*Yazılım Yaşam Döngüsü (SDLC: Software Development Life Cycle )*



Yazılım temelde bir üründür ve bu ürünlerin bir yaşam süreci olur. Bir yazılımın geliştirilmesi ve bakımı süresince izlenen adımların tümüne yaşam-döngü modeli denir. Yazılım yaşam süreci birkaç temel aşamadan oluşur. Bu aşamalar sırasıyla şunlardır: Gereksinim, Analiz, Tasarım, Gerçekleştirme ve Bakım. Bu aşamalar başka kaynaklarda daha fazla ve detaylı da gösterilebilir ancak hepsinin temeli benzerdir ve anlatılmak istenen şey aynıdır.

Yazılım yaşam döngüsünü kendi hayatımızda yaptığımız işlerde uygulanabilir. Örneğin bir fatura ödeme işleminde veya bir inşaatın yapımında bu adımlara rastlamamız mümkündür. Aynı şekilde yazılım ürününde de bir döngü olur ve bu döngü bize izleyeceğimiz yolun temelini oluşturur. Teoride yazılım geliştirme, sıfırdan başlanarak gereksinimler, analizler, tasarım ve gerçekleştirme adımlarının doğrusal olarak tanımlanması ile gerçekleşir.

**1-** Gereksinim Aşaması:

Bu aşamada yazılım ürününü geliştirmeden önce sorunun ne olduğunu, ne istediğimizi bulmaya çalışırız. Bunu yaparken ürünün sahibine (müşteri) ne istediğini neye ihtiyacı olduğunu sorgularız. Örneğin, kamu için yapılmış bir otoparkın yazılım ürünü bizden istenirse biz bu evrede otoparkın neye ihtiyacı olduğunu, nasıl bir yazılım ürününe ihtiyacı olduğunu ek olarak da ürün sahibinin varsa ayrıca bir isteği olup olmadığını anlamaya çalışırız. Bunu yaparken hem kendi fikirlerimiz ve araştırmalarımız, hem inşaatında görevli bilirkişiler, hem de müşteriden yardım alırız.

**2-** Analiz Aşaması:

Gereksinim aşamasında verileri topladıktan sonra analiz aşamasına geçeriz. Bu aşamada elimizdeki verileri analiz eder düzenlemeler yaparız. Müşterinin istediğinden çok gerçekten neye ihtiyacı olduğunu tespit etmeye uğraşırız. Bunu yaparken Yazılım Proje Yönetim Planını da bu safhada hazırlarız. Bu plan ile yapılacakları bütün detayları ile not eder kurgularız. Tabi bu plan daha öncesinde de kabataslak hazırlanabilir. Bu evre sonunda analizin sonucu **şartname dokümanı** olarak yazılır

**ŞARTNAME DÖKÜMANI NEDİR?**

İhtiyacı karşılayacak ürün veya hizmetin verimlilik ve fonksiyonelliğini sağlayacak şekilde, teknik veya uygulamalı her türlü özellikleri tereddütte bırakmayacak, açık ve anlaşılır biçimde tarif edilen müşteri ile yazılım şirketi/yazılımcı arasında yapılan yazılı anlaşmadır.

**3-** Dizayn Aşaması:

Bu aşamada belirlenmiş gereksinimlere çözüm olacak yazılımın ilk ve temel hali oluşturulur, projenin nasıl sürdürüleceği tasarlanır. Bu tasarım ilk iki aşama olan gereksinim ve analiz aşamaları temelinde yapılır. Bu tasarım yazılım modelinin bir sonraki aşamaları için bir temel görevi görür. Bu aşama ikiye ayrılır, bunlar:

**a**. Üst Seviye ve Mimari Tasarım:

Bu tasarımda bir taslak plan oluşturulur, modüllerin isimleri ve işlevleri ile ilgili kısa açıklamaları yazılır, modüllerin birbirleri ile olan ilişkileri tanımlanır ve detayları ile beraber yazılımın mimari diyagramları oluşturulur.

**b.** Ayrıntılı Tasarım:

Bu tasarımda ise yazılımın veri tabanı tabloları, kullanılacak algoritmalar, arayüzün detayları, her modül için girdi-çıktılar tasarlanır/oluşturulur.

**4-** Gerçekleştirme Aşaması:

Bu aşama gereksinimleri oluşturulmuş, analizi yapılmış ve taslak tasarıma sahip ürünümüzün kodlamasına geçtiğimiz aşamadır. Bu aşamada her modül için kodlama yapılır, kodlama planlama sırasında belirlenmiş olan programlama diliyle ve belirlenmiş geliştirme ortamında yazılır. Birbirinden bağımsız olarak modüller test edilir. Yazılım ürünümüzün bir bütün olarak çalışması beklenmektedir bu yüzden modüller birleştirilir ve varsa birleştirme ile çıkan hatalar giderilir. Birleştirme gerçekleştikten sonra gerçek veri kullanılarak ürün test edilir. Testler yapıldıktan sonra son olarak sistem uygulama testine sokulur. Uygulama testi müşteri tarafından yapılır. Kabul gören yazılım ürünü artık bakım aşamasına geçer.

**5-** Teslim Sonrası Bakım Aşaması:

Bu aşama yazılımın geri kalan bütün yaşamı boyunca devam eden bir süreçtir. Yazılıma eklenen yeni güncellemeler, sonradan çıkan hatalar gibi işlemler giderilir. Ürünün daha iyi sonuç vermesi için uygulanan bakımın 2 çeşidi vardır. Bunlar;

**a.** Düzeltici Bakım:

Sistemde sonradan oluşan/oluşabilecek hataları düzeltmek amacıyla yapılan bakım türüdür. Kısaca buna yazılım tamiri de denebilir.

**b.** Ürün Özelliklerinin Arttırılması:

Bu tür bakımda ise ürüne sonradan eklenmek istenen güncellemeler, ek özellikler için yapılır. Ürünün işlevselliğini arttırmak için veya sonradan eklenmek istenen ve şartname dökümanında yazılı değişikliklerin mevcut yazılım ürününe uyarlanması için yapılan bakımdır. Bu bakım kendi içerisinde iki türlü incelenebilir.

**\*\***Mükemmelleştirici Bakım: Müşterinin görüşleri doğrultusunda ürünün yararlılığını arttırmak amacıyla yapılan bakımdır.

**\*\***Uyarlanabilir Bakım: Mevcut sistemin/ürünün başka platformlarda (oyun motoru, tarayıcı motoru, işletim sistemi gibi) çalışmasını sağlamak üzere yapılan bakımdır.

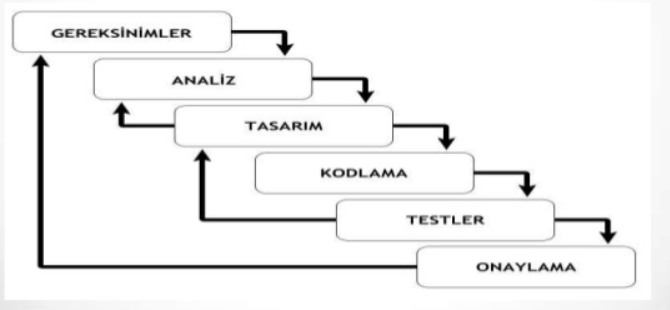
**6-** Emeklilik Aşaması:

Yazılım vadesini doldurduğunda, yazılımın kullanılma durumudur. Yazılım şirketinn belirli süre hizmetinden sonra yazılımın teslimini tamamen şirkete vermesi ve kontrolü elden bırakması aşamasıdır. Bu aşama ile yazılım emekliye ayrılır.

Yazılım yaşam döngüsünün daha verimli sonuç vermesi için Yazılım Geliştirme Süreci Modellerine başvurulabilir. Bu modellerin kullanılması doğrultusunda projelerin karmaşıklığı hafifletilip ürünün üretilmesi kolay hale gelir. Bu modelleri bir nevi rehber olarak tanımlayabiliriz. Peki bu modeller nedir?

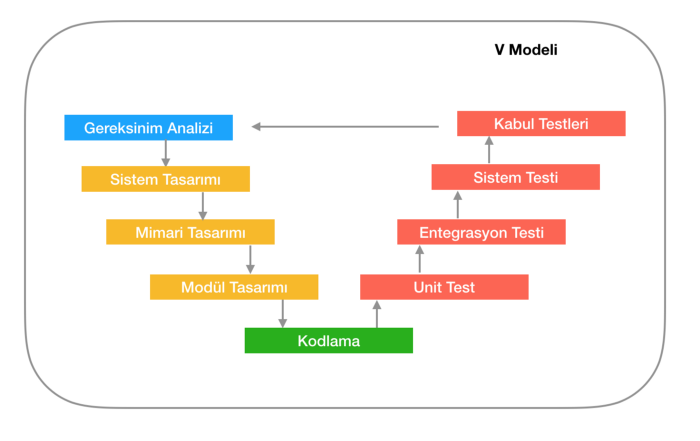
YAZILIM GELİŞTİRME SÜRECİ MODELLERİ

**a.** **Çağlayan (Şelale) Modeli (Waterfall Model):**

Bu model çok yaygın olarak kullanılan geleneksel model olarak bilinen bir SDLC modelidir. Yazılım mühendisliğinde ilk anlatılan modellerden birisi budur. Bu model daha çok statiktir, her şey aşamalı gerçekleşir ve gereksinim, analiz, tasarım, kodlama, test ve bakım gibi safhalardan oluşan kademeli bir yol izler. Her kademe kendisinden önce gelen kademenin ürettikleri doğrultusunda ilerler yani bir kademenin çıktısı ondan sonra gelen kademe için bir girdi olarak kullanılır. Örnek verecek olursak bir işletme açılması istendiğini farz edelim. Bu işletme için her hafta gelirler ile yeni ürünler getirelim, elimizdeki ürünleri sırası ile stok edelim, eğer ciro istenen kademeye ulaşırsa yeri bir tık daha büyütelim, sonrasında ihtiyaç halinde kamera sistemi kuralım, bir süre sonra ürün satışında barkotlu sisteme geçiş yapalım gibi sürekli değişen bir yapı izlemek istersek çağlayan modeli bunun için çok terstir. Günümüzde yazılım sistemleri üzerine her geçen gün yeni fikirler türetildiği ve sürekli gelişen bir yapıya sahip olduğu için çağlayan modeli aslında bu yaklaşımlar arasında en kötüsüdür denilebilir. Ancak ilkler arasında yer alması, basit oluşu ve diğer modeller için bir temel olması nedeniyle gayet önemli bir modeldir.

Bu model iyi tanımlı projeler ve üretimi az zaman gerektiren projeler için uygundur.

**b**. **V Süreç Modeli ( V-Shaped Model):**

 Bu modeli şelale modelinin gelişmiş hali olarak düşünebiliriz, her aşama kendi kontrol aşaması ile eşleştirilip V harfini oluşturduğu için bu ismi almıştır. Bu model belirsizliklerin az olduğu, iş tanımlarının iste yoğun geçtiği projeler için kullanıma uygundur. Yine şelale modelinde olduğu gibi aşağı doğru kademeli akan bir yapıya sahiptir, ancak kodlama safhasından sonra yukarı doğru akış başlar. Birim testleri yapılır. Bu modelin sol tarafı üretim sağ tarafı sınama kısmıdır. Bu döngü temelde 3 modelden oluşur.

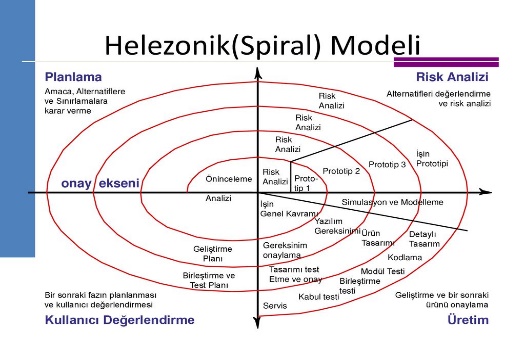
V Süreç Modelinin Temel Çıktıları;

**1- Kullanıcı Modeli:** Geliştirme sürecinin kullanıcı ile olan ilişkilerini tanımlamaktadır. Bu ilişkilerden kastımız gerek üretim gerekse sınama açısından açısından kullanıcıdan beklenenleri ortaya koymaktadır.

**2- Mimari Model:** Sistem tasarımı ve oluşacak altsistem ile tüm sistemin sınama işlemlerine ilişkin işlevleri içerir.

**3- Gerçekleştirim Modeli:** Yazılım modüllerinin kodlanması ve sınanmasına ilişkin işlevleri (fonksiyonları) içermektedir.

**c.** **Helezonik (Spiral) Model:**

 Bu model günümüz modellerine yakın bir modeldir, bazı kaynaklarda sarmal model olarak da karşımıza çıkar. Risk analizinin göz önünde tutulması ve prototiplerin oluşturulması bu modeli diğer modellerden ayrı tutan bir özelliktir. Yinelemeli arıtımsal bir yaklaşım vardır. Risk analizleri yapıldığı için hataları geç olmadan gidermek mümkündür. Spiral modelin 4 temel aşaması vardır. Bu aşamalar planlama, risk yönetimi, üretim ve kullanıcı değerlendirmeleri olarak adlandırılır.

Helezonik modelin en büyük getirisi, her döngü başında risk analizinin yapılması nedeniyle zaman ve maliyet bileşenlerinin kolay tahmin edilebilmesidir. Ancak, helezonik yazılım geliştirme modeli küçük projelerde kullanılmasının uygun olmaması, modeli uygulayanların bu konuda tecrübeli olması gerekliliği gibi dezavantajları da vardır.

Bu model risk analizi ve prototip üzerine kurulmuş olduğundan önceden geliştirilmiş yazılım bileşenlerinin yeniden kullanıldığı projeler için gayet uygundur. Küçük ve riskin çok olmadığı yazılım projelerinde kullanılması uygun değildir, çünkü devamlı dokümantasyon ve prototip yapmak gerekir .

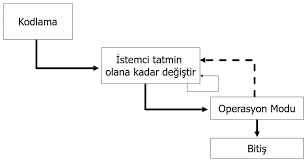
**d.** **Gelişigüzel Model:**

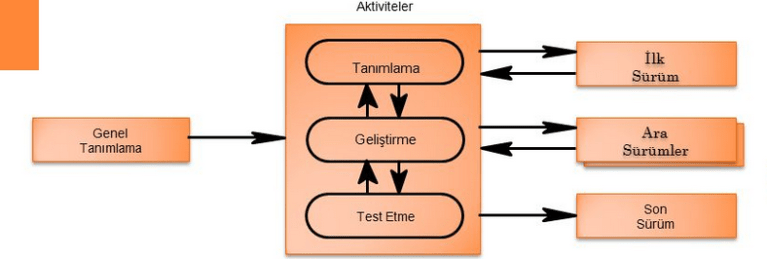
Bu model 60lı yıllarda kullanılan ilkel bir modeldir. Herhangi bir tasarım veya yöntem ele alınmadan yapılır bu yüzden bu yöntemi aslında bir model olarak tanımlamak doğru sayılmaz. Geliştiren kişiye bağımlıdır. Bu modellerin yapımı kişiye bağlı olduğu için takip edilebilirliği ve sonradan bakımı oldukça zahmetlidir. Çok basit programlamaya sahiptir ve üretim tek kişi üzerinden olur. Bu model genellikle basit programlama içeren, çoğunluğunda tek kişinin üretim yaptığı bir yöntemdir

**e.** **Barok Modeli:**

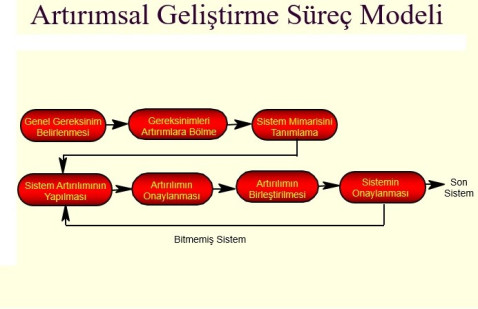
Bu modelde yaşam döngüsünün başlıca adımları doğrusal olarak gerçekleşir. 70 li yıllardan kalma bir modeldir ve bu model belgeleme işlemini ayrı bir süreç olarak işler. Oysa günümüzde belgeleme yapılan işin doğal bir ürünü olarak değerlendirilmektedir. Aşamalar arası geri dönüşün nasıl yapılacağı belirsizdir. Bu model gerçekleştirme aşamasına daha fazla önem vermektedir. Dokümantasyon (belgeleme işlemi) günümüzde üretimin başından sonuna kadar yapılırken bu modelde test aşamaları bittikten sonra yazıldığından günümüzde pek tercih edilmeyen bir modeldir.

**f. Kodla ve Düzelt (Code and Fix):**

 Yazılım geliştirmenin en kolay yoludur. Herhangi bir planlamaya ihtiyaç duymaz, uzmanlık gerektirmediği için herkesçe kullanılabilir bir modeldir. Ancak bu model kontrolsüzdür, kaynak planlaması yoktur ve bitiş süresi belirsizdir. Bir dokümantasyona da sahip değildir bu yüzden hata tespiti ve bakımı zordur. Bu model bireysel çalışmalarda uygun olup takım çalışmaları için terstir, çok küçük projeler veya kısa ömürlü prototiplerde kullanılır, büyük projelerde ise kullanılması uygun değildir. Çünkü kodlamadan sonra bir hatayı değiştirmek yüksek maliyetlere sebep olur ve yazılım şirketleri bu gibi durumları minimum seviyede tutmak ister.

**g. Evrimsel Geliştirme Modeli (Evolutionary Development):**

Bu model ilk tam ölçekli olan modeldir. Başarısı ilk evrimin başarısına bağımlıdır. Bütün aşamalarda üretilen ürünler, üretildikleri bölge için tam işlevselliği içermelidir. Diğer modeller ile kıyaslamaya kalkarsak bu modelin ilerleyişi yavaştır. Bu yüzden büyük alanlara yayılmış, büyük yazılımlar/firmalar için uygundur.

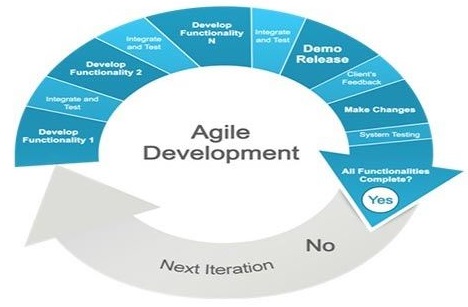
**h. Artımsal Geliştirme Modeli (Incremental Development):**

Bu modelde çekirdek yapı korunur ve bunun üzerinden üretilen her bir yazılım sürümünün kendinden önceki sürümü kapsayacak şekilde ve giderek artan sayıda işlev içerecek şekilde geliştirilir. Bir taraftan kullanım, bir taraftan üretim yapılır yani kullanım-üretim eşzamanlıdır ve sistem arıtılması yapılarak devamlı olarak daha gelişmiş sürümler çıkar. Bu döngü program tamamen hazır olana dek devam eder. Bu modelde ürünün üstüne düzenli olarak bir şeyler katıldığı için ve bu işlev basamaklı olduğu için batma riski azdır. Bu döngü süresince yazılım ürününün daha fazla test edilmesi imkanı sağlanır. Artımsal geliştirme modeli uzun zaman alabilecek ve ürünün eksik olmasına rağmen çalışabilecek türdeki projeler için kullanılması uygundur.

**i. Araştırma Tabanlı Süreç Modeli:**

"Yap-At" tarzı model olarak da nitelendirilebilir. Bu model üzerinden yapılan işlerde sonuç belirsizdir, bu yüzden kesin bir maliyet hesabı yapmak mümkün değildir. Geliştirilen yazılımlar genellikle sınırlı sayıda üretilir ve kullanım süresi geçtikten sonra işe yaramaz hale gelir ve atılır. Araştırma ortamları tamamen belirsizlik üzerine çalışan ortamlardır.

**Çevik (Agile) Yazılım Geliştirme:**

****Yazılım geliştirme uzun ve sıkıntılı bir süreçtir. Çevik modelleme yazılım sistemlerini verimli kullanmak piyasaya daha hızlı ürün verebilmek, değişen isteklere yanıt verebilmek için çıkarılmış daha hızlı çözümler sağlayan metotların genel adıdır. Amaç geleneksel modelleme yöntemlerinden daha esnek bir tavırla yazılım geliştirme projelerinde daha hızlı tepki vermektir. Yani gerek müşterilerin gerek ise üreticilerin daha az baş ağrısı ve daha hızlı değer vermelerine yardımcı olan proje yönetimi ve yazılım geliştirmeye dayalı yinelemeli artımlı bir yaklaşımdır. Her şeyi ele almaktansa çevik bir ekip küçük ama etkili ilerlemeler kaydeder. Gereksinimler, analizler, planlar ve sonuçlar devamlı değerlendirilir, böylece ekip olası bir değişime açık doğal bir mekanizmaya sahiptir. Genel olarak bu metotlar bizlere daha hızlı, daha kolay ve daha az maliyetli bir yazılım yaşam döngü süreci sağlar.

--Çevik Modelin Avantajları--

● Çevik modellemede ekip kısa planlama döngüleri sayesinde değişime açıktır. Yazılım ürünleri zaten yaşayan bir döngüye sahip olduklarından olası bir değişiklik ihtimali yüksektir, bu konuda çevik modelleme hem üretkenliği hem de üretilen yazılım ürünlerinin kalitesini arttırır.

● Tabi bunun yanında bitişi belirli olmayan projeler için de faydalıdır. Zaten sürekli bir planlama ve sonuca varma içinde olan ekip yeni istekleri kolaylıkla karşılayabilir ve projenin ilerlemesi konusunda bir aksilik çıkmaz. Çünkü her yinelemede ayrı projeymiş gibi bir yaklaşım ele alınır. Projenin de küçük parçalara/modüllere ayrılarak ele alınması hataların telafisinin kolaylaştırır.

●Projelerin devamlı ele alınması sonradan fark edilmeyen hataların düzeltilmesine yarar. Sürekli döngü sayesinde yazılım artımlı olarak gelişir ve verimliliği artar. Her yineleme sırasında test yapmak hataların daha hızlı ve kolay çözümlenmesini sağlar.

●Çevik geliştirme sürecinde müşterinin ürüne katkısı için fırsatı çoktur. Her planlama ve testte müşterilere ürünün gerçekleştirilen kısımları ve sonraki basamakları hakkında bilgiler verilip, yeni isteklere açık olması müşteri beklentilerini karşılamak hem müşteriyi memnun eder hem de ekibin işini kolaylaştırır.

●Ayrıca ekip sürekli birbiri ile iletişim halinde olduğundan çevik yazılım geliştirme süreci kısadır ve ekip hızlıdır. Bu iletişim sayesinde ekibin motivasyonu yüksektir ve üretime istekli bireyler piyasaya daha kaliteli ürünler sağlar.

--Çevik Modelin Dezavantajları--

●Modelde sürekli bir yineleme olduğundan teslimat tarihi istendiği gibi yetişmeyebilir. Ayrıca projenin herhangi bir zamanında yeni bir istek projeyi uzatabilir.

●Çevik takımlar genelde 8-10 kişiliktir ve bu modelin işlemesi için takımın çeşitli alanlarda becerilere sahip olması ve iyi bir altyapı gerekir. Aksi halde modele ayak uydurmak zor olabilir.

●Çevik modeller kesin bir plana sahip olmadığından başta beklenenden çok farklı bitebilir. Zaten geliştirme ortalarında gelen istekleri de eklemesi kolay olduğundan ürünün ilk başta beklenen hali ile yapıldığında karşılaşılan sonucun birbirinden uzak olması olasıdır.

,●Çok esnek olduğundan ve müşterinin geri bildirimleri de sürekli olarak proje gidişatını değiştirebileceğinden gidilen yol hedeften uzaklaşabilir.

**Çevik Yazılım Geliştirme İlkeleri:**

1- Yazılımın erken ve sürekli teslimini sağlayarak müşterileri memnun etmek.

2- Çalışan yazılım belirli aralıklar arasında düzenli olarak müşteriye sunulmalıdır.

3- İş sürecinin sahipleri, bilirkişileri ile yazılım ekibi proje süresince beraber çalışmalıdır.

4- Kodlamanın ileri kısımlarında gelebilecek bir değişikliğe açık olunmalıdır, esnek yazılım söz konusudur.

5- Ekip içerisinde kaliteli bir bilgi akışı sağlanması açısından yüz yüze olmak önemlidir.

6- En önemli öncüllerden biri de yazılımın çalışmasıdır. Yinelemeler ile beraber yazılım devamlı olarak gelişir ve bu süreçte bile durmaz, çalışır.

7- Basitlik önemlidir, yazılımın olabildiğince sade ve ihtiyaçları karşılamaya yönelik olması gerekmektedir. Gereksiz fonksiyonlar, modüller veya istekler yazılımın kalitesini düşürür.

8- Ekibin organize olması çok önemlidir çünkü en iyi ürünler ve tasarımlar iyi organize olmuş ekiplerden çıkar.

9- Yeterli düzeyde bilgi ve iyi bir altyapı, iyi tasarım konusunda gösterilen özen çevikliği arttırır.

10- Kaliteli projeler motivasyonu yüksek ekiplerce üretilir. Ekibe duyulan ihtiyaç ve güveni ekibe yansıtmak, yazılım ekibine iyi bir ortam sağlamak önemlidir.

11- Çevik süreçler sürekli geliştirmeyi teşvik eder.

12- Takım düzenli olarak nasıl daha verimli bir üretim yapılacağı konusunda düşünür ve ona göre hareket eder.

-Çevik yazılım metotları kendi içinde temeli aynı fakat uygulama kısmı farklılaşan çeşitli metodolojilere ayrılır.

Popüler Çevik Metodolojiler:

1- Extreme Programming (XP)

2- SCRUM

3- Agile Unified Process

4- Feature-Driven Development (FDD)

5- Test-Driven Development (TDD)

6- Lean Development

7- Dynamic System Development Methodology (DSDM)

8- Microsoft Solution Framework (MSF)

**Extreme Programming (XP) (Aşırı Programlama):**

En popüler çevik yazılım süreçlerinden biri olarak bilinir. 1999 yılında Kent Beck tarafından bir yazılım geliştirme disiplini olarak çıkarılmıştır. Kolay, grup içi iletişime önem veren ve geri dönüşlerin sık olduğu bir yazılım geliştirme yöntemidir. XP, tam belirli olmayan ve çabuk değişime uğrayan müşteri gereksinimlerine kolayca ayak uydurur. İletişim eksikliğini ortadan kaldırmayı amaçlar. Yazılım ekibi ile kullanıcıların sık sık iletişim halinde olmaları olabilecek herhangi bir hatanın erkenden fark edilmesini sağlar.

Ekstrem Programlamada 4 temel değer vardır.

1.İletişim 2.Basitlik 3.Geri Bildirim 4.Cesaret

**Ekstrem Programlamada İletişim**

Projelerin işleyişine bakıldığında çıkan problemlerin iletişimde eksiklikten kaynaklı olduğunu görürüz. Bir projenin başarıya ulaşması için ekibin sağlıklı iletişim halinde olmaları gerekir. XP bu eksikliği ortadan kaldırmayı amaçlar. XP de iletişim yüz yüze olmalıdır çünkü takım insanlardan oluşur ve insan sosyal bir varlıktır. Bu yüzden en iyi haberleşme yüz yüze olanıdır. XP tüm takımın bir araya gelerek çalışmasını ve her bireyin kendi uzmanlık alanı içinde çalışmasını önerir. Sadece ekip değil müşteri de bu ekibin bir parçası olarak kabul edilmelidir. Müşterinin gereksinimleri ve yazılım ürünün özellikleri hakkında bilgi müşteri tarafınca yazılım ekibine aktarılır.

**Ekstrem Programlamada Basitlik**

Her şeyden önce bizler insan olduğumuzdan her şeyin basit olması işimize gelir. Çünkü karmaşıklık kafamızı karıştırır. Basitlikten kastımız ise zorunlu olan işlerin yapılmasıdır. Çünkü insan olarak kapasitemiz sınırlıdır ve karmaşıklığın azaltılması hata ayıklama, tasarlama, kodlama gibi konularda bizlere kolaylık sağlar. Karmaşık çözümler XP'nin mantığına aykırıdır.

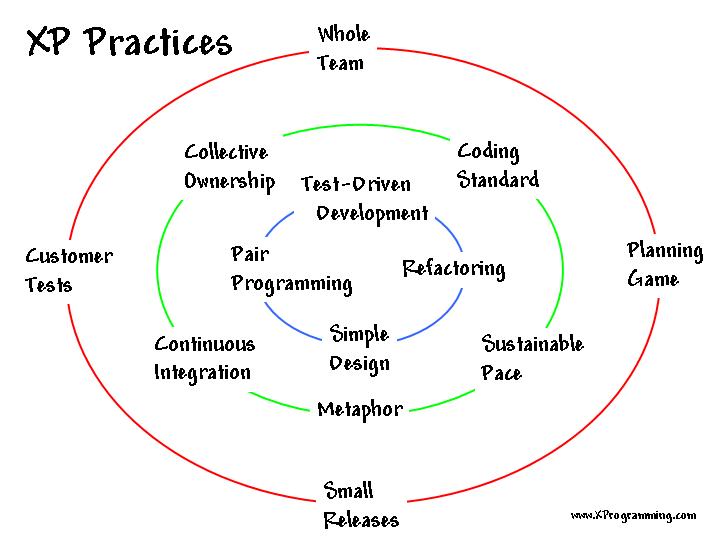
**Ekstrem Programlamada Geri Bildirim**

Geri bildirim yazılım sistemlerinde Hint kumaşı niteliğinde bir fırsattır. Çünkü geri bildirimler sayesinde ortaya çıkabilecek hatalar kaldırılır, müşteri, yönetici ve diğer proje çalışanlarının görüşleri dikkate alınır. XP için de en önemli geri bildirim aracı testlerdir. Yazılan en ufak kod parçasını bile test ederek yolun başında doğru yolda olup olmadığımızı saptayabilelim. Özellikle birim testleri çok önemlidir ve XP bu testlerin yapılması konusunda ısrarcıdır.

**Ekstrem Programlamada Cesaret**

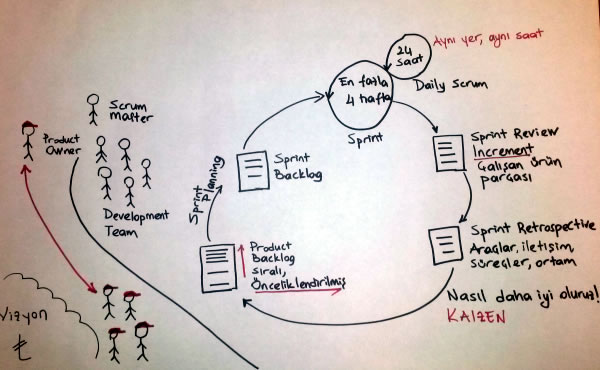
XP'nin dört temel noktasından en zoru cesarettir. Projelerin üzerine yılmadan gidilmesi, projelerin gelişimi açısından oldukça önemlidir. Yazılım projelerinde korku geliştirmenin gittikçe kötüleşmesine sebep olur, yazılımda başarısızlık ise ürünün çoğunlukla çöpe atılmasına sebep olur. Bu yüzden başarısızlıktan korkmak yerine onun üzerine gitmek XP için çok önemlidir.

XP yazılım geliştirmede kolaylık ve esnekliği sağlamak için 12 farklı uygulama vardır.

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

**SCRUM**

****Scrum karmaşık yapılı yazılım süreçlerinin yönetilmesi için kullanılan bir yazılım geliştirme yöntemidir. En temel özelliği gözlemci, geliştirmeci ve tekrara dayalı olmasıdır. Bunu yaparken bütünü küçük birimlere (sprint) parçalayan bir yol izler. Düzenli geri bildirim ve planlamalar ile hedefe ulaşmayı sağlar. Bu metodoloji karmaşık ortamlarda adım adım yazılım geliştiren ekipler için uygundur. Scrum yaklaşımını sadece yazılım ürünlerine uyarlamamız gerekmez. Yaptığımız tüm işlerde bu yöntem kullanılabilir. Bu metodolojide yinelemeler sıktır. Bir yinelemenin tamamlanması 30 günü aşmamakla beraber günlük 15-20 dakikalık toplantılar ile devamlı iş takibi yapılır. Amaç beklenen ürünün hızlı, ucuz ve kaliteli biçimde üretilmesidir. Scrum çevik yazılım prensiplerini hayata uyarlayan bir yöntemdir.

Scrum birçok günümüz projelerinin oldukça karmaşık ve planlamasının zor olacağı şekilde varsayım yaparak bu karmaşıklığı 3 ilke ile azaltmayı amaçlar.

1. Şeffaflık: Projelerde sağlanan ilerlemeler ve problemler günlük olarak tutulur ve herkesçe izlenebilir olması amaçlanır.

2. Denetleme: Ürünün parçaları düzenli aralıklarla teslim edilir ve değerlendirilir.

3. Uyarlama: Ürün için gereksinimler tek seferlik belirlenmez. Her teslimat tekrar değerlendirilir ve gidişat bu değerlendirmeler dahilinde bir yol izler

Scrum Yapı Taşları

**a.** Product Backlog: Proje için gerekli gereksinim listesidir. Müşteri ile product owner arasında olur. Toplanan kullanıcı verileri ürünün sahibi tarafınca önceliklerine göre düzenlenir ve gerçekleştirilme zamanı takımın yardımı ile tahmin edilir. Ürün içeriği geliştirilmekte olan ürünün önceliklerine göre sıralanmış fonksiyonları kapsar. Amaç kullanılabilir yazılım elde etmektir.

**b.** Sprint Backlog: Sprint içeriği halledilmesi gereken görevleri gösterir. Genellikle 15-30 günlük bir proje zaman dilimidir. Product Backlog' un olduğu Planning toplantısının ikinci çıktısıdır. Sprint Backlog'u Scrum tahtası olarak hayata geçirirler, böylece takımın çalışma ortamında fiziksel olarak yer alan bu tahta üstündeki iş listesi sayesinde şeffaflığı sağlar. Scrum tahtası üzerindeki işlerin durumuna bakılarak gidişat değerlendirilir. Gerektiğinde değişikliğe gitme kararı alınır.

**c.** Scrum Daily Meeting: Bu bir statü toplantısı değildir. Ayakta yapılır ve maksimum süresi 30 dakikadır. Amaç takımca genel durum değerlendirmesi yapmak ve takım üyelerini bilgilendirmek, kopuk kalmamaktır. 3 soruya cevap verilir.

1- Dün ne yaptım?

2- Bugün ne yapacağım?

3- İşimi engelleyen bir durum var mı?

Günlük yapılan bu toplanmalar ekip içerisinde iletişimi iyileştirir. Başka toplantılara olan ihtiyacı ortadan kaldırır, geliştirmenin önünde duran engellerin tespit edilmesini sağlar, hızlı karar almaya teşvik eder ve takımın bilgi seviyesini arttırır. Bu günlük etkinlik bir kilit gözlem ve adaptasyon toplantısıdır.

Temel Kavramlar

1- Roller

**\***Ürüm Sahibi (Product Owner)

**\***Scrum Ustası (Scrum Master)

**\***Scrum Takımı (Scrum Team)

2- Toplantılar

**\***Sprint Planlama

**\***Sprint Gözden Geçirme

**\***Günlük Scrum Toplantısı

3- Bileşenler/Araçlar

**\***Ürün Gereksinim Dokümanı

**\***Sprint Dokümanı

**\***Sprint Kalan Zaman Grafiği

Roller

**1. Ürün Sahibi (Product Owner):** Projenin iş değeri açısından geri dönüşü ile sorumludur. Yalnızca kendisi teslimat, maliyet ve işlevsellik gibi kararlardan sorumludur.

**2. Scrum Yöneticisi (Scrum Master):** Scrum ustası Scrum'un başarılı olmasını sağlamaktan sorumludur. Takımın Scrum değerlerine, pratiklerine ve kurallarına bağlı kalmasını garantiye alır. Takımı ve organizasyonu Scrum' a adapte eder. Bunu başarmak için yazılım ekibi ile beraber çalışır ama takıma tabii olmaz.

**3. Geliştirme Takımı (Scrum Team):** Ekibin görevi ise ürün sahibinin taleplerine ve sıralamasına uygun ürünün işlevselliğini sağlamak ve belirli kalite standartlarına uyarak ürünü hazır etmektir. Scrum Takımı sürekli iletişim içerisinde olan, sayıları 5-9 arasında değişen ve aynı hedefe sahip kişilerden oluşur. Bir Scrum Takımında olması gereken başlıca özelliklerden birisi kendi kendine organize olabilmesidir.

Toplantılar

**1. Sprint Planlama Toplantısı:** Bu toplantıda geniş kapsamlı gereksinim listeleri çıkarılır, Kullanıcı hikayeleri önceliklerine göre sıraya dizilir, risk değerlendirilmeleri yapılır. Takımlar belirlenir Gözden geçirmeler ve olası gereksinim değişiklikleri belirlenir, geliştirme altyapılarına karar verilir ve onaylanır. Dağıtım, geliştirme ve pazarlama maliyetleri hesaplanır. Yönetim ve destekler gözden geçirilir ve onay verilir.

**2. Sprint Gözden Geçirme Toplantısı:** Her sprint sonunda yapılır.Değerlendirme takım tarafından yapılır ve başlangıçta belirlenen hedeflerin ürün kapsamında olma durumu ürün sahibi tarafından değerlendirilir. Eğer teslim edilen işlev eksikse o kullanıcı hikayesi tekrardan ürün içeriğine ürün sahibi tarafından gönderilir ve bir öncelik sırası verilir.

**3. Günlük Scrum Toplantısı:** Her iş günü başlamadan önce belirli saatte 15 dakikalık bilgi paylaşımı için günlük scrum toplanışı yapılır. Takımın ilerleyişi ve engelleri görmek için bir fırsattır. Eğer 15 dakika içerisinde bazı sorular cevapsız kalırsa Scrum Ustası bu soruları not alır ve bir sonraki toplantıda sunar veya kendisi çözüm üretir.

Bileşenler

**1. Ürün Gereksinim Dokümanı:** Proje için gerekli olan gereksinimlerin listesidir. Bu doküman canlıdır, geçerli ve kullanışlı olması için devamlı bakım gerekir. Birçok yeni eleman eklenebilir veya silinebilir, bazı gereksinimler artık geçersiz sayılabilir. Listedeki sıralamalar her gün değişiklik gösterebilir. Bu doküman kullanıcı bakış açısı tarafından bakılarak oluşturulur.

**2. Sprint Dokümanı:** Ürün ve gereksinim dokümanından elde edilen verileri kapsar. Amacı, sprint sonunda son ürünün parçası olan bir fonksiyonu veya çalışabilir parçayı elde etmektir. Sadece takım tarafından değiştirilebilir.

**3. Sprint Zaman Grafiği:** Bu grafik sprint boyunca işlerin normalde yapılması gereken ile ne kadar yapıldığını karşılaştırmayı sağlar. Yatay ekseni sprintin günlerini dikey ekseni sprintte kalan işi gösterir. Scrum'un temel ilkelerinden olan şeffaflığı sağlar.

SCRUM Günümüzde Neden Popüler?

Scrum bir analizden çok bir grup uygulamadır. Bir Scrum uygulaması yapılmadan önce ekip oluşturulur, bir dizi görevler ekip içinde dağıtılır, sürekli olarak etkileşimde bulunulur ve müşteri her zaman işe dahil olur. Düzenli yapılan toplantılar sayesinde sorunlara kolaylıkla çözüm üretilebilmesi, yinelemeli bir yol izleyişi ile hataların erkenden fark edilmesi, geliştirme uygulamalarında sahip olduğu esnekliği Scrum'ı günümüzde oldukça popüler hale getirmiştir. Çünkü yazılım yaşam döngüsü sürekli gelişen, değişen bir yapıya sahiptir. Bu da uzun süreçli modellerde maliyeti yükseltir ve gerek ekibi gerekse müşteriyi zor duruma sokar. Scrum ise bunun tersine değişime açık, çevik bir modeldir. Düzenli toplantılar yapılması ek olarak ekibin motivasyonunu yüksek seviyelerde tutan bir uygulamadır. Buna neden ihtiyacımız var? Çünkü insanların iyi projeleri piyasa sürmesi için yaptıkları işi severek ve isteyerek yapmaları gerekir. Yüksek motivasyon ile yapılan işler ürünün kalitesini arttırır, bu da yazılım şirketleri için bir beklentidir. Scrum bu beklentiyi karşılamamız için kullanıma uygun bir metottur.