Ceren SÜRMELİ

No: 200601066

Yazılım Yaşam Döngü Modelleri

Yazılım yaşam döngüsü (“SDLC”, “Software Development Life Cycle”), yazılım ürünlerinin oluşturulması ve geliştirilmesini sağlayan süreçtir. Yazılımın üretim ve kullanım aşamalarını kapsar.

Geleneksel Yazılım Yaşam Döngüsü doğrusaldır. Gereksinim, Analiz, Tasarım, Gerçekleştirme, Test ve Bakım aşamalarından oluşur. Ancak pratikte doğrusal olarak ilerlemesi çok zordur. Proje ilerledikçe gereksinimler değişir (Moving Target Problem). Proje müşterinin ve kullanıcının inisiyatifine göre şekil alır. Bu nedenle, bu aşamalar dinamik olarak; bir döngü halinde kullanılmaktadır.

* Gereksinim: *“Müşteri ne istiyor?”* Müşterinin istekleri ve ihtiyaçları belirlenir. Bu verilere göre müşteri ile uygulanabilir durumlar değerlendirilir. Kullanıcı ihtiyaçlarını en iyi tamamlayacak veriler oluşturulur.
* Analiz: *“Konuyu detaylı değerlendirelim.”* Belirlenen sistem gereksinimleri üzerine detaylı araştırma çalışmaları yapılır. Oluşabilecek sorunlar hakkında değerlendirme yapılır.
* Tasarım: *“Projeyi gerçekleştirirken nasıl bir yol izleyeceğiz?”* Projenin en verimli şekilde gerçekleştirilebilmesi için gerekli tasarımlar yapılır. Bir yol haritası oluşturulur. Soyutlama tekniği ile nesneler, olaylar ve durumlar gereksiz detaylardan arındırılarak proje net bir şekilde belirlenir. Bu aşamada yapılacak bir hata maliyet aşımına ve zaman kaybına sebep olur.

Ayrıca Prototipleme, Tasarım aşamasının bir parçası olabilir. Prototipler yazılımın en temel halidir bu sayede yazılıma dair ilk çerçeve oluşturulmuş olur.

* Gerçekleştirme: *“Şimdi sıra yazılım geliştirmede.”* Tasarlanan adımlara uygun olarak ürün, belirlen dil ve özelliklerle inşa edilir. Kodlama yapılırken okunabilirliği ve bakımı kolay olan bir çalışma yapılmaya özen gösterilmelidir. Takım çalışmalarında uygun iş bölümleriyle yazılım aşaması tamamlanır.
* Test ve Bakım: *“Ürünümüz sınavları geçebilecek mi?”* Test, genellikle tüm aşamaların bir alt kümesidir. Ürün kalite standartlarına ulaşıncaya kadar hatalar rapor edilir, takip edilir, düzeltilir ve yeniden test edilir. Proje teslim edildikten sonra (kullanıcının geri dönüşü de dikkate alınarak) belirli aralıklarla iyileştirme veya onarımı kapsayan bir bakım yapılır.

Yazılım Yaşam Döngü Modelleri; “Geliştirme ekiplerinin yazılım sistemlerini geliştirmek için kullandıkları modellerden ve metodolojilerden oluşur; bu metodolojiler, tüm geliştirme sürecini planlamak ve kontrol etmek için çerçeve oluşturur [1]. “. Böylece oluşabilecek karmaşıklıklar önlenir. Projeye uygun yöntem kullanılarak verimliliği arttırmak ve zaman tasarrufu sağlamak mümkün hale gelir.

Geçmişten günümüze birçok farklı yöntem kullanılmıştır. Başlıca yazılım yaşam döngü modelleri aşağıdaki gibidir;

* Gelişigüzel Model
* Kodla ve Düzelt (Code and Fix)
* Barok Modeli
* Şelale Modeli (Waterfall Model)
* V Modeli (V-shaped Model)
* Prototipleme (Prototyping)
* Helezonik (Spiral) Model
* Evrimsel Geliştirme (Evolutionary Development)
* Artımlı Gerçekleştirme Modeli(Incremental Development)
* Yinelemeli Model (Iterative Model)
* Formal Sistem Geliştirme (Formal System Development)
* Birleşik Süreç (Unified Process)
* Çevik Modeller (Agile Models)

**Gelişigüzel Model:**

Aslında bir model olarak adlandırmak pek doğru değildir. Geliştiren kişiye bağımlıdır ve belli bir tekniği-yöntemi yoktur. Bu nedenle izlenebilirliği ve bakımı çok zordur. 1960’lı yıllarda daha çok bireysel çalışma ortamlarında kullanılmıştır.

**Kodla ve Düzelt (Code and Fix):**

Daha çok bireysel çalışmalar ve kısa projeler için uygundur. Projenin ilk olarak bir prototipi yapılır ve istenen sonuç elde edilene kadar üzerinde düzenleme yapılır. Özellikle uzun çalışmalarda düzelttiğimiz kod başka bir kısmın bozulmasına sebep olabilir ve düzenleme aşamaları içinden çıkılmaz bir hal alır. Müşterinin sürece dahil edilmemesi kullanıcı ihtiyaçlarına uygun olmayan bir proje oluşmasına sebep olabilir. Takım çalışmaları için uygun değildir. Bakım aşaması vardır ancak dokümantasyon olmadığından çok zordur.

**Barok Modeli:**

Eski bir model olan Barok modeli 1970’lerde kullanılmaya başlanmıştır. Bu modelde yaşam döngüsünün temel adımları doğrusal olarak ele alınır yani bir döngü içinde değil de sırayla gerçekleşir. Aşamalar arası geçişlerin nasıl olacağı tanımlı değildir. Günümüzdekinin aksine Belgeleme temel adımların sonucu olarak değil de ayrı bir adım olarak ele alınır; gerçekleştirme ve test aşamaları bittikten hemen sonra yapılır. Bu projenin Belgelenmesi sırasında hatalara veya zaman kaybına sebep olur, bakım aşamasını zorlaştırır. Gerçekleştirme aşamasına daha çok önem veren bu model günümüzde tercih edilmemektedir.

**Şelale Modeli (Waterfall Model):**

Tanıtılan ilk süreç modeli olan Şelale Modelinde Yaşam döngüsünün temel adımları en az bir kez tekrar edilir. Barok modelinin aksine belgeleme temel adımların doğal sonucu olarak ele alınır. Bir önceki aşamanın sonucu, bir sonraki aşama için girdi görevi görür. Aşamalar çakışmaz, bir şelale gibi birbirini takip ederek (bir aşama bitmeden diğer aşamaya geçmeye izin vermez) “ileriye” akar. Dokümantasyon ve test bitmeden aşama tamamlanmış sayılmaz. Bu da projenin hızını sınırlandırır.

Geleneksel Model olarak da bilinen bu model geçmişte çok fazla kullanılmasına rağmen etkisini gittikçe kaybetmektedir. Sebebi; doğrusal bir ilerlemenin, gereksinimlerin başta kusursuzca belirlenmesini gerektirmesidir. Bu büyük projelerde gerçekleşmesi zor bir istektir. Proje ilerledikçe farklı gereksinimler ortaya çıkması (Moving Target Problem) veya kullanıcının ihtiyaçları farklılaşması kaçınılmazdır. Atlanan ufak bir detay projeyi durdurabilir. Ayrıca kullanıcının sürecin içinde yer almaması daha sonradan bakım ve geliştirme maliyetlerini arttırır. Ancak küçük (gereksinimlerin net bir şekilde belirlenebilen) projelerde bir alternatif olarak kullanılabilir.

**V-Modeli:**

V- Modeli, Şelale modelinin belirli düzenlemeler yapılımış bir uzantısıdır. Her temel yazılım geliştirme aşamasına paralel olarak bir test aşamasını içerir. Bu aşamalar şematik olarak V harfine benzediği için bu şekilde adlandırılır. Sol taraf üretim sağ taraf sınama süreçlerini kapsar. Şelale modelinin aksine kullanıcıyı olabildiğince projenin içinde tutmayı amaçlar. Üç kısımdan oluşur, yukarıdan aşağıya doğru;

* Kullanıcı Modeli: Daha çok gereksinim analizi kısmını kapsamaktadır. Sistemin ihtiyaçları belirlenir, ihtiyaçlara yönelik detaylı araştırmalar yapılır. Kullanıcının değerlendirmeleri ve bakış açısı dikkate alınarak geliştirmeler yapılır. Müşteri ile sürekli iletişim içinde olunur.
* Mimari Model: Bu aşamada mimari(tasarımsal) ögeler anlaşılır ve tasarlanır. Dış dünya ile iletişim ve veri aktarımı tanımlanır. Sistemin tüm modülleri için ayrıntılı iç tasarım belirlenir. Maksimum hatanın erkenden belirlenmesi beklenir.
* Gerçekleştirim Modeli: Tasarlanan sistem modülleri kodlanır. En uygun programlama dili belirlenen gereksinimlerle seçilir. Bitirilen kod çok sayıda kod testinden geçirilir.

V-modelinde gereksinimler iyi tanımlanmış olmalıdır. Şelale modelindeki gibi sıralı olduğundan bir geri dönüş mekanizması bulunmamaktadır. Bu nedenle benzer problemleri doğurur. Dinamik bir dünyada gereksinimlerin değişmeden kalmasını bekleyemeyiz. Gereksinimlerin iyi anlaşıldığı projeler için uygundur. Kesinlikle disiplinli bir model olduğu için gereksinimlerin net belirlendiği Tıbbı Geliştirme alanında ve BT(Bilgi Teknolojileri) alanında kullanılmaktadır.

**Prototipleme:**

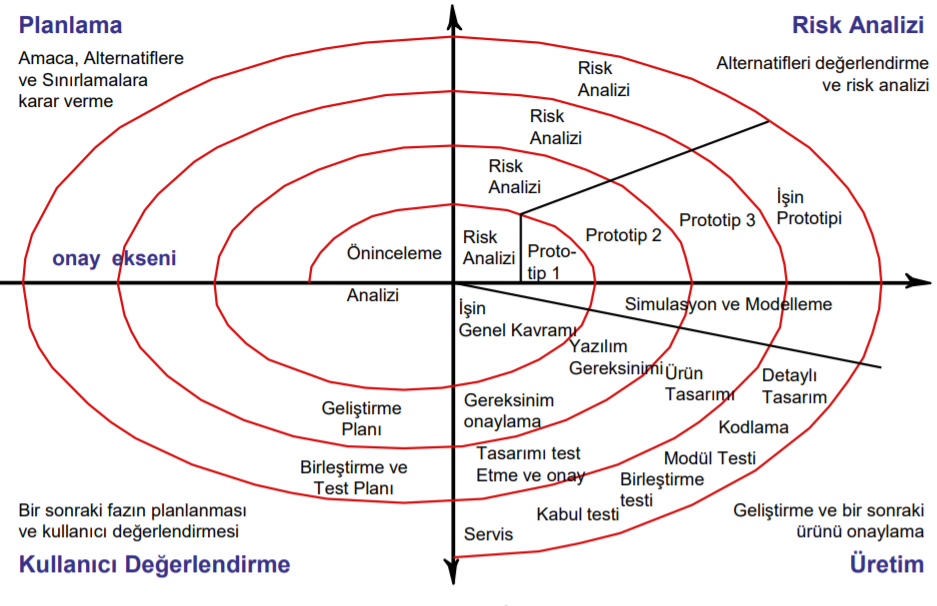
Prototip, sınırlı işlevlere sahip minik yazılım modelidir. Projenin ilk olarak prototipi oluşturulur. Daha sonra gereksinimler kullanıcı ve müşterinin isteklerine göre şekil alarak ana proje oluşturulur. Bu sayede karmaşa ve yanlış anlamalar engellenmiş olur. Müşteri ihtiyaçlarına uygun bir proje ortaya çıkar. Sorunlar erken fark edildiğinden maliyeti azdır. Fakat Belgelendirmesi olmayan prototipler oluştuğundan dolayı karmaşaya yol açabilir. Düzgün izlenmezse harcanan tüm çaba gidebilir. Bu nedenle riskli bir yöntemdir.

**Helezonik (Spiral) Model:**

Tasarımı doğrusal bir süreç olarak değil de döngüsel bir süreç olarak kabul eder. Döngüleri genişleyen bir spiral şeklinde tanımlar. Yinelemeli geliştirme fikri ile şelale modelinin sistematik disiplinini -özellikle net gereksinim analizini- bir araya getirir. Spiral model dört aşamadan oluşur. *Şekil-1* deki gibi proje bu aşamalardan tekrar tekrar gerçekleştirir her döngü bir ‘fazı’ ifade eder.

* Planlama: Gereksinimler toplanır ve değerlendirilir. Oluşacak ara ürün için planlamalar yapılır, amaç belirlenir. Bir önceki ara ürün, ürün olarak kullanılır ve geliştirilerek yeni bir ara ürün elde edilir.
* Risk Analizi: Projede oluşabilecek program kayması, maliyet aşımı gibi riskler değerlendirilir. Bunlara yönelik çözümler sunulur.
* Üretim: Gerçek yazılımı oluşturacak ara ürünün her fazda üretildiği aşamadır.
* Kullanıcı Değerlendirmesi: Ara ürün ile ilgili gerekli testler yapıldıktan sonra müşterinin ara ürünü değerlendirdiği aşamadır.

Spiral modelde risk analizi ön plana çıkar. Her fazda bir ürün geliştirilir ve sonraki faz için planlamalar yapılır. Bu sayede Şelale modelinin aksine önceki fazda oluşan bir hata sonraki fazda telafi edilebilir. Prototip yaklaşımı vardır. En riskli kısımlar önce tasarlanarak karmaşıklık engellenir. Aslında pek çok yazılım modelinin sorunları ele alınarak harmanlanmış bir halidir. Kompleks bir yapısı yönetimi daha karmaşık yapar. Büyük projelerde kullanışlı olmasına rağmen küçük projeler için pahalı bir yöntemdir. Proje süresi belirlenemeyebilir. Çok sayıda ara aşama ve dokümantasyon gerektirir.



*Şekil-1: Spiral Model Yaklaşımı*

**Evrimsel Geliştirme Modeli:**

Evrimsel Geliştirme Modeli ilk tam ölçekli modeldir. Projenin başarısı ilk evrimin başarısına bağlıdır. Gereksinimleri basitleştirir. Genel tanımlama yapıldıktan sonra ‘Tanımlama’, ‘Geliştirme’ ve ‘Test etme’ aşamaları eş zamanlı olarak gerçekleşir. İlk sürüm ve ara sürümler tamamlanırken müşteriyi işin içinde tutar. Müşteri geri bildirimlerine göre proje şekillenir. Müşterinin sürekli projede bulunması hız kaybına sebep olur fakat hata oranını azaltır. Takip edilebilirlik ve bakımı zordur. İki çeşit evrimsel geliştirme modeli vardır. Bunlar;

* Keşifçi Geliştirme: Müşteri gereksinimlerini detaylı inceler beraber çalışır. Net gereksinimlerle başlanmalıdır.
* Atılacak Prototipleme: Amaç Sistem gereksinimlerini anlamaktır. Başlangıçta gereksinimler tam oturmamıştır zamanla netleşir.

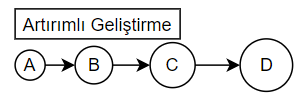
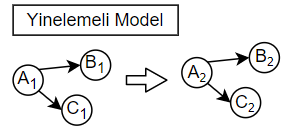
Küçük ve orta boyutlu sistemlerde ya da büyük bir sistemin parçalarının oluşumunda kullanılabilir.

**Artırımlı Gerçekleştirme Modeli:**

Artırımlı Gerçekleştirme modelinde ürün sürümler(artırımlar) şeklinde oluşturulur. Her sürüm bir öncekini kapsayan, geliştirilmiş bir formunu oluşturur. Kullanım ve üretim eş zamanlı gerçekleşir. İlk önce gereksinimler belirlenerek artırımlara paylaştırılır. Daha sonra ürünün basite indirgenmiş ilk sürümü oluşturulur. Bu sürüm kullanıcı tarafından test edilir ve son halini alana kadar artırımlı bir şekilde yeni özellikler eklenerek son ürün oluşuncaya dek devam edilir. Örnek vermek gerekirse: Öğrenci kayıt sistemi bu yöntemle yapılırken ilk önce sadece girilen öğrencinin bilgilerinin kaydedilmesi işlemi gerçekleştirilebilir. Daha sonra buna kayıtları yenileme, silme, toplu işlem yapma gibi özellikler eklenir. Bu modelin önemli bir dezavantajı artımların kendi içinde tekrar etmesine izin vermemesidir.

**Yinelemeli Model (Iterative Model):**

Genellikle Artırımlı modelle karıştırılır fakat aslında birbirinden farklı geliştirme yöntemleri vardır. Artırımlı modelde başlangıç sürümü geliştirme prensibini izlerken Yinelemeli model her sürümü iyileştirme prensibini izler. Aşağıdaki modellerle farkı daha iyi anlayabilirsiniz.

Bu iki model beraber kullanılarak “**Yinelemeli ve Artımlı**” bir yol izleyebilir. Böylece daha verimli bir kombinasyon oluşturulmuş olur. Çevik yazılımlarda bu yöntem kullanılır.

Yinelemeli ve Artımlı Modelde sonuçlar erken ve periyodik olarak alınır. İlerleme takip edilebilir. Riskli kısıma öncelik verilerek yönetmek daha kolay olur. Değişen gereksinimler desteklenir. Kompleks yapısından dolayı yönetimi daha zordur. Büyük projeler için daha uygundur.

**Formal Sistem Geliştirme Modeli:**

Formal Sistem Geliştirme Modeli’nde matematiksel bir teknik kullanılır. Temelinde karmaşık sistemleri geliştirme ve yazılımda doğruluğun geliştirilmesi olgusu yatar. Amaç pahalıya patlayan hata ayıklama işlemine maruz kalmamak için başlangıçta tamamen doğru yazmaktır. Bu sayede hatasız yazılım geliştirmeyi sağlar. Fakat bu çok zaman alan bir süreçtir. Yeteri kadar donanımı olmayan kişilerle kullanımı fazlasıyla zordur. Teknikleri uygulayabilmek için eğitim ve özel beceriler gerekmektedir.

**Çevik Modeller (Agile Models ):**

Genellikle takım çalışmasını içeren Çevik modeller özellikle müşteri ihtiyaçlarını ön planda tutar. Kullanıcı deneyimine ve değerlendirmesine odaklanır. Yinelemeli ve artımlı süreç modellerinin bir kombinasyonudur. Ürün küçük artımlı yapılara bölünür. Bu yapılara yinelemeler yapılır. Her yineleme eşzamanlı çalışan Planlama, Gereksinim Analizi, Tasarım, Kodlama, Birim Testi ve Kabul Testi aşamalarını içerir.

Çevik model her projenin gereksinimlerine en iyi uyacak şekilde farklı bir bakış açısıyla ele alınması gerektiğini savunur. Esnekliği ve uyarlanabilirliği sayesinde sık kullanılmaya başlanmıştır. Ekip çalışması önemlidir. Sık sık denetimler ve müşteri değerlendirmeleri yapılır. Başlıca çevik programlama yöntemleri aşağıdaki gibidir;

* Rational Unified Process (1994)
* Scrum (1995)
* Crystal Clear
* Extreme Programming-XP (1996)
* Adaptive Software Development
* DSDM (1995)



Resim-2: Çevik Model

**Çevik Model ve Geleneksel Yazılım Yaşam Döngüsü Modelleri Karşılaştırması**

Geleneksel SDLC modellerinde ekipler genellikle ayrıntılı planlama ile çalışır. Gereksinimlerin baştan çok iyi belirlenmesi beklenir. Yapılan ayrıntılı planlamadan sonra yapılacak en ufak değişiklik projenin aksamasına sebep olur. Müşteri ile etkileşim yeterli olmadığı için ortaya müşterinin isteği dışında ürünler çıkabilir.

Çevik modellerde ise ayrıntılı planlama yoktur. Yapılacak projede tıpkı bir resim tablosu gibi önce ana hatlar belirlenir; Daha sonra bu çerçevede proje şekil alarak ilerler. Uyarlanabilir bir yaklaşım kullanılır. Bu sayede ekip değişikliklere kolay uyum sağlayabilir. Çok sık testlerden geçirilerek ciddi sorun oluşma riski en aza indirilir. Çevik Modelde müşteri etkileşimi çok önemlidir. Burada müşterinin isteklerini doğru belirlemek önemlidir. İsteklerini aktarmakta zorlanan bir müşteri ekibi yanlış yönlendirebilir.

1. **Uç Programlama (Extreme Programming - XP):**

En popüler Çevik süreçlerden biridir. XP, yazılım geliştirme süreci boyunca son derece kaliteli ve çalıştırılabilir bir kod sunmayı amaçlar. Kısa adımlardan oluşan tekrarlı bir süreçtir. Her bir adımda geliştirilebilir ve kaliteli kod üretimine önem verilmektedir. Kullanıcı gereksinimlerinin değişken veya net olmadığı orta ölçekli projelerde kullanılır.

XP’nin temel prensipleri başlıca aşağıdaki gibidir;

* Hızlı Geri Bildirim Edinme: Sık ve hızlı geri bildirimler projenin daha sağlam bir şekilde ilerlemesini sağlar.
* Basit Çözümler Üretme: Basit çözümlerin kavranması ve anlatımı kolaydır. Ayrıca hızlı geri bildirim oluşmasını sağlar.
* Ufak Değişiklikler Yapmak: Büyük değişimlerdense ufak ve sık değişiklikler yapılarak projenin bozulması önlenir.
* Yeniliklere Açık Olmak: Baştan konulmuş sıkı kurallar olmadığı için proje yeniliklerle şekillendirilebilir.
* Yüksek Kaliteli Yazılım: Ekibe yüksek kaliteli yazılım yapılacak çalışma ortamları sağlanır. Böylece en verimli şekilde çalışılmış olunur.
* Usta-Çırak İlişkisi: Takımlarda kıdem farkı yoktur. Tecrübeli programcılar daha az tecrübe sahibi olan programcılara işi öğretir. Bilgi paylaşımı sağlanmış olur.
* Somut Denemeler Yapmak: Verilen kararların sonuçlarını gözlemleyebilmek adına somut denemeler yapılır. Böylece kararlar daha net sınanmış olur.
* Samimiyet: Ekip üyelerinin birbiri arasında ve müşteri ile olan iletişimlerinde samimiyet ve saygı önemli bir unsurdur.
* Sorumluluk Bilinci: Sorumluluk bir lider tarafından verilmektense her ekip üyesinin sorumluluklarını bilmesi beklenir.

1. **SCRUM:**

Günümüzün popüler metotlarından olan Scrum’da birey etkileşimine ekstra önem verilir. Sık ve kapsamlı dokümantasyonlar değil de çalışan doğru kod ön plandadır. Müşteri ile orta noktada buluşarak ürünü en verimli hale getirmeyi amaçlar. İletişim sayesinde hatalar azalır. XP de olduğu gibi ihtiyaca yönelik ve esnek bir yapısı vardır. Kompleks süreçler için kullanılır.

Proje bütün olarak değil de parçalar halinde değerlendirilir. Her bir parçanın gerçekleşme aralığına ‘Sprint’ denir. Sprintler genelde 1-2 haftalık bir zamanı kapsar. Gereksinim analizi, Tasarım, gerçekleştirme, Test aşamaları eşzamanlı olarak ilerler. Proje yönetimi şeffaftır bu sayede hatalar ve eksiklikler erkenden fark edilir. Yapılan toplantılarla sprintlerin akışı, risk analizleri ve maliyet hesaplamaları değerlendirilir. Bu noktada her gün yapılan 15 dakikalık ayaküstü toplantılar büyük önem taşır.

Scrum da farklı kavramlar vardır. Bunlar;

* Product Backlog: Projedeki gereksinimlerin bir listesidir. Müşteriden alınan ve birlikte belirlenen gereksinimler öncelik sırasına göre sıralanır.
* Product Backlog Item: Product Backlog içindeki herhangi bir gereksinime verilen addır.
* Sprint: Proje Sprint denilen parçalara ayrılır. Böylece daha düzenli ilerlenir.
* Sprint Backlog: Product Backlog Item’lar öncelik sırasına göre Sprintlere paylaştırılmasıdır.
* Scrum Board: Sprint içerisinde yapılacak olan maddeler yani Sprint Backlog’lar burada yönetilir.
* Burndown Chart: Sprintte kalan işi gösteren bir grafiktir. Projedeki şeffaflığı sağlar.

**Kaynakça**

* [1] Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham, Soo Fun Tan, "Software Development Life Cycle AGILE vs Traditional Approaches", Int. Conference on Information and Network Technology, IPCSIT Singapore, Vol. 37, pp. 162, 2012.
* Doç. Dr. Recep ERYİĞİT , “*Yazılım Mühendisliği Süreç Modelleri*” , Ankara Üniversitesi
* Doç. Dr. Resul DAŞ, “*Yazılım Yaşam Döngüsü ve Süreç Modelleri*”, Fırat Üniversitesi
* Doç. Dr. Deniz KILINÇ, “Yazılım Yaşam Döngü Modelleri” Ders notu, İzmir Bakırçay Üniversitesi
* BROOKSHEAR J.G. Bilgisayar Bilimine Giriş. ISBN 978–605–320–361–2.
* <https://phoenixnap.com/blog/software-development-life-cycle#:~:text=Software%20Development%20Life%20Cycle%20is,%2C%20Test%2C%20Deploy%2C%20Maintain>.
* <https://stackify.com/what-is-sdlc/>
* <https://www.tutorialspoint.com/sdlc/index.htm>
* <https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/>
* <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>
* <https://medium.com/@ahmetuyar/extreme-programming-xp-nedir-ddc003a515c4>
* <https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8>
* <https://stackify.com/what-is-sdlc/>