

Yazılım yaşam döngüsü nedir ve yazılım yaşam döngüsüne neden ihtiyacımız vardır?

Öncelikle şu soruyu cevaplamamız daha doğru olacaktır:

Yazılım nedir?

Yazılım: “bir bilgisayarda donanıma hayat veren ve bilgi işlemde kullanılan programlar, yordamlar, programlama dilleri ve belgelemelerin tümü” olarak Türk Dil Kurumu tarafından tanımlanmaktadır.

Bu tanıma göre donanım ve işlenmesi gerekecek farklı veriler sizi yazılımınızı değiştirmeye veya güncellemeye itebilecektir. Bu da bize, sizin yazılımınızın sadece geliştirme evresinde değil kullanım evresinde de bir desteğe ihtiyacı olabileceğini gösteriyor.

Tanımdan ayrı kalarak bahsedersek ise yazılımların içindeki “parçalar” eskidiği ve onların yenisiyle değiştirilmesi gerektiği için değil, genellikle sistemin içerisinde kalan ve keşfedildikçe kaldırılması gereken bazı hatalar olduğu için yazılımların bakıma ihtiyaç vardır.   
Her geliştirme işlemi özellikle de ürün geliştirme işlemi belirli aşamalar içerir. Yazılımların da birer ürün olduğu unutulmamalıdır. Bu bağlamda nasıl bir otomobil, bir uçak veya bir televizyonun geliştirilmesi için çeşitli aşamalardan bahsetmek mümkünse, yazılım için de benzer aşamalar mevcuttur.   
Bu aşamalar ise verimi arttırabilmek, gelecek risklerden kaçınabilmek ve başarısızlık ile hata oranını düşürmek için yazılım yaşam döngüleri adı altında 20. Yüzyılın ikinci yarısından itibaren oluşturulmaya ve kullanılmaya başlanmıştır.

Peki yazılım yaşam döngüsü nedir?  
“Yazılımın ürününün hem üretim hem de müşterideki kullanım süreci boyunca geçirdiği tüm aşamalar yazılım geliştirme aşamalarıdır.” Şeklinde cevaplanabilmiştir.

Yazılım yaşam döngü modelleri temelde 5 kavramı esas alsalar da birbirlerinden çok farklı yöntemler içerebilmektedirler. Üç başlığa ayrılmış olan bu modelleri düzenleyici, birleşik ve çevik yazılım süreçleri olarak belirtebiliriz.

Yazılım yaşam döngülerinin temel aldığı 5 esas kavram:

Planlama: İhtiyaçların toplanması ile planlama evresi başlar. Bu evrede “Yazılım projemiz nasıl kullanılacak?”, ”Kim kullanabilecek?”, ”Yazılım projesinde hangi veri girdi/çıktı olarak kullanılacak?” vb. sorularının cevaplanması gerekmektedir.

Analiz: İhtiyaçlar toplandıktan sonra fizibilite çalışmaları başlar. Yani aslında ihtiyaçların yazılıma eklenebilme yetisi ölçülür ve bunun sonucunda şartname dokümanı yazılır. Ayrıca bu safhada yazılım proje yönetim planı hazırlanır.

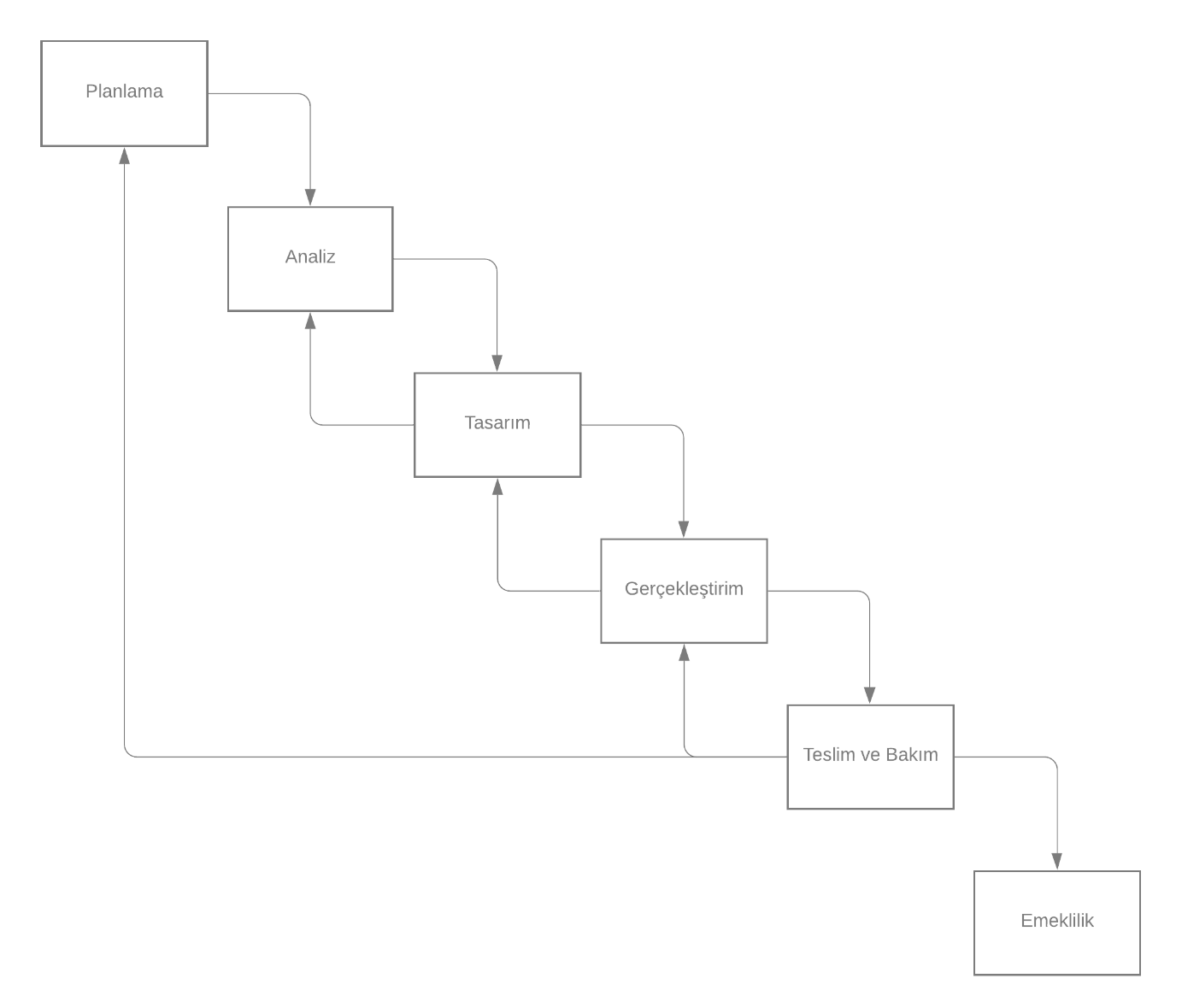
Tasarım: Tasarım safhası kendi içerisinde üst seviye ve mimari tasarım ile ayrıntılı tasarım olmak üzere ikiye ayrılmaktadır. Üst seviye ve mimari tasarım planların genellikle kaba taslak yapıldığı kısımken ayrıntılı tasarım kısmında programın nasıl işleyeceğini, hangi algoritmaların kullanılacağı belirlenir ve diyagram çizimleri yapılır.

Gerçekleştirim: bu safha kodlama evresinin her bir modül için başladığı ve yine her bir modül için birbirinden bağımsız olarak test edildiği evredir. Bağımsız şekilde yapılan bu testlere unit testing ismi verilmiştir. Modüllerin birbiri ile sorunsuz çalışması gerektiği için modüller birleştirilerek integration testing adı verilen diğer test aşaması başlar. En sonunda ise gerçek verilerin girilerek sistemin çalışırlığının ve kabul edilebilirliğinin test edildiği acceptance testing aşaması tamamlanır.

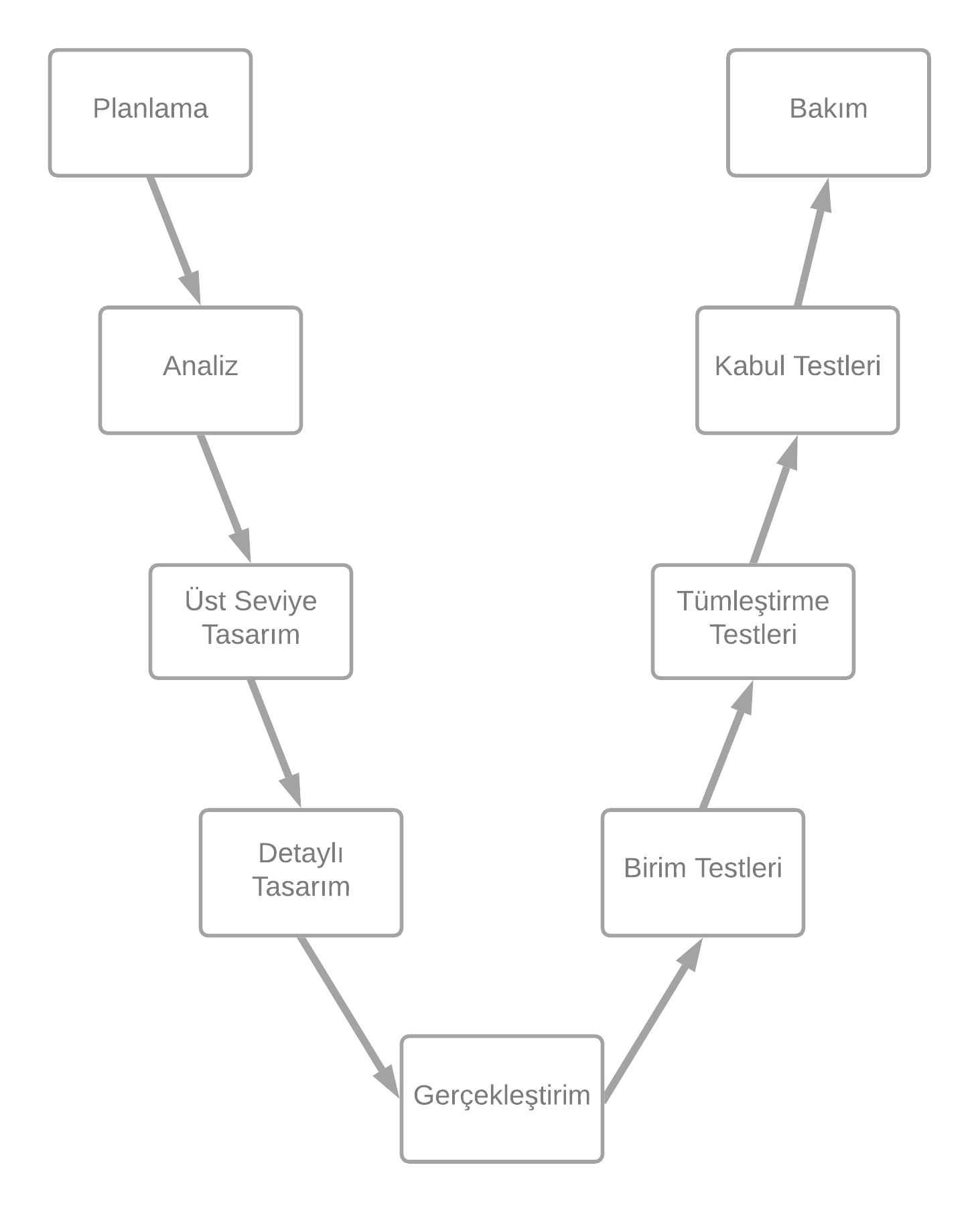
Teslim ve Bakım: Artık proje kullanıcıya teslim edilmiştir ve bu andan itibaren yapılan her değişiklik teslim sonrası safhaya girecektir. Bu safhada da hata düzeltimi ya da müşteri isteği veya ihtiyacı doğrultusunda yapılacak düzeltmeler ve değişiklikler yer almaktadır.

DÜNZENLEYİCİ SÜREÇ MODELLERİ:

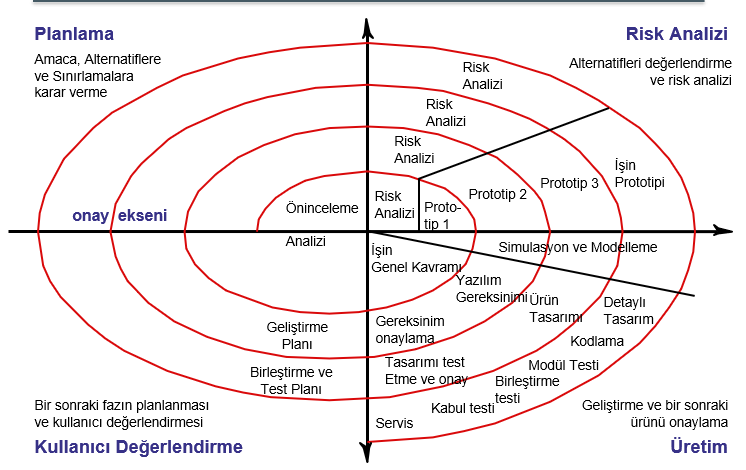
* Gelişigüzel Model:   
  1-Yöntem ve model kullanımı yoktur.  
  2-Projenin işleyişi sadece bireye bağlıdır ve genellikle tek kişi tarafından kullanılır.  
  3-Projenin takip edilebilirliğini ve bakımını zorlaştırır.
* Barok Modeli:  
  1-Temel alınan beş kavramın geliştirildiği modeldir.  
  2-Günümüzde sıkça kullanılan modellerin aksine belgeleme işlemini ayrıca bir süreç olarak ele alır.  
  3-Gerçekleştirim evresine diğer evrelerden çok daha fazla ağırlık verdiği ve adımlar arası geri dönüşün belirsiz olması nedeniyle günümüzde fazla kullanılmamaktadır.
* Kodla ve Düzelt (Code and Fix):  
  1- Hızlıca gerçekleştirim aşamasına geçildiği ve sonuçlar görülebildiği için en sık kullanılan   
  yazılım yaşam döngü modellerinden bir tanesi haline gelmiştir.  
  2-Planlamasız ya da çok ufak planlamalar ile başlar.  
  3-Keşfedilen hataların bakımı daha zor ve daha pahalı olacağı için projenin bakımını zorlaştırır.  
  4- Büyük projeler için oldukça verimsizdir.
* Şelale Modeli (Waterfall Model):  
  1-Bir adım tamamlanmadan sonraki adıma geçilmez.  
  2-Eksiklik veya hata fark edilirse bir önceki adıma geçilir.  
  3-En eski modellerden bir tanesi olmasına rağmen günümüz şirketleri tarafından sıkça kullanılmaktadır.  
  4-Oldukça basit olmasına rağmen fazla zaman tüketir.



* V Modeli (V-shaped Model):  
  1-Şelale Modelinin muaddel versiyonudur.  
  2-Her basamağın gözle görülür sonuçları vardır.  
  3-Fazlasıyla açıktır ve kullanımı basittir.  
  4-Sol taraf üretimi sağ taraf ise sınamayı temsil eder.  
  5-Basamaklar arasında tekrar ve risk çözümleme basamağı bulunmamaktadır.   
  5-Her basamak kendine eşdeğer bir basamağın karşısına gelerek yapıldığı için V ismini almıştır.



* Hızlı Uygulama Geliştirme (RAD) Modeli:  
  1-Hızlı prototip üretimi için planlama evresi çok kısa tutulmaktadır.  
  2-Sürekli ve hızlı bir gerçekleştirim aşaması olduğu için hata çıkma olasılığı çok daha fazladır.  
  3-Büyük projelerde tercih edilebilir bir yazılım yaşam döngü modeli değildir.  
  4-Daha fazla insan gücüne ihtiyaç duyar.
* Evrimsel Geliştirme (Evolutionary Development):  
  1-Diğer modellere göre daha yavaş bir ilerleyişe sahiptir.  
  2-Riski ve hata olasılığını azaltır.  
  3-Değişiklik için bir denetim mekanizması mevcut değildir.  
  4-Tam ölçekli olan ilk modeldir.
* Prototip modeli (Prototype Model):  
  1-Gereksinimler hızlı bir şekilde kararlaştırılır ama kullanıcılar göz ardı edilmez.  
  2-Yanlış analiz ihtimali minimalize edilmiştir.  
  3-Prototipler belgelendirme aşamasından yoksundur.  
  4-Kullanıcı, projenin başından itibaren sistem gereksinimlerini görebilir.
* Spiral Model:  
  1-Fazlasıyla risk analizi ve prototip üzerine kurulmuştur.  
  2-Hatalar, fark edilme süresi kısalacağı için daha erken ve daha ucuz bir şekilde giderilebilirler.  
  3-Fazla dokümantasyon oluşur.  
  4- Ufak çaplı projeler için maliyeti yüksek kalır.



* Formal Sistem Geliştirme (Formal System Development):  
  1- Oldukça basit olmasına rağmen çok zaman alır ve maliyeti yüksektir.  
  2- Projede bulunan hata ve belirsizlikleri saptar.   
  3- Projedeki hata oranını minimalize eder.
* Yeniden kullanıma yönelik geliştirme (Re-use based development):  
  1- Daha önceden var olan bir sistem üzerine çalışılarak yapılır.  
  2-Eski sistem iyice taranır ve saptanan yetersiz parçalar yerine muaddel halleri kullanılır.  
  3-Geliştirme işlemi eski bir sistem üzerine yapıldığı için maliyet ve risk ihtimali düşüktür.   
  4-Eski bir sistemin kalıplarına bağlı kalındığı için kullanıcının ihtiyaçları tam anlamıyla karşılanamayabilir.
* Artımlı Geliştirme (Incremental Development):  
  1-Olabildiğince kısa sürede çalışan bir sistem geliştirilir bunun sayesinde ilk teslim maliyeti düşüktür.  
  2-Yeni işlevsellikler basamak basamak ekleneceği için her sürümde kullanıcıdan dönüt alınabilmektedir.  
  3-Her aşamada test olduğu için risk daha düşüktür ve yönetimi daha kolaydır.  
  4-Maliyeti yüksektir ve planlamanın doğru yapılmasını gerektirir.

BİRLEŞİK SÜREÇ MODELLERİ:  
Genellikle diğer yazılım geliştirme modellerinin en iyi özelliklerini bir araya getirerek oluşturulan modellerdir

* Birleşik Süreç (Unified Process):  
  1-Nesneye dayalı yazılım geliştirme yöntemlerinin en iyi özellikleri bir araya getirilerek oluşturulmuştur.  
  2-Yinelemeli bir yapısı vardır ve her yinelemede bütün bir yazılım projesi varmış gibi davranılır.  
  3-Proje içerisinde erken ürün elde edimi sayesinde kullanıcıdan dönüt alındığı için kullanıcı ihtiyaçlarının ne kadar karşılandığı gözler önüne serilir.  
  4-Erken ürün eldesi sayesinde takımın morali yükselir.

ÇEVİK YAZILIM SÜRECİ:  
Çevik yazılım geliştirme süreçlerinin kendine has manifestosuna göre: Süreçler, araçlar, kapsamlı dokümantasyonlar, sözleşme pazarlıkları ve bir plana bağlı kalmaktan ziyade etkileşimlere, çalışan yazılıma, müşteri ile işbirliğine ve değişime karşılık vermeye yönelik bir süreç yönetimi sağlanacaktır.

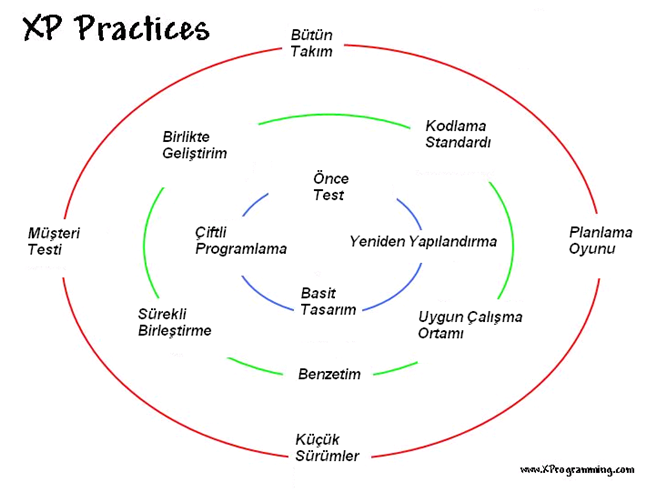
Extreme Programming (XP):  
Extreme programming; basitlik, cesaret, geri bildirim ve iletişim değerleri üzerine kurulmuş bir çevik yazılım modelidir.   
  
Basitlik; sadece projenin gereklerinin yapılıp lüzumsuz durumlardan kaçınılması olarak tanımlanabilecekken bir projenin basitleştirilmesi zor bir süreçtir.  
  
Cesaret; projelerdeki her sorunu yılmadan, bıkmadan çözebilme yetisidir ve Extreme Programming’in en zorlayıcı temellerinden bir tanesidir.  
  
Geri bildirim; kullanıcı ile etkileşimin fazla, proje tehlikelerinin az olmasını sağlayacak temel noktasıdır.  
  
İletişim; Extreme Programming’in temelinin kurulu olduğu, problemlerin hızla çözülmesini sağlayan ve kullanıcı etkileşimini arttırarak ortaya çıkabilecek sorunları minimalize eden temel noktasıdır.

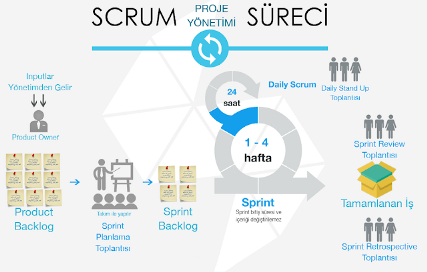
Bu dört temel nokta esas alınarak Extreme Programming için on beş adet Prensip oluşturulmuştur.

1. Hızlı Geri Bildirim (Rapid Feedback)
2. Basitliği Olumlamak (Assume Simplicty)
3. Artımlı Değişim (Incremental Change)
4. Değişimi Benimsemek (Embracing Change)
5. Kaliteli İş (Quality Work)
6. Öğrenmeyi Öğret (Teach learning)
7. Ufak Başlangıç Maliyeti (Little Initial Investment)
8. Kazanmak İçin Oyna (Play To Win)
9. Somut Deneyler (Concrete Experiments)
10. Açık, Dürüst İletişim (Open, Honest Communication)
11. Takımının İçgüdüleri ile Birlikte Çalış, Onlara Karşı Değil (Work With People’s Instincs, Not Against Them)
12. Kabullenilmiş Sorumluluk (Accepted Responsibility)
13. Yerel Uyarlamalar (Local Adaptations)
14. Hafif Taşı (Travel Light)
15. Dürüst Ölçüm (Honest Measurement)

Extreme Programming bütün bu prensipler ve değerler doğrultusunda 12 uygulamaya da sahiptir ve bu uygulamalar şunlardır:

Planlama oyunu, ekipte müşteri, müşteri testi, basit tasarım, çiftli programlama, sürekli tümleştirme, kısa sürümler, yeniden yapılandırma, ortak sahiplenme, benzetim, kodlama standardı ve haftada kırk saat.



* SCRUM:  
  Scrum çevik yazılım metodolojilerine uygulanabilecek bir proje yönetim yaklaşımıdır.  
  Taahüt, cesaret, odaklanma, açıklık ve saygı üzerine kurulmuş olan scrum bu değerlerin varlığı olmadan uygulanamaz.

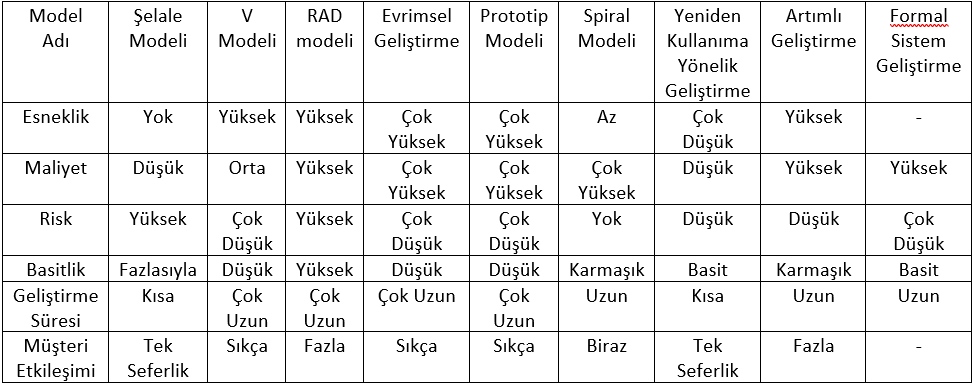
Product Backlog: Müşteri’den gereksinimlerin alınıp takım içerisinde önceliklendirildikten sonra müşteri ile bir anlaşmaya varıldığı evredir.  
Sprint Backlog: İki ile dört hafta arasında seçilmiş bir zaman dilimidir. Projeler birçok sprintten oluşur.  
Daily Stand Up Meeting : En fazla yarım saat ve ayaküstü olmak üzere scrum takımlarının proje gidişatı ve genel durum hakkında bilgi alışverişi yaptıkları günlük toplantılardır.

Scrum üç temel kavrama sahiptir:

1-Roller  
Ürün Sahibi: İş listesini yönetir ve bunu yaparken yatırım getirisi, risk, teknoloji ve iş değeri gibi faktörleri göz önünde bulundurur.  
Scrum Yöneticisi: Ürün sahibine öğreterek, geliştirme takımına ise önlerindeki engelleri kaldırarak yardımcı olur. Aynı zamanda takımın teori ve pratik kurallarına bağlı kalmasından sorumludur.  
Scrum Takımı:  Bir işin kendisine verilmesini beklemez onun yerine işi kendisi alır ve geliştirmeye başlar. Sorumluluğu alınan işte beklentilerin karşılanması yönündedir.

2-Toplantılar  
Sprint Planlama Toplantısı: Risk değerlendirmesinin, takım seçilmesinin, geniş kapsamlı gereksinim listelerinin çıkarttırılmasının, geliştirme araçlarının seçilmesinin ve maliyet hesaplanmasının gerçekleştirildiği toplantıdır.  
Sprint Review Toplantısı: Her sprint başında yapılan, ürün sahibinden geri bildirim alındığı, sprint gereksinim listesinin oluşturulduğu ve ürün sahibi tarafından belirtilmiş gereksinimlerin ne kadarının yapılacağının takım tarafından taahhüt edildiği toplantıdır  
Daily Stand Up Toplantısı: Takım üyelerinin soru ve sorunlarının paylaşıldığı günlük planların ve ilerlemelerin paylaşıldığı ayak üstü ve kısa süreli toplantılardır.

3-Bileşenler/Araçlar  
Ürün Gereksinim Dokümanı (Product Backlog): Projede istenen iş elemanlarını içinde bulunduran ve bu yüzden de zamanla değişebilme potansiyeli olan basit listedir.  
Sprint Dokümanı (Sprint Backlog): Ürün gereksinim dokümanından elde edilmiş iş ve görevleri kapsayan, çalışabilir ya da işlevsel bir parça oluşturulmasını hedefleyen ve sadece scrum takımının kendisi tarafından değişikliğe uğrattırılabilen dokümandır.  
Kalan Zaman Şeması (Burndown Chart): Bu şemada scrum takımı tarafından yapılan ve yapılması beklenen iş verileri karşılaştırılır.

MODELLERİN KARŞILAŞTIRILMASI:  
1-Düzenleyici Süreç Modellerinin Karşılaştırılması:

2-Çevik Süreç Modellerinin Karşılaştırılması:  
Scrum takımları Sprint adı verilen ve iki hafta ile bir ay arası süren yineleme basamaklarına sahiptir. XP takımları ise bu yinelemeleri bir ile iki hafta arasındaki bir uzunlukta yapmaktadırlar.  
Scrum takımları toplantının ardından belirlenen yinelemeleri bitirene kadar onlarda değişiklik yapamazlarken XP takımları bu değişiklikleri belirli kriterler altında yapabilmektedirler.  
XP takımları müşteri tarafından belirlenen katı bir öncelik sıralaması ile çalışırlar. Scrum’da ise bu öncelik sıralaması takım tarafından belirlenmektedir.  
Scrum herhangi bir uygulama yöntemine sahip değilken XP on iki adet uygulama yöntemine sahiptir.

HANGİ PROJEDE HANGİ MODELİ KULLANMALIYIZ?  
Uzun süreli, analiz ile tasarımın çok kritik olduğu ve yapılan herhangi bir hatanın maliyetinin çok yüksek olduğu projelerde Şelale Modeli tercih edilebilir.  
Tanımlamada belirsizlik az ya da yok ise ve proje kısa sürede üretilebilecek ise V modeli tercih edilebilir.  
Coğrafi olarak uzak birimlere ayrılmış müşterilerin projelerinde Evrimsel Geliştirme Modeli tercih edilebilir.  
Büyük ve yüksek riskli projeler için Spiral Model tercih edilebilir.  
Kullanıcı ile sürekli iletişim halinde kalarak düzenli geri bildirimler alınıp buna göre proje geliştirilmek isteniyor ise Prototipleme ya da Artımlı Geliştirme Modeli tercih edilebilir.  
Nesneye dayalı yazılım projeleri geliştirilmek istendiğinde Birleşik Süreç tercih edilebilir.  
Maliyetin düşük tutulmasının istendiği eskimiş ya da artık ihtiyaçları karşılamayan programlara sahip müşterilerde Yeniden Kullanıma Yönelik Geliştirme tercih edilebilir.  
Geliştirilmek istenen sistem karmaşık ve sonradan hata ayıklaması pahalı olacak ürünler için Formal Sistem Geliştirme tercih edilebilir.  
Karmaşık ortamlarda adım adım geliştirilmek istenen ve yazılım geliştirme safhasında geç ortaya çıkan gereksinim değişimlere sahip projelerde Scrum ya da Extreme Programming tercih edilebilir.

SCRUM GÜNÜMÜZDE NEDEN POPÜLER?  
Scrum gelip geçici bir hevesten fazlası. Çeşitli takımlara ve projelere uygulanan başarılı ve sağlam bir çevik model. Üniversiteler, değerli projelerini müşterilerine ulaştırmak için scrum kullanıyor. Askeriyeler gemi imalatlarında Scruma bel bağlıyor. Otomativ sektöründe Wikispeed takımı hızlı, makul ücretli, çok yüksek verimli ve güvenli bir gündelik arabayı 20.000 Doların altına satabilmek için scrum kullanıyor.   
Peki scrum neden bu kadar yaygın?  
Scrum’ın yaygınlığı, basitliği ve yüksek performansından geliyor. Başarım, pozitif geri bildirim ve takım çalışması ile gerçekleştirilen bir projenin üzerine elde hissine karşı olan açlığı doldurabilmenin avantajını elinde tutuyor. Hangi yüksek performanslı proje işlemi içerisinde olursanız olun “dişlilerin yağlanması” gereklidir. Scrum ise bu “dişlilerin yağlanmasını” sağlayarak bir projeyi çok daha kolay ilerleyebilecek hale getirmektedir.

KAYNAKÇA

1. <https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696>
2. Doç. Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş Dersi 2. Ve 3. Hafta Ders Sunumları
3. Apoorva Mishra & Deepty Dubey International Journal of Advance Research in Computer Science and Management Studies
4. Vishal Chandra Comparison between Various Software Development Methodologies
5. https://zone.ni.com/reference/fr-XX/help/371361R-0114/lvdevconcepts/lifecycle\_models/
6. https://docplayer.biz.tr/5692380-Hizli-uygulama-gelistirme-rapid-application-development-rad-model.html
7. https://caglartelef.com/yazilim-yasam-dongusu/
8. Seker, S. E. 2014 Yazılım Geliştirme Yaşam Döngüsü, YBS Ansiklopedi, v. 1, is. 2, pp. 2-5
9. <https://www.geeksforgeeks.org/reuse-oriented-model/>
10. <https://www.yazilimtestmerkezi.com/post/yazilim-gelistirme-modeli-nedir-cesitleri-nelerdir>
11. Süleyman Güneş Birleşik Süreç ve Çevik (Agile) Yazılım Süreç Modelleri Sunumu
12. Highsmith, J., (2002), “Agile Software Development Ecosystems”, Addison Wesley.
13. <https://www.quora.com/Why-is-the-Scrum-process-so-popular-in-the-software-industry>
14. <http://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>
15. <http://www.kurumsaljava.com/2008/11/21/extreme-programming-nedir/>
16. <http://www.yazilimcilardunyasi.com/2017/04/extreme-programing-nedir.html>
17. https://medium.com/@secilcor/scrum-nedi̇r-6a4326951dd8
18. <http://www.ykcgrup.com.tr/selale-modeli-nedir/>
19. Dr. Muhammet Baykara Yazılım Mühendisliği Temelleri 2. Hafta Sunusu
20. http://www.yilmazcihan.com/scrum-roller-ve-sorumluluklar/