**YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ**

**Yazılım Yaşam Döngüsü Nedir?**

Yazılım yaşam döngüsü (Software Development Life Cycle SDLC), geliştirilen bir projenin en baştan başlanarak sonuna kadar geçirdiği aşamalara ve bu aşamalardan oluşan döngüye denir. Yazılım, yaşam süreci var olan bir üründür. Yazılım yaşam süreci tek yönlü değildir ve geriye dönüşlerin olduğu hata var ise sorunların çözüldüğü ya da eksik olan yeni özelliklerin elde edildiği bir döngü halindedir. Bu döngü yazılımın daha rahat, planlı olmasını sağlar. Yazılım yaşam döngüsünün planlı planlı olmasını sağlayan beş temel adımı şöyle sıralayabiliriz: Planlama (Planning), analiz (Analysis), tasarım (Design), gerçekleştirme (Implementation) ve sonuncusu ise bakımdır (Maintenance). Bu adımlar yani aşamalar gerçekleştikten sonra proje kesin olarak tamamlanmayabilir. Bu adımlar bir döngü şeklinde düşünülmeli ve gerçekleştirilmelidir. Proje uygulandıktan sonra hataların düzeltilmesi, eksiklerin giderilmesi için bu adımlar, aşamalar tekrarlanır. Bu birbirini takip eden aşamalara, adımlara yazılım yaşam döngüsü denir. Yazılım yaşam döngüsünün daha iyi anlaşılabilmesi için beş temel adım olan planlama (Planning), analiz (Analysis), tasarım (Design), gerçekleştirme (Implementation) ve bakımın (Maintenance) incelenmesi gerekir.

**Planlama (Planning):** Yazılım yaşam döngüsünün başlangıç aşamasıdır. Temel gereksinimlerin belirlendiği, projede uygulanılacak adımların planlandığı dönemdir.

**Analiz (Analysis):** Analiz aşamasında ki temel amaç gereksinimlerin detaylı bir şekilde incelenip müşterinin istediği şeyler doğrultusunda ilerlenip ilerlenmediğinin ortaya çıkartılmasıdır. Yapılan yazılım projesinin nasıl riskler ortaya çıkarabileceği ve projenin ne kadar süreceği hakkında bilgiler belirlenir. Her şey müşteri ve projeyi yapan kişiler tarafından açığa kavuşunca dokümantasyon yapılır. Kısaca analiz aşaması en önemli aşamalardan biridir. Çünkü projenin tüm detayının ortaya çıktığı evredir.

**Tasarım (Design):** Analiz aşamasında gereksinimler ayrıntılı bir şekilde belirlendikten sonra gelinen adımdır. Projenin nasıl devam edeceğine dair bir tasarlama işlemi gerçekleşir. Yapılan tasarlama işlemi yanı sıra tasarım dokümanı da oluşturulmalıdır. Tasarım dokümanında bulunması gereken bazı bilgiler vardır ve bunlar: Projenin amacı, projenin kapsamı, veri modelleri, UML diyagramları, tasarım dokümanı amacı, referans alınacak dokümantasyon bilgisi. Tasarım aşamasında kullanılan en önemli tekniklerden biri olan soyutlama (Abstraction) tekniği problemi daha basit bir hale getirir ve problemin önemli kısımlarına dikkat edilmesini sağlar.

**Gerçekleştirme (Implementation):** Gerçekleştirme aşamasında projenin test edildiği ve kodlandığı adımdır. Gerçekleştirme adımında istenilenin alınıp alınmadığına bakılır. Eğer istenilen alınmışsa ve hatalar giderildiyse proje yayına alınır.

**Bakım (Maintenance):** Proje sunulduktan sonra ürünle ilgili geri gelen bildirimler sonucu ürün üzerinde güncellemelerin, bakımın, yeniliklerin yapıldığı aşamadır.

**Yazılım Yaşam Döngüsü Modelleri**

Yazılım yaşam döngüsü modelleri sürecin düzeni ve nasıl uygulanacağını belirten modellerdir. Yazılım yaşam döngüsü birden fazla modelle ele alınır. Birden fazla modelin var olmasının sebebi yazılım projesinin kullanılabilirliğinin çok geniş olması gibi sebeplerdir. Birden fazla modelin var olması projelerin yönetimini büyük krizlerden ve karmaşıklıklardan kurtarmış, projenin işleyişinde kolaylıklar sağlamıştır. Her bir model çıktığı dönemin ihtiyacına göre ve donanımına göre çıkmıştır. Bu dönemlerde doğru model ve yöntemi seçmek zaman, ekonomi, uğraş açısından oldukça önemli rol almıştır. Yazılım yaşam döngüsü modelleri temelde iki kısma ayrılmıştır. Bunlar geleneksel yazılım süreç modelleri ve çevik yazılım süreç modelleridir. Geleneksel yazılım süreç modelleri sekiz tanedir ve bunlar: Gelişigüzel model, barok model, çağlayan yaşam döngü modeli (Şelale-Waterfall), V süreç modeli, helezonik(Spiral), artırımlı geliştirme süreç modeli, kodla ve düzelt yaşam döngü modeli, evrimsel geliştirme süreç modelidir. Geleneksel yazılım süreç modellerini daha iyi anlamak için modellerini tek tek incelemek gerekir.

**Gelişigüzel Model:** Belirli bir yöntem ve modele sahip değildir. Kişisel bir yöntemdir denilebilir. Gözetilmesi oldukça zor olan bir modeldir. Gelişigüzel geliştirme modeli için herhangi bir model veya takip edilebilecek bir yol yoktur. Genellikle basit programlar için kullanılır. 1960’lı yıllarda sıklıkla kullanılmıştır ve günümüzde kullanılmamaktadır. Bu model günümüzde güncel bir yaşam modeli olarak adlandırılmaz.

**Barok Model:** 1970’li yıllarda ortaya çıkmıştır. Belli bir gidişatı öngörmektedir ve bu şey yaşam döngüsünün temel modellerini doğrusal biçimde gerçekleştirmektir. Belirli bir döngüye sahip değildir ve belgeleme ayrı bir süreç olarak ele alınmaktadır. Barok modelin belgelemeyi ayrı bir süreçte ele alması en önemli özelliği niteliğindedir. Belgeleme süreci yazılım geliştirme aşamasından sonra yapılır. Belgeleme günümüzde ise her süreçte yer almaktadır ve tek bir aşamada yer alması o modelin zayıflıklarından birisidir. Model aynı gelişigüzel model gibi geriye dönüşler hakkında bilgi vermemektedir. Günümüzde kullanılmamaktadır ve güncel bir yaşam modeli olarak adlandırılmaz.

**Çağlayan Yaşam Döngü Modeli (Şelale-Waterfall):** 1970’li yıllarda ortaya çıkmıştır ve diğer güncel yaşam modelleri temelleri niteliğindedir. Gereksinimleri çok iyi tanımlanmış ve kısa sürede bitebilecek projeler için uygun bir modeldir. Eğer gereksinimler iyi tanımlanmamış ve uzun sürecek bir proje gerçekleştirilecekse model işe yaramaz. Belgeleme üretimin her aşamasında yer almaktadır. Bu model ile birlikte her adım mutlaka bir kere gerçekleşir yani her adım bir önceki adım olmadan gerçekleşmez. Bu özelliği ile yani geriye dönülebilirliği ile barok modelinden daha ön plandadır. Çağlayan yaşam döngü modeli (Şelale-Waterfall) bazı sorunlar ortaya çıkartabilir. Mesela hazırlanması ve sunulması uzun süreçli olan yazılım projeleri için çağlayan yaşam döngü modeli (Şelale-Watefall) uygun değildir. Gereksinim tanımlamaları kesin olmaz veya yanlış olursa bu model ciddi maddi ve zaman kaybına sebep olabilir. Tüm bu kayıpların yanında kişisel olarak motivasyon kaybı olur ve firmanın adı bu durumdan kötü etkilenebilir.

**V Süreç Modeli:** V modeli çağlayan yaşam döngü modeli yani şelale modelinin gelişmiş hali olarak ele alınabilir. Model v harfine benzetilir ve v harfinin sol tarafı üretim, sağ tarafı ise sınamadır. Temel mantığı her sol tarafta bulunan üretim kısmının karşısında bulunan sınama sayesinde hatalar daha kolay ortaya çıkartılabilir ve çoğu şeyden tasarruf yapılabilir. Model belirsizlikleri az olan ve gereksinim tanımlamaları açık, net olan projelerde tercih edilir, kullanılır. V süreç modeli döngüsü temelde üç tanedir. Alttan üste doğru gerçekleştirme modeli, mimari model ve kullanıcı modelidir. Gerçekleştirme modeli yapılan kodlama ve bu kodların denenmesinden oluşur. Mimari modelde projenin tasarlanması ver alır ve son olarak kullanıcı modelinde ise kullanıcının istekleri kullanılır ve son hali yani tamamlanmış hali teslim edilir. V süreç modeli çağlayan yaşam döngü modeli yani şelale modeline projelerde kullanılma konusunda benzerdir. İki modelde belirsizliklerin az gereksinimlerin açık olduğu modelde tercih edilir. V Süreç modelinde karşılaşılan sorun ise tekrar bulunmaması, alternatif değerlendirme ve risk analizi için ayrı bir yerin olmamasıdır.

**Helezonik(Sprial) Model:** Helezonik yani sprial model risk analizi ön planda olan bir modeldir. Helezonik modelin risk analizinin ön planda olması hataları erken gidermeye olanak sağlar. Helezonik modelin diğer ön plana çıkan özelliği ise prototipler oluşturmasıdır. Prototip oluşturma her aşamada yer alır ve bu durum kullanıcının her aşamada yazılım projesinin bir parçasını görmesine imkan sağlar. Bu durum oluşabilecek sorunların azalmasını sağlayabilir. Helezonik(Sprial) modelleme dört temel aşamadan oluşur. Planlama, risk analizi, üretim, kullanıcı değerlendirmedir. Planlama her aşamada var olan ürün için bir planlama yapılmasıdır. Risk analizi; risklerin araştırılması, analizi, belirlenmesi ve çözülmesidir. Üretim ana ve ara ürünlerin ortaya çıkartılması yani kısaca üretilmesidir. Ve son olarak kullanıcı değerlendirme kullanıcıdan alınan geri dönütlerin değerlendirilip bir sonraki aşamaya geçilmesidir. Helezonik modelin önemli avantajları vardır. Bu avantajlara örnek olarak kullanıcının üretime katkısının oldukça fazla olması verilebilir. Kullanıcının proje içerisinde yer alması kullanıcı isteklerinin tamamen karşılanmasını sağlar ve bir hata gerçekleşirse hata tespiti gecikmeden yapılır bu durum maddi ve zaman kazancı sağlar. Helezonik model bazı önemli avantajlara sahip olsa bile bazı sorunlara da yol açabilir. Model komplike bir hal alabilir ve sprialler sonsuza kadar gidebilir. Spriallerin sonsuza kadar gitmesi karışıklığa yol açabilir. Model düşük riskli, küçük projeler için oldukça pahalıdır.

**Kodla ve Düzelt Yaşam Döngü Modeli:** Kodla ve düzelt yaşam döngü modeli yazılı olmayan kaynaklardan çıkan, resmi olmayan bir ürün fikri ile ortaya çıkar. Proje başlangıcından bitişe kadar kodlama yapılarak devam edilir yani ürünü direkt gerçekleştirmeye yönelik çalışılır. Belgeleme ile ilgili şeylere yoktur bu sebepten dolayı takibi ve bakımı zordur. Kodla ve düzelt yaşam döngü modeli küçük projeler için kullanılır, büyük projeler için kullanılmaz. Bazı özelliklerinden dolayı avantajlara sahiptir. Bilir kişi görüşüne ihtiyaç yoktur, küçük proje ya da kısa ömürlü projelerde kullanılır ve herhangi bir planlamaya ihtiyacı yoktur. Dezavantajları ise kaynak planlaması olmadığı için kontrollü değildir ve hataların bulunması zordur. Kodlar kısa projeler için doğrudan yazıldığından dolayı esnek değildir ve değiştirilmesi oldukça zordur. Değiştirilmek istenilirse maliyet oldukça yükselebilir. Kodla ve düzelt yaşam döngü modelinde istenilen sonuca ulaşılabileceği kesin olmadığından ve zaman konusunda belirsizliğe sahip olduğundan kaliteli bir ürünün ortaya çıkması ihtimali oldukça azdır. Kısaca ve sonuç olarak kodla ve düzelt yaşam modeli en basit ürün geliştirme modelidir.

**Evrimsel Geliştirme Süreç Modeli:** Evrimsel geliştirme süreç modeli ilk tam ölçekli modeldir. Çok büyük alanlara yayılmış büyük organizasyonlarda yani firmalarda kullanılacak projeler için önerilmektedir. Evrimsel geliştirme süreç modelinin başarısı ilk evrimin başarılı olmasına bağlıdır. Yani her aşamada elde edilecek ürün her yönüyle işlevselliğe sahiptir. Bu modelde sürekli değişim söz konusudur. Bu durum sürekli değerlendirmeler demektir ve böylelikle geliştirme riskleri azaltılmış olunmaktadır. Sürekli değiştirilebilir olması aynı zamanda yazılım yapısına zararda verebilir. Sürecin görünürlüğü ve izlenebilirliği zorlaşır. Sürekli değişim yapılırken bazen ortaya çıkan gereksinimler değişebilir ve durumda proje bakımı zorlaşır.

**Artırımlı Geliştirme Süreç Modeli:** Artırımlı geliştirme süreç modeli temelde çağlayan yaşam modeli ile evrimsel geliştirme yaşam modelini ön plana alır. Bu modelde proje kullanıcının ihtiyaçları doğrultusunda parçalara bölünerek sıralanır. Artırımlı geliştirme süreç modelinde ürünün gelişimi bir tarafta devam ederken diğer tarafta ürün kullanılmaktadır. Bu kullanılan ürünler her seferinde bir önceki gerçekleştirilen şeyin üstüne eklenerek üretilmeye devam edilir. Bu model üretimi uzun sürebilecek, eksik işlevsellikle bile çalışabilmesi istenilecek projeler için tercih edilir, kullanılır. Artırımlı geliştirme süreç modeli ortaya çıkabilecek hatalar üzerinden çok süre geçmeden zamanında tespit edilip hatanın zamanında düzeltilmesinde etkilidir. Bunun sebebi yani hatların hemen düzeltilebilmesinin sebebi modelin ürün gelişimi olurken diğer taraftan kullanıcı tarafından kullanılmasındandır. Hataların azalmasının bir diğer sebebi proje aşama aşama ilerlediği için erken teslimler bir sonraki aşama için bir prototip görevi görür ve böylelikle projenin komple çökmesi engellenir, hatalar üretim aşamasında kolaylıkla tespit edilebilir. Her ne kadar hatların tespitinde bu model avantaj sağlasa da bu modelde var olan dezavantajlarda vardır. Mesela her bir parça birbiri içinde yer almaz, tekrar halinde değildir. Bir ara ürün oluşmadan bir sonraki ara ürüne geçilmez, değişiklik yapılmaz. Ana parçanın tamamlanması için öncelikle bu ara ürünlerin detaylı bir şekilde tasarlanması gerekmektedir ve böylelikle zamandan kayıp elde edilir. Bu model sonucunda elde edilecek sonuçlar kesin bir şekilde belli değildir. Tüm bu sebepler bu modelin zaman ve maliyeti hakkında kesin bir bilgi vermediğinden proje başı sabit fiyata, zamana anlaşmak uygun değildir.

Tüm bu modellerin incelenmesi ve anlaşılması geleneksel yazılım süreç modellerinin daha iyi anlaşılmasını sağlamaktadır. Hangi projede hangi modeli kullanırsak daha avantajlı olunur bu anlaşılmış olunur ve böylelikle zamandan, maliyetten tasarruf yapılır. Geleneksel yazılım süreç modellerinin dışında bir diğer incelenmesi gereken modeller çevik yazılım süreç modelleridir. Çevik yazılım süreç modellerinin de incelenmesi ile birlikte projelerde kullanılabilecek modeller daha net anlaşılmış olunur.

**Çevik Modeller:** Çevik modeller temelde başarı oranını arttırmak ve ekip içi etkileşimi arttırma amacıyla ortaya çıkmıştır. Günümüzde teknolojinin gelişimi ile birlikte büyümekte olan yazılım dünyasında kullanıcılara ürünleri daha uygun maliyete ve daha kısa sürede kaliteli bir şekilde sahip olmalarını sağlayan bir modellemedir. Bu modelin asıl ortaya çıkmasının sebeplerinden bir diğeri ise yazılımsal geliştirmelerin sıkıntılı bir süreç ve uzun zamanlar almasındandır. Yazılımsal geliştirmeler zaman zaman müşteri tarafından kesin bir memnuniyete sahip olmamaktadır. Yazılım hataları geç farkedildiğinde firmaların statülerinin zedenlenmesine yol açıp, maliyet ve zaman bakımından olumsuzluklara sebep olması gibi birçok olumsuzluk ortaya çıkmıştır. Tüm bu sorunlar giderilmek için 1990’lı yıllarda çevik modeller ortaya çıkartılmıştır. Ortaya çıkan çevik modeller projede ön planda olan bazı problemlerin sonucuna daha hızlı ulaşıp zamandan kazanç elde etmemizi sağlamıştır. Çevik modellerin bir diğer amacı risk oranını daha aza indirip yüksek performansta daha kaliteli ürünler elde etmektir. Sonuç olarak çevik modellerin prensipleri ele alınırsa; basitlik ön planda olmalı, geliştirmeyi destekleyen sürdürülebilir ve çevik bir yapıda olmalı, modellemeyi projelerinde kullanan ekibin etkileşimi iyi olmalı, projeleri yapmaya istekli olan başarılı kişiler etrafında projeler kurulabilir, çevikliğin olması için sağlam bir altyapı ve gereksinimler gerekmektedir ve zaman zaman ekip projeyi geliştirme adına bir araya gelerek yöntemlerini gözden geçirerek verimliliği arttırırlar. Yazılım projeleri gerçekleştirilirken kullanılan çevik modeller şöyledir; extreme programming(XP), rational unified process, feature-driven development(FDD), test-driven development(TDD), LEAN development, dynamic system development methodology(DSDM), Microsoft solution framework(MSF) ve en önemlilerden biri olan SCRUM denilebilir. En önemli olan XP ve SCRUM açıklanıp anlaşılması gerekmektedir.

**Extreme Programming (XP):** 1996 yılında Kent Beck ve arkadaşları tarafından ortaya çıkartılmıştır. Extreme Programming (XP) temelde dört maddeden oluşmaktadır. Bunlar; basitlik, cesaret, geri dönüş(feedback) ve iletişimdir. Disiplin olarak ekip içi çalışma söz konusudur. Bu adı geçen dört maddeden biri olan geri dönüş yani feedback sayesinde ekip içi geri dönüşler yapılarak çalışmanın neresinde olunduğu bilinir ve bundan sonraki çalışmalar bundan sonra ki çalışma temposu ona göre belirlenir. Aynı zamanda XP disiplini 12 pratik yönteme sahiptir. Adı geçen dört temel incelenince XP nin nasıl başarıya ulaştığına ulaşılabilir.

**İletişim:** Proje gerçekleştirilirken genellikle karşılaşılan en büyük sorunlardan birisi ekip içi iletişiminin sorunları olmasıdır. Extreme Programming (XP) sorun olabilecek ekip içi iletişimin olmamasına yönelik davranır. Ekip içi sağlıklı etkileşime, yüz yüze iletişime önem verir. XP için ekip içinde gerçekleşene iletişim açık ve anlaşılır olmalıdır.

**Basitlik:** XP için yapılan projede uzun uzun dokümantasyona yer verilmemelidir. Basit bir sistem ile gereksinimleri karşılamayı amaçlar. Yapılan iş sadece, kolayca anlaşılabilir olmalı, karışık olmamalıdır.

**Cesaret:** Proje gerçekleştirilirken acaba doğru ilerleniyor mu korkusuna kapılmamalı, eğer bu düşünce içinde devam edilirse panik halinde yapılmaması gereken şeyler yapılabilir. XP bu korkunun olmamasını gerektiğini savunur. Yazılımcının korkusuzca ilerlemesi gerektiğini savunur. Eğer hata gerçekleşmişse sıkıntı olarak görülmeden proje ya tamamen silinir yeniden başlanır ya da yapılan hatanın giderilmek için işe konulması gerektiği XP tarafından doğru bir yaklaşım olarak belirtilir.

**Geri Dönüş (Feedback):**  XP geri dönüş yani feedback ile geriye dönüşler ile proje sonu ortaya çıkabilecek hataları erkenden bulmayı hedefler. Müşteri, yazılım ekibi, yönetici iletişim halindedir. Bu adım ile ilerde çıkabilecek anlaşmazlıklar ortadan kaldırılmış olunur.

**SCRUM:** SCRUM çevik yazılım geliştirme yöntemlerinden en önemli olanlarından biridir. Günümüzde hala gelişmekte olan SCRUM 1990’lı yıllarda Jeff Sutjerland Ve Ken Schawaber tarafından oluşturulmuştur.Scrum günümüzde en çok kullanılan yazılım geliştirme yöntemidir. SCRUM kelime anlamı olarak Rugby oyununda oluşan küçük ekiplerdir. Karışık yazılım süreçlerinde kullanılır. Karışık, büyük projeler parçalara ayrılır. Ayrılan her parçaya “sprint” adı verilir. Ayrılan sprintler ayrı ayrı geliştirilir. SCRUM’da ekip içi iletişim XP de olduğu gibi çok önemlidir. Hatta o kadar önemlidir ki sıklıkla SCRUM meeting denilen görüşmeler olur. Scrum da projede gerçekleşen sorunlar, projenin ilerleyişi izlenebilir olmalıdır. Düzenli geri bildirimler yer almalı belirli planlamalarla hedefe ulaşma hedeflenmelidir. Scrum kullanıma en uygun gereksinimlerin açık olmadığı karmaşık halde olunan projeler için kullanılır. Scrum bilindik bazı şirketler tarafından kullanılmaktır. Microsoft, Facebook, Google gibi şirketler örnek verilebilir. Scrum belirli üç temel kavram doğrultusunda hareket eder, bunlar: Roller, toplantılar, ve araçlardır. Scrum’ın daha net anlaşılabilmesi için bu üç kavram incelenmelidir.

**Roller:** Üç tanedir, bunlar: Ürün sahibi, Scrum yöneticisi ve Scrum takımıdır. Ürün sahibi projenin beynidir, yani projede geri dönüşlerden sorumlu olan müşteridir. Scrum yöneticisi, ekibin scrum’a uygun hareket edip etmediğini kontrol eder ve projelerdeki olayları scrum’a adapte eder. Scrum takımı, birbiri ile sürekli etkileşim halinde olan scruma uyum sağlamış yazılım geliştirme ekibidir.

**Toplantılar:** Scrum için toplantılar çok önemlidir. Her sprint her gün scrum meeting gerçekleştirir ve bir gün öncesinde kim nasıl sorunlarla karşılaştıysa onları dile getirir. Fikir alış verişi gerçekleştirilir.

**Araçlar:** Proje boyunca yapılacak şeylerin basitçe yer aldığı ürün gereksinim dokümanı oluşturulur. Kendi içerisinde sprint dokümanı ve sprint kalan zaman grafiği olmak üzere ikiye ayrılır. Sprint dokümanı ürün gereksinim dokümanına göre oluşur. Sprint kalan zaman grafiği ise planlanan projenin neresinde olunduğu, planan zamana göre ilerlenip ilerlenmediği yer alır.

**SCRUM Neden Popüler?** Scrum günümüzde en çok kullanılan yazılım geliştirme yöntemidir. Peki bunun sebebi nedir? Scrum karmaşık yapıları basite indirgeyerek gereksinimleri basit bir hale getirir. Böylelikle zamandan ve maliyetten zarar etmemizi engeller. XP gibi iletişime çok önem verir ve motivasyonun sürekli yüksek olmasına önem verir. Gelişen günümüz teknolojisi ile sürekli gelişmekte olunan gereksinimlere kolay bir şekilde uyum sağlar. Kullanıcı ile sürekli geri dönüş yani feedback içerisinde olduğu için hataları oldukça azaltmamızı sağlar. Karmaşık yapıları parçalayarak projenin çözümünü kolaylaştırır. Bu sebeplerden dolayı Scrum metodu günümüzde yazılım geliştirmecileri tarafından oldukça popülerdir.

**Hangi Projede Hangi Modeli Kullanmalıyız?** En en basit projeler için gelişigüzel model kullanılabilir. Gereksinimleri çok iyi tanımlanmış, kısa sürede bitecek projeler için çağlayan yaşam döngü modeli yani şelale modeli kullanılabilir. Belirsizlikleri az, gereksinim tanımlamaları açık projeler için V modeli kullanılır. Düşük riskli, küçük projeler için Helezonik (Sprial) kullanılmamalıdır çünkü maliyet çok artar. Ürün direkt gerçekleştirilmek isteniliyorsa, küçük bir proje yapılıyorsa ve kişiye özel ise kodla düzelt yaşam döngü modeli kullanılır. Üretimi uzun sürebilecek eksik işlevsellikte bile çalışabilecek projeler için artırımlı geliştirme süreç modeli kullanılabilir. Çok büyük etkileşim alacak projeler için evrimsel geliştirme, maliyetli ve uzun süren projeler için helezonik (Sprial) kullanılabilir. Orta büyüklükte uzun sürmeyecek projelerde çevik model uygundur.

**Kaynakça:**

https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8

<https://talentgrid.io/tr/yazilim-gelistirme-modelleri/>

<https://enprobilisim.com/yazilim-gelistirme-sureci-modelleri-sdmp/>

<https://medium.com/@tunaytoksoz/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-sdlc-ve-modelleri%CC%87-c3fe40f6e4e8>

<https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742>

<https://medium.com/@brfn.kcr26/yazilim-geli%CC%87%C5%9Fti%CC%87rme-ve-s%C3%BCre%C3%A7-modelleri%CC%87-2131ea5f09b2#:~:text=1.Kodla%20Ve%20D%C3%BCzelt%20Ya%C5%9Fam,kadar%20kodlama%20yap%C4%B1larak%20devam%20edilir.&text=%C3%BC%20%C3%87ok%20k%C3%BC%C3%A7%C3%BCk%20projelerde%20ya,%C3%BC%20Program%20a%C5%9Famalar%C4%B1%20%C3%A7abuk%20ge%C3%A7ilir>.

<https://iskulubu.com/yazilim/yazilim-gelistirme-yasam-dongusu/>

<https://tr.linkedin.com/pulse/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-nedir-veysel-ugur-kizmaz>

<https://medium.com/@denizkilinc/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-temel-a%C5%9Famalar%C4%B1-software-development-life-cycle-core-processes-197a4b503696>

Doç. Dr. Deniz KILINÇ Yazılım Mühendisliği Temelleri ders notları

Muhammed Berkay Tokdam

210601005