak için tasarlanmış bir bilgisayar programı türüdür. İşletim sistemi, sistem yazılımının en iyi bilinen örneğidir. İşletim sistemi, bir bilgisayardaki diğer tüm programları yönetir. Diğer sistem yazılımlarına örnekler ise BIOS, önyükleyici programı, aygıt sürücüsü, derleyici ve yorumlayıcıdır. BIOS, bilgisayar açıldığı anda işlemciye tüm diğer donanımları sırasıyla tanıtır. Önyükleme programı, işletim sistemini bilgisayarın ana belleğine veya rastgele erişim belleğine(RAM) yükler. Aygıt sürücüsü, klavye veya fare gibi bilgisayara bağlı belirli bir aygıt türünü denetler. Ayrıca sistem yazılımı, disk birleştirici ve Sistem Geri Yükleme gibi sistem yardımcı programlarını hata ayıklayıcılar gibi geliştirme araçlarını da içerebilir. Uygulama yazılımı veya kısaca uygulama, kullanıcı için belirli görevleri gerçekleştiren yazılımlardır. Verilebilecek birkaç örnek ise eğitim yazılımları, işletim uygulamaları ve eğlence yazılımlarıdır. Yaşadığımız bu çağa “yazılım çağı” demek pek de yanlış olmaz. Çünkü artık yazılım neredeyse her alanda, her yerde etkisini ve önemini artırmaya devam ediyor. Var olduğu yerde de daha sonrası için yazılımın ve getirdiği kolaylıkların olmadığı bir dönemi faydalanan kimse düşünmek bile istemiyor. Kısacası yazılım ile iç içe geçmiş bir bugünden söz ediyoruz. Teknoloji gelişmiş olmasına rağmen birçok proje başlamadan bitebiliyor. Bunun en temel sebepleri ise geliştirilen uygulamanın yeterli güvenliği sağlamıyor olması, geliştirilen yazılımın kalitesizliği ve yazılım sürecinin iyi yönetilememesidir. Yazılım geliştirmek için ise belli süreç ve zamana ihtiyaç vardır. Tasarlanan yazılımın hem üretim hem de müşterideki kullanım süreci boyunca geçirmiş olduğu tüm evreye “yazılım yaşam döngüsü” denir. Yazılım aşamasında bir hata veya bir eksiklik olursa döngü ya tamamen dönülür ve hata o şekilde düzeltilmeye çalışılır ya da en baştan başlanılır.

**Yazılım Yaşam Döngüsünün Evreleri**

1-Planlama: Analiz edilen gereksinimlerin uygulanması ve projenin asıl hedeflerine ulaşabilmesi için ekip projesinin fizibilitesinin belirlendiği ve proje planının tasarlandığı adımdır. En düşük riski göz önünde bulundurarak başarıya ulaşmanın yolları çizilir.

2-Analiz: İhtiyaç ve beklentilerinin (gereksinimlerin) ayrıntılı şekilde çıkarılması aşamasıdır. Bu analizin amacı yapılacak işlerin doğru algılanıp algılanmadığının değerlendirilmesidir.

3-Tasarım: Analiz edilen gereksinimleri gerçekleştirecek yazılım sisteminin yapısı, modelleri oluşturulur. Kodlamayı içermez. İki başlığa ayrılabilir: A)Üst seviye ve mimari tasarım: Sistem yapısını içerir.(akış şemaları, modüller…). B) Detaylı tasarım: Yazılımı içeren bileşenler ve bunların detayları vardır. (veri yapısı, ekran tasarımı,..)

4- Gerçekleştirim: Sistemin yaşatılmaya başladığı süreçtir. Tasarım detaylarına uygun yazılım geliştirilmeye çalışılır. Kodlamalar yapılır, veri tabanı oluşturulup raporlar hazırlanır.

5-Entegrasyon ve Testler: Geliştirme aşamasında elde edilenler kontrol edilir ve taleplere uygunluğu test edilir. Bir nevi sistem hayata geçirilir. Aynı zamanda donanımların temin edilmesi, bağlantılı olduğu birimlerin entegre edilmesi ve başka yazılımlarla entegrasyonu varsa bunların sağlanması ve de veri tabanı bağlantılarının yapılması gibi birçok aşama burada ele alınır.

6-Uygulama: Ürünün kullanıcılar tarafından kullanılması aşamasıdır. Piyasaya çıkmadan hataların, eksikliklerin değerlendirilmesi ve varsa yeni taleplerin alınması sağlanır.

7-Bakım: Teslimden sonra hata giderme ve yeni eklemeler yapma aşamasıdır. Bu aşama yazılımın ömrü boyunca sürer.

Yazılım Geliştirme Modelleri

1-Gelişigüzel Model: Herhangi bir kuralın bulunmadığı ve kuralların kişiye bağlı olduğu modeldir. 60’lı yıllarda tek kişinin ürettiği projeler bu yöntemle hayata kazandırılmıştır. Basit projeler için kişi isterse kullanabilir.

2-Barok Model: Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği modeldir. 70’li yıllarda kullanılmıştır. Bu modelde dokümantasyon(belgeleme) ayrı bir süreç olarak ele alınır. Yazılımın geliştirilmesi ve test adımından sonra yapılması öngürülür. Modelin aşamaları arasında geri dönüşlerin nasıl yapılacağı tanımlı değildir. Tercih edilmemesinin sebeplerinden en önemlisi de gerçekleştirim aşmasına daha çok zaman ayrılmasıdır.

3-Şelale Modeli(Waterfall Modeli): Geleneksel model olarak bilinir. Şelale modeli ardışık bir yazılım yaşam döngüsüdür. Bu modelde adımlar aşama aşama ilerler. Bir aşama bitmeden diğer aşamaya geçilmez. Her adım bir gözden geçirme işleminden geçmelidir. Diğer basamağa geçmek için dokümantasyon yazılmalıdır. İyi tanımlı projeler ve çok fazla zaman gerektirmeyen projeler için uygundur. Onun dışında bu modelin tercih edilmeme sebepleri yazılımın kullanıcının kullanımına sunma süresi oldukça uzundur. İhtiyaç belirlemeleri her ne kadar geniş kapsamlı yapılmaya çalışılsa da bu durum pek mümkün değildir. Bu da sistemin maliyetini artırır. Bir başka sebep ise bu modelde ürün elde etme süresi bir hayli uzun olduğu için kod yazmaya daha fazla önem verirler. Hata yapılmaya devam edilir böylece.

4-V Süreç Modeli: Bu modelde V harfinin sol kolu üretim sağ kolu ise test aşamasından oluşur. Bu modeli kullanabilmek için ihtiyaçlar çok iyi tanımlanmış ve doküman haline getirilmiş olmalıdır. Kısa süreli projelerde kullanılabilir. Bir uygulama test aşamasındayken geriye dönmek zordur. Kodlamadan sonraki aşamalar testleri içerir. Ve bu testler kodlama aşamasına gelinceye kadar geçen adımların karşılıklarına denk gelen sağlama testleridir.

5-Artırımlı Model: Bir müşteri eğer ürünlerinde değişikliğe ihtiyaç duyacak olursa artırımlı model kullanması mantıklı olabilir. Bu modelde yazılım bölüm bölüm geliştirilir ve teslim edilir. Uygulama aşamasından sonra 2 aşama vardır. Biri canlıya geçiş diğeri ise testleri kapsar. Canlıya geçiş de bir nevi test işlemidir. Böylece bir taraftan üretim bir taraftan kullanım yapılır. Gereksinimleri karşılan temel bir sistem geliştirilir. İlerlemeler ölçülür. Erken artırımlar prototip gibi davranır. Böylece gereksinimlerin daha iyi anlaşılması sağlanır. Değişen gereksinimli projeler için maliyetli olabilir. Aynı zamanda bu model risk yönetimini kolaylaştırır. Tüm bunlarla birlikte projelerin direkt batmasının önüne geçmiş olunur.

6-Helezonik(Spiral) Model: İteratif bir modeldir. Bu iterasyon planlama/risk analizi/üretim/kullanıcı değerlendirmesini kapsar. Yazılımı kullanacak müşteriler sürece erken dahil olur ve ileride istenmeyecek durumla karşılaştıklarında da bunu önceden test etme şansları elde ederler. İsterler planlanır, risk analizi yapılır. Bu aşamada prototip 1 oluşur. Bu prototip müşteriye sadece projeye nasıl yaklaştıkları hakkında fikir vermesi içindir. Daha sonra bu prototip üzerinden tekrar gereksinim analizi yapılır. Analiz üzerinden teknik tasarım yapılır. Yazılım geliştirilip test edilir. Prototip 2 oluşur ve müşteri ile değerlendirme yapılır. Yapılan işten müşteri memnun olduğu sürece proje hep bu şekilde ilerler. Asıl ürün, onaylanan son prototip üzerinden şekillenir ve hayata geçer. Spiral model büyük projeler için uygundur. Küçük ve düşük riskli projeler için uygun değildir. Çünkü küçük projeler için maliyet fazla olacaktır. Ara aşamalarda devamlı dokümantasyon istenir. Aynı zamanda spiralin ne zaman biteceği belli değildir. Bunlar da maliyetin artmasına sebep olacaktır.

7-Çevik (Agile) Model: Bundan önceki metotlar bazı uygulamalarda veya farklı sektörlerde işe yarabilecek olsa da yazılım mühendisliği için pek uygun değillerdi. Yazılım için yaptım oldu bakışı çok uygun değil. Sürekli geliştirmek ve yeni eklemeler yapmak gerekiyor. Geleneksel olanlar agile’a göre katıydılar. Agile metodolojisinde ise plandan çok insana odaklanma hakimdir. Gereksinimler ve çözümler kendi içlerinde düzenli olan farklı grupların ortak çalışmalarıyla gelişir. Çevik modellemesinin başlıca özelliği veri modelleri ve ara yüzü modelleri gibi modelleme tekniklerinin neler olduğunu ve bunların ayrıntılarını söylemek yerine bu tekniklerin nasıl uygulanması gerektiğini söylemesidir. Örneğin yapılan projelerin test edilmesi gerektiğini söylemiş olsa bile nasıl test hazırlanacağına değinilmemiştir. Çevik modelde en temel şey etkili ve hızlı bir şekilde müşteri ihtiyaçlarını karşılayabilmek, her türlü değişikliğe adapte olunması konusunda yol göstermektir. Çevik geliştirme teknikleri, süreç ve belgeleme yerine doğrudan yazılımın kendisine yoğunlaşan ve yinelemeli geliştirmeye dayanan tekniklerdir. Beklentilerinin sürekli değişimine cevap olarak doğmuştur. 2001 yılında çevik yazılım geliştiriciler bir araya gelip çevik yazılım geliştirme manifestosunu ve prensiplerini yayınlamışlardır. Manifestosu : Süreçler ve araçlardan ziyade bireyler ve etkileşimlere, Kapsamlı dokümantasyondan ziyade çalışan yazılıma, Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine, Bir plana bağlı kalmaktansa değişime karşılık vermeye değer vermek.

Prensipleri ise Yazılımın erken ve devamlı teslimin sağlayıp müşteri memnuniyetini sağlamak Değişen gereksinimleri yazılım sürecinin sonunda olsa bile kabul etmek Çalışan yazılımı kısa zamanlar aralıları belirlenerek birkaç haftada ya da birkaç ayda bir düzenli olarak müşteriye sunmak İş süreçlerinin sahipleri ve yazılımcılar proje boyunca her gün birlikte çalışmalıdır Bir yazılım takımında bilgi alışverişinin en etkili yöntemi yüz yüze iletişim kurmaktır. Projelerin temelinde motive olmuş bireyler yer almalıdır. Onlara güvendiğimizi hissetmeliler. Çalışan yazılım ilerlemenin ilk öncülüdür. Sadelik olmazsa olmazdır. Kendinden örgütlü takımlar yer almalıdır. Avantajları için şunlar söylenebilir: Küçük ve orta çaplı projeler için iyi bir seçimdir. Sabit ve değişen gereksinimler için uygundur. Yönetmesi kolaydır ve esneklik sağlar. Dezavantajları için ise şunlar söylenebilir: Kurumsal bir yapıda uygulaması zordur. Dokümantasyon minimum seviye oluşturulduğu için bireysel bağımlılık çok yüksektir. Dokümantasyon eksikliği olduğu için de projeye yeni ekip üyelerini dahil etmek zor olabilir. Sürdürülebilirlik için fazla risk vardır.

AGİLE(ÇEVİK) GELİŞTİRME YÖNTEMLERİ

1-Extreme Programming XP-(Sınırsal Programlama)/ 2-Scrum /3-Kristal Yöntem /4-Dinamik Sistem Geliştirme Yöntemi(DSDM)(Dynamic System Development Method) / 5-Özellik Güdümlü Geliştirme(Feature-Driven Programming-FDD) /6-Lean Development / Microsoft Solution Framework(MSF)

1-Extreme Programming XP (Sınırsal Programlama)

Sınırsal programlamayı besleyen değerler iletişim, basitlik, cesaret, geribildirimdir.

A)İletişim: Yazılımcı ile birlikte müşteri de ekibin bir parçasıdır. Ekip üyeleri bir araya gelerek( yüzyüze) bilgi paylaşımı yapar. Sistemin gelişmesini sağlarlar. Test aşamasında yer alan kişi de orada bulunabilir. Bu da müşterinin değerlendirmesinden geçmesinin bir başka adımı olur. B)Basitlik: XP’de en önemli şey sadeliktir. Sadelik projenin hızlı bir şekilde ilerlemesini sağlar. C)Cesaret: Kent Beck, cesareti “korku karşısında etkili bir eylem” olarak tanımlamıştır. Basit bir örnekle işe yaramayan bir şeyi yapmayı bırakıp başka bir şey denemek için cesarete ihtiyaç duyarız. D)Geribildirim: Testler birer geribildirimdir. Yazılan en küçük kod parçası bile test edilerek yazılım en başından kontrol altında tutulur(birim testi). Daha sonra ise birimler bir arada test edilir (birleştirme testi).Sonraki aşama ise sistem testidir. Bir başka geri bildirim yolu ise bizzat yazılan koddur.

XP’ de var olan pratikler, ekiplerin kendilerini değerlere karşı sorumlu tutmasına yardımcı olur. Bu pratikler ise Planlama Oyunu, Ekipte Müşteri, Önce Test, Basit Tasarım, Çiftli Programlama, Sürekli Entegrasyon, Kısa Aralıklı Sürümler, Yeniden Yapılandırma, Orta Kod Sahiplenme, Metafor, Kodlama Standartı, Haftada 40 saat.

1. Planlama Oyunu: Bir yineleme döngüsünün başında gerçekleşen bir toplantıdır. Geliştirme ekibi ve müşteri, bir ürünün özelliklerini tartışmak ve onaylamak için bir araya gelir. Planlama oyunun sonunda, geliştiriciler her biri için görevler atayarak yaklaşan yineleme ve yayın için plan yaparlar. Bu aşamanın temel amacı projenin tamamlanmasını sağlamaya çalışmaktır.
2. Ekipte Müşteri: Müşterinin varlığına ihtiyaç duyulmaktadır. Çünkü yazılımcının ihtiyaç duyduğu bilgilere daha kolay ulaşmasını sağlar.
3. Önce Test: Kod yazılmadan önce test programı yazılır ve test edilmesi gerekir. Böylece meydana gelebilecek herhangi bir soruna karşı hızlı önlem alınmış olunur.
4. Basit Tasarım: Müşterinin ihtiyaçlarını karşılayabilecek en basit tasarım gerçekleştirilir.
5. Çiftli Programlama: Projelerde iki yazılımcı bir bilgisayarda çalışır. Böylece birbirlerinin deneyim ve tecrübelerinden faydalanmış olurlar (usta-çırak). Bu durum yazılım kod kalitesinin artmasına olanak sağalr. Proje üzerinde çalışan yazılımcıların farklı bakış açıları projeyi hızlandırır.
6. Sürekli Entegrasyon: Projeyi geliştirme aşamasında yapılan değişiklikler hemen test edilerek sisteme entegre edilir. Bu şekilde sistem sürekli ve düzgün bir şekilde gelişir.
7. Kısa Aralıklı Sürümler: Proje için yapılan değişikliklerde her küçük işlev için sürüm çıkarılır. Bu sürümler müşteri tarafından test edilir. Taleplerin karşılanıp karşılanmadığına bakılır. Böylece proje kontrollü bir şekilde sürdürülür.
8. Ortak Kod Sahiplenme: Kodlarda bireysellik yoktur. Proje ekibi kodlar üzerinde ortak sorumluluğa sahiptir.
9. Metafor (Benzetim): Belirli niteliklere sahip basit bir tasarımı ifade eder. İlk olarak bir tasarım ve yapısı yeni insanlar için anlaşılır olmalıdır. Örneğin bir para transfer sistemi için olabilecek metaforlar: hesap, işlem, borç ve para birimidir.
10. Kodlama Standartı: Proje ekibindekilerin kod yazmak için aynı stil ve formatları kullanmaları, ortak dilde bir yazılım geliştirmeleri beklenir. Bu da projeye daha sonradan dahil olacak kişilerin de kolay entegre olmalarına olanak sağlar.
11. Haftada 40 Saat: XP projeleri, ürün kalitesinin sürdürülebilir olmasını ve geliştiricilerinin daha verimli çalışabilmelerini gerektirir. Çalışanların kendilerini dinlenmiş ve zinde hissetmeleri, tükenmelerinin önüne geçilmesi için önemlidir. Haftada sadece bir kez fazla mesai yapılabilir. Bu da iki hafta üst üste olamaz.

2-SCRUM

Scrum, iteratif ve aşamalı bir çevik geliştirme yazılım yöntemidir. Scrumda proje en küçük parçalara bölünür. Görevler koşu adı verilen süreçlerle yönetilir. Koşu sayısı ürünün özelliklerine göre değişir. Çok küçük firmalar için de büyük firmalar için de uygundur. Ekipler ihtiyaç duyulduğunda küçültülüp büyütülebilir.

Scrumın iskeletini roller, seremoniler ve artifactlar(araçlar) oluşturur. Scrum rolleri ise ürün sahibi, scrum uzmanı, scrum ekibidir.

A)Scrum Uzmanı(Scrum Master): Bir yöneticiden ziyade lider özelliklere sahip olmalıdır. Yüksek performansa, verimliliğe sahip, motivasyonu yüksek ekiplerin kurulması için çabalar. Scrumun takibini yapar. B)Ürün Sahibi: Ekibin bir parçasıdır. Ürünü iyice tanımlar. C)Scrum Ekibi: Scrum ekibi küçük takımlardan oluşur. Ürün gereksinimlerini çalışan ürüne dönüştürmeye çabalar. Bu çaba, her koşul için geçerlidir.

Seremoniler(Olay)

1. Sprint Planlama: Koşuya başlanırken yapılacak olan ilk etkinliktir. Ne yapılacağı ve nasıl yapılacağı kararları verilir.
2. Sprint Gözden Geçirme: Önceki koşu tartışılır ve sonraki koşu için nasıl daha üretken olunacağı sorularına cevap bulmaya çalışır.
3. Günlük Scrum Toplantısı: Her gün yapılan toplantıdır. On beş dakika ile sınırlıdır. Şu üç soru yanıtlanabilmelidir: “Dün sen ne yaptın?”, “Bugün ne yapacaksın?”, “Herhangi bir sorun var mı?”

Scrum Artifactları (Araçlar)

1. Ürün Gereksinim Listesi ( Product Backlog): Genellikle kullanıcı hikayelerinden oluşur. Bu liste hiçbir zaman tamamlanmaz. Ürün geliştikçe gereksinim listesi değişiklik gösterebilir.
2. Koşu Dokümanı (Sprint Backlog): Scrum sprint sırasında tamamlanacak görevlerin bir listesidir.
3. Sprint Kalan Zaman Grafiği (Burndown Chart): İş bitim çizelgesi, sürat koşusunda tamamlanan iş miktarını ve kalan toplam işi gösterir. Günlük güncellenmelidir. Bu şema yardımıyla gerekli işlerin belirlenen zamanda tamamlanıp tamamlanmayacağı gözlenebilir.

**Kaynakça**

* caglartelef.com/yazilim-yasam-dongusu/
* Kılınç, Deniz, Yazılım Yaşam Döngü Modelleri, 1-46
* Kılınç, Deniz, Çevik Yazılım Geliştirme, 1-54
* Soylu, Salih, 2017, Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği ve Çevre ve Şehircilik Bakanlığı İçin Öneriler, 3-50
* muhendisportali.com/yazilim-yasam-dongusu/
* www.catsbilisim.net/yazilim-gelistirme-surecleri/
* fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/
* www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/
* agilemanifesto.org/iso/tr/manifesto.html
* www.codex.com.tr/yazilim-gelistirme-modelleri
* www.clouddefense.ai/blog/system-development-life-cycle
* digite.com/agile/extreme-programming-xp/