Hızla büyüyen bir endüstri alanı olan yazılımda ihtiyaçlar gitgide artmaktadır. Bu ihtiyaçlara cevap verebilmek için yazılım alanında çalışan insanlar ürünlerinin yaşam döngüsünü temel aşamalarla ilerletirler. Bu döngü ürünün üretiminden kullanımına ve sonrasında bakımına kadar ki süreci kapsar. Yazılım geliştirme yaşam döngüsü (“software development life cycle”,”SDLC”) olarak adlandırılan bu süreç 1970’lerde ortaya çıkmış ve yazılımın mühendislikte yer almasında önemli bir role sahip olmuştur. Kodlama, planlama ve analiz aşamalarını birbirine bağlamış ve belirli standartlar belirlenerek organize edilen yazılım geliştirme işine önemli fayda sağlamıştır. Winn Royce geliştirme sürecini bir yaşam döngüsü olarak ifade etmiş ve ‘Şelale’(Waterfall) adında bir model önermiştir. Bu model tek yol olarak kabul edilmiş ve standart bir model olarak dayatılmıştır. İlk model olmasından dolayı beraberinde getirdiği hataları ile fazlasıyla eleştiri almıştır. Gereksinimlerin ve ihtiyaçların sürekli olarak genişleyip değiştiği için yeni modeller geliştirilmeye ve farklı projelerde farklı modellerin daha iyi çalıştığı sonucuna varılmıştır. Bu modeller beş temel aşamadan(adımdan) oluşuyor ve bu aşamalar üstüne planlar gerçekleşiyor. Aşamalar; planlama, analiz, tasarım, gerçekleştirme, bakımdır.

Planlama; ilk aşamamız olan planlamada bizden istenilen unsurlar analiz edilir, döngüyü nasıl kontrol edeceğiz sorusu sorulur, proje ihtiyaçları belirlenir, fizibilite çalışmaları yapılır ve diğer adımların başlangıç aşaması bu adımda atılır. Projeyi oluşturan ve proje hakkında tartıştığımız adımda burada atılır. Analiz: Bir çok rolün görev aldığı bu aşamada gereksinimleri ve işlevleri belirler varsa eğer mevcut sistemler incelenir. Bu aşama UML diyagramları çizmeye başladığımız ilk aşamadır. Projenin ne kadar süreceği ve risk analizi de bu aşamada belirlenir. Tasarım: Projenin ilk ve temel hali burada oluşturulur. Kodlamadan uzak kalan bu aşamada projenin sürdürülebilirliği tartışılır. Verilen bir problemin nasıl sonuca varacağı bu aşamada görülür. Bu adım ikiye ayrılı Mantıksal tasarım ve Fiziksel tasarım. Mantıksal tasarımda elde bulunan değil önerilen sistemin yapısı belirlenir. Fiziksel tasarımda ise yazılımın bileşenleri ve modülleri tasarlanır. Tasarım aşamasında tüm kararların verilip sonraki aşama olan gerçekleştirme aşamasına soru bırakmamak hedeflenir. Gerçekleştirme: Tasarım aşamasının aksine kodlama bu adımda öne çıkar. Bu adımda tasarım aşamasında çıkan kararlar ile proje gerçekleştirilmeye başlanır. Malzemeleri belirli bir yemeği yapmak gibi tanımlayabiliriz bu aşamayı. Planlama ve analiz aşamasında belirlenmiş dil, ortam ve teknolojiler bu adımda kullanılır. Bu aşamayı üçe ayırabiliriz; kodlama, test etme, kurulum. Kodlama aşamasında kalite standardınızı okunabilirliği iyi ve bakımı kolay olan kod belirler. KISS(Keep It Simple) prensibi adı altında bilinen bu standartlara uymak ürünün kodunu daha anlaşılabilir ve gerekli durumlarda daha rahat değiştirilebilir bir hale getirmenizi sağlar. Test etme aşaması kodlamadan sonra gelen önemli bir aşamadır. Bu aşamayı erken test yaklaşımı ile analiz aşamasından itibaren bu bakış açısıyla ilerlememiz bizim hata oranımızı maliyetimizi azaltırken beraberinde para, zaman ve prestij gibi önemli değerleri beraberinde getirir. Çok farklı kategoride test türleri bulabiliriz. En çok kullanılan dört test türüne kısaca değinirsek; Unit Tests(Birim Test) Alan Kay ile geliştirilen, başka bir bloğu çağıran yada kullanılan kod bloğunun kendi görevini doğru yapıp yapmadığını test eder, Integration Test, hatalı parçaları bulmamızı sağlar, Functional Tests(Fonksiyonel Testler) belirli veri grubunun sonucunu bir sonuç kümesiyle karşılaştırıp test eder, Acceptance Tests (Kabul Testleri) ürünün müşterinin isteklerini yerine getirip getirmediğini kontrol eden testtir. Kurulum kullanılır hale getirilen yazılımı artık yaşayan yazılım haline getirdiğimiz ve beraberinde yeni problemlerin ve bu problemlere karşın yeni fikir ve ihtiyaçların ortaya çıktığı adımdır. Beraberinde getirdiği bu problem ve fikirler bakım aşamasında değerlendirilmeye geçer. Bakım: Bu aşamada ürün sunulup güncellemelere ve bakıma geçtiğimiz aşamadır. Bakım yapılırken yeni tanımlar, tasarımlar ortaya çıkar ve bu aşama sayesinde yazılım tüm yaşam boyu sürer. Bu aşamalar yazılım geliştirme yaşam döngüsünün temel aşamalarıdır ve modellerin en iyi sonucu vermesi için gereklidir. En iyi sonuç için kullanılan yazılım geliştirme sürecindeki modeller temelde üçe ayrılır; düzenleyici süreç modelleri, birleşik süreç, çevik yazılım süreci. Düzenleyici süreç modellerini incelersek eğer karşımıza birçok model çıkar bunlar; Kodla ve Düzelt(Code and Fix), Şelale Modeli (Waterfall), Gelişigüzel Model,Barok Modeli, V Modeli(V-shaped Model), Helezonik Model (Spiral Model), Evrimsel Geliştirme Modeli (Evolutionary Development), Artırımlı Geliştirme Modeli (Incremental Development), Araştırma Tabanlı Model (Resource Based Model), Formal Sistem Geliştirme (Formal System Development), Bileşen Tabanlı Geliştirme (Component Based Development), Prototipleme (Prototyping), Yeniden Kullanıma Yönelik Geliştirme (Re-use based Development).

Kodla ve Düzelt: Genelde resmi olmayan proje fikirleriyle başlayan ve ürünü hazır hale getirene kadar kodlama yaparak ilerletilen modeldir. Bir planlamaya ihtiyaç duyulmayan bu modelde çok küçük projeler veya kısa ömürlü ürün prototipinde kullanılır. Modelde dokümantasyon olmadığında bakım safhası oldukça zorlu geçer. En kolay ancak en pahalı olan bu yol maalesef küçük firmalar veya tecrübesiz yazılımcılarca tercih ediliyor. Uzman görüşü gerektirmeyen bu modelde kodları düzeltmenin maliyeti fazla olur ve müşterinin ihtiyaçlarını karşılamayabilir. Kodlar esnek değildir ve bundan dolayı da değiştirilmesi zordur. Sonuç olarak da bu model genellikle öğrenci ve bireysel geliştiriciler için uygundur ancak takım çalışmalarında yetersiz kalır. Şelale(Waterfall): Günümüzde kullanımı gitgide azalan bir modeldir. Ancak geçmişe bakıldığında en popüler yazılım geliştirme modelidir. Bu model en eski, tanınmış ve temel modeldir. Bu modelde aşamalar en az birer kez tekrarlanır. Bu modelde analiz aşamasında mümkün mertebede gerekli detayların tasarım aşamasına yansıyabilmesi için müşteri ve proje gereksinimleri iyi belirlenmelidir. Tasarım aşamasında da yazılım sürecinin tüm gereksinimleri karşılanacak şekilde olmalıdır. Dolayısıyla bu modelde analiz ve tasarım aşaması en çok üzerinde durduğumuz aşamadır. Kullanımı ve anlaması kolay olan bu model proje safhalarını iş bölüm ve planlamasını en başta net bir şekilde görmemizi sağlar bu sayede proje yönetimi de kolaylaşır. Ancak karmaşık ve nesne yönelimli projelerde bu model yeterli değildir. Yapılacak değişiklikler büyük maliyet çıkartır ve müşteri memnuniyeti sağlaması zorlanır. Gelişi Güzel Model: Aslında bir model olarak adlandırılması doğru değildir. Bir modeli veya yöntemi yoktur. Müşteri ve geliştiriciye bağımlıdır. Takip edilebilirlik ve bakım süreci açısından zordur. Tek kişilik üretim ortamına sahiptir. Basit bir programlamaya sahiptir. Barok Modeli: Yazılım geliştirme yaşam döngüsünün başlıca adımlarının geliştirildiği modeldir. Belgelemeyi ayrı bir süreç olarak ele alınan bu model gerçekleştirme aşamasına daha çok önem verir. Test aşamaları bittikten sonra dokümantasyon işlemleri başlatılır. Bundan dolayı günümüzde pek tercih edilmeyen bir modeldir. V Modeli: Model, ismini kodlama aşamalarından sonra yukarı eğim ile devam ettiğinden dolayı aldığı V şeklinden almıştır. Sol tarafı üretim sağ taraf ise sınama bölümü olarak ele alınır. Ele alınan bölümler tüm teslim edilebilir ürünlerde uygulanılabilir. Proje takip açısından kolaylık sağlayan bir modeldir ve ayrıca kullanımı da kolaydır. Ancak modelde risk çözümleme aktiviteleri eksiktir ve yazılım da diğer sistemler ile zamanla evrimleşir. Modelde doğan sıkıntılardan biri de son ürüne ulaşma düz bir çizgi ile ifade edilemiyor oluşudur. Waterfall’ın gelişmiş haline benzetilen bu modelin üç temel çıktısı vardır. Kullanıcı modeli: Geliştirme süreci ile kullanıcı arasındaki ilişkiler planlanır. Mimari model: Sistem ve alt sistemlerin tasarımları ile tüm sistemlerin sınanmasına ilişkin planlar yapılır. Gerçekleştirim Modeli: Yazılım elemanlarının kodlanması ve sınanması süreci planlanır ve uygulanır. Bu model daha çok iş tanımları belirgin projeler için uygundur.BT (Bilgi Teknolojileri)projeleri bu model için uygundur. Prototipleme: İsminden anlaşılacağı üzere bu model prototipler üretilip üzerine düşünülüp bir şeyler katarak geliştirilmesi hedeflenen bir modeldir. Bu model için gereksinimler bir an önce toplanıp geliştiriciler ve kullanıcılar ile proje hakkında geniş çaplı kararlar verilir. Örneğin; yazılımdan elde edilecek çıktılar, bu çıktılar için gerekli girdiler sağlamaya, korumaya yönelik kararlar. Modelde hızlıca tasarımlar ile örnekler üretilip müşterinin değerlendirilmesine sunulur ve değerlendirmeler sonucu alınan kararlar ile yeni örnekler oluşturulup süreç tekrarlanır. Bu sayede müşteri ve geliştirici arasında periyodik halde bir bilgi alışverişi gerçekleşir ve ürünü sürekli taze tutar. Buda bize üründe görülebilecek karmaşa ve yanlış anlaşılmaların önüne geçebiliyoruz. Ancak bu modelde belgeleri olmayan prototipler işi dezavantaja sürüklüyor. Düzeltme aşamasında yapılan eksiklikler performans açısından düşüklüğe yol açar. Hedeflerin net olmaması güvenlik açığı problemlerine yol açar. Müşterinin prototiplerden son ürün performansı beklemesi de bazen olumsuzluklara yol açabiliyor. Helezonik (Spiral) Model: Bu model 4 aşamadan oluşur; planlama, risk analizi, üretim ve kullanıcı değerlendirmesi. Şelale modelinde yok sayılan riskleri göz önünde bulunduran bu modelde her bir risk ayrı ayrı ele alınır çağdaş modellere yakındır. Planlama: Ürün için planların amacın belirlendiği ve bir önceki adımda üretilmiş ürünler ile bütünleştirme yapılır. Risk Analizi: Risklerin araştırılıp belirlendiği safhadır. Üretim: Ara ürünün üretildiği safhadır. Kullanıcı Değerlendirmesi: Kullanıcıdan ara ürün ile ilgili sınama ve değerlendirmeler alınır. Bu model sayesinde müşteri sistemi erken görebilir ve pek çok yazılım modelini içinde bulundurur. Risk potansiyeline olan yaklaşımı sayesinde zorluklar engellenir. Odak noktası hataları erken gidermektir. En riskli kısımları geliştirmeyi küçük parçalara bölerek önden gerçekleştirilir. Modelin kompleks bir yapısı vardır ve spiral sonsuza gidebilir. Küçük projeler için pahalı bir modeldir. Ara adımlar yüzünden fazla dokümantasyon gerektiren bir modeldir. Bu modelde de prototip yaklaşım mevcuttur. Evrimsel Geliştirme Süreç Modeli(Evolutionary Development): İlk tam ölçekli olan model olarak bilinir. Büyük firmalar için uygun bir modeldir. Modelin her aşamasında proje tam işlevseldir. Modelin başarısı ilk evrimin başarısına balı olur. Modelin eksik yanı değişiklik denetimi yönünden sıkıntıları olmasıdır. Değişiklik yöntemi, kalitesi ve sürümü açısından biraz problemli bir modeldir. Hatalara bu modelde daha az rastlanır. Bu modelde iki çeşit evrimsel geliştirme modeli vardır; keşifçi geliştirme(exploratory development), atılacak prototipleme(throw-away prototyping). Keşifçi geliştirmede müşteri ile çalışıp son sistem hedeflenir anlaşılan gereksinimlerle işe başlanır. Atılacak prototiplemede sistemin gereksinimlerini anlamaya ve anlaşılmamış gereksinimlere başlanır. Modelde bakım zorluğu diğer modellere göre sıkıntılıdır. Ürün uzun süre yapılandırılmadığından yazılım yapısı zarar görür. Artırımlı Geliştirme(Incremental Development): Müşterinin üründe değişikliğe gitmek istediği durumlarda sıkça kullanılan model yazılımı takvime bağlı kalarak geliştirmeye dayanır. Divide and Conquer (Böl ve Yönet) prensibini benimser. Ürünün giderek artan işlevleri içerecek hale getirmesi için model kullanılır. Modelde ürünümüz parçalara bölünür ve müşteri önceliğine dayalı bir sistemle parçalar sıralanır. Bu parçalarla ara ürünler oluşturulur ve müşteri bu ara ürünlerden faydalanır. Ara ürünler her seferinde bir şeyler katarak çıkartılır. Uzun zaman alabilecek projeler için uygundur. Bir öncekini tekrar eden ara ürünler olmayacağından bir diğer ürün başlayana kadar bir değişiklik yapılamaz. Gereksinimler müşteri ile belirlenir ve önem sırasıyla teslim edilir, öncelikli ürün gereksinimlerini karşılayan çekirdek bir sistemi geliştirir. Bu sayede projenin tamamının başarısız olma riski azalır. Ürün daha fazla test edime imkanı bulur. Artımların belirlenmesi için tüm ürünün tamamlanması gereksinimi modelin yarattığı dezavantajlardandır. Bu model için tecrübeli personel gerekir. Her teslim ile birlikte müşteride bir değer görünmesi sistem işlevselliğini erken safhalarda ortaya çıkartır. Araştırma Tabanlı Model (Resource Based Model): Yap-At’ a ilk örnek olarak da bilinen bir modeldir. Modelde elde edilen neticeler belirgin değildir. Bu model çoğunlukla öğrenciler tarafından tercih edilip genel olarak proje ödevi, yarışmalar için hazırlanmış projeler, TUBİTAK,KOSGEB projeleri gibi ürünler için kullanılır. Proje sonucu belli olmadığından dolayı maliyet hesaplaması yapmakta pek mümkün değildir ve sonuç odaklı projelerde kullanılan bir model olduğundan dolayı proje bitiminde bir işlevi kalmaz. Örneğin “Sayıların karesini alan program” adlı bir proje bu model için örnek verilebilir. Formal Sistem Geliştirme (Formal System Development): Yazılım ve tasarım gerçekleştirmesinin matematiksel tekniği kullanılır. Modelin temelinde karmaşık sistemleri geliştirmek ve program geliştirmek yatar. Üründeki veya yazılımdaki belirsizlikleri, uyumsuzluk ve eksiklikleri açığa çıkartır. Hatasız yazılım geliştirme imkanı sunar. Bakıldığında karmaşık değildir. Modelde sürekli tümleştirme vardır. Modelin felsefesinde pahalı hata ayıklamaların önüne geçmek ve kodları başta yazılırken doğru yazıp test aşamasında doğruluğunu sağlamak yatar. Ancak model çok zaman alan ve pahalı bir yöntem içerir. Teknik bilgi açısından eksiklikleri olan personellerle iletişimde güçlük çekmemize yol açar. Bileşen Tabanlı Geliştirme (Component Based Development): Önceden geliştirilmiş bileşenlerle yeni ürün oluşturmaya denir. Bu modellemede soyutlamalar ara yüzler aracılığıyla yapılır. Bu modelin temelinde yeniden kullanıma açıklığıdır. Yeniden Kullanıma Yönelik Geliştirme (Re-use based Development): Günümüzde daha çok kullanılan ve kullanım sayısı artan bir modeldir. Ürün önceden yazılmış veya başka taraftan temin edinilmiş yazılımlarla temel alır. Bu modeli kullanmak için tecrübeli personellere ihtiyaç vardır ve ciddi bir birikime ihtiyaç duyulur. Model anlaşılır ve basit bir mantık çerçevesindedir. Maliyeti yüksek bir modellemedir ve modelin başarı garantisi yoktur. Modelde kaynak kontrolü vardır ve maliyet denetimi yapılabilir. Kısa sürede yazılımı geliştirilir. Sınıflar tekrar kullanılabilir. Birleşik Süreç(Unified Process): Nesneye yönelik geliştirmede kullanılan yöntemlerle edinilmiş deneyimler sonucu oluşturulan yazılım geliştirme modelidir. Model yinelemeli, arttırılmalı ve evrimsel olduğu gibi aynı zamanda da risk güdümlüdür. Yinelemeli(Iterative): İstekler bir bütün değildir hedefler bölünür bir kısmı alınır ve sınanmış ürün olarak kayda geçer. Ürün oluşumu ardından sonraki yinelemeye(iterasyon) geçilinir ve yeni değerlendirmeler sonucu istekler ele alınıp geliştirilir. Her yineleme sonucunda müşterinin istediği ürüne yaklaşılır ve her iterasyon bir ürün gibi ele alınır. Yinelemeler tüm aşamalardan geçirilir ve test edilmiş, çalışır bir ürün ortaya konulur. Arttırılmalı ve Evrimsel (Incremental and Evolutionary): Yinelemelerin ardından yeni değerlendirmeler ele alınır ve bu sayede ürünlerin özellikleri artar ve ilerleme kaydeder. Bu işlemle istenilen ürüne bir adım daha yaklaşmış oluruz. Risk Güdümlü(Risk-Driven): İlk yinelemede elde edilen riskli kısımlar ele alınır. Böylece proje ilerlemeden büyük problemler görülür ve çözüme ulaştırılabilir. Modelleme de geliştirme aşamaları şu şekilde ilerler: Başlangıç, Ayrıntılandırma, Tamamlama, Yayım. Başlangıç: Planlama da alınan kararlar, devam etme yada bitirme kararları alınır. Ayrıntılandırma: Gerçeğe daha yakın çözümleme, yinelemelerle yüksek riskli kısımların incelenmesi. Tamamlama: Düşük risk ve önceliğe sahip kısımlar yinelemeli olarak gerçekleştirilir. Yayım: Test aşamaları, ürünü pazarlama aşamaları. Bu modellemede değişen isteklere erken bir şekilde geri cevaplayabilmek projeye avantaj sağlıyor. Her yinelemede deneyimler biriktirmeyi sağlar ve büyük sistemlerin çözülmesinde tecrübe ve deneyim kazandırarak sistem çözümünde kolaylık sağlanır. Riskler erken giderilir ve müşterinin istediği ürüne erken ulaşarak personel ekibinde moral yükselmesini sağlar. Çünkü sürekli elde edilen ürünler ile moral ve şevk artar. Ancak risk yönetimi düşük ve dokümantasyon ağırlığından maliyet fazla çıkabilir ayrıca model diğer modellere kıyasla karmaşık bir yapıya sahiptir.

Günümüzde en çok kullanılan modelleme Çevik Yazılım Geliştirme Süreci (Agile Programming) adı altında dallanan yinelemeli geliştirme üzerine kurulmuş modellemedir. Yenilikçi bir yazılım süreci gerçekleştirme hedeflenir. Eski, yavaş ve bürokratik sistemlere tepki olarak gelişen bu modelleme “İlerlemenin iyi göstergesi , çalışan yazılımdır” prensibiyle ilerler. Bu sayede devamlı bir şekilde müşteriye ürün sunulur. Birleşik sürecin sonuçlarında olduğu gibi ürünün hızlı ve müşteri memnuniyetiyle ilerlemesi takımın moral ve şevkini arttırır. Takımlar kendi kendine organize olur, kendilerini değerlendirirler ve nasıl daha iyi sonuçlar elde edebilecekleri üstüne konuşurlar bu şekilde devam ederler. Model doğal insan eğilimiyle ilerlediğinden fazla eğitim gerektirmez değişime açıktır ve esneklik seviyesi yüksektir. Dezavantajlarına değinecek olursak bu model kurumsal ortamlar için çok uygun değildir adapte edilmesi zordur. Sürekli değişim halinde olan bu ihtiyaç için aşırı çalışma gerektirir. Ürün başarısı ve proje başarısı ortaktır bu yüzdende kariyer riski doğurur. Takımlar üstünde baskı oluşturur. Kaliteli yazılımlar hızlı bir biçimde geliştiren model en iyi mühendislik uygulamalarını ve yazılım geliştirme ile müşteri ihtiyaçlarını ve firma amaçlarını yan yana getiren business yaklaşımını destekler. Prensipleri arasında; Basitlik, özgün içerik, değişime adapte olma, teknik başarı, uygun tasarım iletişimin en iyi yolunun yüz yüze olduğunu benimsemek, business ve ekip bir çalışır, gereksinimlere sonradan eklenen değişimler dikkate alınır, esas kriter çalışan ürün olması yer alır. Değerleri arasında; sürecin ve kullanılan araçların arasında etkileşimi sağlamak, kapsamlı dokümantasyon üzerinde yazılımın geliştirilmesi, müşteriyle uzlaşılan taahhütlerin dikkate alınması ve değişiklikleri bir plan çerçevesinde yönetmek yer alır. Birkaç çeşit Çevik Yazılım Metotları vardır bunlar; Uç Değer Programlama (Extreme Programing-XP),SCRUM, Çevik Tümleşik Süreç (Agile Unified Process – AUP) ve Özellik Güdümlü Geliştirme (Feature-Driven Development -FDD) en çok kullanılan metotlardır. Uç Programlama (Extreme Programing): Tüm gereksinimlerin, senaryoların işlere bölünüp küçük gruplarla paylaştırıldığı metottur. Aynı iş üzerinde 2 haftadan fazla çalışılmaz. Müşterinin gereksinim ve ihtiyaçlarının değişkenlik gösterdiği durumlarda oldukça kullanışlıdır. Genel olarak küçük ve orta büyüklükteki projelerde kullanılır. Basitlik ve verim sağlanması adına XP de 12 pratik değerler ele alınır bunlar; planlama oyunu, ayakta toplantı, ekipte müşteri, kısa aralıklarla yeni sürümler, geriye bakış, metafor, sürekli entegrasyon, kod standartları, ortak sorumluluk, kalıcı tempo, test, sade tasarımlar, yeniden yapılandırılabilme ve eşli programlama. Dört temel değer ele alınır bunlar; iletişim, basitlik, geri bildirim ve cesaret. İletişim: Üründen sonuç alma sürecindeki aksaklıkları engellemek için önemli bir değerdir. Ekip içi ve müşteriyle olan iletişim sürekli yüz yüze olmalıdır ve bu iletişimlerin hızlı olmasına önem verilir. Basitlik: zorunlu ihtiyaçları hedef alıp daha pratik çözümler üretmek ve problemi daha basit yollarla çözmek hedeflenir. Önemli huşuları atlayabilmek basitliği zorluğu kılar. Geri bildirim: Müşterinin istediği ürünü tam anlamıyla elde etmek için kaçınılmaz bir değerdir. Yazılımcılar, test ekibi ve gerekli ise müşteri ile olan sürümler sunulur. Bu sayede olası hatalar engellenir ve proje istenilen noktaya gelir. Cesaret: En zor olan değerdir başarısızlıktan korkmak ancak sonucu bozar. O korkunun üstüne gitmek ise sonucu güzelleştirir. 12 Pratik değerlere biraz bahsedecek olursak, Planlama Oyunu: müşteri ile beraber her yinelemede işlerin uzunluğu ayırılacak vakit yazılım ekibiyle masaya yatırılır ve herkesin sözü kartlar ile dinlenilebilir hale gelir. Ekipte Müşteri: Projenin gerçekleşmesi için müşteriye ekipte ihtiyaç duyulur bu sayede yazılımcının ihtiyaç duyduğu bilgiler çabucak erişilebilinir. Önce Test: Kod yazmadan test programları yazılır ve olabilecek ve mevcut sorunlar önceden çözüme ulaştırılmaya çalışılır. Basit Tasarım: Gereksinimlerin karşılandığı ve müşteriyi memnun edecek en basit tarımlar gerçekleştirilmeye çalışılır. Kısa sürede anlaşılır, yönetilinebilir ve değiştirilir bir yazılım geliştirilir. Çiftli Programlama: Her personel kendi yeteneklerini ve bilgisini paylaşır ve projeyi gerçekleştirmede yardımcı olurlar bu sayede proje daha hızlı gerçekleşir. Biri diğerinden daha hızlı çözüme ulaşacağından programın yavaşlaması önlenir. İşe yeni başlayan yazılımcılar için çok avantajlıdır. Sürekli Entegrasyon: Yapılan değişiklikler ve yeni bileşenler sisteme otomatik olarak entegre edilerek zamandan kara geçilir ve oluşabilecek hatalar önlenir. Kısa Aralıklı Sürümler: Proje ayrı zaman dilimlerine bölünür(2-4hafta) her dilimin teslim tarihi bulunur ve tarih aşılmadan iş teslim aşamasına getirilir. Bu sayede yaşayan bir uygulamayla müşteri ürünü takip edebilir. Yeniden Yapılandırma: Yazılımcıların ve tasarımcıların sürekli işlerini gözden geçirmesini sağlar; daha kolay ve basit bir biçim, yeni isteklere nasıl cevap verilir ve geliştirilir. Burada amaç müşteri memnuniyetidir. Ortak Kod Sahiplenme: Geliştirilen ürün bütün ekibe mal edilir ve bu durum olumsuz sonuçların ve tartışmaların önüne geçer. Bu sayede tüm ekip her koda erişebilir ve ürünü daha iyiye götürür. Metafor: Her sistem başka bir sisteme benzetilerek yazılım geliştirmesini arttırmak hedeflenir. Kodlama Standardı: Yazılım ekibinde karmaşıklığı engelleme amaçlanır ve kolay anlaşılınabilirliği arttırır. Haftada 40 Saat: Haftada 40 saatlik bir çalışma süresi ayrılır. Fazla mesainin önüne geçmek amaçlanır çünkü fazla mesaide verim artışı söz konusu değildir hatta yapılan hataların artma olasılığı daha fazladır. SCRUM: Adını Rugby sporundan bir hücum taktiğinden almıştır. Jeff Sutjerland ve Ken Schawaber tarafından 90 ların ortasında geliştirilmiştir. Proje yönetim yaklaşımıdır. Üç temel kavramdan oluşur Roller, Toplantılar,Araçlar. Roller: Ürün sahibi, scrum master ve scrum takımından oluşur. Toplantılar: 15 dakika boyunca ayak üstü yapılmış toplantılara “Daily Scrum Meeting” denir. Bu toplantıda önceki gün karşılaşılan sorunlardan bahsedilir ve özet şeklinde konuşmalar yapılır. Bu toplantılara sprint adı da verilir ve her sprintin ardından sprint gözden geçirme yapılır. Bu toplantılar Daily scrum meetinglerden daha uzun sürer. Araçlar: Ürün gereksinim dokümanı başı çeker. Gereken işler listelenir. Product owner bu listeyi yönetir. Listede kullanıcı hikayelerinden oluşur.

Kullanılan Kaynaklar:

<https://www.codex.com.tr/yazilim-gelistirme-modelleri>

<http://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>

<https://medium.com/@HayriRizaCimen/yazılım-yaşam-döngüsü-ve-süreç-modelleri-70fdfb2f8f77>

<https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>

<https://medium.com/@denizkilinc/yazılım-yaşam-döngüsü-temel-aşamaları-software-development-life-cycle-core-processes-197a4b503696>

<https://caglartelef.com/yazilim-yasam-dongusu/>

<https://medium.com/@omerharuncetin/yazılım-yaşam-döngü-modelleri-543c7879a742>

<https://caglartelef.com/cevik-yazilim-gelistirme/>

<http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yazılım-Geliştirme-Modelleri-Yazılım-Yaşam-DöngüsüSDLCYBS.pdf>

Ulaş Can KARAŞAH - 190601064