# YAZILIM YAŞAM DÖNGÜ MODELLERİ (Software Development Life Cycle)

# Yazılım Yaşam Döngüsü (SDCL) Nedir?

# Herhangi bir ürünün üretilmesinde belli plan aşamaların izlenmesi gerektiği gibi yazılım projelerinin de başarılı, zamanında, sürdürülebilir bir şekilde tamamlanabilmesi ve oluşan ihtiyaçları karşılayabilmesi için kullanılması gereken yöntem ve yönteme bağlı farklı modeller vardır. Bu yöntem yazılım yaşam döngüsü (SDCL) olarak adlandırılır. Yazılım yaşam döngüsü belli adımlardan oluşur ve bu adımların doğru takip edilmesiyle planlı ve düzenli bir yazılım ürünü ortaya çıkar. Kısacası bir yazılımın ilk aşamasından kullanım aşaması da dahil olmak üzere son aşamasına kadar geçirdiği tüm aşamalara yazılım yaşam döngüsü denir. Yazılımın işlevleri ve ihtiyaçları sürekli değiştiğinden bu aşamalar doğrusal olarak değil bir döngü olarak gerçekleştirilir.

# Yazılım Yaşam Döngüsü Adımları

# 

# Yazılım Yaşam Döngüsü (SDCL) Nedir? Başlığı altında da bahsedildiği gibi yazılım yaşam döngüleri belirli aşamalardan oluşur.

# 1. Planlama aşaması: Bu aşamada projenin başlangıç aşamasıdır projenin temel ihtiyaçları belirlenir ve fizibilite çalışmaları yapılır. Bu sebepten bu aşama proje yönetimi için oldukça önemli bir aşamadır , diğer proje adımlarına da temel oluşturur.

# 2. Analiz aşaması: Bu aşama sisteminin gereksinimlerinin belirlenip dokümante edildiği aşamadır. Gereksinimler müşterinin ihtiyaç ve istekleri doğrultusunda oluşturulur. Genellikle bu adım müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi gibi kişilerden oluşan grupların ortak çalışmalarıyla gerçekleştirilir. Gereksinimler oluşturulurken müşteriler oldukça iyi dinlenmeli istek ve ihtiyaçlar net bir şekilde belirlenmelidir.

# Yazılım Geliştirme ve Testi Yaşam Döngüleri3. Tasarım aşaması: Bu aşamada projenin nasıl yürütüleceği ve sistemin bütün bileşenlerinin yapısı planlanır. Yazılımın temel yapısının tasarlanır ve önemli konular (örneğin yazılım ekranında nelerin bulunacağı, bir ekrandan diğerine nasıl geçiş yapılacağı vb.) üzerinde kararların verildiği aşamadır. Tasarım aşaması kendi içerisinde iki aşamaya ayrılır: Mimari tasarım ve Detaylı tasarım. Mimari tasarım genel olarak yazılım modüllerinin genel yapıları, organizasyonları ve kendi aralarındaki etkileşimleriyle ilgilenir. Modüller ve akış şemaları kullanımları burada oldukça yaygındır. Mimari tasarım dokümanları oluşturulur. Detaylı tasarım aşamasında ise uygunluk durumuna göre bu dokümanlarda düzenlemeler yapılır. Yazılım içeren bileşenlerin ve ayrıntılarının tasarımları bu aşamada gerçekleştirilir.

# 4. Gerçekleştirim (Kodlama ve Test) aşaması: Tasarımı tamamlanmış olan sistem bu aşamada kodlanır ve test edilir. Şimdiye kadar planlanan ve tasarlanan yazılım projesi bu aşamada somutlaşır. Analiz aşamasından itibaren aslında erken test etme yaklaşımına sahip olmamız maliyet açısından büyük avantajlar sağlar. Buradaki maliyet sadece para olarak düşünülmemelidir. Zaman ve prestij açısından da önemli avantajlar sağlar.

# 5. Teslim ve Bakım aşaması: Gerekli test etme aşamalarından sonra yazılım teslim edilir. Fakat sürece bağlı olarak yazılımın güncelleme, hata giderme, yeni gereksinim ekleme gibi ihtiyaçları oluşur. Bu ihtiyaçlar bakım aşamasında giderilir. Bir bakıma bakım aşaması yazılım yaşamaya devam ettiği sürece sona ermez.

# YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ

# 1. Gelişi Güzel Model

# Herhangi bir yöntem içermez dolayısıyla takibi ve bakımı oldukça zordur. Belirli bir yöntemi ve modeli olmadığından aslında bir model olarak adlandırılmamalıdır. Tek kişilik üretim ve basit yazılım sistemlerinde kullanılır. Genellikle 1960’lı yıllarda basit projelerde kullanılmıştır.

# 2. Barok Modeli

# 1970’li yıllarda ortaya çıkmıştır. Yaşam döngüsünün temel adımlarının doğrusal bir şekilde takip edildiği bir modeldir. Geri dönüşlerin nasıl yapılacağı konusunda bir yöntem tanımlanmamıştır bu durum büyük bir dezavantajdır. Çünkü yazılım geliştirme nispeten uzun bir süreçtir ve bu süreç içerisine yeni gereksinimler, hatalara bağlı olarak adımlar arasında geri dönüşler gereklidir. Bu modelde yazılımın geliştirilmesi ve testinden sonra belgeleme ayrı bir adım olarak ele almıştır. Günümüzde ise belgeleme yapılan işin doğal bir ürünü olarak görülmektedir. Gerçekleştirim aşamasına daha fazla önem veren bu model günümüzde kullanılmamaktadır.

# 3. Çağlayan Yaşam Döngü Modeli

# Yazılım Yaşam Döngü Modelleri. Yazılım yaşam döngüsü, bir yazılımın… | by Berke Canbay | MediumÇağlayan modeli geçmişte en tanınmış ve yaygın kullanılan model olmuştur. Bu sebeptendir ki geleneksel model olarak da bilinir. İyi tanımlı ve üretimi az zaman gerektiren projelere uygundur. Bu model ile gerçekleştirilecek bir yazılım aşamaların en az birer kez tekrarlanmasıyla gerçekleştirilir. Çağlayan yaşam döngüsü modeli için en önemli hususlardan birisi gereksinimlerin net ve doğru bir biçimde belirlenmesidir. Gereksinimlerin analizi ve tasarım aşamasında çok detaylı bir biçimde çalışılması gerekir. Barok modelinden farklı olarak belgeleme ayrı bir aşama olarak alınmamıştır, belgeleme sürecin doğal bir parçası olarak görülmüştür ayrıca geri dönüşler daha iyi tanımlanmıştır. Bu modelde aşama aşama ilerlenir ve bir aşama tamamlanmadan sonraki aşamaya geçilemez. Her aşamada dokümantasyon ve test yapılması gerekir. Dokümantasyon ve test yapılmadan bir aşama bitmiş sayılmaz. Modelin avantajlarını inceleyecek olursak anlaşılabilirliği kolay adımlardan oluşur dolayısıyla müşteri ve son kullanıcılar tarafından da anlaşılabilir. Değişiklik süreci yönetilebilir adımlara bölünmüştür. İş paylaşımı konusunda kolaylıklar sağlar. Gereksinimleri iyi anlaşılabilen ve kalite gereksiniminin bütçe ve zaman kısıtlamasından önemli olduğu projelerde kullanılabilir. Dezavantajlarına bakacak olursak projede gereksinimler süreç içerisinde değişebilir fakat bu model gereksinim değiştirmeye uygun değildir. Kodlama ve test aşamasında yapılacak değişikliklerin maliyeti oldukça fazladır. Genellikle yazılıma başlarken gereksinimler net bir biçimde belirlenemez veya süreç içerisinde kullanıcının istekleri değişebilir. Kullanıcı sürece dahil olmadığından yazılımın tamamlanmasından sonra geri dönüşlerin yaşanması yazılım maliyetini oldukça arttırır. Yazılımın oluşması süreci bu modelde oldukça uzundur bu durum üretim ekiplerinde motivasyon kaybına neden olabilir. Bu duruma bağlı olarak yazılımın kullanıcıya ulaşmasa süresi de uzundur bu durum da dezavantajlar arasında gösterilir.

# 4. V Süreç Modeli

# İsminden de anlaşılacağı üzere kodlama aşamasından sonraki süreç adımları yukarıya doğru eğim oluşturarak bir V görünümü oluşturur. Modelin sol tarafı üretim sağ tarafı test kısmıdır. Proje yönetimi aşağıya doğru yapılır ve sonrasında yukarı doğru çıkılmaya başlanır. Her üretilen modülün testleri gerçekleştirilir ardından modüllerin alt seviye tasarım ürünlerinin entegrasyon testleri ve kabul testleri yapılır. Kabul testleri aşamasında müşteriye gidilir ve müşteri de uygulamayı test eder sonrasında bakım aşaması başlar. Genel sürece bakıldığında bu modelde aslında üç temel çıktı oluşur.

# Bunlardan ilki kullanıcı modelidir. Burada sürecin kullanıcı ile olan ilişkileri ve nasıl kabul edileceğine dair planlamalar yapılır. İkinci tasarımı ve test işlemlerini içeren mimari modeldir. Üçüncüsü modüllerin kodlanması ve sınama fonksiyonlarını içeren gerçekleştirim modelidir. Bu model daha çok belirsizliklerin minimum seviyede, iş tanımlarının net olduğu projeler için uygundur. Modelin avantajlarına ve dezavantajlarına bakacak olursak projenin yönetiminin kolay olması, kullanım kolaylığı avantajları arasında gösterilebilir. Fazlar arasında tekrarlamaların kullanılmaması, risk çözümleme gibi aktiviteleri içermemesi dezavantajları olarak gösterilebilir.

# 5. Helezonik (Spiral) Model

# Geliştirme Metotoları | —SİSTEM ANALİZİ—Uygulama ve TasarımBu modelde diğer modellerden farklı olarak süreci doğrusal değil spiral bir süreç olarak görür. Risk analizi olgusu oldukça ön plandadır. Yinelemeli artımsal bir yaklaşımı vardır. Genellikle iç kısımlar gereksinimlerle ilgili konuları dış kısımlar ise yazılım tasarımı ile ilgili konuları belirtir. Her döngü fazı ifade eder. Diğer modellerde olduğu gibi doğrudan klasik fazlara (gereksinim, analiz…) sahip değildir. Prototip yaklaşımına sahiptir. Planlama, risk yönetimi, üretim ve kullanıcı değerlendirmelerinden oluşan dört bölümden oluşur. Planlama aşamasında hedef belirleme, üretilecek ara ürün planlaması ve önceki adımda üretilmiş ara ürün ile bütünleştirme gerçekleştirilir. Risk yönetimi aşamasında risk analizleri yapılır, seçenekler değerlendirilir. Üretim aşamasında ara ürün üretilir ve son olarak kullanıcı değerlendirmesi kısmında kullanıcı tarafından testler gerçekleştirilir, değerlendirmeler yapılır. Helezonik modelin en önemli avantajlarından biri kullanıcının aslında sürecin içerisinde olmasıdır. Süreç boyunca ara ürün kullanıcı ara ürün değerlendirmesi ve test aşamalarında aktiftir dolayısıyla sistemi erken tanır. Bu durum ileride kullanıcının istemediği bir sonucun son aşamada fark edilip büyük zararlar oluşması riskini azaltır. Bu modeldeki yinelemeli yaklaşım hataları azaltır bu da avantajlar arasında gösterilebilir. Dezavantaj olarak ise ara adımların çokluğundan dokümantasyonun fazla olması ve programın başlangıçtan farklı bir yöne kayması sonucu bütçe aşım riskinin olması gösterilebilir.

# 6. Artımsal Geliştirme Süreç Modeli:

# Bu modelde gereksinimler mantıksal olarak küçük alt parçalara ayrılmıştır. Her bir parça küçük bir şelale modeli yapısında kendi içerisinde çevrim geçirerek yazılım gerçekleştirilir. Ürünün ortaya çıkması için çoklu şelale yapısı kullanılır. İlk çevrimden itibaren yazılım gerçekleştirilir bu sebeple ilk aşamadan itibaren çalışan bir yazılım vardır. Gerekli tüm özellikler yazılıma ekleninceye kadar süreç devam eder. Üretilen her yazılım sürümü birbirini kapsayacak ve giderek hedefe yaklaşan, işlev sayısı artan şekilde geliştirilir. Bir taraftan kullanım diğer taraftan üretim yapılır.

# Avantajlarını sıralayacak olursak; her aşamada test gerçekleştirildiğinden hataların ayıklanması daha kolaydır, risk yönetimi kolay olan bir modeldir çünkü risk parçaları tanımlanır ve yinelemeler sırasında kullanılır, her teslimatta müşteri bir değer görür ve bu da işlevselliğin ilk aşamalarda ortaya çıkmasını sağlar ayrıca erken teslimler sayesinde projenin tümden batması riski azalmış olur, bu modelde gereksinimleri değiştirmek daha az maliyetlidir. Dezavantajlardan bahsedecek olursak aşamalandırma ve planlamanın çok dikkatli ve doğru bir şekilde yapılması gerekir, maliyet olarak şelale modelinden yüksektir.

# 7. Kodla ve Düzelt Yaşam Döngü Modeli

# Yazılım geliştirmenin en kolay yolu olarak görülür fakat dokümantasyon olmadığından bakım aşaması oldukça zordur. Bu modelde diğer modellerden farklı olarak emeklilik emeklilik (retirement) safhası vardır. Doğrudan yazılım ürünü gerçekleştirilir ve hedefe ulaşıncaya kadar devamlı olarak geliştirilir. Küçük projelerde veya kısa ömürlü projelerde kullanılır. Uzun vadede oldukça maliyetli bir yöntemdir. Çünkü belirli bir dokümantasyonu, şartnamesi tasarım dokümanı olmadığından değişiklik yapmak oldukça zordur. Daha çok küçük veya tecrübesiz firmalardaki yazılım projelerinde kullanıldığı görülür.

# ÇEVİK (AGILE) YAZILIM GELİŞTİRME

# Agile at ALTRES Technologies - ALTRES Technologies IndiaYazılım geliştirme uzun ve sıkıntılı bir süreçtir. Yazılım projeleri genellikle yönetimsel eksikliklerden dolayı kısmı başarı ve memnuniyet ile tamamlanır. Buna bağlı olarak yazılım sektöründe zaman içerisinde bazı problemler ortaya çıkmıştır. Bu problemler arasında; değişik isteklerine çabuk cevap verememe, yazılımın zamanında ortaya çıkarılamaması, yazılım hatalarının geç fark edilmesi, zaman içerisinde gelen isteklere göre sistemin kendi yapısını geliştirememesi gibi problemler vardır. Çevik(agile) metotlar bu sorunların çözülmesine yönelik olarak 1990’lı yılların sonlarında geliştirilmiştir. Çevik yazılım piyasaya hızlı ürün çıkarabilme, değişen gereksinimlere çabuk yanıt verme, kısa sürede bir yazılım ürünü geliştirerek müşteriyle buluşturmayı hedefler. Ayrıca çevik yazılım geliştirmede verimlilik yüksek, esnek, hata oranı oldukça düşük, hızlı ve ucuz çözümler sağlamayı amaçlar. Burada proje küçük yinelemelere (iteration) ayrılır. Her yineleme bir proje gibi değerlendirilir ve her yinelemenin sonunda müşteriye bilgi verilir. Takımlar oluşturulur ve bu takımlar mini toplantılar gerçekleştirerek durum değerlendirmesi yaparlar. Nasıl ilerleyecekleri, nereleri düzeltecekleri, nasıl daha etkili olacaklarına bakarlar buradan çıkan sonuçlara göre hedefe ilerlerler. İlerlemenin birincil ölçüsü çalışan bir yazılımın olmasıdır. 2001 yılında Kent Beck ve arkadaşları tarafından çevik manifesto oluşturulmuştur. Bu manifestoda süreç ve araçlar yerine birey ve etkileşimlerin, kapsamlı dokümantasyonlar yerine çalışan yazılımın, sözleşme görüşmelerin yerine müşteri ilişkilerinin, plan izleme yerine esnekliğin ve değişikliğe açıklığın daha önemli olduğu belirtilmiştir. Çevik süreçler sürdürülebilir geliştirmeyi destekler. Çevik yazılım geliştirmenin avantajlarını inceleyecek olursak; kısa sürede müşteri memnuniyetinin sağlanması, üretkenliğin artması, yazılım ekibinin motivasyonunun artması, yazılım kalitesinin artması ve maliyetlerin düşmesi gibi durumlar avantajlar arasında sıralanabilir. Uygulanan birçok farklı çevik metodolojiler vardır. Extreme programming (XP), SCRUM, Agile Unified Process, Feature-Driven Development (FDD), LEAN Development, Dynamic System Development Methodology (DSDM), Microsoft Solution Framework (MSF) yaygın olarak bilinen çevik metodolojilerdendir. Biz bu yazımızda bu metodolojiler arasında en yaygın olarak kullanılan ve popülaritesi en yüksek olan SCRUM ‘dan ve Extreme programming (XP)’den bahsedeceğiz.

# Extreme Programming (XP) 1999’lu yıllarda Kent Beck tarafından ortaya çıkarılmıştır. Basit ve geri dönüşlere imkan sağlayan bir modeldir. Bu modelde iletişim oldukça önemlidir. Dört temel değere sahiptir. Bunlar; basitlik, iletişim, geri bildirim ve cesarettir. Basitlik karmaşık çözümler yerine basit ve anlaşılabilir çözümler kullanılması gerektiğini vurgular. Sistem için uygun olan en basit çözüm kullanılmalıdır. İletişim kısmında ekip üyeleri arasındaki, müşteri ve proje yöneticileri arasındaki iletişimin güçlü olası gereklidir. Çünkü ancak doğru iletişimle gereksinimler doğru bir şekilde saptanabilir. Sadece gereksinimler değil geliştirme aşamasında ekip üyeleri arasındaki iletişim motivasyon, eksikliklerin tespiti vb. konuların doğru anlaşılabilmesi ve karşı tarafa doğru aktarılabilmesi için oldukça önemlidir. Geribildirimler sayesinde ortaya çıkabilecek problemler daha erken tespit edilir, yazılım ekibi ve müşteri arasında oluşabilecek anlaşmazlıklar bu sayede tespit edilerek erken safhalarda uzlaşma sağlanır. Cesaret kısmı belki de işin en zor tarafıdır. Hata yapmaktan korkulmamalıdır, hataların üzerine gidilerek çözümler geliştirilmeye bakılmalıdır. Gerektiğinde bütün kod çöpe atılabilmelidir. Elimizi korkak alıştırmadan memnun olmadığımız bir kodu atıp yenisi yazma cesareti gösterilmelidir. Yazılım geliştirmede kolaylık ve esnekliği gerçekleştirebilmek için 12 farklı pratik ön görülmüştür. Bunlar“ Planlama oyunu, ekipte müşteri, önce test, basit tasarım, çiftli programlama, sürekli entegrasyon, kısa aralıklı sürümler, yeniden yapılandırma, ortak kod sahiplenme, metafor, kodlama standartı, haftada 40 saat” şeklindedir. Her birini kısaca açıklayacak olursak;

# 1-*Planlama oyunu*: Her bir yinelemede yazılım ekibi ve müşterinin de olduğu bir toplantıda işin ne kadar zamanda yapılacağı hakkında bir tahminde bulunarak planlama yapılmasıdır. 2- *Ekipte Müşteri*: Müşteri temsilcisi yazılımın gerçekleştirilmesi süresince yazılım geliştirme ekibiyle bulunmalıdır. Böylece ihtiyaç duyulan bilgileri çok hızlı bir şeklide cevap bulunabilir. 3-*Önce test:* Mevcut sorunların daha erken tesit edilerek daha güvenilir bir yazılım oluşturmayı hedefler. Bu sebepten kod yazılmadan önce test programı yazılmalıdır. 4-*Basit tasarım*: Müşterinin ihtiyaçlarını karşılayacak en basit tasarım gerçekleştirilir. Bu sayede anlaşılması basit dolayısıyla da değiştirilmesi ve yönetilmesi kolay bir yazılımın geliştirilmesi sağlanır. 5-*Çiftli programlama*: Aslında usta çırak ilişkisi gibi de görülebilecek bir yaklaşımdır. Yazılımcıların hızları farklıdır ve birbirlerine yardım ederler. Bu durum programın yavaşlamasını engeller. 6- *Sürekli Entegrasyon (Continuous Integration- CI)*: Yapılan yeni değişiklikler veya eklenen yeni bileşenlerin hemen sisteme entegre edilmesi ardından günlük derlemelerle test edilmesi kısmıdır. 7-*Kısa Aralıklı Sürümler*: Proje zaman dilimlerine bölünerek gerçekleştirilir. Ve bölünerek gerçekleşeceği zaman dilimi belirlenen görevler bu zaman dilimini aşmayacak şekilde tamamlanır. Her yinelemenin sonunda müşteri projenin ilerleyişini bu sayede takip etmiş olur. 8-*Yeniden Yapılandırma (Refactoring)*: Müşteri memnuniyeti için müşteriden gelen istekleri geliştirmek ve geliştirilen yazılımın daha basit bir şekilde yapılabilmesi için yeniden yapılandırma pratiği kullanılır. 9- *Ortak Kod Sahiplenme*: Yazılan kod tüm ekip üyelerinin ortak kodu olarak görülmelidir. 10- *Benzetim (Metafor)*: Tıpkı yap bozun parçalarının bütünü oluşturması gibi gerçekleştirilecek yazılım sistemler birbirine benzetilerek gerçekleştirilmeye çalışılır (ör: gerçekleştirilecek yazılım parçalara ayrılır). 11- *Kodlama Standartı*: Ekip üyeleri genel isimlendirme şekilleri, dizi kullanımı gibi birçok konuda önceden tanımlanmış kodlama standartlarına uygun olarak yazılım geliştirirler. 12- *Haftada 40 Saat*: Haftada 40 saatlik bir çalışma süresi ayrılır. Fazla mesai sadece haftada bir kez yapılabilir. Bu yaklaşıma 40 saatten fazla çalışmada verim düşer ve hata yapma riski artar.

# Avantajlarını sıralayacak olursak; hata oranı azdır, projenin geliştirilme süresi kısadır, değişikliklere izin verir, müşteri ile beraber bir ilerlendiğinden müşteri ile oluşabilecek problemler önlenmiş olur. Ayrıca geleneksel yazılım geliştirme modellerinden farklı hatanın ilerleyen safhalarda düzeltilmesi durumunda oluşabilecek maliyet miktarı geleneksel yöntemlere göre pekte artış göstermez.

# SCRUM

# SCRUM NEDİR?. Zaman içerisinde projelerin daha büyük… | by Seçil Çor | MediumScrum çevik yazılım geliştirme metodolojilerinden biridir. Adını Rugby sporundaki bir hücum taktiğinden alır. Jeff Sutjerland ve Ken Schawaber tarafından 1990’ların ortalarında geliştirilmiştir. Aslında sadece yazılım geliştirmeye yönelik değil genel olarak her yere uygulanabilecek proje yönetimi yaklaşımıdır. Daha çok kompleks yazılım süreçlerini yönetmek için kullanılır. Yani gereksinimlerin kolay belirlenemediği, karmaşık durumlar içeren projeler için en uygun metodolojidir denilebilir. SCRUM metodunda bütün küçük parçalara(sprint) bölerek ve tekrara dayalı yani yinelemeli bir yöntem izleyerek gerçekleştirir. Bu yinelemeler 30 günden fazla sürmeyecek şekilde günlük 15 dakikalık toplantılarla iş takibi yapılarak tamamlanmalıdır. Aralıklı olarak müşteriye demolar sunulur. Düzenli geri bildirimler (feedback) oldukça önemlidir. Bu geri bildirimlere göre planlamalar yapılarak süreç ilerletilir. Dolayısıyla ihtiyaca yönelik esnek bir yapısı vardır. Süreci fikirleri ürüne dönüştürme yeteneğine sahip insanlardan oluşan scrum takımları ilerletir.

# Bir scrum projesindeki rolleri inceleyerek olursak ilk olarak “Product Owner” yani yapılacak işle ilgili gerekli vizyona sahip olan ürün sahibi belirlenir. Sonrasında bu işi yapacak “takım (team)”oluşturulur. Ayrıca “Scrum master (scrum ustası)” olarak ifade edilen kişi de belirlenmelidir. Scrum master sürecin işletilmesi konusunda yol göstericidir ve çıkan problemlerin çözümünde yardımcı olacak kişidir. Proje sahibinin yapılacak işlerin öncelikli bir sıralamasının olduğu “Product Backlog” adı verilen bir liste oluşturur. Süreç içerisinde bu listenin sürekli güncel olması gerekmektedir. İşleri yapacak olan kişiler bu işlerin neler olduğunu, yapılmasının ne kadar zaman alacağını listeye bakıp söyleyebilmelidir. Oluşturulan iş listesi team, scrum master, ve product owner ile ilk toplantı olan sprint toplantısında parçalara bölünür. Bu toplantıda bir sonraki sprint plalama toplantısına kadar yapılacak işler belirlenir. Yukarıda da bahsettiğimiz gibi bu toplantılar arasındaki aralık bir aydan daha uzun olmamalıdır. Pratikte genellikle bir iki haftalık hedefler belirlenir. Ardından herkes sorumluluğu dahilindeki işleri geliştirmeye başlar. Bahsettiğimiz gibi günlük 15 dakikalık toplantılar yapılır. Bu toplantılarda “Dün ne yaptın? , Bugün ne yapacaksın? , İşini yapmana engel bir durum var mı?” gibi sorulara yanıt aranır ve sürecin verimli ilerlemesi sağlanır. Bu mini toplantılar takım ilerleyişi ve karşılaşılan engelleri görmek açısından oldukça önemlidir. İş bitene kadar sprintler devam eder.

# SCRUM NEDEN POPÜLER ?

# Scrum yaklaşımı en popüler Agile(çevik) yöntemidir. Günümüzde birçok büyük firma SCRUM kullanmaktadır. Popüler olması farklı sebeplere bağladır. Projenin açık ve net olması projenin başarısını arttırırken zamandan kazançta sağlar. Projenin parçalara ayrılarak ilerletilir ve bu bir hata durumunda hızlı tespit, kolay çözüm sağlar. Bütün ekip projenin tüm akışından haberdardır ve takım içerisindeki iletişim oldukça iyidir. Belki de en önemli sebeplerden biri olarak müşteri ile iletişimin oldukça iyi olması sayılabilir. Müşteri sürecin içine alınır, müşteri süreç içerisinde prototipler görür, süreç içerisindeki gelişimi takip eder. Bununla beraber müşterinin istediği değişiklikler kolaylıkla gerçekleştirilebilir. Bu durum müşteri ve geliştirici arasındaki güveni sağlar ve başarı oranını arttırır. Ayrıca ekip çalışması sayesinde ekip içi iletişimi geliştirerek sorunların çözümlenme süresi minimuma indirilir. Bu gibi etmenler projenin başarı oranını oldukça arttırmıştır. Başarının artması SCRUM yaklaşımının popülaritesi de giderek artmıştır.

# Kaynakça :

# Bakırçay Üniversitesi Yazılım Mühendisliği Temelleri ders 3. Ve 4. Hafta ders sunumları

# <https://denizkilinc.com/yazilim-yasam-dongusu-temel-asamalari-software-development-life-cycle-core-processes/>

# <http://ybsansiklopedi.com/wp-content/uploads/2015/08/Yaz%C4%B1l%C4%B1m-Geli%C5%9Ftirme-Modelleri-Yaz%C4%B1l%C4%B1m-Ya%C5%9Fam-D%C3%B6ng%C3%BCs%C3%BCSDLCYBS.pdf>

# <http://bilgisayarkavramlari.com/2014/05/12/sdlc/?highlight=sdlc>

# <https://koraypeker.com/2018/06/18/sdlc-v-model/>

# <https://www.beagile.tn/blog-post/12-keys-to-the-successful-implementation-of-agile-methodologies/>

# <https://www.javatpoint.com/software-engineering-sdlc-models>

# <https://en.wikipedia.org/wiki/Software_development_process>

# 