210601046 Şilan Ekin

Yazılım Yaşam Döngüsü Nedir?

Bir yazılımın üretim aşaması ile kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalara yazılım yaşam döngüsü denir. Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için, söz konusu olan aşamalar bir döngü biçiminde ele alınır. Döngü içerisinde herhangi bir aşamada geriye dönmek ve değişiklik yapıp tekrar ilerlemek söz konusudur. Yazılım yaşam döngüsü tek yönlü ve doğrusal olduğu düşünülmemelidir. Yazılım sürekli değiştiği ve geliştiği için bir döngü şeklinde düşünülmelidir.

Yazılım yaşam döngüsünün 5 temel adımı vardır:

1-Gereksinim(Requirements): Gereksinimlerin belirlenmesi yazılım geliştirme sürecinde önemli bir adımdır. Kullanıcının gereksinimleri tam olarak anlaşılmazsa, yazılım son kullanıcının beklentilerini karşılayamaz. Bu nedenle gereksinimlerin belirlenmesi yazılım geliştirmenin ilk adımıdır. Fizibilite çalışması(yapılan ön değerlendirmeler ile bir proje ya da iş fikrinin uygulanabilirliğini sorgular) bu aşamada yapılır.

2-Analiz(Analysis):Kullanıcılar istenen yazılıma ilişkin gereksinimlerin tamamını sağlayamayabilir. Bu nedenle, belirli bir zaman içerisinde yazılımı geliştirmek için kullanıcıların gereksinimlerinin doğru analizi oldukça önemlidir. Yazılım geliştirme ekibi ile kullanıcılar sürekli bir iletişim olmalıdır. Yazılım geliştirme ekibinin, geliştirme sürecinde kullanıcıların gereksinimlerinin değişebileceğini göz önünde bulundurmalıdır. Bu aşamada gereksinimler ayrıntılı olarak incelenir.

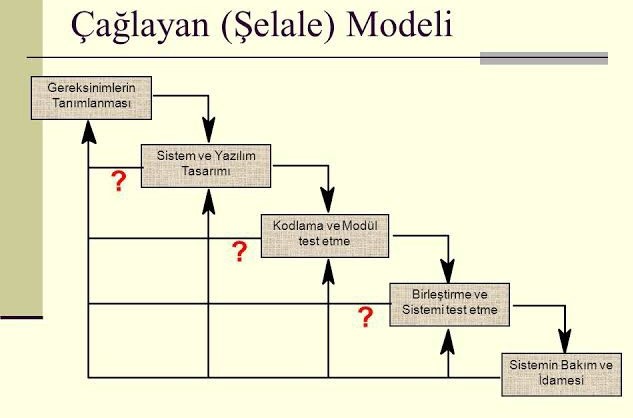
3-Tasarım(Design): Yazılımın mantıksal bir modelini ya da temel yapısını geliştirmeyi içerir. İncelenen gereksinimlere göre yazılımın temel yapısının oluşturulduğu aşamadır. Yazılım tasarımı sistem tasarımı(system design) ve ayrıntılı yazılım tasarımı(detailed design) olmak üzere iki aşamadan oluşmaktadır. \*Sistem Tasarımı: Genel sistemin yapısını kurar. Yazılım sistemi ile diğer yazılım ürünleri arasındaki arabirimleri ve ayrıca temel donanım veya ana bilgisayar işletim sistemini de ele alır.(modüller, akış şemaları…) \*Ayrıntılı Yazılım Tasarımı(detailed software design): Bir veya daha fazla yürütülebilir programa dönüştürülebilen bir biçimde yazılım sistem işlevlerini temsil eder. Spesifikasyon(Anlamı belirtme),yazılacak yazılımı muhtemelen titiz bir şekilde tam olarak tamamlama görevidir.(veri yapıları, ekran tasarımları…)

4-Gerçekleştirme(Implementation): Yazılımın farklı modüllerinin kodu geliştirilir.Farklı modüllerin kodu,her modülün tasarım özelliklerine göre geliştirilmiştir.Yazılım geliştirme ekibindeki programcılar bu amaçla geliştirme araçlarını kullanırlar.Önemli ve genellikle gözden kaçan bir görev,gelecekteki bakım ve geliştirme amacıyla yazılımın dahili tasarımını belgelemektir.Yazılımdaki herhangi bir hatanın yaygınlığını tespit etmek ve bu hataları düzeltmek için test edilir.Yazılımda hataların veya kusurların ortaya çıkmasının nedenleri: 1-Müşterinin gereksinimlerinin tam olarak anlaşılamamasıdır.2-Bir programcının kodu geliştirirken yaptığı yaygın hatalardır(common mistakes).Test sırasında gerçekleştirilen iki önemli aktivite verification ve validation. Verification(soruşturma): önceden tanımlanmış bazı spesifikasyonlara dayalı olarak kontrol edilmesidir. Validation(doğrulama): Ürünün kullanıcı gereksinimlerini karşılayıp karşılamadığını tespit etmek için test edilmesini içerir. Validation sırasında,test eden kişi,yazılımın orijinal gereksinimlere göre doğru çıktı üretip üretmediğini belirlemek için farklı değerler girer.Yeni geliştirilen ve tam olarak test edilen yazılım,hedef ortamına kurulur.Kullanıcılara ayrıca yazılım arayüzü ve diğer işlevleri hakkında eğitim verilmektedir.

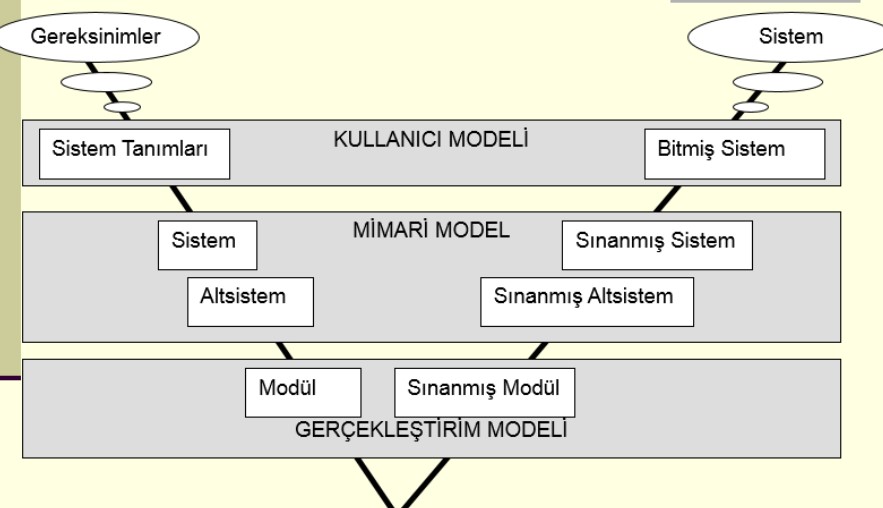
5-Bakım(Maintenance):Bu aşamada geliştirilen yazılım devreye alınır.Kullanıcılar,yazılım geliştirmenin bir sonraki aşamasına yol açan birçok soru ve yazılım sorunu yaşayacaktır.Yazılım başarıyla dağıtıldıktan sonra,tam zamanlı kullanılabilirliğini sağlamak için ona sürekli bir destek sağlanır.Ortam bir değişikliğe uğrarsa yazılımın değiştirilmesi gerekebilir.Yeni keşfedilen sorunlar veya yeni gereksinimlerle başa çıkmak için yazılımın bakımı ve geliştirilmesinden çok daha fazla zaman alabilir.

Yazılım Geliştirme Süreç Döngüleri

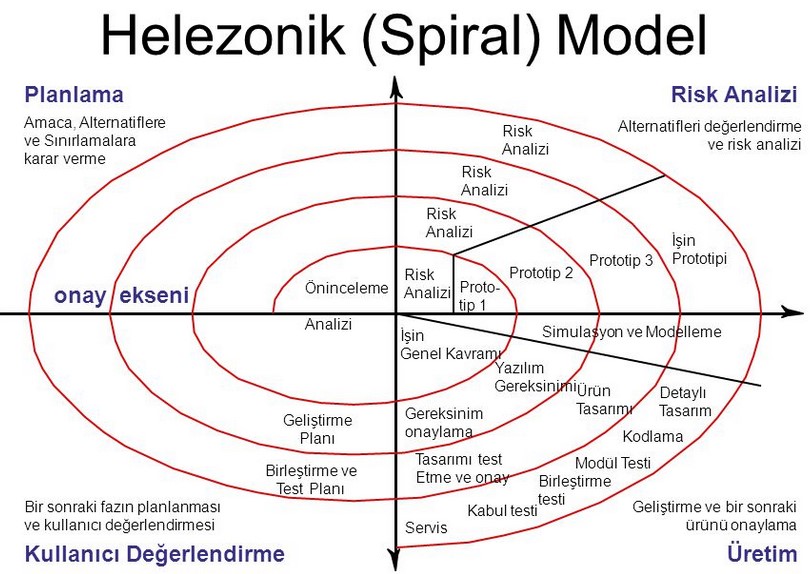
Gelişigüzel Model: Bu yönteme model demek doğru değildir.Gelişigüzel yöntemde belirli bir model veya yöntem yoktur.Çoğunlukla kişiye bağlı yazılım geliştirmede kullanılır.Bundan dolayı yazılım izlenebilirliği ve bakım yapılabilirliği zordur.1960’larda kullanılan bu yöntem basit programlama ve genelde tek kişinin üretim yaptığı yöntemdir.                                                                                                                                                                      Barok Modeli: 1970’lerde kullanılan eski bir modeldir.Bu modelde yaşam döngüsü aşamaları doğrusaldır ve bir aşamayı bitirip diğerine nasıl dönüleceği belirsizdir.Bu modelde diğer aşamalara göre gerçekleştirme aşamasına daha fazla önem verilir.Dokümantasyon ayrı bir süreç olarak ele alınır.Yazılan programın geliştirme ve test aşamaları bitirildikten sonra yapılır.Günümüzde kullanılan bir model olmaktan çıkmıştır. Çağlayan(Waterfall) Yaşam-Döngü Modeli: Yazılım tamamen bitirilip teslim edildikten sonra ortaya çıkan hataların düzeltilmesi prensibine dayalıdır. Çağlayan modeli, yazılım projelerinde uygulanan faaliyetlerin art arda süreçler halinde yapılan, yazılım mühendisliğinin en eski ve temel modelidir. Günümüzde büyük şirketler tarafından her türlü projenin yönetim standardı olarak kabul görmektedir.Bir adım tamamlandıktan sonra diğer adıma geçilir.Eksiklik ya da hata farkedilirse bir önceki adıma geri dönülür.İlerleyen adımlarda karşılaşılan hataların düzeltilmesi zorlaşmaktadır.Ürünün bitirilmesi uzun süreceğinden müşteri sabırlı olmalıdır.Bu modelde müşteriler gereksinimlerini eksiksiz ve kesin olarak belirtmelidir.Rutin projelere uygun bir modeldir.Her adım sonunda o adıma ait bir dokümantasyon oluşturulur.Oluşturulan bu dokümantasyon aynı zamanda kendinden sonraki adım için girdi olarak kullanılır.Bütün adımlar birbiriyle bağlantılı olacağı için karşılaşılacak olası değişimlerin projeye maliyeti yüksek olacağı için adımlar arasında atlama veya büyük çapta geri dönüşler yapılamaz. Çağlayan modelinin uygulandığı projelerde adım bitişleri net bir şekilde tanımlanmıştır ve her adım değişiminde katı incelemeler, yoğun dokümantasyon ve yönetim onayı gibi kontrollerle proje yoğun bir denetim altında yapılır.Bundan dolayı uygulanması için katı disiplin gerekir.Bu modelde kodlamadan önce büyük bir analiz ve tasarım yapılır. Sistem ve yazılım gereksinimleri, gereken her şeyin baştan bilindiği ve değişikliğin olmayacağı kabullenmeler kabul edilerek en başta belirlenir. Bu yüzden belirsizlik içermeyen tam anlaşılmış ve değişme ihtimali olmayan projeler için daha uygundur. Çünkü Çağlayan Modelinin en büyük zayıflığını bu kabullenmeler oluşturmaktadır. Müşteri, gereksinimlerin belirlendiği gereksinim aşamasında projenin içindedir.Analiz,tasarım ve kodlama  aşamalarında görünmez.Test ve bakım aşamalarında yeniden ortaya çıkar. Çağlayan modelinin temel amacı proje süresince değişikliklere izin verilmeyerek, kapsam, zaman ve kaynaklar gibi faktörleri baştan sabitlemek ve büyük bir risk faktörü olan değişimi etkisiz kılmaktır. Çağlayan modelinin ortaya çıktığı zaman ve şartlar değerlendirildiğinde, iş süreçleri bilgi sistemlerine günümüzde olduğu kadar bağımlı değildi ve projelerin aceleyle bitirilmesi gerekmiyordu. Teknolojik gelişme ve iş yaşamındaki değişim günümüzde olduğu kadar hızlı değildi; bu yüzden projeler günümüzde olduğu kadar hızlı bir tempoyla ilerlemiyordu. Teknoloji ve kaynak kısıtlarından dolayı bilgi sistemleri donanım merkezliydi ve yazılımın ana maksadı, depolama kapasitesi ve işlemci gücü gibi kısıtlı donanımsal kaynakları en optimum şekilde kullanabilmekti.



V Süreç Modeli: Çağlayan modelinin geliştirilmiş bir uzantısıdır ve ilgili her geliştirme aşaması için bir test aşamasının ilişkilendirilmesine dayanır.Geliştirme döngüsündeki her bir aşama için doğrudan ilişkili bir test aşaması olduğu anlamına gelir.Doğrusal bir şekilde ilerlemez.V modelinin soş tarafını üretim,sağ tarafını test(sınama) işlemleri oluşturur.V süreç modelinin temel çıktıları; \*Kullanıcı Modeli:Bu modelde proje geliştirme süreci ile kullanıcı arasındaki ilişkiler tanımlanmakta ve sistemin nasıl kabul edileceğine dair test belirtileri ve planları ortaya çıkarılmaktadır. \*Mimari Model:Sistem tasarımı ve oluşacak alt sistem ile tüm sistemin test işlemlerine dair işlevler bu modeldedir. \*Gerçekleştirim Modeli:Yazılım modüllerinin kodlanması ve test edilmesine dair fonksiyonların olduğu modeldir. Belirsizliklerin az,iş tanımlarının belirgin olduğu projeler için uygun bir modeldir.(Örn:BT(Bilgi Teknolojileri))Model, kullanıcının projeye katkısını çoğaltmaktadır.Disiplinli bir modeldir ve sonraki aşamaya ancak önceki aşama tamamlandıktan sonra geçilebilir.



Helezonik(Spiral) Model: Büyük projelerde gereksinim analizini ve planlamayı yapmak zordur.Çünkü proje yapılırken bilinmedik risklerle karşılaşılabilir.Bu yüzden helezonik modelde bu riskler baz alınarak bir model geliştirilmiştir.Amacı riski seviye seviye azaltarak projenin başarılı bir şekilde bitirilmesini sağlamaktır.Bu modeldeki risk analizi ve yönetimi çağlayan ve v süreç modelleri içerisinde bu şekilde ele alınıp detaylı bir şekilde incelenmez.Özellikleri belli olmayan bir proje, Gereksinimleri tam olarak belirlenmemiş,Belirsiz pek çok konu ve benzeri durumlar olduğunda aynı döngü risklerin azaltılarak projenin geliştirilmesi hedeflenir.Buradaki fazlar:hedeflerin belirlenmesi ve çözüm yollarının bulunması,çözüm yolunun seçilmesi ve riskin ortadan kaldırılması,ürünün bir sonraki aşaması için geliştirme yapılması,bir sonraki fazın planlanması şeklindedir.Doğrudan gereksinim,analiz,tasarım ve benzeri fazlar yoktur.Prototip yaklaşımı vardır.Helezonik Modelleme ürünü bir salyangoz kabuğu gibi giderek üstüne büyüyen bir geliştirme sistemi olarak tanımlanabilir. Helezonik model yinelemeli modeldeki geliştirme fikrini,çağlayan modelinin sistematik,kontrollü yönleriyle birleştirir. Kullanıcı üretim sürecinde ara ürün üretme ve ara ürünün test edilmesi kısımlarında rol alır.Yazılımı kullanacak personel sürece erken katılır böylece ileride oluşabilecek istenmeyen durumlar engellenir.Proje sahibi,yüklenici tarafındaki yöneticiler, çalışan yazılımlarla proje süreci boyunca karşılaştıkları için daha kolay izleme ve hak ediş planılaması yapılır.Bu modelde yazılımın kodlanması ve test edilmesi daha erken başlar.



Artımsal Geliştirme Süreç Modeli: Sistem tek seferde teslim edilmez.Geliştirme ve teslim parçalara ayrılır.Her parçanın teslimi beklenen işlevselliğin bir kısmını karşılar.Kullanıcı gereksinimleri öncelik sırasına göre ayrılır ve öncelikli gereksinimler erken teslimlere dahil edilir.Gereksinimler birbirleriyle olan bağlantıları ve önemlerine göre sıralanır.Her yinelemede bunların bir kısmı tamamlanır.Bir parçanın geliştirilmesine başlandığında gereksinimler değiştirilmez.Olası değişiklikler sonraki teslimlerde dikkate alınır.Ortaya çıkarılan her yazılım sürümü birbirini kapsayacak ve giderek artan sayıda işlev içerecek şekilde geliştirilir.Bu model ortaya çıkarılması uzun sürecek ve eksik işlevsellikle çalışabilecek projeler için uygun olabilir.Bu modelde bir taraftan üretim yapılırken bir taraftan kullanım yapılır.Artımsal geliştirme süreç modelinde temel süreç akışı şu şekildedir:

\*Genel gereksinimlerin belirlenmesi \*Gereksinimleri artırımlara bölme \*Sistem mimarisini tanımlama \*Sistem artırılmasının yapılması \*Artırılımın onaylanması \*Artırılımın birleştirilmesi \*Sistemin onaylanması. Sistem onaylandıktan sonra eğer bitmemiş sistem ise yeniden sistem artırılımın yapılması adımına dönülüp diğer adımlar takip edilir. Yazılım geliştirme yinelemeli bir süreçtir. Her süreçte sonuca biraz daha yaklaşılmaktadır. Her teslimle birlikte müşteriye verilen bir sonuç olduğundan, sistemin işlevselliği erken safhalarda ortaya çıkar. Erken teslimler, sonraki teslimler için prototip görevi görür. Projenin tamamen batması ve hezeyana uğraması riskini azaltır.Öncelikli gereksinimleri karşılayan sistem işlevleri daha fazla sınanır. Kodla ve Düzelt Yaşam-Döngü Modeli: Program birkaç yüz satırdan oluşuyorsa bu model kullanılabilir.İlk safhada yazılım ürünün ilk hali geliştirilir.Direkt olarak yazılım ürünü geliştirilir.Sistem istenilen hale getirilinceye kadar sürekli olarak geliştirilir.Bu modelde bakım aşaması vardır fakat yazılım ürününe ait bir dokümantasyon olmadığı için bakım yapmak zordur.Ayrıca geliştirilen yazılım ürünün emeklilik(retirement) aşaması vardır.Kodla ve düzelt yaşam döngü modelinde adımlar sırasıyla şu şekilde gerçekleşir:İlk versiyon geliştirilir,Müşteri gereksinimlerini karşılayana kadar düzenlenir,teslim sonrası bakım yapılır,Emekliliğe ayrılır. Yazılım geliştirmenin en basit yoludur.Fakat maliyeti en çok olandır.Kodlamadan sonra bir yazılım ürünündeki değişikliklerin neden olduğu maliyeti düşünülünce,bu maliyet çok yüksektir.Ayrıca gereksinimler ve tasarım dokümanı olmadan ürünün bakımını yapmak fazlasıyla zordur.Yazılım geliştirmek kolay olduğu için küçük şirketlerde pek çok yazılım projesinde bu model kullanılır.

Prototipleme(prototyping)Modeli:En çok çevrimiçi sistemler gibi yüksek düzeyde kullanıcı etkileşimlerine sahip sistemlerin geliştirilmesinde yararlıdır.Gerçek yazılım geliştirilmeden önce bile tam görünüm vermek ve his vermek için prototiplemeyi çok etkili bir şekilde kullanabilir.Bu modelde izlenen adımlar sırasıyla şu şekildedir: Temel gereksinim tanımlanması,ilk prototipin geliştirilmesi,prototipin incelenmesi,Prototipin gözden geçirilip geliştirilmesi şeklindedir. Türleri:Evolutionary(evrimsel)prototyping,Incremental(artırımlı)prototyping,extreme prototyping Büyük Patlama(Bigbang)Modeli: Bir süreci izlemez ve çok az planlama gerektirir.Müşterinin gereksinimleri belirsizdir ve gereksinimler çok fazla analiz yapılmadan yerine getirilir.Bir veya iki geliştiricinin birlikte çalıştığı küçük projeler idealdir ve aynı zamanda akademik veya uygulama projeleri için de uygundur.Gereksinimler geldikçe anlaşılır ve uygulanır. Gerekli herhangi bir değişiklik,tüm yazılımı yenilemeye ihtiyaç duyabilir veya gerekmeyebilir.Gereksinimlerin iyi anlaşılmadığı ve son çıkış tarihinin verilmediği ürün için ideal bir modeldir.

Çevik(Agile)Yazılım Geliştirme: Çevik yazılım geliştirme süreçleri kendi kendini organize edebilen takımlar tarafından geliştirilir.Son derece işbirlikçi kurumsal bir anlayışla uygun maliyetli ve paydaşların değişen ihtiyaçlarını karşılayan yüksek seviyede çözümler üreten iteratif ve artımlı(evrimsel) bir yaklaşımdırÇevik yazılım geliştirme, geleneksel yöntemlerin yetmediği durumlarda değerlendirilen konular için alternatif çözümler olarak ortaya çıkmıştır.Süreçlerdeki işlevlerden ziyade paydaşların etkileşimlerine,detaylı dokümantasyondan ziyade üretilen koda, sözleşme pazarlıklarından ziyade müşteriyle işbirliğine,bir plana bağlı kalmaktan ziyade değişimi fark etmeye odaklanılır.Değişen gereksinimler,teknik riskler gibi önceden belirlenemeyen durumlara ve yazılım ürününü etkileyebilecek her türlü değişikliğe karşı esneklik sağlayan süreçtir. Yazılım geliştirme yönetiminde bir ilerleyiş olmadan sadece sürekli uyum sağlamak başarı değildir. Çünkü kod yazılımı yinelemeli gelişir, müşteriye erken ve sık ürün teslimi sorun çıkarmaz, başarının birincil ölçütü yazılımın çalışmasıdır. Çevik yazılım geliştirmenin temel ilkeleri vardır. Bunlar;

Yazılım erken ve belirlenen aşamalardaki işlevlerine göre teslimi sağlanarak müşteri memnuniyeti sağlanır. Yazılım sürecinin son safhasında ortaya çıkan değişen gereksinimler kabul edilir. Çevik yazılım geliştirme, değişimi müşterinin avantajı için kabul eder.

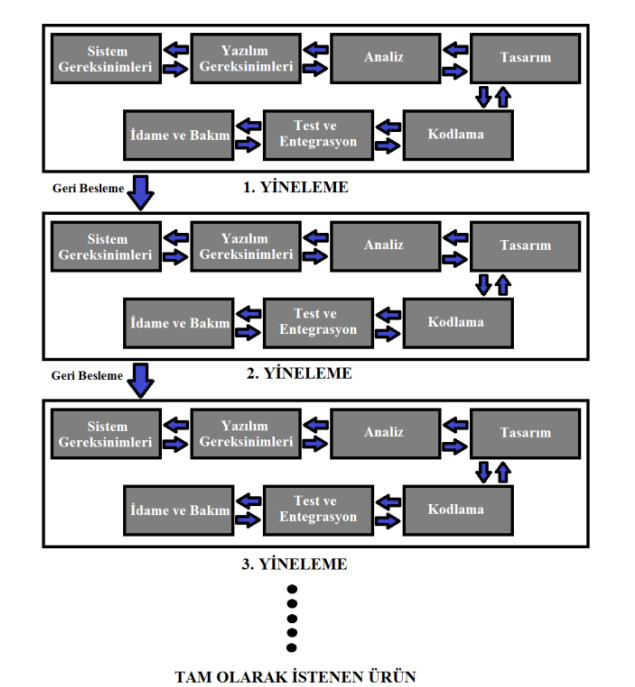
Çalışan yazılım, belirlenen zamanlarda düzenli olarak müşteriye sunulur.

Çevik yazılım geliştirme süreçlerinin sahipleri ve yazılımcılar proje boyunca birlikte çalışır.

Projelerin temelinde motive olmuş, odaklı bireyler yer almalıdır. Onlara ihtiyaç duydukları ortam ve destek sağlanmalı, işi başaracaklarına inanılmalıdır.

Bir yazılım takımında bilgi alışverişinin en verimli ve etkin yöntemi yüz yüze iletişimdir.Çalışan yazılım ilerlemenin birincil ölçüsüdür.

Çevik yazılım geliştirme süreçleri sürdürülebilir işlevler geliştirmeyi teşvik etmektedir. Teknik alt yapı desteği, mükemmeliyet ve iyi tasarım konusundaki sürekli özen çevikliği artırır. En iyi mimariler, gereksinim ve tasarımlarda kendi kendini örgütleyebilen takımlardan ortaya çıkar. Takım, nasıl daha etkili ve verimli olabileceği üzerine beyin fırtınası yapar ve buna göre görevler ayarlanır ve düzenlenir. Çevik yazılım geliştirmede uygulanan yinelemeler, önceki yinelemelerde elde edilen tecrübeler ve tespit edilen eksiklikler doğrultusunda şekillenir. Yapılması gereken görevler, taşıdıkları iş değerine göre önceliklendirilir. Yapılacak işin en iyi nasıl yapılacağını, mevcut kaynaklar ve kısıtlar çerçevesinde proje takımı kendi belirler. Takım belli görevleri belli sürelerde (yineleme süresi içinde) tamamlanır. Yineleme sonunda teslim edilecek ürünü meydana getirmekten sorumlu olan takımdır. Bu nedenle takım içi işbirliği önem arz eder. Bu kapsamda çevik yazılım geliştirmenin dayandığı temel esaslar, deneysellik, önceliklendirme, kendi kendini örgütleme, zaman planlaması ve iş birliği şeklinde sıralanabilir. Çevik yöntemlerde yazılım geliştirme faaliyetleri yinelemeli süreçler halinde uygulanır. Bu yinelemeler sonucunda kullanılabilir ürün ortaya çıkarılır. Paydaşların, Müşteri veya kullanıcıların tecrübe geri beslemeleri ışığında ve varsa değişen gereksinimler doğrultusunda geliştirme süreçleri tekrarlanır. Bu yinelemeli süreçler müşterinin tam olarak istediği ürün ortaya çıkana kadar sürdürülür. Çevik yöntemlerde uygulanan süreçler aşağıdaki şekilde gösterilmiştir:



Çevik yazılım geliştirmenin avantajları:

\*Proje takımının motivasyonu sürekli yüksek seviyededir.Bu da üretkenliği ve yazılımın kalitesini arttırır.

\*Çevik yazılım geliştirme işleyişi gereği proje süresince değişikliklerin hoş karşılanması ve hatta teşvik edilmesi nedeniyle ve tekrarlamalı süreç yapısı sayesinde esneklik yüksektir ve değişen şartların yönetilmesi kolaydır.

\*Kısa süreli yinelemeler ve yineleme sonucunda teslim edilen çalışır ürün sayesinde projenin ilerlemesi daha sağlıklı bir şekilde gözlemlenir ve proje riski azalır, muhtemel hatalar daha erken tespit edilir, teslimat öngörülebilirliği artar ve ürün pazara daha çabuk ulaşır.

\* Müşteri veya kullanıcıların proje sürecinde proje elemanları ile yüz yüze iletişim kurabilmeleri sayesinde iş süreçleri ile yazılım geliştirme süreçlerinin entegrasyon seviyesi yükselir, istenen özelliklerin tam olarak ifade edilebilmesiyle yazılımın fonksiyonel kalitesi artar ve müşteri memnuniyeti sağlanır.

\*Yazılım projelerinin başarısı %55’e kadar yükselebilir.

Geleneksel şelale modeli ile çevik yazılım geliştirme arasında yapılan işin başarı oranları uygulanan metodolojiye göre de farklılık göstermektedir. Şelale modeli ile çevik yazılım geliştirme aynı amaç doğrultusunda ortaya çıkmış yaklaşımlardır. Her iki yaklaşımda da planlama önemli bir yer tutar. Şelale modelinde bütün süreci kapsayan uzun vadeli ve detaylı bir planlama yapılırken, çevik yazılım geliştirmede değişen şartlara cevap verebilmek için uzun vadeli planlamadan ziyade kısa vadeli ve daha detaylı planlamalar yapılır. Bütün süreç sürüm planlarına, sürüm planları yineleme planlarına, yineleme planları ise günlük planlara bölünür. Böylece daha detaylı hedefler belirlenebilir. Her iki yaklaşımda da süreç içinde detaylı incelemelere önem verilir. Şelale modelinde bu incelemeler süreçler arası geçişte yapılan detaylı kontroller ve dokümantasyon vasıtasıyla yapılırken, çevik yazılım geliştirmede yinelemeler sonucu üretilen çalışan ürünler ve müşterinin bütün sürecin içinde bulunması ile detaylı bir inceleme sağlanmış olur. İki metodolojide de yapılan faaliyetler ortaktır. Bu faaliyetlerin işleyiş şekilleri farklıdır. Şelale modelinde bu faaliyetler her gereksinime uygulanan süreçler olarak karşımıza çıkar ve ana sürecin alt bileşenleridir. Çevik yazılım geliştirmede ise her yinelemenin uygulandığı faaliyetler olarak karşımıza çıkarlar ve ana sürecin alt bileşeni yinelemeleri oluşturan faaliyetlerdir.

En Yaygın Kullanılan Çevik Metodolojiler:

\*Extreme Programming(XP)

\*SCRUM

\*Agile Unified Process

Extreme Programming(XP):

Extreme Programming’in 4 temel değeri vardır: iletişim, basitlik, geri bildirim, cesaret.

İletişim:

Projenin başarıya ulaşabilmesi için ekibin kendi içerisinde sağlıklı bir iletişimi olmalıdır.

XP’de iletişim yüz yüze olmalıdır.

Yazılımı ekibi ile yazılımı kullanacak kişiler arasında sıkı bir iletişim bağı olmalıdır. Bu sayede yazılımdaki sorunlar erken fark edilir.

Herhangi bir noktada bir bilgiye ihtiyaç duyuluyorsa ,kısa bir süre içerisinde bu bilgiye ulaşılarak, yazılımın gelişme hızı kesilmeden projenin devamlılığı sağlanır.

Basitlik:

Basitlik sağlanması zor olan bir konudur. Basitlik zorunlu olan işlerin yapılmasıdır.

O anki en basit çözüm yolu kullanılmalıdır.Karmaşık çözümler XP mantığına uymaz.

XP bu basitliği en iyi şekilde sağlayabilmek için günün ihtiyaçlarını ele alarak esnek ve basit bir sistem oluşturmaya çalışır.

Geri Bildirim:

Geri bildirim,sistemler için çok değerli bir fırsattır.Geri bildirim sayesinde oluşabilecek hatalar ortadan kaldırılır.

Müşteri, yönetici ve diğer proje çalışanlarının görüş ve düşünceleri geri bildirimdir.

Müşteri proje takımının bir üyesidir.

Müşterilerle yazılım ekibi belli aralıklarla buluşup, o ana kadar gerçekleştirilen yazılım çalıştırılarak gelinen noktayı incelerler.Sonradan ortaya çıkabilecek büyük anlaşmazlıklar daha önceden giderilir.

Cesaret:

XP’in dört temel değerinden en zoru cesarettir. Projelerin üzerine yılmadan çalışılması, projelerin geliştirilmesi açısından son derece önemlidir.

Başarısızlıktan korkmak yerine başarısızlığa neden olan sebeplerin üzerine gitmek gerekir.

Yapılan işten memnun olunmazsa onu sonlandırıp bir yenisine en baştan başlanılabilmesi gerekir.

XP, başarısızlıktan korkmayı değil, başarısız olununca bu başarısızlığı en kısa sürede telafi etmeyi önerir.

Başarısızlıktan korkmak yazılımcının yavaşlamasına neden olur ve bu da projenin ilerleme hızının düşmesine sebep olur.

XP Pratikleri:Müşterinin belirlediği her bir yineleme için yazılım ekibi ve müşterinin bir arada bulunduğu bir toplantıda, o işin ne kadar sürede gerçekleştirileceğine dair tahminde bulunmasıdır.Böylece müşteri bir sonraki yinelemede hangi işlerin yapılacağına karar verecek ve yazılım ekibi de belirtilen sürelere mümkün olduğunca bağlı kalarak, bu işleri ilgili yinelemede yapacaktır.Müşteri temsilcisi yazılım ekibi ile birlikte, yazılım gerçekleştirilmesi sürecinde ekiple birlikte çalışır. Bu sayede yazılım gerçekleştirilirken yazılım ekibinin ihtiyaç duyduğu bilgilere çok kısa zamanda erişebilme imkanı sağlanmış olur. Kod yazılmadan önce test programı yazılır. Bu sayede mevcut sorunların erkenden ortaya çıkması sağlanarak, daha güvenli bir yazılım gerçekleştirilmeye çalışılır. Bu pratikle, müşterilerin gereksinimlerini karşılayacak en basit tasarım gerçekleştirilir. Böylece kısa sürede anlaşılması, yönetilmesi ve değiştirilebilmesi basit bir yazılım geliştirilmesi sağlanır. Çiftli Programlama(Pair Programming):Programın geliştirilme aşamasında yazılım ekibindeki her bir yazılımcı sahip olduğu yetenek ve bilgi birikimiyle projeye katkıda bulunurlar. Bu da projenin geliştirilme hızını arttırır. Genellikle yazılımcılardan biri diğerinden daha hızlı problemi çözer ve bu da programın yavaşlamasına engel olur. Çiftli programlamanın diğer bir avantajı ise farklı bakış açılarının proje üzerinde birleşmesiyle gerçekleştirilecek projenin ilerleyen aşamalarında ortaya çıkabilecek problemleri proje planlanırken çözülmesini sağlamasıdır. Sürekli Entegrasyon(Continuous Integration-CI): Yazılım geliştirilirken yapılan değişiklikler ve yapılan tüm değişiklikler,yazılımcıların sistem üzerindeki değişiklikleri görmelerini ve oluşabilecek sorunları erken tespit etmelerini sağlar. Geliştirilen yazılım kodu yazılım ekibinin ortak yapımıdır. Buna dayanarak bir ekip üyesi yazılımı daha iyi yapabilmek adına herhangi bir ekip üyesinin yazdığı koda erişebilir.Yazılım ekibindeki üyeler önceden belirlenmiş kodlama standartlarına göre yazılımı geliştirirler. Burada amaç yazılan kodun karmaşık olmasını engellemek ve bütün ekip üyelerini anladığı bir biçimde olmasıdır.

SCRUM:

Scrum adı Rugby sporundaki bir hücum taktiğinin ismidir. Bu taktik bütün takımın topu birlikte karşı sahaya taşıyarak atak yapmasıdır. Scrum çevik yazılım geliştirme metodolojileriyle birlikte çalışabilecek bir proje yönetim yaklaşımıdır.Karmaşık yazılım işlerini küçük parçalara bölerek geliştirmeyi öngörür. Gereksinimlerin belirsiz olduğu ve kaotik durumların beklendiği projeler için en uygun metodolojidir. Bu metodolojide bir yinelemenin tamamlanması 30 günü aşmamakta ve günlük 15 dakikalık toplantılarla sürekli olarak iş takibi yapılmaktadır.

Scrum’da üç temel kavram vardır:

1-Roller(Roles):

\*Ürün Sahibi(Product Owner): Projenin iş değeri açısından geri dönüşüyle sorumludur.

\*Scrum Yöneticisi(Scrum Master):Takımı ve organizasyonu Scrum’ın kurallarına ve pratiklerine adapte eder.

\*Scrum Takımı(Scrum Team):Devamlı iletişim halinde ve hedefe ulaşmak için mücadele eden kişilerden oluşur.

2-Toplantılar(Meetings): \*Sprint(Koşu) Planlama(Sprint Planning):Geniş kapsamlı gereksinim listesinin çıkarılması, Başarılı bir geliştirme için dağıtım gereksinimlerinin belirlenip dağıtımlar için gereksinimlerin eşleştirilmesi, Dağıtımlar için ekiplerin belirlenmesi, Geliştirme araçlarının ve altyapının onaylanması, Dağıtım, geliştirme ve pazarlama maliyetlerinin hesaplanması. \*Sprint(Koşu) Gözden Geçirme(Sprint Review):Her koşu başlangıcında toplantı yapılır. Toplantının ilk kısmında Ürün sahibi ile takım, ürün gereksinim listesini birlikte gözden geçirir ve gereksinim elemanlarının hedeflerini ve içeriklerini belirler. Toplantının ikinci kısmında takım üyeleri ürün gereksinim listesinin en üstünden başlayarak koşu sonucunda gerçekleştirilecek gerekesinimlerden oluşan koşu gereksinim listesini oluştururlar. Bu Scrum’ın Anahtar uygulamasıdır. \*Günlük Scrum Toplantısı(Daily Scrum):Koşu başladıktan sonra günlük 15 dakikalık bir toplantı yapılır. Bu kısa toplantıya tüm ekip katılır. Takımın nasıl ilerlediğini ve karşılaştıkları engelleri gözlemlemek için önemlidir.

3-Bileşenler/Araçlar(Artifacts)

\*Ürün Gereksinim Dokümanı(Product Backlog):Proje boyunca yapılması gereken işlerin basit bir listesidir.Ürün gereksinim listesi canlı bir üründür. Değişiklik gösterebilir. Geçerli ve kullanışlı olabilmesi için sürekli olarak bakım yapılması gerekir.Listedeki sıralamalar her gün değişiklik gösterebilir.Listeye kullanıcı bakış açısından bakılır.

\*Sprint (Koşu) Dokümanı (Sprint Backlog):Mevcut koşu için product backlog’dan elde edilmiş iş ve görevleri kapsar. Koşu dokümanının amacı koşu sonunda son ürünün bir parçası olan bir işlevselliği ya da çalışabilir bir parçayı elde etmektir.

\*Sprint Kalan Zaman Grafiği(Burndown Chart):Bu grafik koşu boyunca işlerin ne kadarının yapıldığını ve normalde ne kadar yapılması gerektiğini karşılaştırılabilmesini sağlar.

Yazılım Modellerin Karşılaştırılması:

Yönetim ve Roller: Merkezi ve hiyerarşik yapılı yönetim sistemleri, şelale modeli ile uyumludur. Liderlik, dayanışma ve çalışanların işlerini kolaylaştırıcı anlayışta yönetim tarzı ise çevik yöntemler için uygun ortamlardır. Roller açısından da iki metodoloji arasındaki en temel fark yöneticilerdir. Çevik yöntemlerde takımı motive eden, engelleri kaldıran, süreçleri kolaylaştıran bir koçluk ortaya koyulurken, şelale modelinde klasik komuta-kontrol yönetim tarzı uygulanır. Kendi kendini organize bağlamında proje yönetimi faaliyetleri liderler, müşteriler ve takım elemanları arasında paylaşılır. Görünürlük: Şelale modelinde proje görünürlüğü uygulanan süreçler arasında yapılan kontroller ve dokümantasyonla sağlanırken, çevik yöntemlerde yinelemeler sonunda ortaya çıkarılan çalışır ürünler ve ürün iş listeleri üzerinden sağlanır. Çevik yöntemler bu kapsamda müşteriden doğrudan çalışan ürünün onayını alarak proje ilerlemesini gerçekleştirdiği için şelale modeline göre avantaj sağlar. Çevik yöntemlerin görünürlük olarak avantajlı olduğu projeler, çalışan ürünün kullanıcı ara yüzlü ürün olduğu projelerdir. Kullanıcı tarafından değerlendirilemeyen ya da fonksiyonel olarak küçük parçalara ayrılamayan projelerde dokümantasyon daha iyi bir görünürlük sağlayacağı için şelale modeli daha uygundur. Karmaşıklık: Üretilecek yazılımın karmaşıklık seviyesi arttıkça analiz ve yönetim ihtiyacı da artar. Örneğin teknik ve yönetsel açıdan en karmaşık projeler olan gerçek zamanlı savunma sistemleri için kodlama aşamasına kadar detaylı analizler yapılmış olmalıdır. Bu şartlarda şelale modeli, plan tabanlı bir yaklaşım olduğu için en iyi seçenektir. Çevik yöntemler en erken zamanda müşteri için değer üretmeyi amaçlar ve eldeki mevcut girdilerle kodlamaya geçilerek yinelemeler içerisinde durumun olgunlaşması amaçlanır. Bu nedenle çevik yöntemler daha az analiz gerektiren nispeten daha az karmaşık sistemler için daha uygundur. Detaylı analiz ve planlama şelale modelinin çevik yöntemlere göre avantajı iken, yineleme sonlarında yapılan çalışır ürün teslimleri vasıtasıyla yapılabilen gözden geçirme, alınan geri besleme ve değişim yapabilme imkânı, proje sürecinde karmaşıklık ile başa çıkabilme açısından çevik yöntemlerin şelale modeline göre avantajıdır. Büyüklük: Proje büyüklüğü ile ilgili genel kanı, büyük projelerin detaylı dokümantasyon ile daha iyi yönetilebileceği, küçük projelerin ise katılımcılar arasındaki doğrudan iletişim sayesinde daha etkin sürdürülebileceği, bu nedenle büyük projelerin şelale, küçük projelerin ise çevik yöntemler için daha uygun olduğu yönündedir. Proje büyüdükçe çalışan sayısı ve bütçe artar. Bu da daha çok koordinasyon gerektirir. Şelale modeli, kapsamlı planlama, dokümantasyon ve büyük takımlar için daha iyi iletişim ve koordinasyon süreçleri sunarak bu ortamı destekler. Proje büyüdükçe çevik yöntemlerin maruz kaldığı dezavantajlara karşı tedbirler alınmakta ve büyük ölçekli projelerde kullanılmak üzere özelleşmiş geniş kapsamlı çevik metodolojiler ortaya çıkmaktadır. Altyüklenici Kullanımı: Altyüklenici kullanılacak durumlarda, yapılması istenen ürün veya ürün parçasının detaylı tanımlanması şarttır. Altyüklenici kendisinden istenen ürüne göre bir maliyet, zaman ve efor tahmini hazırlayarak bir teklif verir ve sözleşme yapılır. Maliyetleri arttıracağı için ihtiyaçlarda doğabilecek değişiklikler kabul edilmez. Bir tedbir alınmadığı müddetçe ihtiyaç sahibi müşteri ve proje takımı ile alt yüklenici arasında herhangi bir iletişim olmaz. İletişim olmadan müşteri etkileşimi ve proje takımı ile dayanışmadan söz edilemez. Bu nedenlerle çevik yöntemler, altyüklenici kullanımına uygun değildir. Tanımlanmış isterler ve zaman kısıtları nedeniyle şelale modeli alt yüklenici kullanımına daha müsaittir. Müşteriler(Customers) Müşteri katılımının istenen seviyede sağlanabildiği durumlarda çevik yöntemler proje sonucunda elde edilecek yazılımın nitelikleri açısından şelale modeline göre daha avantajlıdır. Gereksinimlerin ve iş süreçlerinin sağlıklı bir şekilde tanımlanması, ara teslimatlar ve geri beslemeler sayesinde elde edilen ürünün tam olarak istenen ürün olması sağlanabilir. Bunun yanında proje sürecine müşteri katılımının sağlanabilmesi için müşteri açısından istek, zaman ve bilgi gerekir. Proje elemanlarıyla birlikte proje süresince çalışacak eleman bulundurmak müşteri için külfetli olabilir ve istenmeyebilir. Geleneksel proje yönetimine alışkın müşteriler proje başında ihtiyaçlarını tanımlayıp bir daha zaman ayırmak istemeyebilir çünkü müşterilerin kendi günlük işleri her zaman daha önemlidir. Aynı yerde ve zaman diliminde olmayan müşteriler ile yüz yüze iletişim kurulamaması faaliyetleri aksatabilir. Müşteri adına projeye katılan personel, iş süreçlerine hâkim olmayabilir. Böyle durumlarda gereksinimlerin sağlıklı tanımlanamaması ve teslim edilen ürünlerden geri besleme alınamaması muhtemeldir. Bu gibi nedenlerle müşteri katılımının sağlıklı olarak sağlanamayacağı ortamlar açısından şelale modeli daha uygundur çünkü çevik yöntemlerin en önemli dayanaklarından biri müşteri katılımı iken şelale modelinde proje başlangıcından sonra teslime kadar bir müşteri katılımı beklenmez.

Scrum Günümüzde Neden Popüler?

Çevik yazılım geliştirme kısa vadeli planlar ve küçük parçalar halinde yazılımın kısa sürede ve yüksek başarıyla geliştirilmesini öngörür. Scrum, ilk bakışta çok basit kuralları olan bir yönetimsel modeldir. Gereksinimleri açıkça belirli olmayan, değişime açık, karmaşık yazılım projelerinin yönetimi için uygulanmaktadır. Scrum, detaylı bir şekilde projede izlenmesi gereken adımları belirtmemekte, onun yerine basit ama önemli birkaç olmazsa olmaz kuralıyla esnek bir yönetim sunmaktadır. Her projeye uygulanabilirliği ve basit kurallardan oluşması günümüzde daha çok tercih edilmesinin sebebidir.

MEDYA BAĞLANTILARI:

MEDİUM: https://medium.com/@silanekin.1904/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-nedir-c038b6f05da3

LINKEDIN: https://www.linkedin.com/in/%C5%9Filan-ekin-869653233/

GITHUB: https://github.com/silanekinn?tab=repositories

KAYNAKÇA:

<https://ckk.com.tr/ders/softwareeng/Yaz%C4%B1l%C4%B1m%20M%C3%BChendisli%C4%9Fi.pdf>

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

Deniz Kılınç Yazılım Mühendisliğinin Temelleri Ders Notları

Okan Öztürkmenoğlu Bilgisayar Sistemlerinin Temelleri Ders Notları

<https://medium.com/architectural-patterns/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-modelleri-62915545c51e>

<https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRTWpKU2lNIFoTyY_HkNyL8PAOiw_szKnpdNw&usqp=CAU>

[https://miro.medium.com/max/640/1\*UTQuwGdhy8wLrcsEcIoYyQ@2x.jpeg](https://miro.medium.com/max/640/1*UTQuwGdhy8wLrcsEcIoYyQ@2x.jpeg)

<https://i0.wp.com/furkanalniak.com/wp-content/uploads/2017/05/vModeli.png>

<https://i2.wp.com/furkanalniak.com/wp-content/uploads/2017/05/helezonikModel.png>