İZMİR BAKIRÇAY ÜNİVERSİTESİ

MÜHENDİSLİK VE MİMARLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

YAZILIM YAŞAM DÖNGÜ MODELLERİ

BİL102 ÖDEV SUNUMU

HAZIRLAYAN

AHMET BİLGİN

200601016

DOÇ. DR. DENİZ KILINÇ

İSTANBUL

2021

YAZILIM YAŞAM DÖNGÜSÜNUN AŞAMALARI VE MODELLERİ

Yazılım Yaşam Döngüsü Nedir?

Yazılımın hayatımızda çok önemli bir yere geldiği dünyamızda canlılar zaman ilerledikçe gelişir ve geliştikçe ihtiyaçları, işlevsellikleri değişime uğrar. Yazılımdaki durum da bundan farklı değildir. Üretildiği andan itibaren ihtiyaçları sürekli değişir. Bu değişiklikler yazılımı genişletir. Böylece yazılım prosesi bir döngüye girer. Yazılım yaşam döngüsü (SDLC), bir yazılım ürünü üretilirken geçirdiği süreçtir. Bu süreç birkaç aşamadan oluşmaktadır. Aşamaların sayısı farklı kaynaklarda farklı sayılarda belirtilmiştir. Yazılım yaşam döngüsü doğrusal değildir. Yazılım ürününün ihtiyacına bağlı olarak ileri veya geri gitmek mümkündür. Ayrıca bu metot, sadece yazılım değil herhangi bir sistemi geliştirmekte de kullanılabilir. Yazılım yaşam döngüsü temel olarak 5 aşamadan oluşmaktadır.

1. PLANLAMA: Yazılım yaşam döngüsünün ilk ve en önemli adımıdır. Geliştirilecek ürün hakkında müşteri ile istişare edilip gereksinimler belirlenir. Projenin niteliği ortaya konur. Bu aşamanın diğer bir önemli noktası fizibilite çalışması yapılmasıdır.

2. ANALİZ: Bir önceki aşamada belirlenen gereksinimler analiz edilir ve ayrıntılı olarak tanımlanır. Böylece sistemin teknik özellikleri daha net bir çerçeveyle belirlenmiş olur. İş projelendirilir, projenin tüm detayları ortaya çıkarılır ve bunlar düzenli bir şekilde dokümante edilir. Geliştirilen projenin ne yapması istenildiği öğrenilir. Ayrıca UML diyagramları ilk bu aşamada çizilmeye başlanır.

3. TASARIM: Belirlenen gereksinimlere göre sistemin tasarımı yapılır. Gereksinimler donanım ve yazılıma paylaştırılır. Planlama ve analiz aşamaları dikkate alınarak sistemin projesi çizilir. Kapsamlı sistem mimarisi oluşturulur. Bu aşamada iki tür tasarım yapılabilir: Üst Seviye ve Mimari Tasarım; Detaylı Tasarım. Mimari tasarım yazılım ürünü geliştirilirken kullanılan mimariyi açıklar. Modül ve ara yüz için geliştirilen temel parçaları tanımlayarak sistemi genel olarak bir gözlememizi sağlar. Bunun sonucunda bir doküman oluşur. Detaylı tasarımda ise bu dokümanlar revize edilir. Yazılım tasarımında temel sistem soyutlanır ve bunların ilişkisi belirlenir.

4. GERÇEKLEŞTİRİM (Kodlama ve Test): Bu aşamada yazılım tasarımını kodlanma işlemi gerçekleştirilir. Yazılım projesi ürüne dönüştürülür. Sistem yaşatılmaya başlanır ve ilk çıktılarını verir. Kodlama süreci ve sonrasında testler yapılır. Testlerle birlikte sistem artık hayata karışır. Program birimleri bütün bir sistem olarak birleştirilip belirlenen gereksinimleri karşılayıp karşılamadığını anlamak için sınanır. Bu esnada metot veya nesne seviyesinde birim testleri, bileşenler arasında entegrasyon testleri, bütünleşik sistem testi, performans ve yük testleri yapılabilir. İhtiyaca bağlı olarak tasarım ya da kodlama değişikliğe gidilir.

5. TESLİM VE BAKIM: En uzun yaşam döngüsü evresidir. Ürün kullanıma hazır hale geldikten sonra piyasaya sürülebilir bir hali müşteriye teslim edilir. Teslim edildikten sonra bakım aşaması başlar ve ürünün hayatı boyunca sürer. Sistem kullanılmaya başlandıktan sonra olgunlaşmaya başlar ve bu da birtakım ihtiyaçları beraberinde getirir. Bu aşamada hataların giderilmesi, sistem iyileştirmesi, sistem servislerinin güçlendirilmesi gibi bakım çalışmaları yapılır. Zaman geçtikçe yeni ihtiyaçlar ortaya çıkar, bunlar için yeni tasarımlar yapılır ve bu şekilde döngü hiçbir zaman durmaz.

Yazılım yaşam döngüsünün birçok avantajı vardır. Bizlere risk azaltıcı çözümler sunar. Projeyi sistematik bir şekilde analiz etmemizi sağlar. Projeyi parçalı bir şekilde ele alma imkânı verir. Bu sayede herhangi bir aşamada orta çıkan bir problemin diğer aşamaya geçmesi engellenebilir. Sistem genel olarak incelenebilir. Dinamik ortamlarda kullanılmak için oldukça elverişlidir. Bilgi yönetimi için oldukça önemli olan doküman tabanı oluşturulur. SDLC yaklaşımının bu avantajlarının yanında bazı dezavantajları da vardır. Proje yönetimi için tecrübeye gerek vardır. Daha çok büyük sistemler için tercih edilir. Zaman ve maliyeti kat ve kat artıracağından küçük sistemlerde pek tercih edilmez. Normalde projelerde her adım sonrası test yapılabilir ama yazılım yaşam döngüsünde temel amaçlardan biri hızı artırmak olduğundan bu adım testlerini pek göremeyiz.

Yazılım Süreç Modelleri

Gelişigüzel Model: Herhangi bir model ya da yöntem içermez, sistemi geliştiren kişiye bağlıdır. Daha çok tek kişilik ve basit düzeydeki projelerde kullanılır. Bu uygulama ile oluşturulan projelerin okunurluğu ve bakımı oldukça zordur. 1960'larda ortaya çıkmıştır.

Barok Modeli: Yazılım yaşam döngüsünün adımları doğrusal bir şekilde işlenir. Bu model için önemli olan kısım, "Belgeleme" aşamasının diğer süreç modellerinden farklı olarak ayrı bir adım olarak ele alınmasıdır. Bu aşamanın geliştirme ve test aşamalarından hemen sonra yapılması gerektiğini belirtir. Adımlar arası geri dönüşlerin nasıl yapılacağı belirsizdir ve gerçekleştirim aşamasına çok ağırlık verdiğinden artık günümüzde uygulanması tavsiye edilmemektedir.

Çağlayan (Şelale) Modeli: En popüler ve en temel modeldir. Yaşam döngüsü temel adımları sırasıyla en az bir kez izleyerek gerçekleştirilir. Bir aşamadan diğerine art arda akış vardır. Statik bir modeldir. Ancak yazılım projeleri dinamiktir, çok esnektir ve çok sık değişir yani her an yeni şeyler olabilir. İyi tanımlanmış ve kısa süreli yazılım projeleri için uygun olsa da günümüzde kullanımı zaman geçtikçe azalmaktadır. Gereksinimler, Analiz, Tasarım, Üretim, Test ve Bakım aşamalarından oluşur. Çağlayan modelinde bize bir evrenin bittiğini gösteren imzalı onaylanmış belgelerdir. Bir aşama bitmeden diğer aşama başlamaz. Yazılım geliştirmede aşamalar arası bilgi aktarımı söz konusudur. Dolayısıyla yazılım süreci evreler arası geri bildirimlerle beslenir. Barok modelinden farklı olarak dokümantasyon sürecin doğal bir parçasıdır. Yazılım geliştirmeden önce süreç etkinlikleri ve zaman ayrıntılı bir şekilde programlanır. Bu modelde analiz ve tasarım aşamaları çok önemlidir. Sistem gereksinimleri ayrıntılı bir şekilde belirlenip bu sistem gereksinimlerini karşılayacak detaylı bir tasarım yapılması gerekmektedir. Çağlayan modelinin uygun olduğu sistem türlerine şunlar örnek olarak verilebilir: Yazılımın donanım sistemleriyle arazyüzlenmesinin gerektiği gömülü sistemler, yazılım spesifikasyonu ve tasarımının kapsamlı emniyet ve güvenlik analizi gerektirdiği kritik sistemler, birçok ortak şirket tarafından geliştirilen ve daha geniş mühendislik sistemlerinin parçası olan büyük yazılım sistemleri.

Çağlayan Modelinin Avantajları: Anlaşılması ve kullanılması kolaydır. Projeler daha yönetilebilir bir hale gelir. Bütçe tahmin etmede kolaylık oluşturur. Uzman görüşüne ihtiyaç duyulmadığından herkesçe yönetilebilir. Her aşamanın sonucunda belgeleme işlemi sayesinde sistem test edilir ve korunur.

Çağlayan Modelinin Dezavantajları: Proje ihtiyaçları zaman geçtikçe değişebilir. Müşteriler sistemin ihtiyaçlarını net bir şekilde ifade edemeyebilir. Risk ve belirsizlik içerir. Süreç bitmeden ortaya çalışan bir model çıkmaz. Proje ilerledikçe son halini gözümüzde canlandırmak oldukça zordur. Zaman geçtikçe ilk aşamalara ait kaynak eksikliği ortaya çıkabilir. Değişiklik yapmak ya da hataları düzeltmek oldukça zor ve masraflıdır. Esnek değildir. Gereksinimin gerçekleştirilmesi için çok fazla masraf çıkacağı anlaşılırsa gereksinimin çıkarılması için belgenin değiştirilmesi gerekir. Bu durumda müşteri onayı gerektiğinden tüm süreç aksar. Uzun ve sık ihtiyaçları değişen projeler için hiç uygun değildir.

V Modeli: Süreç akışının V şeklinde gerçekleştiği bir modeldir. Sol taraf üretimi sağ taraf ise test işlemini gösterir. Sol taraf çağlayan modeline oldukça benzerdir. Hatta çağlayan modelinin gelişmiş bir hali olarak düşünülebilir. Çağlayan modelinden farklı olarak üst seviye ve detaylı olmak üzere iki aşamalı tasarım bulunur. Sol taraftaki her bir geliştirme aşaması sağ taraftaki bir test aşamasıyla ilişkilendirilmiştir. Az belirsizlik içeren ve iyi tanımlanmış bilgi teknolojileri projeleri için oldukça uygundur. Bir senaryonun bu modele uygun olabilmesi için ürün tanımının sabit, teknolojisi değişkenlik göstermeyen, tüm ihtiyaçları ayrıntılı bir şekilde tanımlanmış ve kısa süreli olması gerekmektedir. Bu modelin çıktıları kullanıcı, mimari ve gerçekleştirim modelleridir. Kullanıcı modelinde müşteri ile olan ilişkiler iyi bir şekilde tanımlanır ve sistemle ilgili yol haritası çizilir. Mimari modelde tüm sistemin sınanması gerçekleştirilir. Gerçekleştirim modelinde ise yazılım ürünü kodlanır ve sınanır.

V Modelinin Avantajları: Aşamalar sırayla tanımlanmıştır ve bundan dolayı oldukça disiplinli bir modeldir. Basit ve kolay anlaşılan bir yapısı vardır. Yönetilmesi kolaydır. Üretim ve test planları son ürünü değil teslim edilebilir tüm ürünlerde uygulanabilir.

V Modelinin Dezavantajları: Karışık yapılı ve nesne yönelimli projeler için kullanımı uygun değildir. Uzun ve sürekli projeler için tavsiye edilmez. Test aşamasına gelindiğinde önceki aşamalardaki herhangi bir özelliği değiştirmek oldukça zor maliyetlidir. Aynı anda gerçekleştirilmesi gereken olaylara imkân sağlamaz. Aşamalar arasında tekrarlamalar yapmayı kullanan bir model değildir.

Helezonik (Spiral) Model: Süreci düz ve sıralı bir şekil olarak değil de spiral olarak ele alan yazılım modelidir. Yineleyici bir modeldir. Dolayısıyla süreci oluşturan aşamalardan tekrar tekrar geçilir. Bu sürede döngüler de genişler ve spiral model oluşur. Spiraldeki her halka bir döngüyü gösterir. Her geçişte projeyi ilerletmek ve projeye bir şeyler katmak amaçlanır. Risk değerlendirmesi yapmak, projeyi parçalara ayırarak riski minimuma düşürmek bu modelin öne çıkan özelliklerinden biridir. Süreç boyunca risk analizi yapılır. Bu sayede zaman ve maliyet hesabı daha kolay bir şekilde yapılabilir. Yinelemeli artımsal ve prototip yaklaşımlarını içerir. Spiral dörde bölündüğünde ortaya 4 aşama çıkar: Planlama, Risk analizi, Üretim, Kullanıcı değerlendirmesi. Planlama aşamasında sistem gereksinimleri toplanıp analiz edilir. Bu aşamada müşteri ve sistem analisti sürekli diyalog halindedir. Risk analizi aşamasında riskler araştırılır, tartışılır ve belirlenir. Üretim aşamasında ara ürün üretilir. Kullanıcı değerlendirmesi aşamasında ise kullanıcı tarafından prototip sınanır. Doğrulama ve onaydan sonra bir sonraki döngü başlar ve süreç böyle devam eder. Her döngüde daha gelişmiş prototipler yapılır. Son ürün onaylanan son prototip üzerinden hayata tutunur. Sistemde yukarı gittikçe maliyet, sola gittikçe ürünü gözden geçirme süresi artış gösterir. Spiral model orta-yüksek riskli, uzun vadeli, müşterinin gereksinimlerden tam emin olamadığı büyük yazılım projeleri için idealdir.

Spiral Modelin Avantajları: Kullanıcılar sistemi erkenden görebilir. Sistem ihtiyacı değişirse bunu karşılayabilir. Sistemi parçalara ayırır, yüksek riskli olanlar öncelik verilerek çözülür, potansiyel zorluklar engellenir. Bu da daha iyi bir risk yönetimi sağlar. Gerçek gereksinimler daha mantıklı bir şekilde belirlenebilir. Pek çok modeli bünyesinde barındırır. Yazılım-donanım sistemine çerçeve oluşturur.

Spiral Modelin Dezavantajları: Küçük ve düşük riskli projeler için elverişli değildir. Yönetimi diğer modellere göre daha zordur. Projenin son hali ve süresi başlangıçta kestirilemeyebilir bu da bir noktada spirali sonsuza sürükler. Ara basamakların fazlalığı nedeniyle çok fazla dokümantasyon işlemi gerektirir. Öznel risk değerlendirmesi içerir.

Artırımsal Geliştirme Süreç Modeli: Küçük bir yazılım ürününün basit bir uygulamasıyla başlayıp bütün sistem oturana kadar gelişen sürümleri birleştirip sistemi tekrarlı bir şekilde geliştirir. Diğer bir deyişle kullanıcılardan gelen geri bildirimlere dayanarak asıl sistem geliştirilene kadar birçok farklı versiyonla birlikte yazılımın evrimleştiği yazılım yaşam döngü modelidir. Ayrıntılı tanımlama, geliştirme basamakları bağımsız değil birbirine geçmiş durumdadır. Basamaklar arası hızlı geri bildirim içerir. Amaç projeyi bir seferde teslim etmek değildir, geliştirim ve teslim parçalara bölünür. Müşteri ihtiyaçları önemlerine göre sıralar ve en gerekli olan en erken teslim edilir. Yazılım projesi geliştirildikçe üretilen ürünler birbirini kapsar ve işlev sayısı giderek artar. Kullanım ve üretim aynı anda götürülür. Uzun zaman alan ve eksik özellikle çalışabilen projeler için elverişlidir. Spiral model gibi yinelemeli bir modeldir. Burada döngüler daha küçük ve kolay yönetilebilen parçalara bölünür. Her parça gereksinim, tasarım, uygulama, test aşamalarından geçer. Her döngüde ürün yeni özellikler kazanır ve bir önceki döngüde üretilen parçayla birleştirilir. Bu modelde birden fazla döngü eş zamanlı gerçekleşebilir. Bu modelin başarılı bir şekilde kullanılabilmesi için birim seviyesindeki testler titiz bir şekilde her döngüde yapılmalı ve genişletilmelidir. Günümüzde kullanılan en yaygın yaklaşımlardan biridir. Bu modelde yazılım ürünün geliştirirken değişiklikler yapmak maddi olarak daha uygun ve daha kolaydır. Bu modelde her artırımın müşteriye teslim edilmesi gerekmez. Bazen bir artırımla ilgili bir görüş alınmak istenebilir ve bu da sistemi kullanıma geçirmeden mümkün olabilir. Çünkü yazılımı uygulayıp denemek normal süreci aksatabileceğinden geri bildirim her zaman mümkün olmaz. Artırımlı modelin çağlayan modeline göre başlıca avantajları bulunmaktadır. Bir ihtiyacın değiştirilmesinin maliyeti daha düşüktür. Çünkü hazırlanması gereken doküman miktarı daha azdır. Kullanıcıdan geri bildirim almak daha kolaydır. Kullanıcılar yapılanları görebilir ve bununla ilgili görüşlerini dile getirebilir. Halbuki doküman üzerinden yazılım projesini takip etmek oldukça zordur. Müşteriler ürünü çağlayan modeline göre daha erken teslim alabilir.

Artırımlı Modelin Avantajları: Projedeki ilerleme kolay bir şekilde ölçülebilir. Küçük yinelemelerde test ve hata ayıklama kolaydır. Risk analizi ve yönetimi içerir. Projenin gidişatına bağlı olarak farklı ihtiyaçları karşılayabilir. Teslimlerde müşteriye bir değer gösterildiğinden projenin işlevselliği erkenden belli olur ve bu ön değerler prototip gibi davranır. Bütün sistemin başarısız olma olasılığını düşürür.

Artırımlı Modelin Dezavantajları: Sistem için gereken kaynak miktarı artar. İlerleyen zamanlarda tasarımla ilgili sorunlar belirebilir çünkü tüm gereksinimler projenin en başında belirlenemez. Küçük projeler için uygun değildir. Proje yönetimi karmaşık ve zordur. Bazı durumlarda projenin sonu kestirilemez. Projenin ilerlemesi risk analizi aşamasına bağımlı bir haldedir.

Kodla ve Düzelt Modeli: Herhangi bir yöntem içermeyen ve belgeleme gerektirmeyen bir modeldir. Ürünün ilk hali geliştirilir ve proje direk gerçekleştirilir. Süreç, proje istenilen hale gelinceye kadar işler. Yazılım geliştirmenin en kolay ancak en maliyetli yoludur. Çünkü gerçekleştirim aşamasından sonra yapılan değişikliklerin maliyeti oldukça yüksektir. Genellikle öğrencilerin ya da tecrübesiz kişilerin oluşturduğu yazılım projelerinde kullanılır.

Evrimsel Geliştirme Modeli: İlk tam ölçekli modeldir. Büyük coğrafi alanlarda sürdürülen çok birimli uygulamalar için idealdir. Modelin aşamalarında üretilen ürünler üretildiği alan için işlevsellik içerir. Modelin tamamındaki başarı geçirilen ilk evrimin başarısına bağlıdır. Müşteriden alınan dönütler ile sistem yavaş yavaş geliştirilir. Çok birimli banka uygulamaları bu modele iyi bir örnektir. Öncelikle sistem geliştirilir ve ilk birime yüklenir. Daha sonra ortaya çıkan problemler çözülerek sistem ikinci birime yüklenir. Sistem tekrar geliştirilip üçüncü birime yüklenir ve belli zaman dilimlerinde bu birimler için güncellemeler yapılır. Evrimsel geliştirme modelini temel olarak ikiye ayırabiliriz. Bunlar Keşifçi Geliştirme ve Atılacak Prototiplemedir. Keşifçi geliştirmede amaç kullanıcı ihtiyaçlarını analiz etmek için kullanıcı ile çalışıp sistemi ona teslim etmektir. İyi kavranan ihtiyaçlarla başlanması önerilir. Atılacak prototiplemede ise amaç sistemin ihtiyaçlarını araştırıp analiz etmektir. Burada ise tam anlaşılamayan ihtiyaçlarla başlanması önerilir.

Evrimsel Geliştirme Modelinin Avantajları: Kullanıcılarım sistemle ilgili gereksinim analizini daha iyi yapmasını sağlar. Modelin sunduğu sürekli geliştirme erken safhalardaki risklerin kalkmasında büyük rol oynar ve bu sayede hatalar azalır.

Evrimsel Geliştirme Modelinin Dezavantajları: Ortada sistematik bir şekilde teslim edilebilen bir ürün yoktur. Yazılım sistemi genellikle iyi yapılandırılamaz. Bakımı zor bir modeldir. Bazen gereksinimleri değiştirmek gerekir ancak sürekli değişiklik sistemin yapısını bozar.

Prototipleme Modeli: Bu modelde amaç düzenli bir şekilde prototip üretip geliştirmektir. Yazılım ürünü gerçekleştirilirken yazılımın bitmemiş prototipini oluşturma sürecini kapsar. Temel gereksinimlerin belirlenmesi, ilk prototipin geliştirilmesi, prototipin gözden geçirilmesi, geri bildirim ve prototip geliştirme aşamalarından meydana gelir. İlk olarak gereksinimler toplanır. Geliştiriciler ve müşteriler yazılım ürünleri ve bu ürünler için gerekli materyallerin nasıl işleneceği hakkında ortak bir karara varırlar. Daha sonra hızlı bir şekilde sistemin ana hatlarını gösteren ilk prototip üretilir. Kullanıcılar bu prototipi değerlendirmeye alır ve yeni gereksinimlere göre gerekli değişiklikler yapılır. Prototipin yenilenmiş versiyonu kullanıcı tarafından tekrar değerlendirmeye alınır ve yeni görüşlere göre prototip tekrar revize edilir. Böylece kullanıcı sisteme dahil edilerek kullanıcının istekleri doğrultusunda bir anlaşmaya varılır. Birkaç yinelemeden sonra artık prototipin son hali müşteriye teslim edilir. Bu model geliştiriciler ve kullanıcılar arasındaki bilgi ile fikir alışverişi sayesinde gelişir. Prototip modelde yazılım uygulamasındaki mantık bazen tutmaz ve bizim tahmin için ek bir çaba sarf etmemiz gerekir. Prototiplerin yatay ve dikey boyutları olabilir. Bu prototiplerin amacı birbirinden farklıdır. Yatay prototip, tüm sistemi geniş bir şekilde görmemizi sağlar. Bizlere kullanıcı arabirimi ve sistem ihtiyaçları hakkında bilgi verir. Dikey prototip ise yazılım ürününün herhangi bir işlevinin detaylı bir şekilde hazırlanmasıdır. Alt sistemlerin işleyişini anlamak için uygundur. Hızlı prototipleme, evrimsel prorotipleme, artırımlı prototipleme ve aşırı prototipleme bu modelin yaygın olarak kullanılan türleridir. Hızlı prototiplemede minimum gereksinim analizi ve minimum çaba ile bir prototip oluşturulur. Daha sonra asıl gereksinimler ortaya çıkınca asıl prototip oluşturulur ve gerçek sistem geliştirilir. Evrimsel prototiplemede az işlevsellik ile gerçek prototipler ortaya çıkar. Oluşturulan bu prototip ilerleyen zamanlarda oldukça gelişecek olan sistemin kalbidir. Bu türde sadece iyi anlaşılan gereksinimler sisteme eklenir. Artırımlı prototipleme alt sistemlerin işlevsel modüllerini geliştirmek ve daha sonra hepsini birleştirmeyi kapsar. Aşırı prototipleme web geliştirmede kullanılır. Üç evreden oluşur. İlk evrede sayfalarla birlikte prorotip html formatına dönüştürülür. Daha sonra prototip hizmetleri aracılığıyla veri işleme simüle edilir. En son hizmetler uygulamaya açılır ve asıl prototiple birleştirilir.

Prototipleme Modelinin Avantajları: Kullanıcının sisteme etkin bir şekilde dahil edilmesi sağlanır. Kullanıcılar ara ara sistemin prototiplerin görünce sistemi daha iyi kavrar. Hatalar ortaya erken çıktığından kullanıcılara zaman kazandırır ve maliyetleri düşürür. Sistemin eksik parçaları kolay bir şekilde tespit edilebilir.

Prototipleme Modelinin Dezavantajları: Prototipe aşırı bağlılık oluşursa ihtiyaç analizinde aksaklıklar ortaya çıkar. Sistemin yapısı orijinal planın dışına taşabilir, bu da sistemi bilinmezliğe sürükler. Prototip oluşturmak için belirlenen sürenin dışına çıkılabilir. Hataları düzeltme aşaması atlanırsa bu performans düşüklüğü meydana getirir. Bazı durumlarda belgelendirilemeyen hızlı prototipler oluşabilir.

Çevik Yazılım Geliştirme: Günümüz dünyası sürekli değişmekte ve gelişmektedir. Yazılım artık hayatımızın çok önemli bir parçası olduğundan şirketlerin de bu değişimime ayak uydurması gerekir. Yeni iş fırsatları, rekabet baskısı gibi nedenlerle yazılımın sürekli bir yenilik halinde olması kaçınılmazdır. Dolayısıyla yazılım, birçok firma için kritik bir öneme sahiptir Yazılım piyasasında yazılım ürünlerinin zamanında üretilememesi, değişikliklere zamanında cevap verememesi, hataların geç fark edilmesi ve kendini geliştirememesi gibi sorunlar ortaya çıkmıştır. Hayat değişir, müşterilerin istekleri değişir, gereksinimler değişir ve bu da hızlı yazılım geliştirme ihtiyacını doğurur. Yukarıda anlatılan yazılım yaşam döngü modelleri ne yazık ki bu ihtiyacı karşılayamaz. Bu metodolojiler artık eskimiştir ve yazılım mühendisliği için artık iyi bir seçenek değildir. Çünkü yazılım mühendisliği değişen dünyamızda bir üretim sistemi değil artık bir tüketim sistemidir. Bütün ortaya çıkan problemler ve bunları giderme isteği çevik modellemeyi oluşturmuştur. Çevik yazılım geliştirme, küçük adımlar atan ve her adımda yeni özellikler geliştiren, 2-3 haftalık aralıkla sistemi kullanıcıya gösteren yinelemeli bir modeldir. Sistemi hızlandırmak için müşteri sürece dahil edilir. Müşteri ile iletişim artırılarak dokümantasyon en aza indirgenir. Bu sayede bürokrasi azaltılır ve büyük ihtimalle kullanılmayacak olan belgeler azaltılır. Çevik yazılım geliştirme; hızlı yazılım ürünü üretme, değişiklikleri hızlı yanıtlama, ürünü en kısa sürede müşteriye ulaştırma gibi temel değerleri amaçlar. Verimli, hızlı ve ucuz çözümler üretmektedir.

Çevik geliştirmede ortak bir standart oluşması için çevik yazılım geliştirme liderlerinden olan 17 kişilik bir grup 2001 yılında toplanarak beyin fırtınası yaptılar ve çevik yazılım manifestosunu oluşturdular. Manifestoya göre:

"*Bizler daha iyi yazılım geliştirme yollarını uygulayarak ve başkalarının da uygulamasına yardım ederek ortaya çıkarıyoruz. Bu çalışmaların sonucunda:*

- *Süreçler ve araçlardan ziyade bireyler ve etkileşimlere*
- *Kapsamlı dokümantasyondan ziyade çalışan yazılıma*
- *Sözleşme pazarlıklarından ziyade müşteri ile iş birliğine*
- *Bir plana bağlı kalmaktan ziyade değişime karşılık vermeye değer vermeye kanaat getirdik.*

Özetle, sol taraftaki maddelerin değerini kabul etmekle birlikte sağ taraftaki maddeleri daha değerleri bulmaktayız."[1]

Çevik manifestonun altında 12 adet temel prensip yatar. Bunlar:

1. *En yüksek önceliğimiz, değerli yazılımların erken ve sürekli teslimi yoluyla müşteriyi memnun etmektir.*
2. *Değişen gereksinimleri, geliştirmenin sonlarında bile karşılayın. Çevik süreçler, müşterinin rekabet avantajı için değişimden yararlanır.*
3. *Daha kısa zaman ölçeğini tercih ederek, çalışan yazılımı birkaç haftadan birkaç aya kadar sık sık teslim edin.*
4. *İş adamları ve geliştiriciler proje boyunca günlük olarak birlikte çalışmalıdır.*
5. *Motive olmuş bireyler etrafında projeler oluşturun. Onlara ihtiyaç duydukları ortamı ve desteği verin ve işi tamamlamaları için onlara güvenin.*
6. *Bir geliştirme ekibine ve içinde bilgi aktarmanın en verimli ve etkili yöntemi yüz yüze görüşmedir.*
7. *Çalışan yazılım, ilerlemenin birincil ölçüsüdür.*
8. *Çevik süreçler, sürdürülebilir gelişimi destekler. Sponsorlar, geliştiriciler ve kullanıcılar süresiz olarak sabit bir hızı koruyabilmelidir.*
9. *Teknik mükemmelliğe ve iyi tasarıma sürekli dikkat, çevikliği artırır.*
10. *Basitlik -yapılmayan iş miktarını maksimize etme sanatı- esastır.*
11. *En iyi mimariler, gereksinimler ve tasarımlar kendi kendini organize eden ekiplerden ortaya çıkar.*
12. *Ekip, düzenli aralıklarla nasıl daha etkili olabileceği üzerine düşünür, ardından davranışını buna göre ayarlar ve düzeltir.*[2]

Çevik yöntemler iki tür sistemi geliştirirken oldukça başarılı sonuçlar doğurur. Birincisi küçük-orta ölçekli yazılım ürünleri geliştirilen sistemdir. İkincisi ise müşterinin her zaman sürecin içerisinde yer aldığı, yönetmeliklerin az etkilediği sistemlerdir. Müşteriler bu modelde projede kilit rol oynar.

[1] http://agilemanifesto.org/
[2] https://agilemanifesto.org/principles.html

Çevik Model Avantajları: Projelerin hayatın değişen akışına uymasını sağlar. Kaynak ihtiyacı en alt seviyededir. Risklere ve hatalara erken cevap oluşturur. Aynı anda teslimatı ve geliştirmeyi sağlar. Yönetilmesi kolaydır. Kısa döngüler içermesi nedeniyle verim artışı yaşanır. Yazılım kalitesi artar, maliyetler düşer. Değişime her zaman açıktır.

Çevik Model Dezavantajları: Karmaşık sistemlerde önerilmez. Müşteri yeterince iyi anlaşılamazsa yazlımın süreci yanlış yerlere gidebilir. Bu modelin amaçlarından biri de dokümantasyonu en aza indirmek olduğundan buna bağlı olarak bireyselleşme artabilir. İhtiyaçlar sürekli değiştiğinden yoğun bir çalışma temposu içerir.

En yaygın çevik geliştirme modellerine Extreme Programming (XP), Scrum, Kristal Yöntem, Dinamik Sistem Geliştirme Yöntemi (DSDM), Özellik Güdümlü Geliştirme (FDD) örnek olarak verilebilir. Bu yazıda Extreme Programming (XP) ve Scrum modellerini inceleyeceğiz.

Extreme Programming (XP, Uç Programlama): Yazılım boyunca çabuk değişen müşteri gereksinimlerine ayak uydurabilen, oldukça kaliteli çalıştırılabilir kodlar üretmeyi amaçlayan çevik modeldir. Kısa adımlardan oluşur ve yinelemeli bir yapısı vardır. Uç programlama ekip çalışmasını vurgular. Takım içi etkileşim ve geri bildirimin yoğunlaştığı bir yapısı vardır. Planlama, tasarlana, kodlama ve test aşamalarından oluşur. Bu modelde mümkün oldukça her şey test edilir. Mimari, özellikler adım adım geliştirilirken arkada inşa edilir. Genellikle dokümantasyon yapılmaz. Müşteri geliştiriciler ile sürekli iletişim halindedir. Extreme Programming 4 temel değer üzerine kurulmuştur: İletişim, Basitlik, Geri bildirim, Cesaret.

- İletişim: Yazılım projelerinde oluşan problemlerin çoğu insanların birbiriyle sağlıklı iletişim kuramamasından kaynaklanır. Bu model de proje ekibini bir araya getirerek, müşteriyi ekibin bir parçası yaparak, geliştiriciler ile müşteri arasında sıkı bir bağ kurarak iletişim problemlerini ortadan kaldırmayı amaçlar.
- Basitlik: Bu metodun en temel özelliği sadeliktir. Her bir yazılım parçası oldukça basit yapıda olmalı ve sadece ihtiyaçlar üzerine şekillenmelidir. Karmaşık bir yapı sistemin mantığıyla ters düşer. Yazılım esnek ve basit olmalıdır.
- Geri bildirim: Bu özellik hatalardan erken bir zamanda dönme fırsatı verir. Birim testleri ile yazılım ürünü sürekli sınanır. Müşteri de belli zamanlarda geliştiriciler ile buluşup yazılımın geldiği nokta hakkında yorumlarını dile getirir. Bu sayede anlaşmazlıkların çıkması büyük bir ölçüde engellenir.
- Cesaret: En zor olan özelliktir. Projede beğenilmeyen bir duruma doğru sürüklendiğinde korkmadan değişiklik yapılmalı, gerekiyorsa yeniden oluşturulmalıdır. Bu müşteriye güven verir.

Extreme Programming 12 pratik içerir. Bunlar: Planlama Oyunu, Ekipte Müşteri, Önce Test, Basit Tasarım, Çiftli Programlama, Sürekli Entegrasyon, Kısa Aralıklı Sürümler, Yeniden Yapılandırma, Ortak Kod Sahiplenme, Metafor, Kodlama Standardı, Haftada 40 Saat. Planlama oyunu, ekip ve müşteri tarafından işin ne kadar sürede yapılacağının tahmin edilmesidir. Ekipte müşteri, yazılım gerçekleştirme sürecinde müşteri varlığına ihtiyaç duyulduğunu ifade eder. Önce test, kodlamadan önce test yapılmasıdır. Basit tasarım, projenin en basit halde geliştirilmesini söyler. Çiftli programlama, iki yazılımcını ortak çalışmasıdır. Bu sayede sorunlar daha hızlı çözülür, projeye farklı bakış açısı kazandırılır. Sürekli entegrasyon, yapıların yeniliklerin sisteme günlük olarak eklenmesidir ve bu sayede hatalar erken tespit edilir. Kısa aralıklı sürümler, projenin zaman dilimlerine bölünmesidir. Yeniden yapılandırma, müşteri memnuniyetini artırmak için sistem sürekli gözden geçirilir. Ortak kod sahiplenme, geliştirilen kodun bireye değil tüm ekibe ait olmasıdır. Metafor, sistemlerin birbirine benzetilerek geliştirilmesidir. Kodlama standardı, kodun karmaşıklığını azaltmak ve kolay okunabilirliğini artırmak için belli standartlara göre geliştirilmesidir. Haftada 40 saat, çalışmaların verimli geçmesi için hafta içi her gün 8 saat çalışılması gerektiğini söyler.

Extreme Programming'in Avantajları: Projeler sağlam bir yapıya sahip olur. Geliştirme esnasında esneklik oluşturur. Proje maliyetini azaltır ancak yazılım geliştikçe değişiklik yapma maliyeti artar. Riskleri ve bireysel kodlayıcılığı azaltır. Çalışan memnuniyetini gözetir.

Extreme Programming'in Dezavantajları: bu model müşterinin katılımına bağlıdır ama bazı müşteriler sürekli katılımdan hoşlanmaz. Tasarımsal değil merkezi bir yaklaşım içerir. Dokümana fazla önem vermemesi, ekip üyesi ayrıldığında ya da yeni bir üye katıldığında sorun oluşturabilir.

Scrum: Günümüzde en çok tercih edilen yaklaşımlardan biridir. Karmaşık yazılım projelerini küçük parçalara ayırıp geliştiren tekrara dayalı çevik yönetim metodolojisidir. Adını rugby sporundaki bir hücum taktiğinden alır. Bu yöntemde müşterilerin ihtiyaçlarına hızlı cevap verilir. Gereksinimlerin tam olarak anlaşılamadığı ve tanımlanamadığı karışık projeler için uygundur. Bu modelde ekip iş birliği hat safhaya ulaşır, dolayısıyla üretkenlik artar. Scrum; şeffaflık, inceleme ve uyum olmak üzere 3 temel kavram üzerine kurulmuştur.

Scrum Temel Kavramları:

1. Roller

- **Ürün Sahibi (Product Owner):** Ürünü tanımlar ve kapsamını belirler.
- **Scrum Yöneticisi (Scrum Master):** Ekip işlevselliğini ve verimliliğini yönetir.
- **Scrum Takımı (Scrum Team):** Sürekli iletişim halindeki göreve yönelik çalışan ekiplerdir.

2. Toplantılar

- **Sprint (Koşu) Planlama (Sprint Planning):** Bu toplantıda koşunun amacı belirlenir. Gereksinim listesi çıkarılır, dağıtım gereksinimleri belirlenir, takımlar kurulur. Risk değerlendirmesi yapılır. Geliştirme araçları onaylanır ve maliyetler hesaplanır. Yaklaşık 2 saat sürer.
- **Sprint (Koşu) Gözden Geçirme (Sprint Review):** Koşudan önce planlama toplantısı yapılır. Önceki koşu tartışılır ve koşuyu daha üretken hale getirmek için bir sonraki koşuda neler yapılacağı belirlenir. Toplantı üç aşamadan oluşur.
- **Günlük Scrum Toplantısı (Daily Scrum Meeting):** Her iş günü ekip üyeleri tarafından her gün yapılır. Dün ne yaptın, bugün ne yapacaksın ve önünde herhangi bir engel var mı? Soruları yanıtlanır. En fazla 15 dakika sürer.

3. Bileşenler/Araçlar

- **Ürün Gereksinim Dokümanı (Product Backlog):** Proje için gerekli olan özelliklerin bir listesidir. Dinamik bir yapıya sahiptir. Ürün geliştikçe liste de gelişir. Bazı gereksinimlere zamanla gerek kalmazsa listeden silinebilir. Ürün sahibi listedeki sıralamayı değiştirebilir.
- **Sprint (Koşu) Dokümanı (Sprint Backlog):** Koşu hedefine ulaşma planını ve product backlog'dan gelen görevleri içerir. Ürüne hangi işlevselliklerin kazandırılacağını içerir. Hedefe ulaşmak için yapılacakları görüntüler. Hedefi gerçekleştirmeye ne kadar yakın olduğunu gösterir Sürekli değişir ve zamanla belirginlik kazanır. Sadece ekip tarafından değiştirilebilir.
- **Sprint Kalan Zaman Grafiği (Burndown Chart):** Koşunun bitmesi için yapılması geren iş miktarını gösterir. Gün sonunda güncelleme yapılır. İşlerin ne kadar yapıldığı ve normalde ne kadar yapılmış olması gerektiği karşılaştırılabilir.

Scrum Avantajları: Zaman ve maliyet kazandırır. Sürekli iletişi hali bedeniyle kolayca kontrol edilebilen bir sistemdir. Döngüsel bir model olduğundan kullanıcı geri bildirimleriyle beslenir. Yeniliklere kolayca ayak uydurabilir. Günlük toplantı yapmak bireysel üretkenliğin gelişmesine yardımcı olur. Ortaya gayet kaliteli bir ürün çıkar. Daha hızlı bir şekilde daha ucuz sonuçlar ortaya çıkar.

Scrum Dezavantajları: Eğer proje sonlanış tarihi kesinleştirilmezse kullanıcılar sürekli yenilik talep eder. Deneyimli çalışanlarla daha iyi sonuçlar ortaya çıkar, yeni ekip üyeleri projeyi bitiremeyebilir. Scrum master ekip üzerindeki etkisi çok büyüktür. Dolayısıyla sürekli motive edici ve verimliliği artırıcı davranmalıdır yoksa işler istenildiği gibi gitmez. Üyeler proje bitmeden ayrılırsa proje üzerinde olumsuz bir etki oluşturabilir.

Scrum günümüzde neden popüler?

Çünkü scrum basit ve uygulanabilirliği kolay bir modeldir. İşbirliğini ve sürekli iletişimi temeline almasıyla projeleri başarıya ulaştıran bir yapısı vardır. Müşterilerin talep ettikleri değişime hızlıca yanıt verebilir ki bu müşteriler için oldukça önemli bir detaydır. Ortaya çıkan ürün oldukça kalitelidir. Takım merkezli çalışır. Ekip üretkenliğini artırmayı amaçlar.

Kaynakça

Agile Nedir? (tarih yok). ACM-Agike Eğitimi ve Danışmalığı: https://www.acmagile.com/agile-nedir/ adresinden alındı

Alnıak, F. (2017, Mayıs 22). *Yazılım Mühendisliği: Yazılım Süreç Modelleri*. Furkan Alnıak Web Sitesi: https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/ adresinden alındı

DAŞ, D. D. (2018). Yazılım Yaşam Döngüsü ve Süreç Modelleri.

Hasan. (2018, 09 03). *Yazılım Süreç Yönetim Modelleri ve Karşılaştırılması*. Fikir Jeneratörü Web Sitesi: https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/ adresinden alındı

KILINÇ, D. (2016, Kasım 20). *Yazılım Yaşam Döngüsü Temel Aşamaları*. Medium web sitesi: 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 adresinden alındı

KILINÇ, D. (tarih yok). Çevik Yazılım Geliştirme. *Yazılım Sistemlerinin Temelleri Ders Notları*.

KILINÇ, D. (tarih yok). Yazılım Yaşam Döngü Modelleri. *Yazılım Sistemlerinin Temelleri Ders Notları*.

Kızmaz, V. U. (2012, Aralık 28). *Yazılım Yaşam Döngüsü Nedir?* Veysel Uğur Kızmaz Web Sitesi: http://www.ugurkizmaz.com/blog/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-dongusu-nedir adresinden alındı

Learn SDLC. (tarih yok). Tutarialspoint Web Sitesi: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm adresinden alındı

Özaydın, O. (2020, Kasım 29). *Yazılım Yaşam Döngüsü ve Agile Yazılım Geliştirme*. Osman Özaydın Web Sitesi: https://osmanozaydin.com/yazilim-yasam-dongusu-ve-agile-yazilim-gelistirme/ adresinden alındı

SEKER, S. E. (2015, Eylül). Yazılım Geliştirme Modelleri ve Sistem/Yazılım Yaşam Döngüsü. *YBS Ansiklopedi Cilt 2, Sayı 3*.

SEKER, S. E. (2015, Ağustos 19). *Yazılım Geliştirme Modelleri, Yazılım Yaşam Döngüsü (SDLC)*. Youtube: https://www.youtube.com/watch?v=u2rU8Wss4bw adresinden alındı

Sommerville, I. (10. basımdan Çeviri (Ocak 2018)). Yazılım Süreçleri. I. Sommerville içinde, *Yazılım Mühendisliği* (s. 30-37). Nobel Yayınları.

SOYLU, S. (2017). Kamu Kurumlarında Yazılım Yaşam Döngülerinin Uygulanabilirliği ve Çevre ve Şehircilik Bakanlığı İçin Öneriler Uzmanlık Tezi. Ankara.

Telef, Ç. (2015, Haziran 22). *Yazılım Yaşam Döngüsü*. Çağlar Telef Web Sitesi: https://caglartelef.com/yazilim-yasam-dongusu/ adresinden alındı

Why is Scrum so Popular? (tarih yok). Professional Development: https://www.professionaldevelopment.ie/who-uses-scrum adresinden alındı