YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ

Yazılım Yaşam Döngüsü Nedir?

Yazılım yaşam döngüsü (“software development life cycle”, “SDLC”), yazılım mühendisliğinin en temel bilgilerini içerisinde barındırır. Planlı, verimli ve kalitesi yüksek projeler ortaya çıkarmak isteyen her yazılım mühendisi için hayati önem taşıyan bilgilerdir bunlar. Yazılım yaşam döngüsü, doğrusal bir şekilde adımların takip edildiği bir yöntem değildir döngüseldir. Bunun sebebi yazılımların ihtiyaçlarının ve işlevlerinin sürekli değişmesi ve gelişmesidir.



Şimdi yazılım yaşam döngüsü elemanlarını açıklayalım.

0) Anlama:

Bu bir ön aşamadır yazılım yaşam döngüsü için. Bu adımda uygulama veya proje hakkında bilgi toplanır ve gerçekleştirilebilmesinin mümkün olup olmadığı kararlaştırılır. Diğer adımlar için ön eleme aşamasıdır aslında.

1) Planlama:

Yazılım yaşam döngüsünün ilk adımıdır. Projedeki ilk fizibilite testleri yapılır kullanıcı istekleri ve projenin ihtiyaçları belirlenir. Projenin maliyeti, projede yer alan insanların iş bölümü ve projenin her adımı için gereken zamanlama gibi konular bu adımda gerçekleştirilir.

2) Analiz - Çözümleme:

Bu adımda sistem gereksinimleri ve işlevleri ayrıntılı bir şekilde incelenir ve varsa problemler ortaya çıkarılır. Problem çözümleri için çözümler üretilir. Ayrıca bu adımda müşteri, yazılım mühendisi, iş analisti gibi farklı pozisyondaki kişilerle toplantılar yapılarak ihtiyaçların tam olarak anlaşılıp anlaşılmadığı gözlemlenir. Projede çalışan uzman kişiler tüm bu süreci raporlayarak somutlaştırır.

3) Tasarım:

Bu adım tüm proje gereksinimleri tamamlandıktan sonra gerçekleştirilir. Projenin temel yapısı bu adımda oluşturulur mantıksal, fiziksel tasarımı bu adımda gerçekleşir.

4) Uygulama:

Bu adımın anahtar kelimeleri; **kodlama**, **test** ve **kurulumdur.** Tasarım aşaması belli bir seviyeye geldikten sonra “kodlama” aşaması başlar. Kodlamadaki amaç temiz ve okunabilir bir kod oluşturmak olmalıdır.

5) Test:

Bu aşama genel bir sistem testidir, yani sadece bu aşamada test yapıldığı düşünülmemelidir. Her adımın kendine has test aşaması vardır, bu adımdaki test başta da bahsettiğim gibi genel bir sistem testidir bu aşama da tamamlandıktan sonra proje müşteriye teslim edilmeye hazır hale gelir.

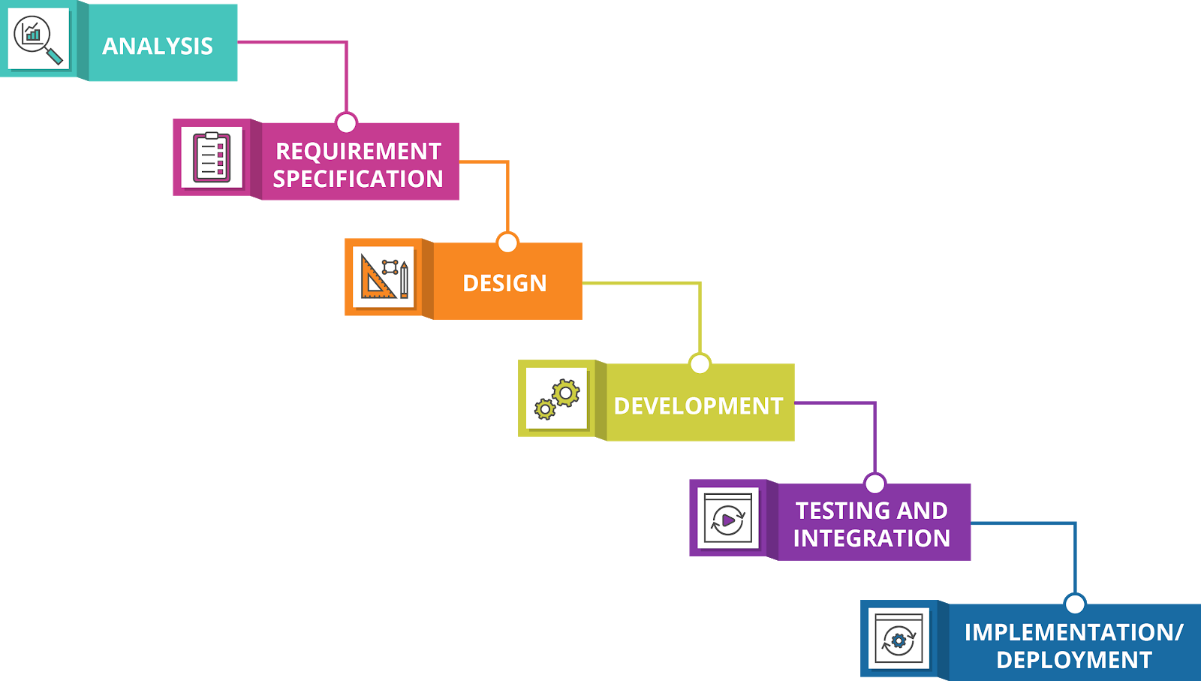
6) Bakım:

Bakım aşaması yazılımın ömrü boyunca devam eder. **“En iyi tamir bakımdır”** misali uygulamanın bakımları ne kadar özverili bir şekilde yapılırsa kullanıcı deneyimi de o kadar iyi olacaktır. Bakımdan kastımız da “Hata giderme, eklenti yapmak vs.” gibi şeylerdir.

Şu ana kadar “yazılım yaşam döngüsü nedir?”, “yazılım yaşam döngüsünün temel faktörleri nelerdir?” gibi sorulara cevap verdik, şimdi de yazılım yaşam döngüsü süreçlerinin nasıl ve ne şekilde uygulanacağını tanımlayan **“yazılım yaşam döngüsü modelleri”** ‘ne göz atalım.

Yazılım Yaşam Döngüsü Modelleri:

* Şelale Modeli (Waterfall Model):



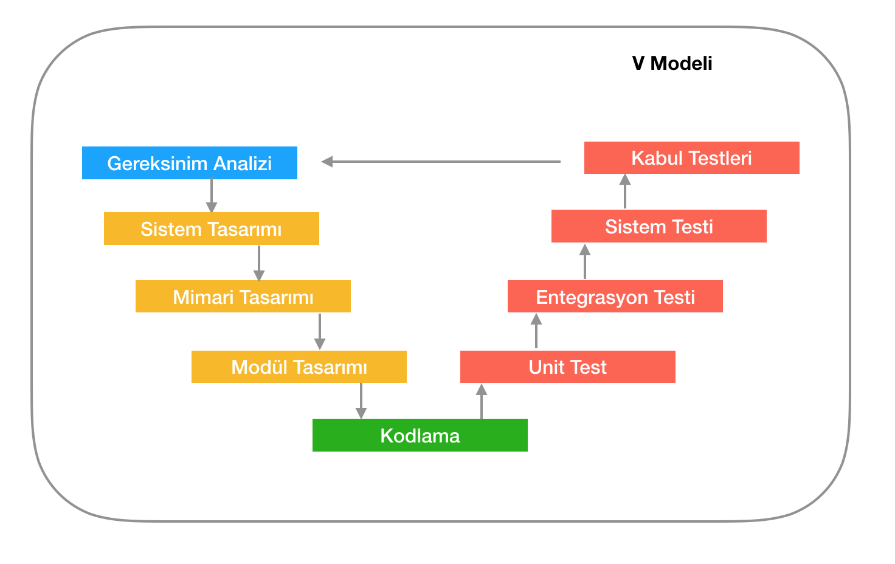
Şelale yöntemi, sıralı basamaklardan oluşur. Bu modelde bir sonraki adıma geçebilmenin koşulu bir önceki adımdaki gereksinimlerin tamamlanmasıdır. Her adım sonrası rapor oluşturulur. İlk adım analiz aşamasıdır, tüm yazılım gereksinimleri net ve ayrıntılı bir şekilde belirlenerek dokümanı oluşturulur. Tasarım adımında yazılımın arayüz, veritabanı vb. tasarımları yapılır ve dokümanı oluşturulur. Sonraki basamakta yazılım kodlanmaya başlanır. Kodlama; tasarım ve analiz aşamasındaki verilere göre yapılır. Test aşamasında, analiz ve tasarım aşamasındaki tüm gereksinimler için test senaryoları yazılır ve bu test senaryoları gerçekleştirilip sistem test edilir. Eğer testler sonucu bir hata ile karşılaşılırsa hata hangi adımda ise o adıma dönülerek hata giderilir ve dokümanlar güncellenir. Hata kodlama aşamasında ise maliyet görece küçüktür ancak hata analiz veya tasarım aşamasında çıkarsa maliyet de artar. Tasarım veya analiz kısmı değiştirileceği için yeniden kodlama yapılması gerekir. Hatalar giderildikten sonra sistem tekrar test edilir eğer hata ile karşılaşılmazsa yazılım müşterinin kullanımına açılır. Şelale modelinin Barok modelinden farkı, proje içindeki dokümantasyonu ayrı bir süreç olarak değil üretimin doğal bir parçası olarak ele alır. Şelale modeli, analiz ve tasarımın çok önemli olduğu ve bu unsurlara uzun zaman ayırmayı gerektirecek projelerde kullanılır. Örneğin askeri projeler. Bu tarz projelerde analiz veya tasarım aşamasında hata yapmak çok yüksek maliyet yaratacaktır. Şelale modeli, oldukça detaylı analiz ve tasarım aşamalarına sahip olduğu için yazılımın gereksinim ve tasarımı net bir şekilde ortaya konur bu sayede uzun analiz ve tasarım sürecine karşın oldukça kısa kodlama ve test aşamalarına sahiptir. Test aşamasında çıkan hata sayısı da azdır. Şelale modelinde, üst basamaklarda (analiz, tasarım vs.) yapılan hatalar çok büyük zaman ve para kaybına yol açmaktadır. Bu modelin diğer bir dezavantajı ise tüm aşamalar bitmeden ürünün ortaya çıkamamasıdır. Müşteri bir prototip isterse ürünün tamamlanmasını beklemek zorunda kalacaktır ve müşteri sabırsız ise bu oldukça yıpratıcı bir sürece dönüşecektir müşteri ve proje ekibi açısından.

* Barok Modeli:



1970’li yıllarda ortaya çıkmıştır. Yazılım yaşam döngüsü doğasına aykırı bir şekilde bu modelde doğrusal olarak tanımlanmıştır. Aşamalar arası geri dönüşlerin nasıl yapılacağı belli değildir. Barok modelinde dokümantasyon yazılımdan ayrı bir süreç olarak ele alınır. Yazılımın geliştirme ve test faaliyetleri tamamlandıktan sonra yapılır. Günümüzde kullanılan bir model değildir artık.

* V Modeli:

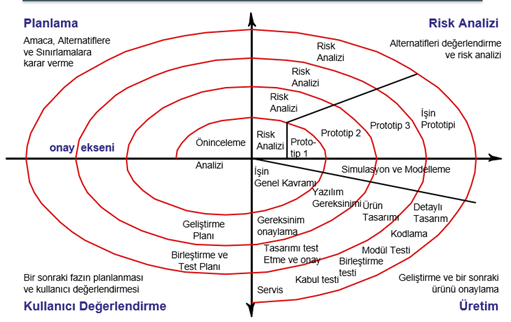


V modelinin sol tarafı **üretim,** sağ tarafı ise **test** işlemleridir. Model, sınama işlemlerinde hata bulma durumunda nereye dönüleceğini de belirtmektedir. Sağ tarafta yapılan bir hata yatay olarak solundaki tasarımın düzeltilmesi gerektiğini belirtir. Örneğin, sistem testinde hata varsa sistem tasarımına dönülür. Bilgi teknolojileri projeleri için uygun bir modeldir. Şelale modelindeki; maliyetli geri dönüşler, ekibin projeye dahil olmasının gecikmesi vb. gibi problemlerin V modelinde de görülmesi V modelinin dezavantajları arasındadır. V modelinde aşamaların nasıl test edileceği belirgin bir şekilde gösterildiği için hataların erken ve kolay bir şekilde fark edilmesi sağlanmıştır.

* V Modeli Çıktıları:

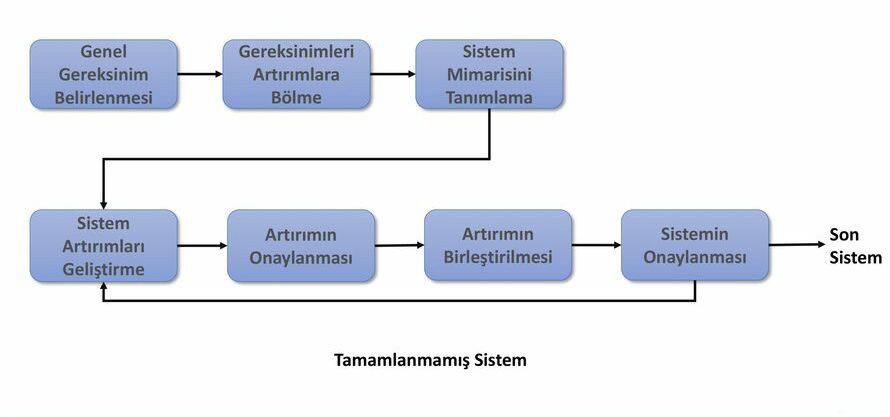
**Kullanıcı Modeli:** Geliştirme sürecinin kullanıcı ile ilgili olan kısmını tanımlar. Üretim ve test kısmı tarafında kullanıcıdan beklenenleri ortaya koyar. **Mimari Model:** Sistem tasarımı, oluşacak alt sistem ve tüm sistemlerin sınama işlemlerine ait işlevleri içerir. **Gerçekleştirim Modeli:** Yazılım modüllerinin kodlanması ve sınanmasına ilişkin işlevleri içerir.

* Helezonik (Spiral) Model:



Bu modelde kullanıcı katkısı en ön plandadır, şekilde de görüleceği üzere ürün her turda kullanıcı değerlendirmesine sunulur. Yöneticiler, proje boyunca prototipler sayesinde çalışan yazılımları gördükleri için daha kolay proje takibi ve daha kolay maaş planlaması yaparlar. Prototipler sayesinde yazılımın kodlanması ve sınanması daha erken başlar ve geliştirici ekip için bir motivasyon kaynağı olur. Şelale modeline göre somut bir proje oluşumu gözlemlemek prototipler sayesinde helezonik modelde daha hızlı gerçekleşir. Helezonik (Spiral) Model, artımsal yinelemeli bir yaklaşımla oluşturulmuştur. Yani her spiral turunda bir aşama kaydedilir ve öncekinin üzerine koyulur. Bu modelde prototip yaklaşımı mevcuttur. Helezonik model, belirsiz iş ihtiyaçları olan projeler, inovatif ihtiyaçları olan projeler, büyük ve karmaşık projeler, Ar-ge faaliyeti gibi büyük çaplı projelerde tercih edilir.

* Artımsal Geliştirme Süreç Modeli:

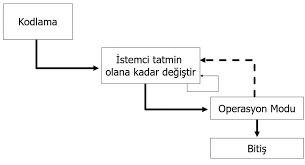


Modelde üretilen ve uygulamaya alınan ürün sürümleri birbirini kapsayacak şekilde giderek artan bir sayıda geliştirilir. İlk olarak ana sürüm geliştirilir uygulamaya alınır daha sonra ana sürüme yeni eklentiler eklenerek ana sürümün zamanla işlevselliği arttırılır. Prototip mantığı bakımından **“helezon modeli”** ile benzerlik gösterir. Ürün parça parça yayınlanıp müşteriden dönüt alındığı için projenin batma riski azalır. Öncelikli gereksinimleri karşılayan sistem işlevleri daha çok test edilir. Bu modelin dezavantajları ise; deneyimli ekip gerektirmesi, iyi proje tanımlamasına ihtiyaç duyması denebilir. Son olarak artımsal geliştirme süreç modeli, uzun veya kısa vadeli fark etmeksizin her tarz proje için uygundur projeler için uygundur.

* Big Bang Modeli:

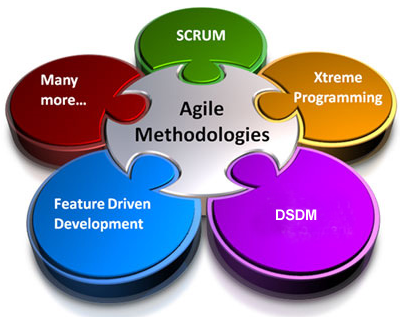
Belirli bir plan programın çok az olduğu büyük çoğunlukla kodlamaya odaklanılan bir yazılım geliştirme modelidir. Küçük çaplı proje gruplarının tercih ettiği bir yöntemdir. Akademik yazılım geliştirme projeleri için kullanılabilir.

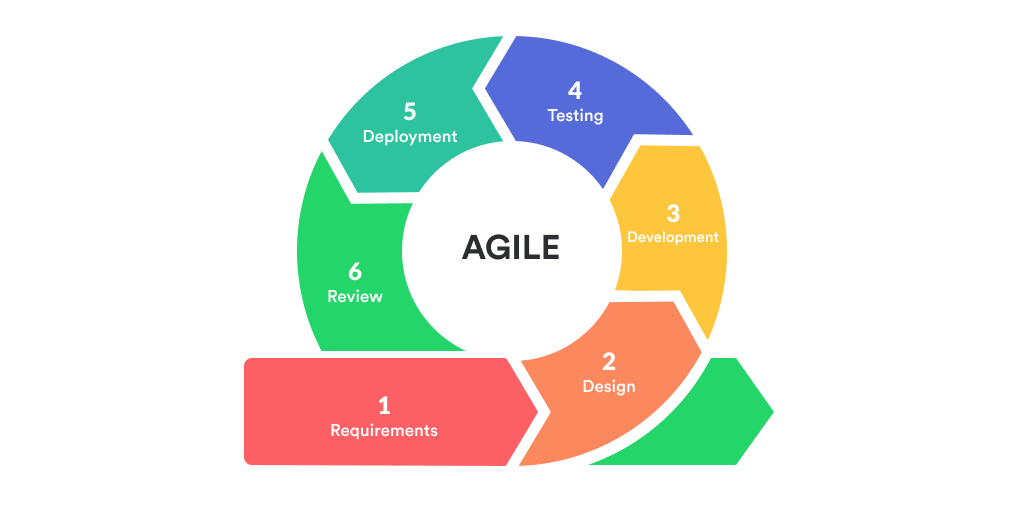
* Kodla ve Düzelt Modeli:



Plan ve program kısmı kodla ve düzelt modeli için arka plandadır. Bu modelde önemli olan ve odaklanılan kısım kodlama kısmıdır. Bu yüzden direkt olarak yazılım ürünü gerçekleştirilir. Ürünün ilk safhasında bir prototip oluşturulur ve müşteri ile temasa geçilerek müşteri dönütleriyle ürün geliştirilmeye devam edilir. Bu yönüyle “kodla ve düzelt” modeli “helezon modeli” ile benzerlik gösterir Yazılım geliştirmenin en kolay yoludur ancak en pahalısıdır. Sürekli yapılan işlerde değişiklik yapmak gerekir. Dokümantasyon olmadığı için bakım safhası çok zordur. Yazılımı kodlamak kolay olduğu için küçük firmalardaki küçük işler için tercih edilir genellikle.

* Agile (Çevik) Yazılım Geliştirme:



Agile (Çevik) yazılım geliştirme, içinde farklı yazılım geliştirme metotlarını barındıran bir metotlar topluluğudur. Çevik yazılım geliştirme metotlarının amacı; piyasaya hızlı bir şekilde ürün sunabilmek, değişen istek ve ihtiyaçlara hızlı yanıtlar verebilmek ve en kısa sürede ürünü müşteri ile buluşturabilmektir. Çevik yazılım geliştirme metotlarının ortak anahtar kelimesi hız ve esnekliktir. Çevik yazılım geliştirme metotlarının temel prensipleri aynı olsa da pratikte uygulanmaları bakımından farklılaşmışlardır. Çevik yazılım geliştirme metodolojisinde **böl ve fethet** **(divide and conqure)** işler yürütülür. Projenin ölçeği ne olursa olsun proje küçük parçalara ayrılır **– bu parçalara “yineleme (iteration)” adı verilir –** ve her bir parça başlı başına bir projeymiş gibi tanımlanarak geliştirilir. Her “yineleme” sonunda müşteriye projenin durumu hakkında bilgi verilir. Ekip üyelerinin sürekli birbiri ile iletişim halinde olmaları, proje küçük parçalara ayrıldığından dolayı oluşan hataların düzeltilmesinin kolaylığı çevik yazılım geliştirme modellerinin avantajları arasındadır.

* Çevik Yazılım Geliştirme Manifestosu:

2001 yılında Kent Beck ve arkadaşları tarafından oluşturulmuştur. Manifestoda müşteri memnuniyetinin önemi belirtilmiştir. Değişen gereksinimler yazılımın son adımında dahi olsa kabul edilebilmelidir. Çalışan yazılım kısa zaman aralıkları belirlenerek birkaç ay ya da haftada bir müşteriye sunulmalıdır. İş sürecinin sahipleri ve yazılımcılar her gün birlikte çalışmaları gerekir. Projede çalışan insanların moral ve motivasyonu için gerekli ortam şartları sağlanmalı ve onlara güven duyulmalıdır. Manifestoda bilgi aktarımının en iyi yolunun yüzyüze iletişimden geçtiği belirtilmiştir. Sadelik ve basitlik olmazsa olmaz niteliklerdendir. Sonuç olarak çevik yazılım geliştirme manifestosunun öne çıkan anahtar kelimeleri ise şöyledir; **hız, esneklik, iletişim, güven, sadelik, süreklilik.**

* Çevik Yazılım Geliştirmenin Avantajları:

Üretkenliğin artması, maliyetlerin düşmesi, kısa sürede müşteri memnuniyetinin sağlanması, yüksek motivasyonlu yazılım ekiplerinin oluşması.

Çevik (Agile) yazılım geliştirme yöntemlerinden en çok kullanılanlardan biri olan **SCRUM** metodunu inceleyelim şimdi de.

* SCRUM:

SCRUM, bir proje yönetim modelidir ve çevik yazılım geliştirme metodlarıyla birlikte uygulanır. Bu özelliği sayesinde her konuda SCRUM yöntemi kullanılabilir, sadece yazılım ürünlerinde kullanılabilir demek yanlış olur. SCRUM, karmaşık yazılım işlerini daha küçük birimlere (sprint) bölerek sonuca ulaşmayı hedefler. SCRUM modeli karmaşık ve büyük projelerde yazılım geliştiren ekipler için oldukça ideal bir yöntemdir. Çevik yazılım geliştirme sürecinin özelliklerini taşıdığı için SCRUM, karmaşa yaşanan projeler için akılcı çözümler geliştirir. Bu modelde bir yinelemenin (sprint) tamamlanması 30 günden fazla sürmemektedir. Ayrıca günlük olarak 15 dk’lik kısa toplantılarla iş takibi yapılmaktadır. SCRUM modelinin olmazsa olmaz yöntemlerinden biri ise iletişimdir. Yazılımın her bir adımında farklı pozisyonlardaki insanlarla temas kurarak projenin sorunları giderilmeye çalışılır. Örneklendirmek gerekirse; **Product backlog**, müşteri ile yapılan görüşme sonucu oluşturulan ihtiyaç listesidir. **Sprint backlog,** maksimum 30 günlük aralıklarla yapılan yazılımın ilerleme durumunun tartışıldığı toplantılardır. **SCRUM Daily meeting,** Her gün SCRUM takımıyla projenin genel gidişatıyla ilgili konuların konuşulduğu kısa toplantılardır.

* Peki SCRUM Günümüzde Neden Popüler?

SCRUM, sosyalliği ve paylaşmayı öne çıkardığı için popülerdir. Proje ekipleri yoğun iş temposu altında çalışırken moralleri ve motivasyonları zamanla düşecektir ki bu durum da verimin düşmesine neden olacaktır. SCRUM metodolojisi sayesinde yönetim birimi ile proje ekipleri, proje ekipleriyle müşteriler arasındaki iletişim bağı her daim güçlü kalacağı için tüm ekip monoton olmaktan çıkmış bir çalışma düzeniyle ürün gelişimini sürdürecektir. Sonuç olarak ortaya kaliteli bir iş çıkacaktır ve öte yandan geliştirici ekip yüksek verim ve motivasyonla proje üzerinde çalıştıkları için moral bozuklukları ve stres asgari düzeye inecektir, müşteri tarafında ise kaliteli ürüne hızlı bir şekilde ulaşılması memnuniyet yaratacaktır. Bu avantajları göz önüne alındığında benim de gözde çalışma modelimin ilerleyen yıllarda SCRUM olması kaçınılmazdır.

Kaynakça:

<https://osmanozaydin.com/yazilim-yasam-dongusu-ve-agile-yazilim-gelistirme/>

<https://furkanalniak.com/yazilim-muhendisligi-yazilim-yasam-donguleri/>

<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://www.ykcgrup.com.tr/selale-modeli-nedir/>

<http://w3.bilecik.edu.tr/bilgisayar/wp-content/uploads/sites/75/2019/02/ders-2.pdf>

<https://muhammetbaykara.com/2017/02/16/ymh-114-2-hafta/>

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

<https://medium.com/@ryoldash/agile-%C3%A7evik-yaz%C4%B1l%C4%B1m-geli%C5%9Ftirme-nedir-ve-nas%C4%B1l-uygulan%C4%B1r-93e85ffc866>

Doç. Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş Dersi 2. ve 3. Hafta Sunumları

Medium Makale Linki:

<https://tunaaydin6161.medium.com/yazilim-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-modelleri%CC%87-e0ad6f0e743a>

**Bahadır Tuna AYDIN**

**200601002**