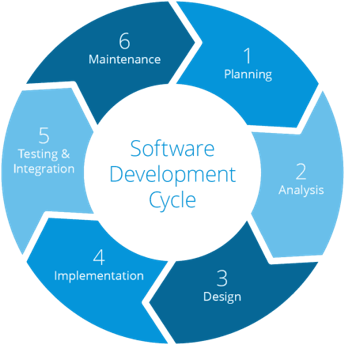
**Yazılım Yaşam Döngüsü**

Yazılım yaşam döngüsü (software development life cycle, SDLC) benim gibi bilgisayar mühendisi adaylarının, bilgisayar ve yazılım mühendislerinin, kısaca tüm programcıların kullandığı, herhangi bir projenin hazırlık, üretim ve kullanım aşamaları dahil geçirdiği tüm aşamalardır. Yazılım ile ilgili gereksinimler, müşterinin istekleri gibi birçok değişik etken olması ve bunların zamanla değişebilmesi sebebiyle bu aşamalar sürekli bir döngü içerisindedir. Bu nedenle programcı, projenin herhangi bir döneminde önceki aşamalara geri dönme veya projeyi baştan tasarlama gibi ihtimallerle karşı karşıyadır. Yazılım Yaşam Döngüsündeki “Döngü” kelimesi buradan gelmektedir. Yazılım yaşam döngüsü; planlama, analiz, tasarım, gerçekleştirme ve bakım olmak üzere toplam 5 adımdan oluşur. Bu adımların hepsini yazımın ilerleyen kısımlarında elimden geldiğince açıklamaya çalışacağım fakat oraya gelmeden önce bu aşamalarla ilgili eklemek istediğim birkaç genel düşüncem var. Öncelikle “Planlama” hakkında konuşmak istiyorum. Planlamanın kulağa ne kadar basit ve kısa bir süreç olarak geldiğinin farkındayım ama ben bu 5 aşama arasındaki en önemli aşamayı planlama olarak buluyorum. Çünkü projenin tüm planı ve nasıl ilerleyeceği, projenin temel gereksinimleri ve müşterinin istekleri bu aşamada belirlenir. Müşterilerin isteklerini eksiksiz ve yanlışsız bir şekilde anlamak çok büyük önem arz eder. Çünkü bir programcının projesindeki en önemli amaç, müşterinin isteklerini eksiksiz bir şekilde yerine getirebilmektir. Eğer müşterinin isteklerini tam olarak anlayamayıp tüm projeyi bu yanlış isteklere göre planlayıp hazırlarsak müthiş bir zaman kaybı ve mali problemlerle karşı karşıya kalmamız işten bile değil. Bu yüzden planlama aşamasının en önemli aşama olduğu kanaatindeyim. Böyle sorunlarla karşılaşmamak için planlama adımına büyük bir önem ve zaman vermeliyiz. İkinci aşamamız olan “Analiz”e gelirsek, kısaca gereksinim ve işlevlerinin ayrıntılı olarak incelendiği aşamadır diyebiliriz. “Tasarımdan ise kabaca analiz aşamasında belirlenen gereksinimler doğrultusunda yazılım projesinin nasıl sürdürüleceğinin tasarlandığı aşama olarak bahsedebilirim. “Gerçekleştirme”, kodlamanın yapıldığı, projenin test edildiği ve projeyle ilgili “oluşturma” aşamasının bittiği adımdır. Daha sonra son aşamamız olan ve programın tüm ömrü boyunca devam edecek olan “Bakım” aşaması geliyor. Bu aşamada tamamen bitmiş ve sahaya sürülmüş olan programdaki hataları giderilir ve ihtiyaç duyulan yeni güncellemeler eklenir. Şimdi bu döngü adımlarının tam olarak tanımlarını yapmaya gelelim:



**Planlama (Planning) Aşaması:** Müşterilerin dinlenerek isteklerinin tam, eksiksiz ve net bir şekilde alındığı ve gereksinimlerin belirlendiği aşamadır. Programın yapımında kullanılacak olan yazılım, donanım ve personel tarzında ihtiyaçların türü ve miktarı belirlenir. Ve son olarak tüm projenin planlaması yapılır.

**Analiz (Analysis) Aşaması:** Yazılım işlevleri ile gereksinimlerinin ayrıntılı olarak incelendiği, bunun ne kadar süreceği ve risklerinin belirlendiği aşamadır. Temel UML diyagramlarının çizilmeye başlanıldığı ilk adımdır.

**Tasarım (Design) Aşaması:** Yazılımın, planlama ve analiz aşamasından elde edilen bilgiler doğrultusunda yazılımın özellikleri, yetenekleri ve arayüzlerinin nasıl olacağının belirlendiği adımdır. Tasarım aşaması “Mimari Tasarım” ve “Detaylı Tasarım” olmak üzere 2 bölümden oluşur. Mimari Tasarımda genel bir plan yapılır ve modüller belirlenir. Detaylı Tasarımda ise programın hangi dil veya dillerde yazılması, kullanılacak olan algoritmaların belirlenmesi, veri tabanı tasarlanması ve bunun gibi daha ayrıntılı noktalar belirlenir.

**Gerçekleştirme (Implementation) Aşaması:** Bu aşamaya gelindiğinde artık tüm planlama ve analizler bitmiş, tasarım aşamasında programın nasıl olması gerektiği belirlenmiş olmalıdır. Ve bu belirlenen kriterlere göre programın kodlanmaya başlandığı, yani sistemin yaşatılmaya başlandığı aşamadır. Kodlama, test etme ve kurulum çalışmaları olarak 3’e ayrılır. Kodlama aşamasında KISS (keep it simple) prensibine göre kodlar, yani okunabilirliği ve bakımı kolay olan, arı kodlar yazılmalıdır. Test aşamasında ise kodlama sürecinde ve kodlama sonrasında yazılan kodlar sürekli test edilir. Programı edebildiğimiz en kısa aralıklarla test etmeliyiz ki herhangi bir hata durumunda hatanın üzerine fazla zaman geçmesin ve devamında benzer hatalar yapılmasın. Yapılan test sayesinde hatayı ne kadar erken fark edersek zamandan, maliyetten ve prestijden kaybı bir o kadar azaltacaktır. Birim testleri, duman testleri, yanlış değer testleri, kabul testleri, kullanım senaryo testleri, yük testleri, kullanıcı kabul testi, yoldan geçen adam testi, test otomasyonu gibi programımızı test edebileceğimiz çok sayıda test yöntemi bulunmaktadır. Daha sonra ürünün kurulumu tamamlanır ve piyasaya sürülür. Buradan sonra projenin kalan hayatı boyunca “Bakım” aşaması devam edecektir.

**Bakım (Maintenance) Aşaması:** Kodlama ve tüm test aşamaları tamamlandıktan sonra ürün piyasaya çıkartılıp müşteriye teslim edilir. Buradan sonra “Bakım” aşaması başlar. Bakımın temel amacı, programda var olan hataları gidermek, muhtemel hataların yaşanmaması için çalışmalar yapmak ve programda eksik olan ya da ihtiyaçlar doğrultusunda eklenmesi gereken yenilikleri getirmektir. Bu aşama programın tüm yaşamı boyunca sürecektir.

**Yazılım Yaşam Döngü Modelleri**

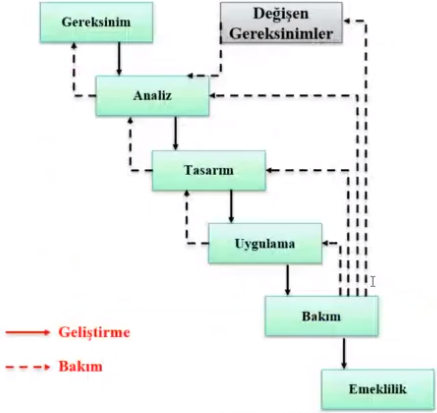
Yazılım yaşam döngüsünün daha aktif olması ve en iyi sonucu verebilmesi için yazılım geliştirme süreç modelleri (Software Process Models) kullanılır. Yazılım yaşam döngüsünde birden fazla model vardır. Bunun nedeni farklı gerekli hedeflere ve amaçlara daha kolay bir şekilde hizmet edebilmektir. Bu modellerin ortaya çıkmasında kendi dönemlerinin yazılımsal ve donanımsal ihtiyaçları ve sektörün ihtiyaçları büyük etkendir. Bu süreç modelleri kendi arasında “Düzenleyici Süreç Modelleri”, “Birleşik Süreç” ve “Çevik Yazılım Süreci” olmak üzere 3’e ayrılır.

Düzenleyici Süreç Modelleri

* **Gelişigüzel Model:** Bu modelde belirli bir yöntem yoktur. Bu yüzden kendisine “Gelişigüzel Model” denir ve model olarak adlandırmak pek de doğru olmaz. 1960’larda kullanılan bu yöntem, genellikle tek bir kişinin yaptığı basit programlamalı üretimlerde kullanılır. Genellikle kişiye bağlı yazılım geliştirme şeklinde kullanıldığı için bu yazılımın izlenebilirliği ve bakım yapılabilirliği çok olası değildir.
* **Barok Modeli:** Barok Modelinde Yazılım Yaşam Döngüsündeki temel adımların (Planlama, Analiz, Tasarım, Gerçekleştirme ve Bakım) hepsi vardır ve doğrusal olarak ele alınır. Ancak bu adımlar arasında olması gereken geri dönüşlerin nasıl yapılacağı tanımlı değildir. Ayrıca 1970’lerde kullanılan Barok Model’inde belgeleme apayrı bir süreç olarak ele alınır. Günümüzde ise Barok Modeli kullanılmamaktadır.
* **Çağlayan Yaşam-Döngü (Waterfall) Modeli:** Waterfall Modeli, geçmişte en popüler model olarak görülmüş, en temel, en tanınmış ve en popüler modeldir. Geleneksel yazılım geliştirme modeli olarak da bilinir. Bu modelde tüm aşamalar en az birer kez tekrarlanır. Bir aşama tamamıyla bitmeden bir sonraki aşamaya asla geçilmez. Her aşamada belgeleme yapılmak zorundadır. Eğer bir aşamada belgeleme ve test yapılmamış ise bu, o aşamanın tamamıyla bitmemiş olduğu anlamına gelir ve o aşama tamamlanmadan bir sonraki aşamaya geçilmez. Çağlayan Yaşam-Döngü Modeli ile Barok Modeli arasındaki farklara bakacak olursak, bu farklardan en önemlisi Barok Modelinde belgeleme apayrı bir süreç olarak ele alınmasına rağmen Çağlayan Yaşam-Döngü Modelinde her aşamanın sonunda bir belgeleme yapılır. Başka bir fark ise Barok Modelinde aşamalar arasındaki geri dönüşlerin tanımlı olmamasına karşın Çağlayan Yaşam-Döngü Modelinde bu geri dönüşler tanımlıdır. Çağlayan Yaşam-Döngü Modeli çok statiktir ve değişimlere elverişsizdir. Çünkü detaylı bir gereksinim aşamasından sonra gereksinimlerin doğru bir şekilde elde edildiğini ve bunun üzerine bir daha değişmeyeceğini ele alır. Fakat hayat devam ediyor, süreçler ve işler değişiyor, bu sebeple değişim kaçınılmaz oluyor. Tasarım aşaması da tüm gereksinimleri karşılayacak şekilde detaylı bir çalışma gerektirdiğinden bu iki adımda oldukça uzun bir süre harcanır ve bundan dolayı zamanla değişen gereksinimlerden kaçınmak imkânsız olacaktır. Bu iki aşama bittikten sonra, kodlama ya da test aşamasına denk gelen bu

değişiklikleri programa uygulamak oldukça maliyetli olacaktır. Bu da Waterfall modelinin bir eksisi diyebiliriz. Kullanıcı, geliştirme sürecinin içinde yer almadığından yazılımın kullanıcıya ulaşma zamanı uzundur. Bu yüzden olası bir sorun çok geç fark edilebilir, kullanıcının güncellenen istekleri çok geç alınır ve maliyetin büyük oranda artmasına ve sürecin daha da uzamasına sebebiyet verir. Çağlayan Yaşam-Döngü Modelinde kodlama dışındaki aşamalar çok uzun sürdüğünden dolayı bir an önce kodlamaya başlayıp programı bitirmek isteyen programlama ekibi mutsuzlaşır ve kod yazma dışındaki adımlara çok önem vermemeye başlar. Çok iyi tanımlanmış ve kısa sürecek projelerde kullanılması uygun bir modeldir. Kısa sürede tamamlanacak, küçük çaplı projelerde Waterfall modeli kullanılabilir. Bunun haricindeki uzun sürecek ve büyük projelerde yukarıda bahsettiğim sebepler yüzünden pek tercih edilmemektedir. Bu yüzden günümüzde kullanımı gittikçe azalmaktadır.

Waterfall Modelini güzel bir şekilde özetleyen bir görsel:



* **V Süreç Modeli (V- Shaped Model):** V Süreç Modelinde en önemli şey test-sınamadır.V süreç Modelini V harfinin sol tarafındaki çizgide üretimler, sağ tarafındaki çizgide ise (sol tarafın karşısında kalan bölümde) o üretimlerin test (sınama) işlemleri olarak düşünebiliriz. Bu şekilde süreç aşağı doğru eğimli bir şekilde inerken üretimler, yukarı doğru eğimli bir şekilde çıkarken testler olduğundan bu modele V Süreç Modeli denmiştir. V Süreç Modelinin 3 tane temel çıktısı vardır. Bunlar:

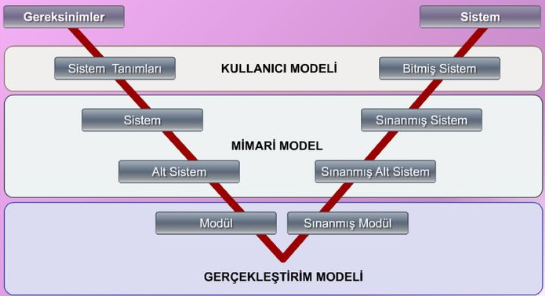
**Kullanıcı Modeli:** Kullanıcının istekleri ve proje için tanımları kullanılır.

**Mimari Modeli:** Projenin tasarımı ve alt sistem ile tüm sistemin sınama işlemlerinden oluşur.

**Gerçekleştirme Modeli:** Programın kodlanması ve bunların sınama işlemlerinden oluşur.

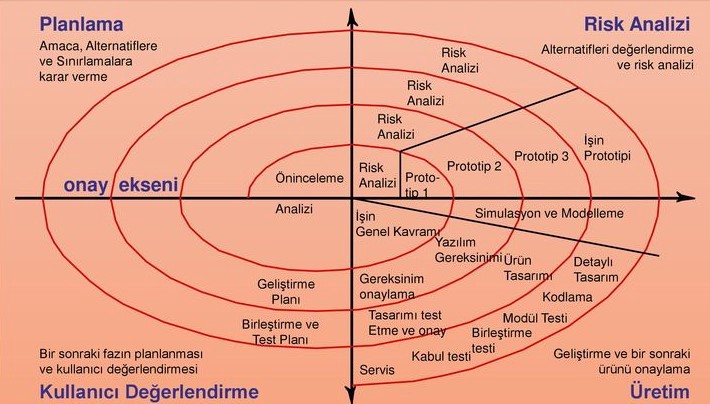
Bu Modelde çok sayıda test yapıldığı için kullanıcının projeye katkısı yüksektir. V Süreç Modelinde aşamalarda tekrar olmamasını ve risk çözümlemenin bulunmamasını bu modelin eksileri içinde sayabiliriz.

V Süreç Modelini anlatan bir görsel:



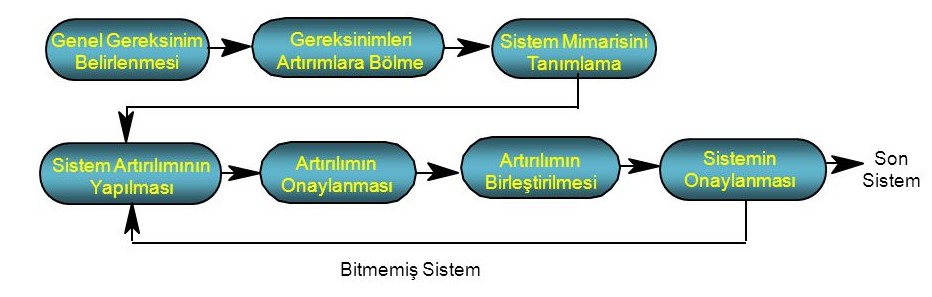
* **Helezonik (Spiral) Model:** Helezonik Model, planlama, risk analizi, üretim ve kullanıcı analizi olmak üzere toplam 4 ana aşamadan oluşur. Bir spiral oluşturacak şekilde dönerek bu aşamaların hepsi tekrar eder. Helezonik Modeli diğer modellerden ayıran en büyük farklardan biri prototiplere sahip olmasıdır. Bu prototipler sayesinde kullanıcılar sistemi erkenden görebilirler. Bir diğer ve Helezonik Modeli Çağlayan Yaşam-Döngü Modelinden ayıran farkı ise sürekli risk analizi yapılmasıdır. Sürekli risk analizi yapıldığından dolayı sistemdeki sıkıntılar en kısa sürede çözülür ve oluşabilecek muhtemel sıkıntıların da farkına varılmış olunur. Her aşamada kullanıcı değerlendirmesi olduğundan, olası bir gereksinim güncellemesi durumunda fazla zaman kaybetmeden güncelleme yapılabilir ve bu sayede fazla maliyet problemlerinin önüne geçilir. Planlamayı üretilecek ara ürün için planlama, projedeki amacı belirleme ve bir önceki ara ürün ile bütünleşme olarak tanımlayabiliriz. Risk analizi ise basit bir şekilde olası risklerin araştırılması ve belirlenmesidir. Üretim aşamasında ana ürün üretilir. Kullanıcı değerlendirmesinde ise ara ürün ile ilgili olarak kullanıcı tarafından yapılan test ve değerlendirmeler incelenir. Helezonik Modelde projenin büyüklüğüne göre prototip sayısı artıp azalabilir, aynı şekilde spiral sayısı da projenin büyüklüğüne göre değişiklik gösterebilir. Küçük ve düşük riskli projeler için devamlı olarak prototip çıkarılıp dokümantasyon yapıldığından maliyeti fazladır. Kullanıcı katkısının fazla olması, risk analizi yapılması, yönetici bakışı olması, yinelemeli artımsal bir yaklaşımı olması gibi sebeplerden dolayı çağdaş modellere son derece yakın bir modeldir.

Helezonik Modelini anlatan bir görsel:



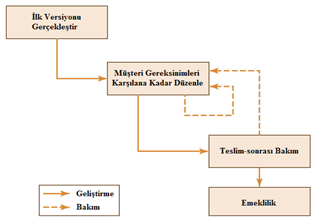
* **Artımsal Geliştirme Süreç Modeli:** Artımsal Geliştirme Modelinde proje parçalara bölünür ve kullanıcının gereksinim önemine ve birbirine bağımlılıklarına göre bu parçalar sıralanır. Sıralanan bu parçalar bittiğinde sırasıyla birer ara ürün geliştirilmiş olur ve bu ara ürünler de kullanıcı tarafından kullanılır. Ara ürünler birbirini kapsayacak ve işlevsellikleri giderek artacak şekilde geliştirilir. Bu modelde bir taraftan üretim kısmı sürerken diğer tarafta ise kullanım kısmı sürer. Her türlü projeye uygun olan bu model, genellikle uzun zaman alabilecek ve ürünün eksik işlevsellikle çalışabileceği türdeki yazılımlar için kullanılır. Sistem artırılması yapılarak devamlı bir şekilde gelişmiş sürümlerin ortaya çıkmasından ve bunların müşteriye teslim edilmesinden dolayı her teslimle birlikte müşteriye bir ürün geldiği için sistemin işlevselliği erken aşamalarda ortaya çıkar. Bu yüzden projenin tamamen batma riski azalır. Erken teslimler, sonraki teslimler için gereksinimleri çıkarmada prototip görevi görür.

Artımsal Geliştirme Süreç Modelini gösteren bir görsel:



* **Kodla ve Düzelt Yaşam-Döngü Modeli:** Birkaç yüz satırdan oluşan, küçük programlar için kullanılabilir. Herhangi bir planlamaya ihtiyaç duymaz. Direkt olarak yazılım ürünü gerçekleştirilir. Program en son istenilen şekle gelinceye kadar düzenlenir. Sisteme ait dokümantasyon olmadığından bakım safhası oldukça zordur. Emeklilik (retirement) safhası vardır.

Kodla ve Düzelt Yaşam-Döngü Modelini gösteren bir görsel:



**Birleşik Süreç (Unified Process)**

**Yinelemeli (Iterative):** Problemdeki istekler bir bütün olarak görülmez. Problemlerdeki hedeflerin bir kısmı ele alınır ve sınanmış bir ürün ortaya konulur. Her ürün oluşumunun sonunda bir sonraki yinelemeye(iterasyon) geçilir ve yeni istekler ele alınarak geliştirilir. Her bir iterasyon sonunda hedeflenen ürüne yaklaşılır. Her bir iterasyona ayrı bir proje gibi bakılır ve tüm aşamalardan geçerek sınanmış ve çalışır bir ürün elde edilir.

**Arıtmalı ve Evrimsel (Incremental and Evolutionary):** Her yinelemenin ardından farklı ve yeni istekler ele alındığı için elde edilen ürünlerin özellikleri artarak gelişir ve istenilen ürüne daha da yaklaşılır.

**Risk Güdümlü (Risk-Driven):** İlk iterasyonda en riskli kısımlar ele alınır. Böylece proje daha ilerlemeden oluşabilecek büyük problemler görülür ve gerekli önlemler alınabilir. Örneğin bütçe, personel ve zaman gibi etkenler kontrol edilebilir.

Çevik Yazılım Süreci (Agile Programming)

Yazılım sektöründe, yazılım sürümlerinin zamanında ortaya çıkarılamaması, yazılım hatalarının geç fark edilmesi, müşterilerin değişiklik ve yenilik isteklerine kısa sürede yanıt verilememesi gibi çeşitli sorunların aşılmasına yönelik 1990’lı yılların sonlarına doğru “çevik (Agile)” olarak isimlendirilen metotlar geliştirilmiştir. Dolayısıyla çevik yazılımın amaçları olarak; hızlı ürün üretme, değişen isteklere en kısa sürede yanıt verebilme ve bir yazılım ürünün en kısa sürede müşteriye ulaştırmayı sayabiliriz. Çevik yazılım geliştirme metotları, yüksek verimli, hata oranı düşük, hızlı ve ucuz çözümler sağlamaktadırlar. Çevik yazılımda yinelemeli bir sistem kullanılır. Her yineleme kendi içerisinde apayrı bir proje gibi düşünülür ve 2-4 hafta sürer. Tamamlanan her yinelemenin sonunda müşteriye çalışan bir program teslim edilerek müşteri memnuniyetinin artması sağlanır. Projenin küçük parçalardan oluşması geriye dönük hataların düzeltilmesini kolaylaştıran bir etkendir. Çevik’in bu kadar hızlı olmasının sebeplerinden birisi de projede çalışan ekip üyelerinin sürekli bir iletişim içinde olmalarıdır.

* **Extreme Programming (XP):** 1999 yılında Kent Beck tarafından ortaya çıkarılan yazılım geliştirme disiplinidir. Bu metot basitlik, cesaret, iletişim ve geri bildirim olmak üzere 4 adet temel maddeden oluşur.

**XP’nin Temel değerleri:**

* **İletişim:** Sorunlu projeler incelendiğinde, sorunun çıkmasını sağlayan en büyük etkenlerden biri iletişimdir. Ekip arasındaki iyi bir iletişim, tüm projenin çok daha sağlıklı ilerlemesini sağlar. Bu yüzden XP, projenin başarıya ulaşabilmesi için ekibin sağlıklı bir şekilde iletişim kurmasının önemini vurgular. XP’ye göre iletişim yüz yüze olmalıdır. Sadece ekibin içindeki iletişim değil, proje ekibiyle müşteri arasındaki iletişimin de iyi bir seviyede olması gereklidir. Bu iletişim sağlandığı durumda sorunlar daha erken fark edilir. Bilgi alınması gereken noktada iyi iletişim sayesinde kısa bir sürede alınan bilgi, projenin hız kesmeden devam etmesinde önemli rol oynar.
* **Geri Bildirim:** Geri bildirim, sistemler için olmazsa olmazdır. İyi bir iletişim sayesinde kaliteli bir geri bildirim mekanizması sağlanır. Bu sayede ortaya çıkabilecek hatalar ortadan kaldırılır.2-4 hafta aralıklı sürümlerle sistemin durumu belirlenir. Yazılım ekibiyle müşterilerin belirli aralıklarla yüz yüze olacak şekilde buluşarak değerlendirmeler yapması sayesinde ilerde oluşabilecek hatalar engellenir ve projenin istenilen doğrultuda gittiğinden emin olunur.
* **Basitlik:** Basitlik, zorunlu işlerin yapılmasıdır. O anki gereksinimleri kapsayan en basit çözüm kullanılmalıdır. Ancak basitliği sağlamak zordur. Çünkü basitleştirmek istenirken ayrıntı gibi görünen önemli hususları atlamak, başka problemlere sebebiyet verebilir.
* **Cesaret:** XP’nin 4 temel noktası arasından en zor olanı cesarettir. Yapılan işte cesur olunmalıdır, projelerin üstüne korkmadan yürünmelidir. Başarısızlık durumunda mühendisler hız kesmeden devam etmeli, gerekirse tüm kodu silip en baştan başlayabilmelidir. XP’nin anlayışında başarısızlıktan korkmak değil, başarısızlığı en kısa sürede telafi etmek vardır. Başarısız olmaktan korkmak, mühendisin yavaş hareket etmesine sebep olur ve bu da projenin hızının düşmesine neden olur. Bu yüzden cesur olmak önemlidir.

**XP Pratikleri:** Yazılım geliştirmede kolaylığı ve esnekliği sağlamak için XP 12 farklı pratiği ön görür. Bunlar;

1. Planlama oyunu
2. Ekipte Müşteri
3. Önce Test
4. Basit Tasarım
5. Çiftli Programlama
6. Sürekli Entegrasyon
7. Kısa Aralıklı Sürümler
8. Yeniden Yapılandırma
9. Ortak Kod Sahiplenme
10. Metafor
11. Kodlama Standardı
12. Haftada 40 Saat

* **SCRUM:** SCRUM, kelime anlamı olarak Rugby sporundaki bir hücum taktiğidir**.** Bu taktikte tüm oyuncularla birlikte topun karşı sahaya taşınarak atak yapılması amaçlanmaktadır. SCRUM, 1990’ların ortalarında Jeff Sutjerland ve Ken Schawaber tarafından geliştirilen bir proje yönetim yaklaşımıdır. SCRUM, kompleks yazılım süreçlerinin yönetilmesi için kullanılır ve bütünü parçalayan, tekrara dayalı bir yöntem izler. Düzenli geri bildirim ve planlamalarla hedefe ulaşmayı amaçlar. Müşterinin ihtiyacına göre şekillendiği için müşterinin geri bildirimini almak ve iletişim çok önemlidir. Şeffaflık, denetleme ve uyarlama olmak üzere 3 temel prensipten oluşur.
* **Şeffaflık:** Projenin ilerleyişi, sorunlar ve gelişmeler herkes tarafından görülebilir olmalıdır.
* **Denetleme:** Projenin ilerleyişi düzenli olarak kontrol edilir.
* **Uyarlama:** Proje, değişen gereksinimler sonucu yapılabilecek değişikliklere ve yeniliklere uyum sağlayabilir olmalıdır.

**Kavramlar:**

1. **Product Backlog:** Proje için gerekli olan gereksinimler listesidir. “Projenin sonunda neyi üretmeyi amaçlıyoruz?” sorusuna cevap aranır. Product owner müşteriden gereksinimleri alır, öncelik sırasına göre sıralar ve product backlog’a ekleme veya çıkartma yapar ve bu sayede müşterinin gereksinimleri projenin her aşamasında projeye kolayca işlenir.
2. **Sprint(koşma):** Proje sprint denilen daha küçük parçalara ayrılır. SCRUM içerisindeki tüm aktiviteler sprint içerisinde gerçekleştirilir. 1-2 haftalık süreçlerdir.
3. **Sprint Backlog:** Mevcut sprint için product backlog’dan elde edilmiş iş ve görevleri kapsar. Sprint backlog’daki işlerin amacı, sprint sonunda son ürünün bir parçası olan bir işlevselliği ya da çalışabilir bir parçayı elde etmektir. Sprint backlog sadece takım tarafından değiştirilmektedir.
4. **Scrum Daily Meeting:** Her gün SCRUM takımıyla maksimum 30 dakika ve ayakta olacak şekilde düzenlenen toplantıdır. Projenin genel durumu hakkında bilgi alınır.” Dün neler yapıldı? Bugün neler yapılacak? Seni engelleyen ne idi?” gibi sorular sorulur.

**Roller:**

* **Ürün Sahibi (Product Owner):** Projenin iş değeri açısından geri dönüşünden sorumludur. Geliştirme takımı ve müşteri arasındaki iletişimi sağlar. Projenin özelliklerini tanımlar**.**
* **SCRUM Yöneticisi (SCRUM Master):** SCRUM kurallarını, teorilerini ve pratiklerini iyi bilen kişidir ve takımın bu kuralları uygulamasından sorumlu olan kişidir. Takımı rahatsız eden, verimli çalışmayı engelleyen durumları ortadan kaldırır.
* **SCRUM Takımı:** Bir sprinte alınan, devamlı iletişim halinde olan ve tek bir hedefe ulaşmak için çalışan kişileri barındıran ekiptir. 5-9 kişiden oluşur.
* **Chicken Roller:** SCRUM’ın işleyişinde aktif olarak yer almayan kişilerdir. Müşteriler, satıcılar bunlara örnek olarak gösterilebilir.

**Toplantılar:**

* **Sprint Planlama (Sprint Planning):** Product Backlog ile belirtilen gereksinimler, bu toplantı ile geliştirme takımı tarafından küçük görevlere ayrılır. Bu toplantıya product owner, geliştirme takımı ve scrum master katılır. Risk değerlendirilmesi ve risk kontrolleri yapılır. Geliştirme açarları ve altyapısı onaylanır. Dağıtım, geliştirme ve pazarlama maliyetleri hesaplanır.
* **Sprint Gözden Geçirme (Sprint Review):** Her sprintin sonunda yapılır. Yapılan sprint gözden geçirilir, ortaya çıkan ürün değerlendirilir. Buradaki amaç yazılımın ürün sahibinin istekleri doğrultusunda uygun olarak gerçekleştiğinden emin olmaktır.
* **Günlük Scrum Toplantısı (Daily Scrum):** Her gün aynı yerde ve aynı saatte, belirlenen yerde, ayak üstü 15 dakikalık yapılan toplantıdır. Tüm takım bu toplantıya katılır. Takımın ilerleyişini ve karşılaştıkları engelleri görmek için önemli bir etkinliktir. Takımın tüm üyeleri “Dün ne yaptım? Bugün ne yapacağım? Önümde olan engeller ve karşılaştığım sorunlar neler?” sorularına cevaplar verir.

**Scrum’ın Günümüzde Popüler Olma Sebepleri:** Scrum’ın daha fazla şeffaflık, proje görünürlüğü, günlük toplantılar sayesinde tüm takımın kimin ne yaptığını bilmesi, product owner sayesinde müşteri ve takım arasındaki iletişimin sağlanması, bu sayede karışıklıkların olmayıp projenin hızlı bir şekilde tamamlanması, basit ve düşük maliyetli olması Scrum’ın günümüzde bu kadar popüler olmasının en büyük sebeplerindendir.

**Ad Soyad:** Muhammet Emin AKYOL

**Öğrenci no:** 200601004

<https://m-e-akyol.medium.com/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-modelleri-5a1a9c462bae>

<https://github.com/ImPrepective/Yazilim-Yasam-Dongusu-Modelleri>