YAZILIM YAŞAM DÖNGÜ MODELLERİ

Begüm Keleş - 220601054

İzmir Bakırçay Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, İzmir, Türkiye

Özet – Bu makale; SDLC (Software Development Life Cycle) olarak da bilinen yazılım geliştirme yaşam döngü modelinin ne olduğu, hangi modelleri kapsadığını ve yazılım geliştirme modellerinin kendi içerisinde kıyaslanmasını içerir. Yazılım yaşam döngü modellerinin karşılaştırılmasının yanında çevik yazılım geliştirme metodolojisi olan SCRUM metodolojisinin günümüzdeki popülerliği hakkında inceleme yapılmıştır.

1. Yazılım Yaşam Döngü Modeli Nedir?

Herhangi bir yazılımın geliştirilmesi ve bakım aşamasında icra edilen aşamalar topluluğuna yaşam döngüsü denir. Yazılım işlevleri ve ihtiyaçları devamlı değişip geliştiğinden mutlaka bir döngü halinde düşünülmelidir. Doğrusal yani tek yönlü düşünülmemelidir. Yazılım projelerinde bazı başarısızlıklar kendini ön plana çıkarmaktadır. Bu başarısızlıklar başlıca şunlardır:

- Müşteri isteklerinin analizinin doğru yapılmaması,
- Maliyet hesaplarında meydana gelen eksiklikler,
- Ekiplerin görev tanımlarının belirsiz olması.

Bunlar gibi durumların olabildiğince az yaşanması, tekrarlanmaması hatta mümkünse hiç yaşanmaması adına yazılım geliştirme yaşam döngü modelleri kullanılır ve birden fazla modele sahiptir. Şelale model (waterfall model), v-şeklinde model (v-shaped model), artırımlı model (incremental model) ve spiral model yazılım yaşam döngü modellerinden bazılarıdır. Yazılım yaşam döngü modeli aşamalardan oluşur. Bu aşamalar kaynaktan kaynağa 5 ile 10 arasında değişebilir fakat temelde her kaynakta aynı olan 5 aşama vardır. Bu aşamalar; gereksinim (requirements), analiz (analysis) – çözümleme, tasarım (design), gerçekleştirme (implementation) ve bakım (maintenance) aşamalarıdır.

2. Yazılım Yaşam Döngü Aşamaları

2.1. Gereksinim (Requirements)- Gereksinim aşaması yazılım yaşam döngüsünün en önemli aşamasıdır ve bir binanın temeli gibi düşünülebilir. Başta müşteri olmak üzere yapılan pazar araştırmalarından, uzmanlardan alınan girdiler kullanılarak projenin başarılı bir şekilde hayata geçirilmesi için izlenecek yaklaşımlar tanımlanır ve ilk adım planlanır. Bu aşamaya planlama aşaması da denilir. Gereksinim aşaması atlanmış bir projenin başarılı olma ihtimali çok düşüktür.

2.2. Analiz (Analysis) – Çözümleme- Gereksinim aşamasından sonra yaşam döngüsünün en önemli adımıdır. İlk aşamada planlanan gereksinimler net bir şekilde tanımlanır. Projenin yazılım işlevleri ve gereksinimleri detaylıca ortaya çıkarılır.

2.3. Tasarım (Design)- Analiz aşamasından sonra ortaya çıkan detaylar göz önüne alınarak belirlenen gereksinimleri karşılayacak yazılım ya da bilgi sisteminin temel yapısını oluşturma sürecidir. Bu aşamada hazırlanan tasarımlar inşaat projeleri gibi düşünülebilir. Ürünün

tasarımı için birden fazla yaklaşım önerilir ve bir tasarım belgesi oluşturulur. Oluşturulan tasarım belgesi gözden geçirilerek değerlendirmeler sonucu ürün için en iyi tasarım seçilir. Bu çalışmalar mantıksal tasarım ve fiziksel tasarım olarak iki gruba ayrılır.

- Mantıksal Tasarım: Önerilen sistemin yapısı anlatılır.
- Fiziksel Tasarım: Yazılımı içeren bileşenler ve bunların ayrıntılarını içerir.

2.4. Gerçekleştirme (Implementation)- Bu aşama tasarımı tamamlanan bir projenin somutlaşmaya başladığı adım olarak düşünülebilir. Proje verilen kararlara uygun olarak gerçekleştirilmeye başlar. Gerçekleştirme aşaması yazılım projeleri için kodlama olarak düşünülebilir veya tasarımı tamamlanmış bir binanın inşaat aşaması olarak düşünülebilir.

2.5. Bakım (Maintenance)- Proje geliştirilmesi tamamlandıktan sonra oluşabilecek hataların giderilmesi, yazılım iyileştirilmesi ve geliştirilmesinin yapıldığı süreçtir.

Yazılım veya oluşturulan herhangi bir sistem canlı bir organizma gibidir. Belli bir yaşam süreci vardır. Var olur, gelişir ve sonunda ölür. Yazılım ise üretilir ve bu süre zarfında ufak müdahalelerle uzun yıllar boyunca yaşaması beklenir. Tam bu noktada yazılım yaşam döngüsü ortaya çıkar. SDLC'nin bakım aşamasından sonra döngü sona ermez ve yazılım projesi işe yaramayacak noktaya gelinceye kadar döngü tekrarlanır. Su döngüsünde nasıl ilk önce su buharlaşır, bulut kümeleri oluşur, yeryüzüne su damlacıkları halinde döner, su birikir ve tekrar buharlaşırsa yaşam döngüsünde de buna benzer bir tekrarlama vardır. SDLC var olan projede düzenlemelerinin nasıl yapılacağının ve izlenilecek yolun en büyük yardımcısıdır.

3. Yazılım Yaşam Döngüsünün Bazı Modelleri

3.1. Çağlayan – Şelale Model (Waterfall Model) – Geleneksel yazılım geliştirme modeli olarak bilinen *Çağlayan Modelinde* aşamalar en az birer kez tekrarlanır. Tanımı iyi yapılmış projeler ve üretimi az zaman gerektiren yazılım projeleri için uygun bir modeldir. Belgeleme işlemi sistemin doğal bir parçası olarak görülür ve ayrı olarak ele alınmaz. Her aşama sonunda bir belge oluşturulur. Bir aşama tamamlanmadan diğerine geçilemez. Gelişen fikirlere tam olarak uyum sağlayamaz. Günümüzde yazılım alanında gelişmeler çok sık yaşandığından çağlayan modeli yazılım projeleri için en kötü yaklaşımlardan biridir. Fakat diğer modellere öncelik ettiğinden ve uygulanması basit olması açısından önemli bir modeldir.

3.1.1. Çağlayan – Şelale Model Aşamaları

- *Gereksinim Tanımlama* – Planlama aşamasıdır. Gerçekleştirilecek projenin gereksinimleri belirlenir, analizler yapılır. Analizler genelde durum analizi ve istek analizleri olmak üzere ikiye ayrılır.
- *Sistem ve Yazılım Tasarımı* – Projenin gerekli analizleri tamamlandıktan sonra detay ve yapısal tasarımı oluşturulur.
- *Gerçekleştirme ve Birim Test* – Tasarımı tamamlanmış olan bir yazılım projesinin kodlanma aşaması olarak düşünülebilir. Bu aşama aynı zamanda geliştirme aşamasıdır ve birden fazla alt aşaması vardır. Bu aşamada oluşturulan testler projedeki olası tasarım ve analiz hatalarına geri dönülmesi açısından önem taşımaktadır.
- *Birleştirme ve Sistem Testi* – Projenin tamamının test edildiği aşamadır. Sonrasında ürün müşteriye teslim edilir.
- *İşlem ve Bakım* – Ürün müşteriye teslim edildikten sonra değişen ihtiyaçlara ek ve değişen müşteri taleplerine göre güncelleme işidir.

Çağlayan modeli aşamalarının birinde hata olduğunda önceki aşamaya dönülemez. Tekrardan planlama yapmak gerekir. Bu durum da büyük projeler için oldukça zor ve uzun bir süreçtir. Bu yüzden çağlayan modelin büyük projelerde kullanılmaması gerekir. Daha küçük projelerde uygulanması basit bir model olduğundan projelerin hazırlanmasını kolaylaştıracaktır.

3.1.2. Çağlayan – Şelale Model Avantajları

Basit bir planlamaya sahip olduğundan müşteriler ve kullanıcılar tarafından anlaşılabilir aşamalardan oluşur. Maliyeti diğer geliştirilen modellere göre oldukça düşüktür. Ayrıca proje yöneticileri için görev dağılımını kolaylaştırır. Planlama adımı tamamlandıktan sonra sağlam bir temel oluşur. Tekrarlamalar önceki ve sonraki adımlarla gerçekleşir. Kalite gereksinimlerinin zamana ve bütçeye göre ön planda olduğu projelerde çok iyi çalışır.

3.1.3. Çağlayan – Şelale Model Dezavantajları

Yazılım geliştirme ve üretim ekipleri genelde kodlama aşamasına önem verdiğinden diğer aşamalar ideal şekilde tamamlanamaz. Çağlayan modelinin en önemli aşaması olan gereksinim (planlama) aşaması çoğu kez net şekilde tamamlanmadığından yanlışların düzeltime ve eksiklerin giderilme maliyeti yüksektir. Bir aşama tamamlanmadan diğerine geçilmemesi olası riskleri artırır. Bir ya da iki önceki aşamalara gitmek oldukça maliyetlidir. Kullanıcı süreç içerisinde yer almaz, yazılım süreci tamamlandıktan sonra geri dönüşleri arttırılabilir. Yazılımın son kullanıcıya ulaşma zamanı uzundur. Çağlayan modelinde geriye dönülememe sorunu özellikle değişken yazılım projeleri için oldukça büyük bir problem olduğundan diğer modellerin geliştirilmesine sebep olmuştur.

3.2. V-Şeklinde Model (V-Shaped Model) – V-Şeklinde model, çağlayan modelinin gelişmiş hali olarak düşünülebilir. Çağlayan modeli gibi doğrusal yönde ilerlemek yerine süreç adımlarının kodlama evresinden sonra yukarıya doğru eğim alarak V şeklini oluşturmasıyla çağlayan modelinden ayrılır. Kodlama evresinden sonra test evreleri başlar. Buna dayanarak modelin sol tarafı için üretim, sağ tarafı için test işlemleri denilebilir. Kullanıcıların katkısı bu modelde çağlayan modele göre daha fazladır. Belirsizliklerden oluşmayan veya az belirsizlik içeren, iş tanımlarının belirgin olduğu projeler için uygun bir modeldir.

3.2.1. V-Şeklinde Modelin Temel Çıktıları

- *Kullanıcı Modeli* – Sistem tanımlamaları ve tamamlanan sistem sınaması bu aşamadadır. Geliştirme sürecindeki projenin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin kabulüne ilişkin testleri, planları ortaya çıkarılmaktadır.
- *Mimari Model* – Sistem ve alt sistem aşamalarını içerir. Sistem tasarımı bu bölümde gerçekleşir. Ayrıca oluşacak alt sistem ile tüm sistemin sınama işlemleri de yine bu bölümde gerçekleşir.
- *Gerçekleştirim Modeli* – Geliştirilen projenin yazılım modülleri bu aşamada kodlanır ve yine testleri bu aşamada gerçekleşir.

3.2.2. V-Şeklinde Model Avantajları

Kullanıcının projeye katkısını arttıran bu model aynı zamanda proje yönetim ve takibini de kolaylaştıran bir modeldir. Kullanımı anlaşılır ve kolaydır. Doğrulama işlemi sadece son üründe değil tüm teslim edilebilir ürünlerde uygulanır. BT için uygundur.

3.2.3. V-Şeklinde Model Dezavantajları

V-Şeklinde modelin çağlayan modelinin test kısmının standartlaştırılmış bir türü olduğu düşünülmektedir ve yeni bir şey geliştirmediği öne sürülmektedir. Çağlayan modelinde görülen geri dönüşlerin maliyetli olması, bazı önemli ihtiyaçların sonraki aşamalarda belli olması gibi problemler v-şeklinde modelde de kendini gösterir. Aşamalar arasında tekrarlama yoktur ve risk çözümleme aktivitelerini içermez. Aynı anda gerçekleştirilebilecek olaylara kolay imkân tanımaz. Herhangi bir aşamada değişiklik yapılırsa test belgeleri ve diğer tüm belgeler güncellenmelidir.

3.3. Spiral Model – Bazı kaynaklarla *Helezonik Model* olarak da geçen bu yaşam döngü modeli; planlama, risk analizi, üretim ve kullanıcı değerlendirilmesi bölümlerinden oluşur. İsmini spiral şeklinden almıştır. Diğer modellerden farkı her aşamadan tekrar tekrar geçilebilmesi ve her geçişte projenin gelişmesini hedeflemesidir. Risk analizi olgusu ön plandadır ve her döngü bir aşamayı ifade eder. Hedefler, alternatifler ve kısıtlamalar belirlenir. Alternatifler değerlendirilir ve kısıtlamalar çözülür. Aşama ürünü geliştirilir. Sonraki aşama planlanır. Doğrudan gereksinim, tasarım aşamaları yoktur. Yinelemeli artımsal ve prototip bir yaklaşım vardır. Süreç sıralı bir şekilde değil spiral şeklindeki döngü halinde ilerler. Her bir halka bir aşamayı ifade eder. Yazılım mühendisliğindeki spiral modelden ilk olarak Barry Boehm 1986 tarihli makalesinde bahsetmiştir. Karmaşık yapılı büyük projeler ve risk, maliyet değerlendirmesi önemli olduğunda uygun bir modeldir fakat küçük projeler için maliyetli olur.

3.3.1. Spiral Model Temel Aşamaları

- *Planlama* – Her aşamada olan ara ürün için gereksinimler belirlenip planlama yapılır.
- *Risk Analizi* – Riskler araştırılır, belirlenir ve çözümlemesi yapılır.
- *Üretim* – Ürünün veya ara ürünün üretimi yapılır.
- *Kullanıcı Değerlendirilmesi* – Oluşturulan ürün/ara ürün kullanıcıya sunularak geri dönütler alınır ve sonraki aşamaya geçilir.

3.3.2. Spiral Model Avantajları

Geliştirme aşaması parçalar halinde hızlıdır ve riskli aşamalar en önce gerçekleştirilir. Riske duyarlı yaklaşımı sayesinde potansiyel zorluklar aşılabilir. Kullanıcılar sistemi erkenden görebilir. Sonraki aşamalarda ek işlevler veya değişiklikler yapılabilir. Olası hataları erken gidermeye odaklıdır. Yazılım kodlanması ve test edilmesi erkenden başlar.

3.3.3. Spiral Model Dezavantajları

Karmaşıktır ve döngü sonsuza gidebilir. Küçük ve basit projeler için oldukça maliyetlidir. Kontrat tabanlı yazılımlar adım adım geliştirmeye uymadığından spiral model kontrat tabanlı yazılımlar için uygun değildir. Ara adımlar çok fazladır bu sebeple çok belge gerektirir. Risk uzmanlığı gerektirir.

3.4. Artırımlı Model (Incremental Model) – Proje bir bütün değildir. Geliştirme ve teslim parçalarından oluşur. Kullanıcı gereksinimleri önceliklendirilir. Üretilen her yazılım sürümü birbirini kapsar ve işlevselliği giderek artar. Gereksinimler mantıksal olarak alt sınıflara ayrılmıştır. Her sınıf küçük çağlayan modeline benzer. Ürünün ortaya çıkması bu küçük çağlayan modellerinin birleşmesi ile oluşur. Her alt sınıf kendi içinde kolay yönetilen modüllerdir. İlk sınıf oluştuğunda aynı zamanda çalışan ilk yazılım da elde edilir. Bu yüzden

ilk aşamadan itibaren çalışan bir yazılıma sahip olunur. Her artımda (sınıfta) mevcut yazılım üstüne yeni işlevler eklenir ve planlanan ürün ortaya çıkana kadar bu süreç devam eder. Güncelleme gerektiren büyük projeler için uygun bir modeldir.

3.4.1. Artırımlı Modelin Avantajları

Müşteriler aktiftir. Proje için gerekli olan gereksinimlerde müşteriler rol oynar. Gereksinimlerin önemine göre artımlar belirlenir. Daha fazla test imkânı vardır ve önceki aşamalara dönmek daha az masraflı olduğundan büyük projelerde bu model kullanılabilir. Böl ve Yönet yaklaşımı vardır. Projenin başarısızlıkla sonuçlanma ihtimali düşüktür.

3.4.2. Artırımlı Modelin Dezavantajları

Ekibi yönetebilmek için deneyimli kişiler gereklidir. Her aşama kendi içerisinde tekrarlanamaz. Gereksinim atamaları artımlara doğru şekilde yapılamayabilir. Geliştirme başlamadan önce tüm sistem net bir şekilde tanımlanmalıdır. İyi planlama ve tasarım gereklidir. Toplam maliyet çağlayan modelinden yüksektir.

4. Çevik Model Nedir?

2000li yıllarda çoğalan yazılım şirketleriyle birlikte şirketler arasındaki rekabet artmıştır. Buna bağlı olarak oluşturulan yazılım projelerini geliştirme ihtiyacı hızlanmıştır. Yazılım sürümlerinin zamanında ortaya çıkarılamaması, değişiklik isteklerine hızlı cevap verilememesi, yazılım hatalarının geç fark edilmesi ve yazılım programlarının kendi yapısını geliştirememesi sebebiyle çalışmalar yapılmış ve "Çevik (Agile)" olarak isimlendirilen metotlar geliştirilmiştir. Çevik geliştirme modeli bir artımlı modeldir. Yazılımın artımlı, hızlı çevrimler ile geliştirilmesi hedeflenir. Hızlı olarak ürün vermesi gereken uygulamalar için kullanılır. Çevik yazılım geliştirme metotları verimliliği yüksek, esnek, hata oranı düşük, hızlı ve ucuz çözümler sağlamaktadır. Proje, büyüklüğü fark etmeden küçük iterasyonlara ayrılır. Her iterasyon sonu proje ekibi müşteriye projenin ilerleyişi hakkında bilgi verir. Bu model ile her bir iterasyonun 2-4 hafta sürmesi planlanır ve bu durum müşteri memnuniyetini artırır. Tüm ekip birbiri ile iletişim halinde olur ve bu durum proje tamamlama hızını olumlu yönde etkiler. Projenin küçük parçalara ayrılması da oluşabilecek hataların kontrolünü kolaylaştırır.

4.1. Çevik Model Avantajları

Hızlı ve devamlı yazılım teslimatı ile müşteri memnuniyeti devamlı artar, zaman ve araçtansa kişilerle olan etkileşim ön plandadır, düzenli çalışan yazılımlar üretilir, gelişen ve daha iyi olan tasarımlara doğru gidilir, ileri zamanlarda ortaya çıkacak gereksinimler hoş şekilde karşılanır ve kolay uygulanır, takım oyunu vardır, değişime açıklık en üst düzeydedir, iterasyon içeren planlar yapılır, öğrenim gerektirmez ve adaptasyon hızlıdır.

4.2. Çevik Model Dezavantajları

Büyük, kapsamlı projelerde ürün oluşturma zamanı net olarak belirlenemez, belgelendirmede aksaklıklar oluşabilir, projeler ekip ile yönetileceğinden ekip şefi deneyimsiz kişilerden oluşmamalıdır aksi takdirde proje olumsuz etkilenir, kurumsal yapılarda uygulaması zordur, ihtiyaçlar devamlı artacağından sürekli çalışma gerektirir, ürün başarısızlıkla sonuçlanırsa kariyer olumsuz etkilenir, takım üzerinde baskı vardır.

Çevik modellerden Extreme Programming (XP) ve SCRUM ilk akla gelenlerdir.

4.1. Extreme Programming (XP)

1999 yılında Kent Beck tarafından geliştirilmiştir. Basitliğe önem verip grup içi iletişimi ön plana çıkarmıştır. Müşteri de projeye dahil edilir ve olası değişikliklere hızlıca uyum sağlayabilmeyi amaçlar. 4 temel değer ve 12 alt temel prensibe dayanır.

4.1.1. Extreme Programming 4 Temel Değeri

1. *Basitlik* – O anki gereksinimleri karşılayan en basit çözüm kullanılmalıdır. Karmaşıklıklardan kaçınılmalıdır.
2. *İletişim* – bir proje başarısızlıkla sonuçlanıyorsa bunun sebebi yüksek ihtimalle iletişimden kaynaklanan problemlerdendir. XP disiplinine göre müşteri ile yazılım ekibi sık sık iletişim halinde olmalıdır.
3. *Cesaret* – Yazılımcı hata yapmaktan korkmamalı, kodun üstüne gidebilmeli hatta gerektiğinde yazılı çöpe atabilmelidir.
4. *Geri Bildirim* – Müşteri, yazılım ekibinin bir üyesi gibi davranmalı ve düzenli olarak geri bildirim vermelidir.

4.1.2 *Extreme Programming 12 alt disiplini şunlardır:* planlama oyunu, ekipte müşteri, önce test, basit tasarım, çiftli programlama, sürekli entegrasyon, kısa aralıklı sürümler, yeniden yapılandırma, ortak kod sahiplenme, metafor, kodlama standardı, haftada 40 saat.

4.2. SCRUM

Scrum, Jeff Sutjerland ve Ken Schwaber tarafından 1990'ların ortalarında geliştirilen bir başka çevik yazılım modelidir. Aslında sadece bir model ismi olmayan SCRUM aynı zamanda bir Rugby hücum taktiğidir. Bu taktikte oyuncular hep birlikte hücum eder. Scrum yazılım geliştirme modelinde de buna çok benzer bir yöntem kullanılır. Yazılımcılar ekipler halinde toplanır ve müşteri de dahil olmak üzere toplantılar yaparak ürün geliştirir.

Bu modelde 30 gün süreyi aşmayacak şekilde büyük parça halindeki proje küçük parçalara (sprint) bölünerek çalışılır ve her gün -genellikle sabahları- 15 dakika ayakta süren toplantılarla gidişat planlanır. "Dün ne yapıldı?", "Bugün ne yapılacak?" gibi sorulara cevap aranır. Buna "Daily Scrum Meeting" ismi verilir. Bu metodoloji, karmaşık ortamlarda adım adım ilerleyen yazılım ekipleri için uygundur. Scrum modeli karmaşıklığı "şeffaflık", "denetleme" ve "uyarlama" ilkeleriyle azaltmaya çalışır. Gereksinimlerin tam olarak tanımlanamadığı ve kaotik durumların beklendiği projeler için en uygun metodolojidir.

4.2.1 Günümüzde Neden Scrum Tercih Edilir?

Scrum, iletişimde şeffaflığı temel alır. Scrum tahtası ve Scrum takip çizelgesi gibi tüm bilgiler ortaktır, şeffaf bir şekilde paylaşılır. Buna bağlı olarak güven ortamı oluşur. Sprint toplantıları, günlük toplantılar sayesinde devamlı geri bildirim ve ekip içi motivasyon sağlanır. Scrum müşteri odaklıdır. Müşteri geri bildirimlerine göre sürekli gelişim halindedir. Ekip iş birliği ile sorunlara kolay çözüm sağlanır. Zaman kısıtlaması olduğunda gereksiz işleri en aza indirgeyebilir. Devamlı yenilebilir bir formda olması değişimlerin projeye uyarlanmasını kolaylaştırır. Ekip üyeleri projeyi ortak sahiplenir. Bu sayede proje kalitesi artar. Diğer modellere göre çok fazla avantaj sağlar. Günümüzde var olan büyük ve devamlı yenilenen projelerin en iyi şekilde geliştirilmesine olanak sağlar.

Yazılım projeleri durağan değildir ve değişen koşullara göre güncellenmesi gerekir. Yazılım projelerinin dinamikliğinden kaynaklanan problemleri en aza indirgemek için Scrum tercih edilir.

Kaynakça

- *https://www.academia.edu/27236336/SDLC_WATERFALL_MODEL_SDLC_WATERFALL_MODEL*
- *https://www.academia.edu/36996695/Software_Development_Life_Cycle_Models_Comparison_Consequences*
- *https://akademiksunum.com/index.jsp?modul=document&folder=a93e3a2fccf8eb56a557c55c5f0d5cf10789abe2*
- *https://akademiksunum.com/index.jsp?modul=document&folder=a93e3a2fccf8eb56a557c55c5f0d5cf10789abe2*
- *https://www.miuul.com/not-defteri/yazilim-gelistirme-yasam-dongusu-sdlc-nedir*
- *https://www.yazilimtestmerkezi.com/post/yazilim-gelistirme-modeli-nedir-cesitleri-nelerdir*
- *https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm*
- *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*
- *http://www.yilmazcihan.com/agile-modelleme-cevik-modelleme/*
- *https://tr.wikipedia.org/wiki/V-Model_(Yaz%C4%B1l%C4%B1m_geli%C5%9Ftirme)*
- *https://tr.wikipedia.org/wiki/Waterfall_model*
- *İzmir Bakırçay Üniversitesi, Dr. Öğr. Üyesi Zekeriya Anıl GÜVEN, BİL102, Yazılım Mühendisliği Temelleri, Bölüm 2, Yazılım-Yaşam Döngü Modelleri Sunumu*
- *İzmir Bakırçay Üniversitesi, Dr. Öğr. Üyesi Zekeriya Anıl GÜVEN, BİL102, Yazılım Mühendisliği Temelleri, Bölüm 3, Çevik Yazılım Geliştirme Sunumu*

İletişim

- *Github: https://github.com/begumkls?tab=repositories*
- *Medium: https://medium.com/@begumkls_*
- *LinkedIn: https://www.linkedin.com/in/beg%C3%BCm-kele%C5%9F-a50774232*