Yazılım Yaşam Döngü Modelleri

Yazılım yaşam döngüsü; bir yazılım ürünü geliştirilirken izlenecek adımlar ve ortaya çıkan süreçlerden meydana gelir. Her modelin odaklandığı hedefe göre adım ve süreçler değişiklik gösterebilir. Başlı başına aşamalar; planlama, analiz, tasarım, uygulama-test, teslim sonrası bakım ve emeklilik olarak sıralanabilir:
Planlama gereksinimlerin belirlenmesiyle başlar. Müşterinin talepleri net ve doğru şekilde belirlenir. Daha sonraki aşamalar dikkate alınarak gerekli ekipman, personel ve miktarları belirlenir ve projenin planı oluşturulur.

Analiz aşamasında ise gereksinimler tekrar gözden geçirilir ve müşterinin talepleriyle birlikte dokümante edilir.

Tasarım aşamasında önceki aşamalardan elde edilen gereksinimler karşılanacak şekilde, planlama ve analiz aşamalarından faydalanılarak yazılımın projesi oluşturulur. Yazılımda bulunacak özelliklerin nasıl olması gerektiği ve nasıl gerçekleştirilecekleri belirlenir. Kodlamadan uzak, tasarım üzerine gidilir ve bu tasarım 2 yönden ele alınır: mimari tasarım ve ayrıntılı tasarım. Mimari tasarımda genel bir planlama yapılır ve modüller belirlenir. Ayrıntılı tasarımda ise veri tabanı tasarlanır, kullanılacak olan dil-diller, algoritmalar ve bunlar gibi detaylar kararlaştırılır.

Uygulama-test bu aşamada kod yazma, test etme ve kurulum ihtiyaçları giderilir. Kodlama işleminde Tasarım aşamasında kararlaştırılmış dil ve algoritmalar kullanılarak uygulama tamamlanır, daha sonra testler sayesinde gözden kaçmış olabilecek eksiklikler ve yanlışlıklar düzeltildikten sonra ürün müşteriye sunulmaya hazırdır. Teslim edildikten sonra bakım aşaması başlar ve program emekliliğe ayrılana kadar sürmeye devam eder. Bakımın var olmasının nedeni müşteriye sunulan üründe meydana gelebilecek hataların ve eksikliklerin düzeltilebilmesi ve müşteri memnuniyetini korumaktır.

Emeklilik süreci ise yazılımın kullanımı sona ermesinden itibaren başlar ve tekrar kullanılmadığı durumda devam eder.

Düzenleyici Süreç Modelleri

Kodla ve Düzelt Modeli: Bu modelde ürün geliştirme aşamasında planlama olmadığı için ilerledikçe ortaya çıkan eksiklikleri giderilmesiyle ürün ortaya konulmaktadır. En basit ürün geliştirme süreci olarak kabul edilmektedir. Ayrıca yazılım odaklı teknoloji çözümlerinin prototipini oluşturmak için varsayılan yöntemdir. Kodlama ve Sabitleme olmak üzere iki ana adımdan oluşan döngüsel bir süreçtir. Bu modelde kapsamlı bir ürün planlaması somut strateji veya iyi tanımlanmış tasarım düşüncesi yoktur. Bu model bazen zaman kazandırıcı olarak kabul edilir ve düşük bütçeli ürünler için uygun bir seçenektir. Kodlama becerilerini geliştirmek için birçok giriş seviyesi mühendisin bulunduğu yazılım gruplarında çok yaygınca kullanılmaktadır.

Kodla ve Düzelt modelinin avantajlarından en önemlisi herhangi bir planlamaya ihtiyaç duyulmamasıdır. Ayrıca kısa ömürlü projelerde uzman görüşüne ihtiyaç duyulmadan kullanılabilmektedir. Modelin dezavantajlarıysa, kontrolsüzdür, kaynak planlaması yoktur, ne zaman biteceği belli değildir, hataların ve eksikliklerin denetlenmesi ve giderilmesi zordur.

Gelişigüzel Model: Gelişigüzel model, yazılıma yeni başlayanların kullandığı adımlara verilen model adıdır. Miladi olarak 60lı yıllarda ortaya çıkmıştır. Gelişigüzel Model aslında bir model değildir çünkü herhangi bir yöntemi yoktur. Geliştiren kişiye göre değiştiği için bir standart yoktur. İzlenebilirliği ve bakımı oldukça zordur.

Barok Modeli: Eski bir modeldir, yaşam döngüsünün temel adımları hedefe odaklıdır ve aşamalar birbirinden bağımsız olduğu için nasıl geri dönüş yapılacağı belirsizdir. Bu modelde gerçekleştirme aşaması ön plana konulmaktadır. Dokümantasyon, yazılan programın test aşamaları bitirildikten sonra yapılır. Günümüzde ise bu durum bütün sürece yayılmıştır. Bu gibi çeşitli sebeplerden dolayı günümüzde çok tercih edilmemektedir.

Çağlayan Modeli: Çağlayan modeli, projenin başarısını sağlamak için Yazılım Mühendisliğinde yaygın olarak kullanılan ilk SDLC Modelidir. Çağlayan Modelinin farklı isimleri de vardır. Bu isimlerden bir tanesi de 'Şelale Modeli'dir. Yazılım geliştirme sürecini, doğrusal bir sıralı akışta gösterir. Geliştirme sürecindeki bulunulan aşama tamamlanmadan sıradaki aşamaya geçilemeyeceği anlamına gelir, fazlar çakışmaz. Şelale modelinde bir aşamanın sonucu sıradaki aşama için girdi görevi görür. Şelale modelinde sıralı aşamalar şunlardır:

- Gereksinim Toplama ve Analiz: Geliştirilecek sistemin tüm olası gereksinimleri bu aşamada yakalanır ve bir gereksinim belirtimi belgesinde belgelenir.
- Sistem Tasarımı: Bu aşamada ilk aşamadaki gereksinim özellikleri incelenir ve sistem tasarımı hazırlanır. Bu sistem tasarımı, donanım ve sistem gereksinimlerinin belirlenmesine ve genel sistem mimarisinin tanımlanmasına yardımcı olur.
- Uygulama: Sistem tasarımından gelen girdilerle, sistem ilk önce bir sonraki aşamada entegre edilen birimler adı verilen küçük programlarda geliştirilir. Her birim, Birim Testi olarak adlandırılan işlevselliği için geliştirilir ve test edilir.
- Entegrasyon ve Test Etme: Uygulama aşamasında geliştirilen tüm birimler, her bir birimin test edilmesinden sonra bir sisteme entegre edilir. Entegrasyon sonrası tüm sistem herhangi bir hata ve arıza için test edilir.
- Sistemin konuşlandırılması: İşlevsel ve işlevsel olmayan testler yapıldıktan sonra; ürün müşteri ortamında konuşlandırılır veya piyasaya sürülür.
- Bakım: İstemci ortamında ortaya çıkan bazı sorunlar oluşabilmektedir. Bu sorunları gidermek için yamalar yayınlanır. Ayrıca ürünü geliştirmek için bazı daha iyi sürümler yayınlanabilir.

Yinelemeli Model: Her bir yinelemede bir özellik gerçekleştirilir. Uygulama tamamlanana kadar, analiz, tasarım, geliştirme ve test aşamaları sürdürülür.

V Modeli: Çağlayan modeline Doğrulama ve Onaylama mekanizmasının eklenmiş halidir. Amaç her aşamanın karşısında, doğrulama ve onaylama mekanizmalarını koyarak sistemin düzgün ve kaliteli çalıştığından emin olarak ilerlenen geliştirme modelidir. Her adımın devam edebilmesi için bir önceki işlem tamamlanmış olmalıdır. V Modeli, analiz sonucunda sistemi tepeden aşağıya doğru parçalanan ve tasarlanan bir modeldir. Daha sonra ise gerçekleştirme sırasında sistemi toparlandığı ve bu toparlama sırasının her aşamanın karşısına onun ile ilgili doğrulama ve onaylama mekanizması konulmuştur.

V Modelinin avantajlları; doğrulama ve onaylama mekanizmaları sayesinde eksiklikler ve hatalar olabildiğince engellenir. Proje yönetim ekibi tarafından yönetimi ve kullanımı kolaydır. Dezavantajları ise: risk çözümleme ile ilgili aktiviteleri içermeme, yazılımın diğer sistemler gibi değişimi, fazlar arasında tekrarlamaların bulunmaması ayrıca aynı zamanda gerçekleştirilmesi gereken olaylar kolay yapılamaz.

Spiral Model: Çağlayan ve Yinelemeli modellerin harmanlanmasıyla ortaya çıkmıştır. Her bir aşama tasarım hedefi ile başlar ve müşterinin onaylaması ile sonlanır. Bu modelin üzerinde durduğu konular risk analizi ve prototip üretmektir. Büyük projelerde planlama ve gereksinim analizini düzgün bir şekilde yapabilmeniz çok zordur çünkü karşınıza bilmediğiniz birçok risk çıkabilir. İşte spiral model bu riskleri baz alarak geliştirmiştir. Amacı riski aşama aşama azaltılarak projenin başarılı bir şekilde tamamlanması sağlanmaktır.

- Hedeflerin Belirlenmesi ve Alternatif Çözüm Yollarının Bulunması: Bu aşamada müşteriden elde edilen gereksinimler sonucunda hedefler belirlenir ve bu hedeflere göre çözüm önerileri üretilir.
- Çözüm Yolu Seçilir ve Risk Ortadan Kaldırılır: Bu aşamada çözüm yolları değerlendirilir ve en iyi çözüm yolu belirlenip bu konuda prototip geliştirilir.
- Ürünün Bir Sonraki Aşaması için Geliştirme Yapılır: Bu aşamada bu bulunan çözüm yoluna göre geliştirme ve testler yapılır ve ürün bir sonraki aşamaya hazırlanır.
- Bir Sonraki Faz Planlanır: Bu aşamada bir sonraki fazın planlaması yapılır. Yukarıda bahsedilen aşamalar gelecek için tekrardan planlanır.

Spiral modelin avantajlarından bahsedecek olursak sistemi erkenden görülebilir, geliştirmeyi küçük parçalara bölerek riskli kısımlar kontrol altında gerçekleştirilebilir. Pek çok yazılım modelini içerir, hataları erken gidermeye odaklanır. Dezavantajlarıysa küçük ve az riskli projeler için maliyeti yüksek bir yöntemdir, karmaşıktır, ara adımların çok olması nedeniyle dokümentasyona çok gerek duyulur.

Evrimsel Geliştirme Modeli: Tam ölçekli ilk modeldir. Projenin başarıya ulaşması ilk evrimin başarısına bağlıdır. İstenen ürün tam olarak belli değildir müşterinin talep ettiği kriterler doğrultusunda bir ürün prototip oluşturulur. Diğer modeller ile kıyaslandığında ilerleyişi daha yavaştır.
Avantajları: Kullanıcıların kendi gereksinimlerini daha iyi algılamalarını sağlar, sürekli değerlendirme erken aşamalardaki geliştirme risklerini azaltır. Dezavantajlarıysa sürecin görünürlüğü azdır, sistemler sıklıkla iyi yapılandırılamaz bu yüzden bakımı zordur, yazılım gereksinimini yenilemek gerekebilir.

Artırımlı Geliştirme Modeli: gereksinimlerin- yinelemeli modelde olduğu gibi- küçük yapılara bölündüğü, her yapının kendi içerisinde aşamalara sahiptir. Bu modelde aşamalar kolayca yönetilebilen modüllere bölünür. Her modül, gereksinim analizi, tasarım, uygulama ve test aşamalarıyla kontrol edilir. Yazılım geliştirme süreci sistem tamamlanana kadar devam eder.

Avantajlarından bahsedecek olursak gereksinimler müşterilerle belirlenir, erken arttırımlar sayesinde emin adımlarla ilerlenir, projenin başarısız olma riski azaltılır, önemli gereksinimlerin erkenden dahil edilmesi sayesinde sık kullanılması sağlanır. Dezavantajları ise artımları sağlamak için tüm sistemin tamamlanması gerekmektedir, gereksinimleri doğru boyutta artım sağlamak bazen zorluk çıkarabilir. Artımları kendi içlerinde tekrarlatmaz. En önemlisi de deneyimli personel gerektirir.

Prototipleme Modeli: bir sistemin geliştirilmesindeki gereksinimleri analiz etmek, ihtiyaçları tespit etmek amaçlı bir prototipin üretildiği bir modeldir. Prototipleme modelinde, bir yazılım kabataslak durumda müşteriyle paylaşılır. Böylelikle müşterinin isteklerinin ve

ihtiyaçlarının neler olduğu rahatlıkla belirlenebilir. Bir yazılımın prototipi başlıca olması gerekenden oldukça basittir ve eklenmesi gereken pek çok işlev eklenmemiştir. Çünkü prototiplemenin asıl amacı işlevsel bir sistem ortaya koymaktır.
Avantajları sistem gereksinimleri görülebilmektedir, karmaşa ve yanlış anlaşılmaların engellenmesini sağlar bu sayede risk kontrolü elde edilir. Dezavantajları dokümante edilmemiş hızlı ve kirli prototipler oluşturulması, düzeltme düzgün yapılmazsa performans kayıplarına sebep olur, müşteri prototipleri sanki son ürünmüş gibi gözlemler.

Çevik Yazılım Süreci

Uç Programlama: İlk olarak Kent Beck tarafında ortaya atılmıştır. Dört değer üzerine kuruludur. Bunlar basitlik, iletişim, geri dönüm ve cesarettir. Bu yöntem grup içi iletişime önem verir. Bu yüzden projenin başarılı olması için iletişim eksikliğini ortadan kaldırması amaçlanır. Yazılım ekibi ile müşteri arasında sıkı bir iletişim vardır. Basitlik kapsamında karmaşık çözümler kullanılmaz. Gereksinimleri karşılayacak en basit çözüm yolu aranır. İletişim kapsamındaki geri dönüşler sayesinde hatalar erkenden fark edilir. Yeniliklere açık olmak, başarısızlıktan korkmamak, hataları oluşturan nedenlerin üzerine gitmek cesaret kapsamına girer. XP 12 farklı pratiği ön görür. 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 olarak sıralanabilir.

- Planlama Oyunu: Projenin iş yönetimi tarafının, sürümlerin kapsamına, önceliklerine ve teslimat tarihlerine karar vermeleri gerekmektedir. Bu kararları sağlıklı verebilmeleri için teknik yönden gerekli desteğin verilmiş olması gerekmektedir. Teknik insanlar, kestirimler yapmalı, sürece karar vermeli ve planlarda detaylı tarihleri belirlemelidir.
- Ekipte Müşteri: XP anlayışında gerçek müşteri, geliştirme takımından gelecek sorulara hızlı cevap verebilmek, planları kolay yönlendirebilmek ve uyuşmazlıklara çabuk çözümler bulabilmek için takım ile birlikte oturmalıdır.
- Önce Test: Müşteri fonksiyonel testleri yazmakla sorumludur. Teslim alacağı ürünün isteklerini karşıladığından nasıl emin olacağı sorusunun cevabını kendisi vermeli ve bu tespiti yapabilmek için gerekli testlerin içeriğini belirlemelidir. Programcılar her bir işlevsel özellik için birim testlerini yazmak ile sorumludur. Bu, her bir metot için bir test yazmak anlamına gelmemektedir.
- Basit Tasarım: Doğru bir tasarımın özelliklerini sıralamaya çalışırsak; tüm testlerden başarıyla geçmiş, tekrarlanmış mantıklar içermeyen, tüm niyetleri programcıya açıklanmış ve mümkün en az sayıda sınıf veya metot kullanılmış tasarım diyebiliriz. Birim testleri ile tasarım XP içerisinde birbirlerine çok bağlı pratiklerdir. Tasarımın hiçbir şekilde karmaşık olmaması ve testlerden geçebilecek kadar olmalıdır.
- Çiftli Programlama: XP mantığında tüm kaynak kodu üretimi iki kişi ve tek bir makine, tek bir klavye kullanılarak gerçekleştirilir. Bu çiftin içinde iki ayrı rol vardır. Bir kişi elinde klavye ile metodu en doğru şekilde nasıl gerçekleyebilirim diye düşünmekte, diğer eleman ise daha stratejik düşünmektedir. Bu pratik tamamen dinamik olarak işlemektedir ve sabah veya öğleden sonra roller değişebilmektedir.
- Sürekli Entegrasyon: XP yaklaşımında geliştirme günü içerisinde kaynak kod sisteme en az bir defa entegre edilmelidir. Bunu sağlayacak makineler ve simülatörler üzerine çalışmalar yapılmalıdır. Bu altyapıya gerekli kaynağın ayrıldığı ve altyapının

oluşturulduğu göz önünde tutulmaktadır. Çift programlama pratiği ile entegrasyon safhası süresi yaklaşık yarıya indirgenebilmektedir.

- Kısa Aralıklı Sürümler: Her bir sürüm mümkün olduğunca kısa ve en değerli iş gerekleri içerir durumda planlanmalıdır. Yılda iki sefer veya tek sefer planlanan sürümlerden daha iyidir bir veya iki aylık sürümler. Tek seferde büyük bir iş gücünü yüklenmek yerine kısa aralıklarla daha küçük ve işlevsel sürümler az maliyetli olur.
- Yeniden Yapılandırma: Bu pratiğin yazılım geliştirme sürecinin içerisindeki rolü ve tam yeri konusunda çeşitli çalışmalar vardır. Ayrıca pratiklerin en çok tartışılan konusu olarak öne çıkmaktadır. XP'nin bu pratik ile yazılım tasarım aşamasını hafiflettiği öne sürülmektedir. Yazılımın hafifletilmesinden doğan çeşitli soru işaretlerine dikkat çekilmektedir. Yani, bu pratik başlı başına bir tartışma konusu olarak ele alınmaktadır.
- Ortak Kod Sahiplenme: XP yaklaşımında, takım içerisindeki herkes tüm sistem üzerinde sorumluluğu paylaşmaktadır. Tabii ki, herkes aynı düzeyde sistemin tüm parçalarına hakim olacak anlamına gelmemektedir bu yaklaşım. Ama çift programlama pratiğinin ve gözden geçirmelerin etkisi ile bireylerin sistemin tamamı üzerindeki bilgisi en yüksek düzeyde tutulmaya çalışılmaktadır.
- Metafor: Metaforun pratik ile uyum sağladığına dair gerçek kodlardan ve testlerden hızlı ve anında geri bildirimlere ulaşılmalıdır. Metaforun pratikte hangi anlama geldiği anlamalı ve buna paralel yenilemeler yapabilmeniz gerekmektedir.
- Kodlama Standardı: Takım içerisindeki herhangi bir kişinin sistem yazılımlarının herhangi bir yerinde değişiklik yapabilme hakları olduğu bir durumda herkesin kodlamayı rahatlıkla algılayabilmesi için belirli bir standart belirlenmelidir.
- Haftada 40 saat: XP yaklaşımında, sürekli şekilde fazla mesai yapma bulunmamaktadır. Planların ve öngörülerin haftada 40 çalışma saati üzerinden yapılmasını önermektedir. Dağınık şekilde 60 saat çalışma yerine tamamen konsantre olunmuş verimli 40 saat tercih edilmesi gerektiği vurgulanmaktadır. ('Haftada 40 Saat' pratiği 2005 güncellemesinde pratikler arasından çıkartılmıştır.)

Scrum

Yazılım geliştirme ve yazılım Mühendisliği'nde bir uygulama geliştirme çerçevesidir. Atik yazılım geliştirme yöntemi olarak çevik yönetim ve proje yönetiminde karmaşık bir ortamda ürünleri geliştirmek, sunmak ve sürdürmek için bir çerçevedir. Birçok modern yazılım projesinin oldukça karmaşık olduğu ve en baştan tümünü planlamanın zor olacağı şeklindeki bir varsayımdan hareket eder. Bu karmaşıklığı şeffaflık, gözlem, uyumlama adlı üç ilke ile azaltmaya çalışır. Şeffaflık: Projedeki ilerlemeler ve sorunlar günlük olarak tutulur ve herkes tarafından izlenebilir olması sağlanır. Gözlem: Ürünün parçaları ya da fonksiyonları düzenli aralıklarla teslim edilir ve değerlendirilir.
Uyumlanma: Ürün için gereksinimler en baştan bir defalığına belirlenmez, bilakis her teslimat tekrar değerlendirilir ve duruma göre uyarlamalar yapılır.
Müşteri/kullanıcı tarafından istenilen ve tanımlanan işlevler, iki ya da dört haftalık "Sprint" adı verilen dönemler içerisinde geliştirilir ve yeniden gözden geçirilir. Her Sprint sonunda yazılımın fonksiyonel bir parçası bitmiş ve müşteriye teslim edilebilir bir durumda olur.
Scrum Çevik yazılım geliştirme prensiplerini hayata geçiren bir yöntemdir.

Scrum Rolleri

- Ürün Sahibi: Ürün Sahibi stratejik ürün geliştirmeden sorumludur. Şirketin ekonomik faydasına uygun ürün tasarımı ana amaç belirler. Yalnızca o teslimat, işlevsellik ve maliyet gibi kararlardan sorumludur.
- Geliştirme Takımı: Yazılım ekibinin görevi ürün sahibinin taleplerine ve sıralamasına uygun ürünün işlevselliğini sağlamak ve belirlenen kalite standartlarına uymak koşuluyla ürünü teslim etmektir. Ne kadar ve hangi işlevlerin sprinte dahil olacağına kendileri karar verirler.
- Scrum Yöneticisi (Master): Scrum Ustası Scrum'un başarılı olmasını sağlamaktan sorumludur. Bunu başarmak için Yazılım Takımı ile birlikte calışır ama takıma tabii olmaz. Scrum kurallarını bildirir, uyumu kontrol eder ve toplantı moderatörlüğünü yapıp süreçteki düzensizlikler ile ilgilenir. İş aracı olarak engel-birikimini takımın önündeki engelleri kaldırmak için kullanır ve bu anlamda sorumluluk taşır.
- Kullanıcı: Ürünün nasıl bir perspektif ile kullanılacağı konusunda fikir verir ve gerçek hedef kitlesidir. Kullanıcı Sprint başlangıcı ve sonucunda ürünü test etme amaçlı yer alır ve geri bildirim sağlar.

Toplantılar

Sprint Planlama Toplantısı 1

Bu toplantıda ürün sahibi kendi yazılım takımına ürün içeriğinde kararlaştırılan kullanıcı hikayelerini öncelik sırasına göre belirtir ve gereksinimler takım tarafından netleştirilip yazılı olarak kaydedilir. Kullanıcı da işlevsellik konusunda önemli bilgiler verebilir. Haftada 1 defa 60 dakikalık toplantılardır.

Sprint Planlama Toplantısı 2

Genellikle küçük gruplar oluşturulup yapı, test, açık gibi konulara açıklık verilir. Burada beklenen sonuç görevlerin bir günlük süreyi aşmayacak şekilde bitirilmesini kapsar. Görevler belirlenen kullanıcı hikâyelerinden hareketle duvara ya da beyaz tahtaya asılır bu sayede hangisinin işlemde ya da sırada olduğu bilinir. Haftada 1 defa 60 dakikalık toplantıdır.

Günlük Scrum Toplantısı

Her iş günü başlamadan evvel 15 dakikalık bilgi paylaşımı için günlük Scrum toplantısı yapılır. Bu görüşmede herhangi bir problem değerlendirilmez, yalnızca 3 tema işlenir: dün ne yaptım, bugün ne yapacağım. Eğer belirtilen görev bir günde bitmesi mümkün değil ise, görev parçalanıp takıma dağıtılır. Ayrıca 15 dakika içinde bazı sorular cevap bulamadığı durumlarda scrum ustası not alıp bir sonraki toplantıya taşır ya da kendisi çözüm üretir.

Sprint Değerlendirmesi

Değerlendirme sprintin sonunda takım tarafından yapılır ve başlangıçta belirlenen hedeflerin kapsamında olup olmadığı Ürün sahibi tarafından değerlendirilir.

Eğer teslim edilen işlevde eksiklik var ise o kullanıcı hikâyesi tekrardan, ürün sahibi tarafından ürün içeriğine gönderilir ve öncelik sırası verilir.

Scrum'ın Yapı Taşları

Ürün İçeriği taleplerin oluşturulması ve yönetilmesi için merkezi belgedir. Toparlanan kullanıcı hikâyeleri ürün sahibi tarafından önceliklerine göre düzenlenir.

Ürün içeriği, geliştirilmekte olan ürünün önceliklere göre sıralanmış işlevleri kapsar. Değişim taleplerinin alındığı tek yerdir ve ekleme, çıkarma, öncelikler gibi işlemler ürün sahibi tarafından yapılır. Ürün içeriği hiçbir zaman eksiksiz değildir ve böyle bir iddiası da olmaz, tanımlanmış, iyi anlaşılmış gereksinimleri içerir, öncelikler ise ekonomik fayda, risk gibi faktörlerle değerlendirilip uygulanır.

Sprint içeriği halledilmesi gereken görevleri gösterir. Bu amaç için dört sütunlu bir görev tahtası kullanılır:
1. sütunda sprintte bulunan İş Parçacıkları2. sütunda görevler3. sütunda çalışma ve 4. sütunda teslime hazır olan iş parçacıkları bulunur. Yazılım takımı elemanları günlük scrumda önceki gün hangi görev üzerinde çalıştığını ve bitip bitmediği hakkında bilgi verir. Bir günde bitmeyen görevler ise kırmızı bir nokta ile işaretlenir.
İş Bitim Grafikleri yapılmış ve geri kalan çalışmayı görselleştirmek için kullanılır.
Bir Sprint yanık grafiği, x-ekseninde günlük zamanı, y-ekseninde bitirilmemiş görevleri gösterir. Bu grafik, sprintin belirtilen zaman birimi içinde daha iyi tahmin edilmesini sağlar.
Seçili kullanıcı hikâyelerini izlemek içinde hikâye-iş bitim-grafiği kullanılır. Burada eksik kalan görevler değil eksik hikâyeler gösterilir.
Her hikâye eşit büyüklükte olmayacağından, büyüklük bilgisi noktalarla sağlanır. Kalan hikâyelerin toplamı y-ekseninde belirtilir, x-ekseni de Sprint süresini gösterir.
Grafik eğilimleri merdiven şeklindedir. Her azalma değeri bir hikayenin bittiğini gösterir.
Tüm projeyi göstermek için devir grafiği kullanılır. Bu durumda y-eksenine bütün ürün içeriğinde belirlenen bitmemiş kullanıcı hikâyeleri ve hikâye noktalarının toplanmış sekli gösterilir. Böylelikle proje bitimine kadar kaç tane teslimat yapılacağı anlaşılır.

Kaynaklar:

- https://tr.wikipedia.org/wiki/Scrum
- https://medium.com/teknopar-akademi/yazılım-geliştirme-yaşam-döngüsü-modelleri-852217d2a353
- https://medium.com/@baskinozge24/yazilim-yaşam-döngüsü-nedir-89b4417e0751
- https://ofcskn.com/tr/yazilimda-sdlc-modelleri-nelerdir
- https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/
- https://tr.linkedin.com/pulse/yazılımın-yaşam-döngüsü-beyza-nur-karakoç?trk=pulse-article
- https://hayririzacimen.medium.com/yazılım-yaşam-döngüsü-ve-süreç-modelleri-70fdfb2f8f77

Hesaplar:

- https://www.linkedin.com/in/barış-tunçkanat-669148254/
- https://github.com/Baristk74/Baristk
- https://medium.com/@baris.tunckanat