**YAZILIM YAŞAM DÖNGÜSÜ**

**Muhammed DENİZ 180 601 024**

Yazılımla üretilen programın, geliştirilmeye başlandığı ilk günden kullanıldığı son güne kadar geçirdiği bütün süreç “Yazılım Yaşam Döngüsü” olarak adlandırılır. Üretilen yazılım ürünü yazıldığı günün şartlarına göre en uygun yaşam döngüsü modelini örnek alarak oluşturulur. Yazılım yaşam döngüsü: güvenli, müşterinin isteklerini karşılayan, doğru çalışan, maliyeti düşük olan ve zamanında tamamlanan bir program oluşturmada önemli bir rol oynar. Yazılım yaşam döngüsünün geriye dönülebilir bir süreç olduğunu unutmamak gerekir. Bu sayede Herhangi bir aşamada yapılan bir hatayı sonradan fark edip o aşamaya geri dönmeyi kolaylaştırır. Yazılım yaşam döngüsü modellerini anlatmadan önce bu modellerin hangi aşamaları olduğunu ve bu aşamaların neler içerdiği anlatmanın daha doğru olduğunu düşünüyorum.

**Planlama:** Yazılım yaşam döngüsünün hayatına başladığı nokta diyebiliriz. Projenin temel hataları oluşturulur ve yazılım belirli bir temele oturtulur. Proje, yazılım veya program için neler gerekli, neler ne zaman kullanılacak, bütçe nerede dikkate alınacak gibi sorulara cevap aranarak yazılımın ilk aşaması tamamlanır. Aynı zamanda bu program veya proje için düşünülenlerin ne kadar uygulanabilir olduğu belirlenerek bu yazılımın sağlam bir plan dahilinde oluşturulması hedeflenir.

**Analiz:** Bu aşamada projenin ayrıntıları ve yazılım ürününün hangi işlevlere sahip olması gerektiğinin, kullanıcının hangi isteklerine karşılık verebileceği detaylı olarak belirlenir. Programın sistem gereksinimleri ve hangi taleplerin gerekliliği analizler sonucunda ayrıntılı bir şekilde belirlenir.

**Tasarım:** Tasarım aşamasına kadar uzanan süreçte proje detayları göz önünde bulundurularak proje parçalara ayrılır. Parçalara ayırmadaki en önemli nokta ise işi daha kolay halledebilmesini ve belirli uzmanlık alanlarına göre iş kolları oluşturmaktır. Parçalara ayrılan projeler için genelde bir işlem haritası ve proje planı oluşturulur. Bu aşamada aynı zamanda müşterinin istekleri dikkate alınarak yazılım ürününün temel yapısı, işlevselliği ve ara yüz tasarımı belirlenir. Tasarım aşamasında da dokümantasyon aşamasıyla da alakalı bir işlem gerçekleştirilir. Bu işlem Tasarım Dokümantasyonu olarak adlandırılır. Unutmamalıdır ki bir program oluşturulurken veya herhangi bir yazılım ürünü geliştirilirken ilk evreden son evreye kadar belgeleme yapılır. Yani bu evrenin içinde belgeleme olması özel bir durum değildir. Bu belgelerin oluşturulmasının birçok nedeni vardır. Burada kendi konumuzla alakalı olan tasarım dokümantasyonunu açıklayalım fakat diğer nedenleri de belgeleme alt başlığı altında açıklayacağız. Tasarım dokümantasyonunun oluşturulma nedeni: yazılım ürününün geliştirilme aşamasında veya geliştirildikten sonra bu yazılım ürününün geliştirilme ekibine katılan yeni yazılımcılara yol gösterici ve teknik ayrıntıları barındırıcı olmasıdır. Aynı zamanda programa en başından beri dahil olan birisinin büyük bir hatada başa dönerken yazılım özelliklerini hatırlaması konusunda da önemlidir. Tasarım aşamasını bitirmeden tasarımın mimari tasarım ve detaylı tasarım olmak üzere ikiye ayrıldığını bildirmekte fayda var. Mimari tasarımda yazılım ürünü daha yüzeysel olarak ele alınırken detaylı tasarımda mimari tasarımın ayrıntılarına inilir.

**Kodlama (Yazım):** Bundan önceki adımları tamamladıktan sonra geliştirme yani kodlama aşamasına geçilebilir. Bu aşamayla birlikte herhangi bir analiz yapılmasının artık bir anlamı yoktur. Projenin ilerlemesi elbette gelişigüzel değildir, daha önceki aşamalarda da oluşturulan plan dahilinde olur. Kodlama aşamasında yazılım ürününü oluşturan yazılımcının isteği ve daha çok program yapımına uygun olacak şekilde bir programlama dili seçilir. Programlama dili belirlendikten sonra derleyiciye yazılacak kodların belirli bir standartlar dahilinde olması gerekir. Bunun nedeni yazılan programın belli bir süre sonra yazılımcısının değişebilme olasılığı ya da programın bakım aşamasını farklı programcılar üstlenebilmesidir. Bu nedenle kodlama standartları önem taşır. Kodlama standartlarına örnek vermemiz gerekirse örneğin değişkenlere konuyla alakalı isim vermek veya Türkçe karakter kullanmamak. Yazılım yaşam döngüsünde test tek başına bir aşamadır ve genelde program yazıldıktan sonra yapılır. Bu durumu bozan bir unsurda kodlama aşamasında yapılan alfa testleri çünkü alfa testleri program yazılmaya devam edilirken yapılır.

**Test:** Programın yazılım aşaması sona erdikten sonra müşteriye teslim edilmeden önce beta testler yapılır. Bu testler sonucunda hata oluşursa bu hata giderilir. Şunu da söylemekte yarar var eğer bir programda bir hata bulur ve bu hataya sebebiyet veren kodları değiştirirseniz bu olayında belgelendirmesini yapmanız gerekir. Bunu yaptığınız taktirde daha sonra benzer bir hata aldığınızda ne yapmanız gerektiğini daha iyi bilebilirsiniz. Şunu da unutmamak gerekir ki bir programda hata giderilirken başka bir hataya sebebiyet verilebilir tam da bu noktada hatanın bağlantılarının yapılması için de hata belgelendirmesi önemlidir. Test aşamaları sadece yazılım ürününü oluşturanlar veya test ekibi tarafından gerçekleştirilmez. Bazı yazılım ürünleri bütün kullanıcıların erişimine sunulmadan önce belirli ve az sayıdaki kullanıcıya göndererek kullanılması istenmektedir. Bu kullanıcılar yazılım ürününde hata bulursalar bu hatalar giderilir ve yazılım ürünü kullanıcı kullanımına sunulur. Test aşamasında sadece program hataları giderilmez. Elbette program hataları önemlidir ama programın hatasız olması müşterinin veya kullanıcının istediği tek istek değildir. Daha öncede belirttiğimiz gibi programın sistem gereksinimleri, programın hızı ve bellek kullanımı da oldukça önemlidir. Test aşamaları tamamlandıktan sonra program teslimiyle birlikte detaylı bir kitapçığın verilmesinin de oldukça önemli olduğunu belirtmekte fayda olduğunu düşünüyorum.

**Bakım:** Bakım bir yazılımın yaşamı boyunca yani kullanıldığı süre boyunca devam eder. Yazılım ürünü test edildikten sonra kullanıcının kullanımına sunulur. Bakım aşaması, kullanıcının programı kullanmaya başladığı ilk günden başlar. Bu noktada bakım aşamasını iki bölüme ayırabiliriz. Birincisi güncelleme (yenileme) ve ikicisi hata giderme. Güncelleme yapılırken kullanıcının değişen zamanda yeni isteklerine cevap verebilmek amaçlanır. Hata giderme ise programda var olan bir özellik veya işlevin yapılamıyor olması durumunda sorunun giderilmesidir.

Bakımdan da bahsettiğimize göre program yaşam döngüsünü bitirmiş oluyoruz. Burada konu başlığı olmadığı halde değinmemiz gereken konu ise bu aşamaların hepsinin içinde olan belgeleme aşaması. Bir uygulama geliştirirken amatör veya üst düzey geliştirici olmanın önemi olmaksızın yapılan belgeleme, programı birçok açıdan daha iyi bir hale getirir. Programda ileride çıkacak sorunların daha hızlı çözülmesini sağlayan belgeleme aşaması aynı zamanda maliyeti de düşürmüş olur. Amatör yazılımlarda pek kullanılmadığı bilinse de ekiplerle geliştirilen projelerde vazgeçilmez bir parametredir. Özet olarak belgeleme yapılmayan proje rotası olmayan bir gemiye benzer.

**YAZILIM GELİŞTİRME MODELLERİ**

**Gelişigüzel Model: Herhangi bir sisteme bağlı kalmadan yazılım ürününü geliştiren yazılımcının inisiyatifine bağlı olarak geliştirilen modeldir. Bu modelin kullanılması aslında hiçbir model kullanmama seçeneğidir. Amatör olarak geliştirilen yazılım ürünlerinde sıkça görülmesine karşın ekiplerle geliştirilen ve şirketlerin geliştirdiği yazılım ürünlerinde görmek oldukça zordur. Bu modelin en sıkıntılı olduğu tarafı modelin sıkıntılarla karşılaştığı kısımlarda geri dönülmesinin mümkün olmamasıdır. Bu modelde daha öncede belirttiğimiz ve önemli olduğunu vurguladığımız belgelemenin de büyük bir çoğunlukla yapılmadığını söyleyebiliriz. İşte tam d bu yüzden oluşan hatalarda geriye dönmek oldukça zordur.**

**Avantajları:**

* **Bir planlama ve hazırlık aşaması olmadığı için zaman kaybı olmaz.**
* **Küçük çaplı projelerde fayda sağlar.**
* **Az kişi ve az ekipmana ihtiyaç vardır.**

**Dezavantajları:**

* **Planlı bir süreci yoktur.**
* **Hataları çözmek için planlı bir gidiş yolu yoktur.**
* **Bitiş süresi programcıya bağlıdır.**
* **Planlı bir süreç olmadığı için kodlarda geri dönüş çok daha büyük hatalar oluşturur.**

**Barok Modeli: Programlama yaşam döngüsü adımlarının doğrusal bir şekilde işlenmesiyle oluşan bir modeldir. Daha önce belgelemenin bir aşama olmadığını ve her adımda uygulanması gerektiği üzerinde durmuştuk. Barok Modeli belgelemeyi ayrı bir aşama olarak ele alır. Belgeleme eğer doğru zamanda yapılmazsa amacına ulaşmamış olur. Yani belgeleme zamanında yapılmayacaksa hiç yapılmaması daha doğrudur, en azından zaman kaybına sebebiyet vermez. Günümüzde çok fazla kullanılan bir yöntem değildir. Bunun sebebi ise az öncede belirttiğimiz gibi belgeleme gibi çok büyük sorunları barındırmasıdır. Aynı zamanda bu modelde bir aşamada yaptığınız bir hataya nasıl dönüleceği de tanımlanmamıştır. Bu modelde kodlama aşamasının programın en önemli aşaması olduğu belirtilmiş ve bu aşamaya fazlasıyla önem verilmiştir.**

**Avantajları:**

* **Her aşamada belgeleme ile uğraşılmadığı için zamandan kazanç eder.**
* **Kodlama aşamasıyla çok fazla uğraşıldığı için burada az hata görülür.**

**Dezavantajları:**

* **Zamandan kazanç için zamanında yapılmayan belgeleme hata çözümüne yardımcı olmaz**
* **Yazılım aşamaları arası geçişler belirsizdir.**
* **Belgelemenin sadece son kısımda yapılması hata ve geri dönüşlerde etkisiz kalır**

**V Modeli:** Bu modelde ters piramit şeklinde iki koldan adımlar ilerler. Genelde yapının sağ tarafı test adımlarını içerir, sol tarafı ise üretim adımlarını içerir. Genelde BT projeleri için gayet uygun bir modeldir. Bu modelde kullanıcının projeye katkısı oldukça önemsenmektedir. V Modelinin üç adet çıktısı vardır. Şimdi bunları sırayla açıklayalım. Kullanıcı Modelinde müşterinin verdiği notlar dahilinde kullanıcı ile olan ilişki tanımlanmaktadır. Mimari Model sistem tasarımını, sistem gereksinimlerini ve oluşabilecek alt sistemlerin sınamasını içerir. Sonuncusu Gerçekleştirim Modeli yazılım ürününün oluşumunu yani kodlanmasını içerir.

Avantajları:

* Kendi içinde de üç modele ayrıldığı için iş paylaşımını ve yönetimini kolaylaştırır.
* Proje takibi kolaydır.
* Proje planı ilk aşamalardan belirlenir.

Dezavantajları:

* Aynı anda proje devamlılığına imkân vermez.
* Tasarım aşaması doğrusal ilerlemez.
* Hata çözümlemelerinin ön planlaması yoktur.

**Çağlayan (Şelale) Modeli:** Bu modelde ilk olarak yazılım aşamalarına baştan sona bir ön izleme yapılır. Bu izlenim programın maliyetini arttırmakta ve süresini uzatmaktadır. Programa ilk olarak sistem aşamalarına inceleme yapılarak başlanır. Bu incelemenin sonucu olarak SRS (Yazılım İhtiyaç Özellikleri) dokumanı hazırlanır. Hazırlanan bu inceleme dokumanı kaynak alınarak programa bir tasarım hazırlanır. Tasarım bittikten sonra bir doküman daha hazırlanır bu dokümana da SDD (Yazılım Tasarım Dokumanı) adı verilir. Bu belgelendirmelerin ardından yazılım ürünü somut olarak işlenmeye ve program somut olarak görülmeye başlanır. Bu aşamadan sonra hataların ortaya çıktığını da söyleyebiliriz.

Avantajları:

* Anlaşılırdır ve kullanışı kolaydır.
* Model içinde farklı parçalar barındırdığı için ekiplerle hazırlanan projelerde kullanımı kolaydır.
* Orta derecede projelerde hem iyi çalışır hem de proje yönetimini kolaylaştırır.

Dezavantajları:

* Büyük projelerde sorunlara sebep verir.
* Hata oluşumunda maliyeti çok fazladır.
* Program yazımı bittikten sonra değişime elverişli değildir.
* Gelişime ve değişime elverişli olmadığı için müşteri memnuniyetini sağlamaz.

**Helezonik (Spiral) Model:** Daha önceki yazılım modellemelerini anlatırken hep hataların çözülmesi konusundaki yetersizlikten ve yazılım ürününün yenilenmesine imkân verememesinden bahsetmiştik. Programımızı geliştirirken kullanıcı hedefli bir model kadar yazılımcı hedefli bir model olması da önemlidir. İşte tamda bu noktada Helezonik Model hata geri dönüşlerinde ve program yenilemelerinde fazlasıyla kolaylık sağlar. Program, yenileme ve sürekli elde olan verileri arttırmaya yönelik oluşturulur. Prototip yaklaşımının da olduğunu söylemekte fayda var. Helezonik Model süreci 4 ana gruba ayrılır. Bu süreçler planlama, risk analizi, üretim, kullanıcı değerlendirmesidir. Bu 4 grubun tüm program oluşum sürecini kapsadığını hatırlatmakta fayda var. Bu 4 süreç var olan işi paylaşma değildir. Yani bu süreçlerden biri bitirilmeden diğerine geçmek pek de mümkün değil.

Avantajları:

* Müşteri veya kullanıcı yazılım ürünü tamamlanmadan görebilir.
* Program parçaları arasındaki en zor kısmı ilk başta halleder.
* Diğer yazılım modellerinin iyi yanlarını içinde barındır.
* Hata oluşumunu engellemez ama hata giderilmesi konusunda kolaylık sağlar.
* Program güncellemelerinde kolaylık sağladığı için maliyetten tasarruf ettirir.

Dezavantajları:

* Program çok fazla parçaya ayrıldığı için çok fazla belgeleme yaptırır.
* Karmaşık yapıya sahip olduğu için ekip olmadan kullanmak oldukça zordur.
* Küçük projelerde zaman kaybıdır ve risk teşkil eder.

**Birleşik Süreç Modeli:** Bu modelde ilk olarak bir vizyon belirlenir. Bununla birlikte uyumluluk çalışmaları başlatılır. Projenin ayrıntılandırılması için daha ayrıntılı bir inceleme yapılır. Bu inceleme sonucunda minimum program yapısı ve riskli kısımlar programlanır. Bu aşamadan sonra tamamlama aşamasına geçilir. Tamamlama aşamasında riskli kısımlardan geriye kalanlar da kodlanır. Tüm kodlar yazıldıktan sonra beta testler yapılır ve program müşteriye teslim edilir.

Avantajları:

* Esnek bir yapıya sahiptir. İsteğe göre değişkenlik gösterebilir.
* Hata ve yeni eklemelerde hızlıca geri dönülebilir.
* Programa ait somut ürüne erken erişim.
* Karmaşık yapılara çözüm kolaylığı sağlamsı.

Dezavantajları:

* Riskler için herhangi bir çözüm pratiği yoktur.
* Karmaşık problemleri çözer ama kendisi de karmaşıktır.
* Çok parça olması çok belgeleme anlamına gelir bu da zaman kaybı demektir.

**Artırımlı Geliştirme Modeli:** İlk başta bir iskelet oluşturup bunun üstüne ekleme yapılması şeklinde gelişen model olarak düşünülebilir. Burada unutulmaması gereken olay bu ilk iskeletin de çalışan bir program olduğudur. Ayrıntılı Geliştirme Modeline örnek olarak cep telefonlarımızdaki uygulamaların güncellemelerini verebiliriz. Belirli zamanlarda ihtiyaca göre uygulamaya yeni özellik eklenmesi tam da bu modele bir örnektir. Çıkarılan her yeni yazılım bir önceki yazılımı kendi içinde barındırır ve daha gelişmişidir.

Avantajları:

* Daha önceki kodlar tekrar kullanılabilir.
* Yeni eklemelerle gün geçtikçe daha iyi bir program yapılır.
* Basit ve uygulanması kolay bir yöntemdir.

Dezavantajları:

* Amatör yazılım ürünlerinde uygulanamaz.
* Uygulanması oldukça pahalıdır.
* Her aşamada belgeleme yapmak oldukça güçtür.
* Doğru model olduğunu program bitmeden anlayamazsınız.

**Formal Sistem Geliştirme Modeli:** Daha önce anlattığımız modellerde bazı modellerin karmaşık programlarda etkisiz olduğunu anlatmıştık. Bu model ise tam da karmaşık problemlerin çözümlendirilmesi için kullanılır. Aslında Formal Sistem Geliştirme Modeli matematiksel bir yöntemdir. Yazılım ürünü artımlı bir şekilde ilerler. Süreçteki her bir adım bir önceki adımı kapsar ve o adımın üzerine eklentiler yapılmış halidir. Bu nedenle aslında program baştaki projenin geliştirilmiş halidir. Yazılım ürününün üst üste eklenen, yenilemelerle geliştirilmesi belgelemeyi zorlaştırır. Bu nedenle bazen belgeleme aşaması program bittikten sonra yapılır. Bu da başka bir soruna neden olur. Yazılım ürününün oluşum süresi artar bu da maliyeti doğrudan etkiler. Hiç belgeleme yapmamak da bir seçenektir ama bu da bakım aşamasında hata çözülmesi ve program analizini zorlaştırır, zaman kaybına neden olur. Yine maliyet artışına sebep verir. Böyle bir olası ikilemde kalındığında risk analizi yapılıp ortak bir yoldan ilerlemek gerekir.

Avantajları:

* Yazılım ürününde olmaması gereken her şeyi belirleyebilir.
* Yazılım en az hatayla gelişmesini saplar.
* Karmaşık sistemleri çözümleyebildiği halde kullanımı karmaşık değildir.

Dezavantajları:

* Sürekli projeye yeni eklemeler yapılması yapım aşamasını oldukça fazla uzatır.
* Zamanın uzaması doğrudan proje maliyetini de arttırır.
* Kullanımı için belirli bir eğitim düzeyi gerektirir.

**Prototip Modeli:** Bu modelde asıl amaç ilk aşamalarda bir prototip üretip bu prototipi müşteriye sunmak ve müşterinin fikirlerini de dikkate alarak en uygun programı oluşturmak. Burada müşterinin prototip hakkındaki görüşleri kesinlikle önemlidir ve prototipte değişiklikler yapılır. Kullanıcının veya müşterinin yeni istekleri dikkate alınarak yenilenen prototip, yazılacak programın kaynağını oluşturur.

Avantajları:

* Kullanıcının elinde ilk baştan beri bir program örneği (prototip) olduğu için hatanın görülmesi daha kolaydır.
* Program yapımına en alttan başlandığı için karmaşıklık engellenir.
* Prototip program oluşumunu kontrol altında tutar.

Dezavantajları:

* Prototip Modeli iki defa belgeleme yapmayı gerektirdiği için çoğu zaman yazılımcıyı hiç belgeleme yapmamaya iter. Bu durumda da pusulasız programlar ortaya çıkar.
* Prototipin programın altyapısını oluşturduğunun farkında olunması gerekli.
* Prototip müşteriye gösterildiğinde müşterinin prototipi program gibi algılayıp ona göre yorum yapması.

**Çevik Modeller:** Yazılım projesinin iyi ve verimli bir şekilde oluşturulmasını sağlayan ve belgelemede de aynı yöntemi kullanan modellemelere verilen genel addır. Bu modeller daha pratik bir yapıya sahip olduğu için yazılımı daha esnek geliştirme imkânı sağlar. Program geliştirmesi devam ederken yapılacak değişikliklere fazlasıyla esneklik sağlar. Proje geliştirme ekibinin kendi içindeki iletişime oldukça önem verilir. Aynı zamanda projeye müşterinin de belirli zamanlarda dahil olması sağlanır. Müşteriye verilen önem sözleşme dahilinde değildir. Müşterinin sözleşmeden farklı isteklerini çözmeye çalışılır. Proje, fazlasıyla önem verilen yenilemelerle geliştirilir. Her yenilemenin sonunda müşteriye proje hakkında ilerleme bilgisi verilir. Bu yenilemelerdeki programlar çoğunlukla çalışan programlardır. Yani müşteriye verilen program çalışan bir yazılımdır ve asıl programın ön sürümü konumundadır. Bu sayede müşteri programda gördüğü yanlışlıklar daha rahat saptanabilir ve gelişim çok daha kolay yapılabilir. Çevik modellerde başarının en önemli kaynağı projenin diğer parçalarında çalışan programcıların belirli sürelerde birbirleri ile bilgi alışverişi yapmasıdır. Çevik modellerin bazı ortak prensipleri vardır. Örneğin; basitlik, hızlı planlama, müşteri memnuniyeti, müşteriyi bazı süreçlerde olaya dahil etme, ekipler arası iletişim, değişen koşullara uyum…

Avantajları:

* Yüksek kalitede ürün elde edilmesi.
* Hata ile karşılaşmada geriye dönmede sorun yaşatmaması.
* Hataların maliyete etkisinin çok fazla olmasını engellemek.
* Yazılımcıya esnek bir yol haritası çıkartması.
* Takım ruhuna önem vermesi.

Dezavantajları:

* Takım ruhunun öneminin bazı programcılara baskıdan başka bir şey ifade etmemesi.
* Kurumsal yapılarda uygulanabilirliğinin zor olması
* Müşterinin olayın içinde gereğinden fazla olmasıyla çok fazla parametrenin sürekli değişmesi.

**Çevik Model Örnekleri:**

* Extreme Programing (XP)
* Scrum
* Crystal
* Agile Unified Process,
* Feature Driven Development (FDD)
* Test Driven Development (TDD)
* Lean Development
* Dynamic System Development Methodology (DSDM)

**Nerede, Hangisi Kullanılmalı?**

Yapılmasına başlanılacak projede kullanılacak yazılım yaşam döngüsü modelini belirlerken onlarca kriterden bahsedilebilir. Bu başlık altında bu kriterlerden önemli olanlarından bahsedeceğim. Öncelikle belirlenmesi gereken olay projenin profesyonellik düzeyi ve projenin büyüklüğüdür. Başlayacağımız proje eğer tek kişilik amatör bir proje ise kullanılabilecek en uygun model Gelişigüzel Modeldir. Bu model hem az ekipli çalışma gruplarında fayda sağlar hem de projenin hızlı tamamlamasına odaklanır. Gelişigüzel Modelinin Uzmanlık seviyesinin düşük olduğu az modellerden biri olduğunu da söylemekte fayda var. Uzmanlık seviyesinin orta düzeyde olduğu bir model ise Çağlayan model. Bu modelde uzmanlık gerektiren bir olay yok ve kolaylık sağlamayı amaçlar. V Model ve Prototip Modellemede de orta seviyeli bir uzmanlık istendiğini söyleyebiliriz. Spiral Model, Formal Model, Artımlı Geliştirme Model de ise yükse seviyede bir uzmanlık ister. Peki modelde uzmanlaşmayı gerektirmeyen modeller var mı? Elbette var. Buna da en bilindik örnek olarak Evrimsel Geliştirme Modeli örnek olarak verebiliriz. Birazda maliyetten bahsedelim. Hayatın neredeyse her alanında olduğu gibi yazılım yaşam döngüsünde de maliyet oldukça önemlidir. Yanlış model seçimi veya proje devam ederken verilen yanlış kararlar programlama şirketlerine milyonlarca dolar zarar verdiği tarihteki onlarca örneklerle pekiştirilebilir. Şimdi hangi modelin, hangi maliyet seviyesindeki projelerde kullanılmasının gerektiğini açıklayalım. Gelişigüzel, Prototip ve Evrimsel Geliştirme modelleri maliyeti düşük projelerde kullanılabilir. Zira bunlar genelde kişisel veya az kişili ekip projeleri olmaktan öteye gidemiyor. Maliyeti ne çok ne de az olan modellere örnek olarak; Spiral, Çağlayan ve V Modeli verilebilir. Diğer bir kısas da dokümantasyon. Bu yazımın en başından beri dokümantasyonun ne kadar önemli olduğunu birçok kez anlatmaya çalıştım. Burada da Çağlayan Modelinde belgeleme oldukça önemlidir. Prototip Modelde ve Spiral Modelde olması veya olmaması konusunda zorunluluk yoktur. Geri kalan yaşam döngüsü modellerinin hepsinde belgeleme olmazsa olmaz konumunda olmasa da çoğunlukla yapılmaktadır. Şimdi de programın bakımına göre modellerin sunduğu imkanlara bakalım. Birçok modelde aslında bakım yapılabilirlik vardır. Ama Gelişigüzel Modelde bakım yapmak oldukça zordur. Hatırlarsanız Gelişigüzel Modellemede belgeleme yapılmadığını söylemiştik. Eğer yaptığınız programın belgelemesi yoksa aylar sonra programa bakım yapmak için döndüğünüzde elinizde bir karmaşadan başka bir şey olmaz. Gelişigüzel Modelinin aksine Çağlayan modelde ise bu olmazsa olmazdır. Diğer bütün modellerde normal seviyede belgeleme yapılmaktadır. Şimdi ir yazılım ürünü geliştirilirken dikkat edilmesi gereken önemli konulardan biri olan zamana değinmek istiyorum. Konumuzla da direk alakalı olan zaman, yazılım yaşam döngüsünde inanılmaz bir konuma sahiptir. Zaman, program yaşam döngüsü boyunca çok iyi yöneltilmeli aksi halde birçok sorunla karşılaşılabilir, hatta yazılım projesinin iptaline kadar gidebilir. Örneğin öngörülen zamanda hazırlanmayan bir program fazladan uğraşılan her gün maliyet artar. Burada “maliyetin artması sadece yazılımcı ekibin başındaki kişiyi rahatsız eder” diye düşünebilirsiniz. Yazılımcılar belki maliyetten rahatsız olmayabilir ama aynı projeyi sürekli düzeltmeye çalışmak programcıları bunaltabilir ve bu da onların verimliliğini inanılmaz bir şekilde arttırabilir. Piki bu kadar önemsediğimiz zaman yönetimini dikkate aldığımıza göre uzun ve kısa süreli projelerde hangi model kullanılmalı? Çok uzun süreli yazılım projelerinde; Çağlayan, Formal modelleri kullanılır. Uzun süreli projelerde; Prototip, Spiral, Artımlı, V modelleri kullanılır. Kısa süreli projelerde ise Gelişigüzel Model kullanılabilir. Bu anlattığımız yazılım yaşam döngüsü modellerinin çoğunun uygulanması oldukça kolaydır. Ama bazı modellerin uygulanabilirliği biraz karmaşıktır. Örneğin; Spiral Model, Formal Model, Yeniden Kullanıma Yönelik Model.

Burada anlamamız gereken nokta bütün modeller bazı özel noktalarda birbirinden ayrılıyor. Ama ana hatları çoğunlukla aynı ve amaçları sadece yazılım sürecini daha iyi bir şekilde yönetmek. Yukarıdaki yazımızda hangi durum veya parametrelerde hangi model kullanılması gerektiğini anlatmaya çalıştım. Ama şunu da unutmamak gerekir ki bu modellerden hangisinin kullanılacağı biraz da yazılım projesindeki yetkili kişinin inisiyatifine de bağlıdır. Bir yazılım projesinde sadece bir tane model kullanılma zorunluluğu olmadığını da söylemeliyim. Çoğu zaman beraber kullanılırlar. Çünkü bir modelin gideremediği sorunu diğer bir modelle çok daha kolay bir şekilde giderilebilir. Bir yazılım modeli seçilmeden önce doğurabileceği sorunlar, projenin büyüklüğü, projenin karmaşıklığı göz önünde bulundurulmalıdır. Tüm bunları anlattıktan sonra günümüzde en çok kullanılan Çevik Modelin neden bu kadar kullanıldığından da biraz bahsedelim. Çevik Modelde öncelikle zaman yönetimi oldukça önemlidir. Gereksiz ayrıntılara takılmayı önler ve verimli bir şekilde projede ilerlemeyi sağlar. Projede ilerlerken yaşanabilecek bir hatada geri dönüşümü de kolaylaştıran model, bunu yaparken de zaman prensibini çiğnemez ve maliyeti inanılmaz bir biçimde arttırmaz. Hiçbir modelde başarı garantisi yoktur ama Çevik Modellerde başarı yüzdesi oldukça yüksektir. Tüm bu pozitif yönlerini düşündüğümüzde günümüzde en çok kullanılan model olması hiç de şaşırtıcı değil.

**Scrum Nedir?**

Çevik Modeller, yazılım projesindeki bireylerin birbirleriyle daha fazla iletişim kurmasını, dokümantasyondan ziyade çalışan yazılıma odaklanan, sözleşme önceliğinden ziyade müşteri memnuniyetine odaklanan, plana sadık kalmak yerine değişen koşullara karşılık verebilmeyi amaçlayan ve günümüzde en çok kullanılan yazılım yaşam döngüsü modelidir. Scrum ise bu Çevik Modellerden biridir. Karmaşık yazılım projelerinin çözümlendirilmesi için kullanılır. Temel metodu projeyi parçalara ayırıp çözümlemeye çalışmak ve çözüme ulaşana kadar da birçok kez tekrar etmektir. Yazılım ekibinden sürekli iletişim halinde olmayı gerektirir. Ve her bir parçanın ne kadar ileri gittiğini sürekli gözler. Bunun nedeni projedeki parçaları aynı zamanda birleştireceği için parçaların belirli zamanlarda belirli seviyelere aynı anda ulaşmış olması gerektiğidir. Bu yüzden Scrum iletişim ve takım içindeki haberleşmeye çok fazla önem verir. Scrum kendi içinde üç temel prensip barındırır. Birincisi şeffaflık: proje hakkındaki tüm bilgilerin herkes tarafından görülebilir olmasıdır. İkincisi denetleme: projedeki ilerleme seviyesi sürekli olarak kontrol edilir. Üçüncüsü uyarlama: projenin yapılacak değişikliklere vereceği tepkileri ve ne kadar uyumlu olduğu belirlenir.

Scrum takımı; Scrum Master, Ürün Sahibi ve Geliştirme ekibinden oluşur. Ürün sahibi yapılacak ürünün alıcısı ve üründe istediği özellikleri belirleyen kişidir. Geliştirme ekibi yazılım ürününün geliştirilmesi için gerekli yetenek ve bilgiye sahip kişidir. Geliştirme ekibi ürün sahibinin isteklerini dikkate alarak yazılım ürününe şekil verir.

**Scrum Günümüzde Neden Popüler?**

Scrum modelinin yapısını incelediğimizde günümüzde neden bu kadar popüler olduğunu gayet iyi anlayabiliriz. Scrum kendi yazılım ekibinde ürünü temsilen bir kişi bulundurur. Bu ürün temsilcisine özel bazı toplantılar yapılır ve istekleri dikkate alınır. Scrum yazılımcı ve müşteri arasında ilişkiyi üst düzeyde tuttuğu gibi yazılım ekibi içinde de iletişimi üst düzeyde tutar. Yazılım ekipleri kendi aralarında programın parçaları hakkında bilgi sahibi oldukları için parçalara ayrılan projenin ilerde daha sağlıklı bir şekilde birleşmesini sağlar. Scrum toplantıları tamamen verimlilik ve karşılaşılan sorunları çözme üzerine tasarlandığı için verimlilik düzeyi oldukça fazladır. Scrum toplantılarında dün ne yaptın, bugün ne yapacaksın, karşılaştığın sorunlar neler? Gibi sorulara cevap verilir. Bu sorulara cevap verilirken projenin bilinmeyen parçaları hakkında fikir sahibi olunur. Bu toplantılar çok farklı şekilde gerçekleşebilir. Örneğin her sabah 15 dakikalık bir ayaküstü konuşma esnasında tüm bu sorulara cevap verilerek proje ilerlemesine katkı sağlanır. Scrum toplantıları birçok konuda konu kapsamında kullanır. Yazılım ürününün geliştirme süresini belirlerken de bir toplantı düzenler ve eğlenceli bir tahmin oyunuyla zaman belirlenir. Scrum tüm bu anlattıklarımız sayesinde hem proje sahibine hem de geliştirici ekibe kolaylık sağlar ve bu yüzden günümüzde oldukça popüler bir yapıya sahiptir.

**REFERANSLAR:**

* <http://slideplayer.biz.tr/slide/2806563/>
* <http://www.slideshare.net/preetimishra14661/process-models-40421019>
* **https://medium.com/@denizkilinc/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-temel-a%C5%9Famalar%C4%B1-software-development-life-cycle-core-processes-197a4b503696**
* [**https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8**](https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8)
* [**https://www.bilginc.com/tr/egitim-haber/scrum-nedir**](https://www.bilginc.com/tr/egitim-haber/scrum-nedir)
* **http://www.defnesarlioglu.com/agile-sureci-ve-scrum/**