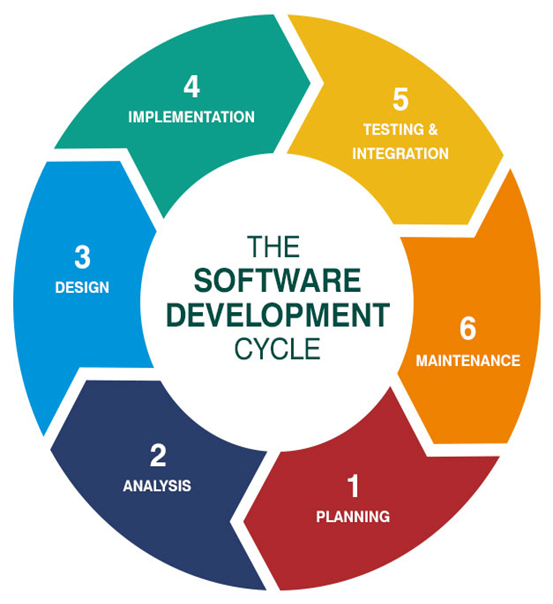
**Yazılım Yaşam Döngüsü**

Bu makalede; yazılım yaşam döngüsünün (SDLC) ne olduğunu ve ne gibi aşamalarının olduğunu, yazılım geliştirme süreci modellerinin neler olduğunu, modellerin nasıl uygulandıklarını, modellerin içerdiği kavramları ve işleyiş biçimlerini, hangi projede hangi modelin kullanımının daha uygun daha avantajlı olduğunu, günümüzde hangi yazılım geliştirme süreç modelinin daha çok kullanıldığını ve sebeplerini irdeleyeceğiz.

**Yazılım Yaşam Döngüsü Nedir?**

Yazılımlar bilinen/düşünülenin aksine cansız değillerdir. Canlı bir varlık gibi bir oluşur, gelişir ve nihayetinde işlevlerini yitirip tabiri caizse ölürler. Bu şekilde düşündüğümüzde “Yazılım Yaşam Döngüsü” nü anlamamız daha kolay olacaktır diye düşünüyorum. Yazılım Yaşam Döngüleri (Software Development Life Cycle) geliştirilen bir yazılımın oluşumunun planlanmasından, müşteriye teslimine kadar hatta teslim sonrası geri dönüş ve kontrollerin yapıldığı ya da hataların düzenlendiği o zaman diliminin de dahil olduğu bir süreçtir.

Yazılım Yaşam Döngüleri genel hatlarıyla planlama, analiz, tasarım ve test evrelerini içerirler. Bu evrelerin sadece bir kez gerçekleştirilmiş olması genelde yeterli değildir çünkü gereksinimler müşteri ile istişareler sonucu sürekli gelişip değişebilmektedir. Bu sebeple bu aşamalar bir döngü şeklinde tekrarlanarak yazılımlar oluşturulur. Bu döngüler sayesinde oluşturduğumuz yazılımlar daha kaliteli olur ve daha az hata içerirler. Aynı zamanda yazılım yaşam döngülerini kullanmak enerji ve maliyetten de tasarruf etmemizi sağlarlar. 

Yazılım Yaşam Döngülerinin evrelerini listeleyip bunları biraz daha detaylı inceleyelim:

1)Planlama Aşaması

Yazılım yaşam döngüsünün ilk evresidir. Üretilecek olan yazılım için müşteri ile iletişime geçilir ve gereksinimler belirlenir. Bu gereksinimler doğrultusunda maliyet, zaman, personel gibi ihtiyaçları en uygun ve verimli biçimde kullanabilmek için yapılması gereken ön hazırlıklar yapılır. Ve tüm süreç bu aşamada yapılan hazırlıklar doğrultusunda planlanır.

2)Analiz Aşaması

Planlama aşaması tamamlandıktan sonra yapılan planların müşteri ile istişare edilerek gereksinimlerin netleştirildiği, ihtiyaçların kesinleştiği, proje ayrıntılarının belirlendiği ve bunların dokümantasyonunun yapıldığı evredir. Bu dokümantasyon proje yaşam döngüsü boyunca geliştirilmesi gereken her şeyi içerir. Bu evre ile proje daha net bir biçimde ortaya koyulmuş olur.

3)Tasarım Aşaması

Analiz aşamasındaki sonuçlar doğrultusunda yazılımın özelliklerinin ve ara yüzlerinin belirlendiği aşamadır. Bu aşamada tasarım dokümantasyonu da oluşturulur. Bu dokümanda proje bilgileri, kullanıcı ara yüz tasarımları ve ayrıntıları, sistem tasarımı ile ilgili bilgiler ve tasarımın ayrıntıları bulunur. Bu aşamada en büyük hedefimiz uygulama/geliştirme aşamalarına olabildiğince eksiksiz ve hatasız geçebilmek, projeyi uygulamaya geçmeden net bir biçimde anlaşılmasını sağlayabilmektir. Tasarım aşamasının düzgün bir biçimde tamamlanması proje ekibine daha sonradan katılacak çalışanların projeyi daha kolay bir şekilde anlayıp, özümseyebilmesi açısından da oldukça kıymetlidir.

4)Gerçekleştirme Aşaması

Tasarım aşaması büyük ölçüde tamamlandıktan sonra “kodlama ve test” yapacağımız gerçekleştirme aşamasına geçeriz. Bu aşamada müşterinin istediği ürün programlanmaya başlanır. Daha önceki aşamalara uygun olarak belirlenen plana göre yazılım projesi kodlanır. Bu aşama ile net bir biçimde üretim başlar. Burada yazılan kod “clean code” olmalı yani okunulabilir, anlaşılabilir, bakımı kolay olmalıdır.. Yazılımı oluştururken genellikle ekip çalışması gerçekleştirildiği için basit ve anlaşılabilir kod yazmak ekip çalışmasının daha sürdürülebilir olması ve devamlılık sağlayabilmesi açısından da oldukça önemlidir.

Gerçekleştirme aşamasında yaptığımız diğer bir aşama da testtir. Kodlama yapmaya başladığımızdan itibaren test aşamasını aklımızda bulundurmalı ve o şekilde ilerlemeliyiz. Bu aşamada uygulayabileceğimiz farklı kategorilerde ve etkinlikte test türleri bulunmaktadır. Yani test aşamasında temel amaç yazılımı müşteriye sunmadan önce geliştirilen yazılımı test etmek varsa hataları ayıklayabilmek ve yazılımın en optimize halini müşteriye sunmaktır.

5)Teslim ve Bakım Aşaması

Tüm aşamalar tamamlandıktan sonra yazılım ürünü ve bir çeşit kullanma kılavuzu müşteriye teslim edilir. Teslimden sonra oluşabilecek ya da oluşan sorunlar ışığında da bakım aşaması başlar. Geliştiriciler müşteriden gelen değerlendirmelere göre üründe geliştirmeler, iyileştirmeler yapar.

Böylece yazılım yaşam döngüsü aşamalarını açıklamış olduk. Şimdi de yazılım yaşam döngü modellerini inceleyip karşılaştıracağız.

**Yazılım Geliştirme Süreci Modelleri**

Yazılım projelerinin büyüklüğünü ve farklılığına bağlı olarak zaman içinde birden çok yazılım geliştirme süreci modeli oluşturulmuştur. Bu modeller: Gelişigüzel Model, Barok Model, Şelale Modeli (Waterfall Model), Evrimsel Geliştirme Modeli, Artımlı Geliştirme Modeli (Incremental Development Model), Kodla Düzelt Modeli, Spiral (Helezonik) Model, V Modeli (V-Shaped Model) ve Çevik Model’lerdir.

**Gelişigüzel Model**

Kullanılan bir yöntem veya model yoktur, yazılım gelişigüzel yapılır. 60’lı yıllarda ortaya çıkmıştır. Geliştiren kişiye bağımlı olduğundan iyileştirme ve bakım yapmak oldukça zordur. Ekip çalışması değil tek kişilik üretim ortamı vardır, çok basit proglamlamaya sahiptir.

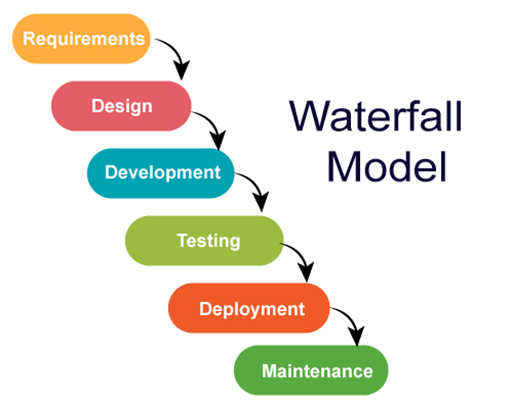
**Barok Modeli**

70’li yıllarda kullanılmaya başlanan bir modeldir. Yazılım yaşam döngü model aşamaları gerçekleştirilir fakat adımlar arası geri dönüşler hakkında net bir yöntem belirlenmemiştir. Belgeleme bugün her projede yer alması gereken bir aşama iken Barok Modeli’nde belgeleme yazılım sürecine dahil edilmemiştir. Belgeleme ayrı bir süreç olarak görülmüştür. Bu model günümüzde pek kullanılmamaktadır.

**Evrimsel Geliştirme Modeli**

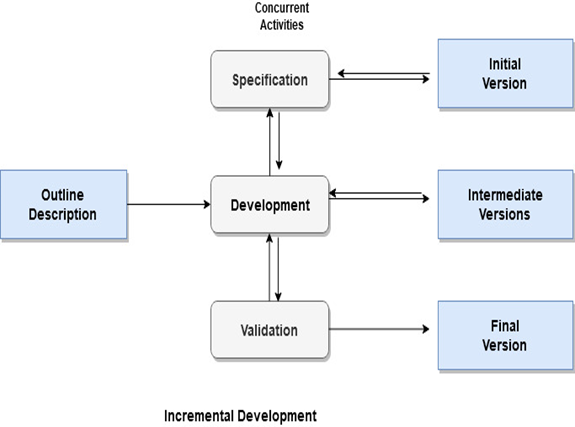
Evrimsel Geliştirme Modeli ilk tam ölçekli modeldir. Modelin başarısı ilk evrimin başarısına bağlıdır. Diğer modellere göre daha ağır işleyen bir yapısı vardır. Gereksinimleri anlamanın kolay olması, hata ve riskin az olması sebebiyle kullanışlı bir modeldir. Herhangi bir değişiklik yapılmasının güç olması, bakımının zor olması, düzenli ürün oluşmaması ise modelin eksileri olarak sıralanabilir.

**Şelale Modeli (Waterfall Model)**



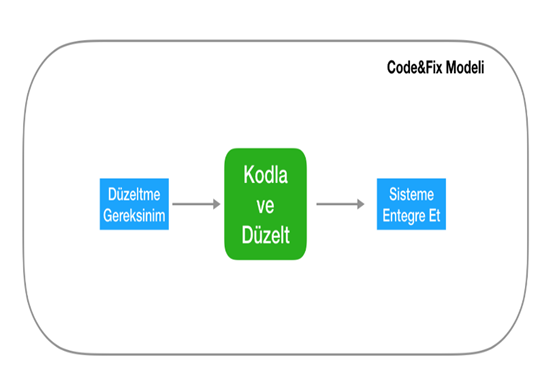
En yaygın kullanılan modellerden biridir. Şelale Model’inde dokümantasyon işlemi oldukça fazladır. Yazılım geliştirme belli aşamalar (yazılım yaşam döngüsündeki temel aşamalar) takip edilerek yapılır. Her bir aşamanın sonucu bir sonrakinin girdisidir ve her aşamada dokümantasyon yapılır. Gereksinimler tanımlanır, sistem ve yazılım tasarımı gerçekleştirilir sonrasında da kodlama ve test etme aşamaları gerçekleştirilir. Yazılım teslim edildikten sonra da bakım işlemi yapılır. Anlaşılması kolay bir modeldir. Gereksinimler iyi tanımlandığı takdirde uzun vadeli olmayan projelerde kullanımı uygundur. Şelale Modeli’nde projelerde müşterilerin gereksinimleri net bir şekilde açıklamaması nedeniyle hataların düzeltilmesi ve eksiklerin giderilmesi gerçekleştirme aşamasına kadar yapılamayabilir. Bu da hem zaman hem de maliyet kaybına yol açmaktadır, modelin en büyük eksisinin bu olduğunu söyleyebiliriz. Geçmişte çok popüler bir model olmakla beraber günümüzde kullanımı giderek azalmıştır.

**Artımlı Geliştirme Modeli**



Proje bir çekirdek(ana) yapının üzerine işlevselliği arttırarak yazılımın tamamlanması üzerine kurulmuştur. Müşterinin belirlediği öncelik sırası ve birbiriyle bağlantılarına göre yazılımlar parça parça tamamlanır. Bu parçalar birbirini kapsayacak şekilde oluşturularak proje tamamlanır. Teslimler de parça parça yapılmaktadır bu sayede projenin bütünü teslim edildikten sonra ortaya çıkabilecek hata ve eksikler de azaltılmış olur. Ayrıca erken teslimler sayesinde müşteri yazılım ürününü yani projenin tamamlanmış halini öngörebilmektedir. Bu sebeplerden ötürü Artımlı Geliştirme Modeli üretim süreci uzun olan ve eksik gereksinimlere rağmen çalışabilen projelerde kullanımı uygun olan bir modeldir.

**Kodla Düzelt Modeli**

Diğer modellere kıyasla daha küçük projeler için uygun olan, sonuç odaklı bir modeldir. Dokümantasyon yapılmadığı için yazılım ürünün bakımı ve hata ayıklaması çok zordur. Yazılımı tamamlamak bu modelde nispeten daha kolaydır ve bu yüzden özellikle küçük firmalarda tercih edilmektedir. Kodla Düzelt Yaşam Döngü Model’inin en büyük dezavantajı ise pahalılığıdır. Bu pahalılığın sebebi ise dokümantasyon olmamasından kaynaklı bakımın oldukça güç olmasıdır.

**Spiral (Helezonik) Model**



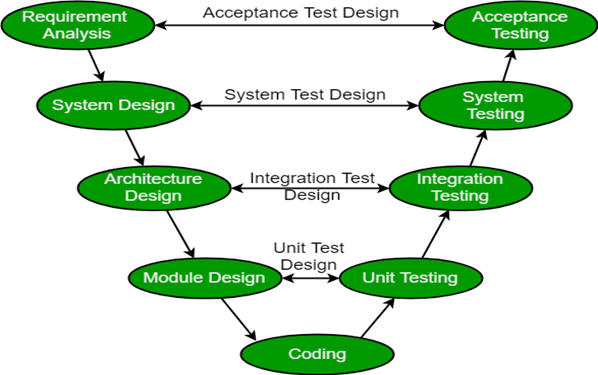
Helezonik Model planlama, risk analizi, üretim ve kullanıcı analizi safhalarının bir döngü, spiral biçiminde uygulanması ile gerçekleştirilir. Planlama aşamasında projenin amacına, alternatiflerine, sınırlarına karar verilir. Risk analizinde alternatifler değerlendirilir ve risk analizi yapılır. Bu modelde risk analizi ön plandadır. Üretim safhasında ürün geliştirilir ve bir sonraki ürün onaylanır. Kullanıcı değerlendirmesinde ise kullanıcı değerlendirmeleri incelenir ve bir sonraki faz planlanır. Gereksinim, analiz, tasarım gibi aşamalar doğrudan gerçekleştirilmez. Prototip\* yaklaşımı vardır. Yinelemeli artımsal bir yaklaşımı vardır.

Helezonik modelde yazılımı kullanacak kişilerin sürece erken dahil olması sebebiyle ileride çözümü de daha zorlaşacak olan hataların engellendiği veya erken fark edildiği bir modeldir. Bu modelin en büyük avantajıdır diyebiliriz. Yazılımın kodlanması ve sınanması da diğer modellere kıyasla nispeten daha erken başlar bu da Helezonik Model’in bir diğer artısıdır.

Helezonik Model oldukça komplike bir yapıya sahiptir. Spiral bazen oldukça uzun zaman devam edebilir bu da dokümantasyonun uzamasına neden olur. Bunu modelin dezavantajı olarak söyleyebiliriz. Helezonik Model küçük ve riski yüksek olmayan projeler için oldukça maliyetli bir modeldir.

\*Prototip: Fiziksel ya da dijital bir ürünün, üretim aşamasından önce bitmiş haline en yakın biçimini öngörebilmek amacıyla yapılan ilk örneği, ilk modelidir.

**V Modeli (V-shaped Model)**



Model aşamaların gerçekleştirilme biçimi “V” harfini andırır. Sol tarafta üretim, sağ tarafta sınama işlemlerinin aşamaları bulunmaktadır. Her üretim aşamasının karşısında bir sınama aşaması olması sayesinde yazılım ürünü oluştururken ortaya çıkabilecek hatalar en aza indirilir.

V Modeli’nin temel çıktıları; kullanıcı modeli, mimari model ve gerçekleştirim modelidir. Kullanıcı modelinde gerçekleştirim esnasında kullanıcı ilişkilerini tanımlanır ve planlar ortaya çıkarılır. Mimari Model’de sistem tasarımı bulunur. Gerçekleştirim Model’inde ise yazılımın kodlanması ve sınanmasına dair işlemler bulunur.

V Modeli takibi ve kullanımı kolay bir modeldir. Modelin en büyük iki eksisi model aşamalarının tekrarının olmaması ve risk çözümlemesinin olmamasıdır. Belirsizliklerin çok olmadığı Bilgi Teknolojileri projeleri için uygun bir modeldir.

**Çevik Yazılım Geliştirme**

1990’lı yılların sonlarına doğru geliştirilen Çevik Yazılım Geliştirme Metotları, hızlı ürün oluşturmayı, müşterinin değişen isteklerine daha etkin ve çabuk yanıt verebilmeyi, projeyi yani yazılım ürününü olabildiğince hızlı bir biçimde teslim edebilmeyi amaçlar.

Bu modelde tüm proje küçük yinelemelere ayrılır ve her bir yineleme başlı başına bir projeymiş gibi üzerinde çalışılır. Her yineleme sonunda proje hakkında müşteriye bilgi verilir bu bağlamda Çevik Yazılım Modelleri için müşteri iletişiminin oldukça yoğun olduğunu söylememiz yerinde olacaktır. Ayrıca bu işleyiş ve iletişim sayesinde hata oranının da epey düşük olduğunu söyleyebiliriz.

2001 yılında Kent Beck ve on altı arkadaşı “Çevik Manifesto” oluşturmuşlardır. Bu manifesto; önceliğin müşteriye hızlı ve sürekli olarak kullanılabilir yazılım teslimatı yapmak olduğunu, kodlamaya geçildiğinde müşteri istekleri doğrultusunda değişiklik yapılabilecek esnekliğin olması gerektiğini ve yazılımı geliştiren ekibin süreç boyunca doğru ve etkin biçimde iletişim kurması gerektiğini belirtir.

Çevik Yazılım’da bu işleyiş ve ilkeleri sayesinde motivasyonu yüksek ekip çalışanları ile beraber hızlı, kaliteli ve düşük maliyetli yazılım ürünleri geliştirilerek müşteri memnuniyeti sağlanır.

En yaygın kullanılan Çevik Yazılım Gerçekleştirme Metotları XP ve SCRUM’dur. Bunların dışında Agile Unfied Process, feature-driven development, Test-driven development, Lean Development, DSDM, MSF gibi çevik yöntemler bulunmaktadır.

**Extreme Programming (XP)**

XP 1999 yılında Kant Beck tarafından geliştirilen bir disiplindir. XP’nin dört temel değeri vardır. Basitlik, iletişim, geri bildirim ve cesaret. Gereksinimleri sağlayan en basit çözüm yöntemlerini kullanmak, ekip üyelerinin birbirleriyle ve müşteriyle sağlıklı iletişim kurmasını sağlamak, sık sık sınama yaparak yahut müşteri ile görüşerek geri bildirim almak ve hata yapmaktan, yeniden başlamaktan korkmadan devam edebilmek XP’nin temel hedefleridir. Bunlar sayesinde XP en ideal yazılım geliştirme metotları arasında yer almaktadır.

Xp yazılım geliştirmeyi olabildiğince kolaylaştırmak ve süreci idealize etmek için on iki uygulamaya sahiptir. Bunlar Planlama Oyunu, Ekipte Müşteri, Önce Test, Basit Tasarım, Çiftli Programlama, Sürekli Entegrasyon, Metafor, Kısa Aralıklı Sürümler, Yeniden Yapılandırma, Ortak Kod Sahiplenme, Kodlama Standardı ve Haftada 40 Saat uygulamalarıdır.

**Scrum Nedir ve Günümüzde Neden Popülerdir?**

Bir çevik yazılım geliştirme modeli olan SCRUM aslında Rugby sporunda tüm oyuncuların beraber rakip takım sahasına saldırdığı bir hücum taktiğinin adıdır. Karmaşık projeler küçük parçalara(sprint) ayrılarak tamamlanır.

Gereksinimler net olarak belirlenemediği, büyük ve karmaşık bir yapısı olan projeler için en ideal yöntemdir. Scrum’ın her bir yinelemeyi tamamlamak için en fazla 30 gün süre verilen ve günlük 15 dakikalık toplantılarla proje işleyişinin daha verimli planlamasını ve sağlayan ve denetleyen bir metodolojisi vardır.

Scrum’ı daha iyi anlayabilmek Scrum’ın için şu üç ana kavramını ve içeriklerini inceleyelim.

1)Roles

Product Owner: Yazılım projesinde geri dönüşlerden sorumlu olan kişi, bir nevi müşteri temsilcisi.

Scrum Master: Ekibin Scrum prensiplerine uygun çalışıp çalışmadığını denetleyen ve kurallara bağlı kalınmasını sağlayan kişi, ekibi Scrum’a adapte eder.

Scrum Team: Sürekli iletişim halinde olan ve Scrum yöntemini kullanan 5-9 kişilik ekip.

2)Meetings

Daily Scrum Meeting: Her gün aynı saatte 15 dakikayı aşmayacak şekilde kısa bir sürede yapılan tüm ekip üyelerinin katılımıyla gerçekleşen günlük toplantılardır. Dün ne yapıldığını, bugün ne yapılacağını ve proje işleyişi konuşulup, engeller aşılmaya çalışılır.

Sprint Planing: Gereksinimlere göre proje küçük parçalara (sprint) ayrılır vetakımlara uygun bir biçimde dağıtılır. Proje ilerleyişi için planlama yapılır. Karşılaşılabilecek riskler belirlenir ve önlem planlaması yapılır. Gerçekleştirme araçları ve altyapısı onaylanıp maliyet hesaplanır.

Sprint Review: Her bir sprint başlangıcında toplantı yapılır. Ürün sahibi ve takım gereksinimleri gözden geçirir. Yazılımın gereksinimlere uygun gerçekleştirilip gerçekleştirilmediği denetlenmiş olur.

3)Artifacts

Product Backlog: Ürün Gereksinim Dokümanı proje boyunca yapılması gerekenlerin listesidir. Doküman zamanla genişleyebilir ekibe yeni üyeler katılabilir yani gün aşırı bile bu dokümanda değişiklik gerçekleşebilir.

Sprint Backlog: Üzerinde çalışılan sprint için product backlog’dan elde edilen iş ve görevlerdir. Sprint Backlog sadece takım tarafından değiştirilir.

Burndown Chart: Üzerinde çalışılan sprint’in ne kadarının tamamlandığını geride tamamlanması gereken ne kadar iş kaldığını gösteren, kıyas yapmamızı sağlayan grafik.

Karmaşık projeleri basit parçalara ayırıp motivasyonu yüksek, cesaretli ve sağlıklı iletişim kurabilen ekip üyeleriyle beraber sistemli bir biçimde yazılım geliştirmeyi sağlayan Scrum; tüm bu özellikleri sayesinde oldukça popüler bir metot. Yapılan toplantılar sayesinde hata oranın düşük olması ve risklerin proje boyunca kontrol altına alınmış olması da bir diğer tercih sebebi. Günümüzde Google, Microsoft, Yahoo!, IBM, IKEA, Siemens, Nokia, BBC gibi birçok şirket Scrum metodunu kullanmaktadır.

CEMRE DOĞAN

Kaynakça

[1] <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>

[2]<https://www.emo.org.tr/ekler/0801491e6797bbd_ek.pdf>

[3]<https://www.roberthalf.com/blog/salaries-and-skills/6-basic-sdlc-methodologies-which-one-is-best>

[4] <https://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>

[5]<https://furkanalniak.com/yazilim-muhendisligi-yazilim-surec-modelleri/>

[6] <https://hackr.io/blog/sdlc-methodologies>

[7]<https://dhalsim.github.io/software/2015/11/14/software-development-processes>

[8]<https://medium.com/architectural-patterns/yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-modelleri-62915545c51e>

[9] <https://enozelders.com/sistem-gelistirme-yasam-dongusu-nedir/>

[10] Doç. Dr. Deniz KILINÇ Yazılım Mühendisliği Temelleri Ders Notları(Hafta 2 ve 3,2022)

[11]<https://softschema.com/incremental-model-of-software-development-life-circle/>

[12]<https://techreceptives.com/development-methodology-spiral>

[13]<https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/>