****

**Yazılım Yaşam Döngüsü**

Yazılım da diğer üretilen ürünler gibi üretilme aşamasına gelene kadar geçirdiği aşamalar vardır hatta yazılım yaşam döngüsü üretilme aşamasını ve üretilme aşamasından sonraki bakım aşamasını da kapsar. Aslında o aşamaları da kapsadığı için baya bir uzun süre süren bir yaşam döngüsüdür eğer yazılım düzgün yapılır ve aynı zamanda bakımı da iyi yapılırsa. Bu açıdan baktığımız zaman aslında yazılımın da diğer ürünlerden bir farkı olmadığını görürüz yani yazılım da bir üründür. Bu nedenlerden dolayı bu döngü gayet önemli diyebiliriz. Yazılım yaşam döngüsü şu aşamalardan oluşur: Planlama, Analiz, Tasarım, Gerçekleştirme, Bakım.

**Yazılım Yaşam Döngüsü Aşamaları**

**Planlama:** Planlama aşaması ilk başta yazılımı sunacağımız potansiyel müşterilerin yazılım hakkındaki fikirlerinin ve isteklerinin alınması ile başlar. Sonraki adımında bu fikirler ve istekler doğrultusunda yazılım, personel, donanım gereksinimlerinin, proje planının belirlenmesiyle ve fizibilite çalışmalarının yapılmasıyla devam eder.

**Analiz:** Analiz aşamasında yazılımın gereksinimleri ve işlevi ayrıntılı olarak analiz edilerek çıkarıldığı aşamadır. Eğer ortada mevcut bir sistem varsa onun incelemesi gerçekleştirilir. Analiz aşamasındaki en temel amaç bilgisayar mühendisi gözüyle mevcut olan sistemdeki yapılacak, yapılması olası işlerin ortaya çıkarılması ve bunların doğru olarak algılandığından emin olmaktır. Analiz aşaması temel olan UML diyagramlarının (Sınıf, Nesne, Use-case…) çizilmeye başlandığı aşamadır.

**Tasarım:** Analiz aşamasında belirlenen yazılım gereksinimlerine çözüm olacak yazılım temel halinin oluşturulması aşamasıdır. Bu aşamada daha kodlamaya başlanmamıştır. Bu aşamanın da 2 farklı hali vardır. Bunlar şunlardır: Mantıksal Tasarım ve Fiziksel Tasarım.

**a-) Mantıksal Tasarım:** Olası sistemin yapısı anlatılır ve olası değişiklikler önerilir.

**b-) Fiziksel Tasarım:** Yazılımı içeren bileşenleri ve bu bileşenlerin ayrıntılarını içerir.

**Gerçekleştirme:** Gerçekleştirme aşamasında analizi ve tasarımı bitmiş sistemin kodlanması aşamasıdır. Kodlama işlemi analiz ve çözümleme aşamasında kararlaştırılmış programlama dili, geliştirme ortamı (IDE) ve teknolojilerin yardımıyla yapılır. Gerçekleştirme aşaması 3’e ayrılır bunlar şunlardır: Kodlama, Test Etme, Kurulum.

**Bakım:** Bakım aşaması kodlama, test etme ve kurulum aşamalarından geçen yazılımın hata giderme, güncelleme gibi işlemlerinin yapıldığı aşamadır. Bakım aşaması yazılımın ömrü süresince devam eder.

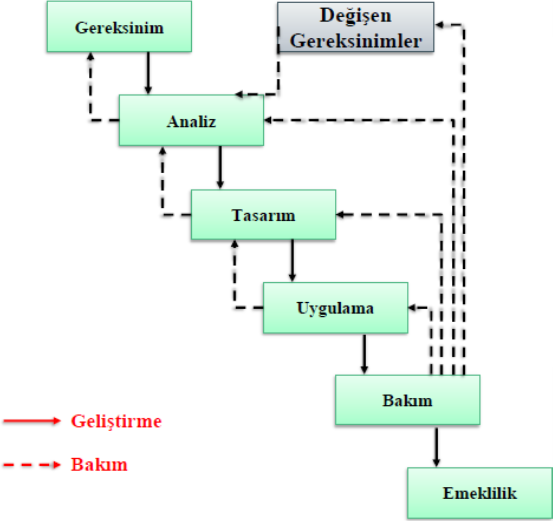
**Yazılım Yaşam Döngüsü Modelleri**

Yazılım yaşam döngüsü modelleri birden fazla modelden oluşur. Yazılım yaşam döngüsü modellerinin birden fazla modelden oluşmasının sebepleri şunlar olabilir: Yazılım projesinin kimler için yapılacağı, yazılım projesinin büyüklüğü, yazılım projesinin bütçesi, yazılım projesinin eleman sayısı, yazılım projesinin kodlama dili gibi sebepler. Ayrıca her modelin kendine özgü avantajları ve dezavantajları vardır. Bunları göz önünde bulundurarak seçim yapılır ve bu neden dolayı da zaten zamanla yeni modeller ortaya çıkmıştır. Artık yazılım yaşam döngüsü modellerini incelemeye başlayalım.

**Gelişigüzel Model:** 1960’lı yıllarda ortaya çıkan ve kullanılan bu modelde bir yöntem ya da model yoktur. Kişiye özgü olduğu için takip edilebilmesi ve uyum sağlamak oldukça zordur. Bu model günümüzde kullanılmamaktadır ve güncelliğini yitirmiştir.

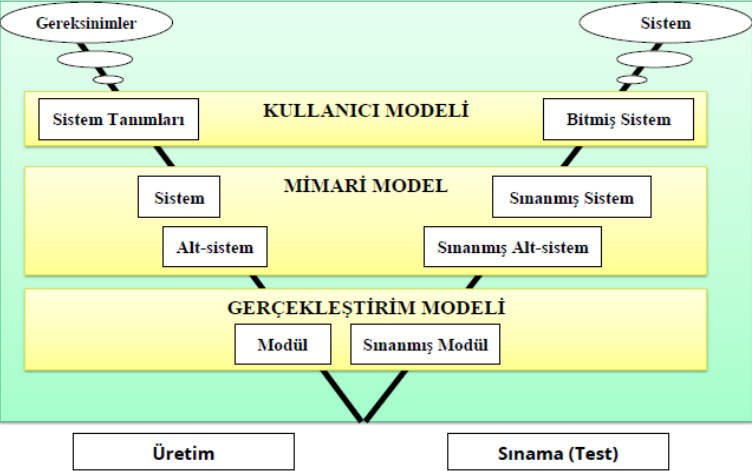
**Barok Modeli:** 1970’li yıllarda ortaya çıkan ve kullanılan bu modelde yazılım yaşam döngüleri yavaş yavaş ve temel olarak kullanılmaya başlanmıştır ama bu modelde de ortada kullanılan ele alınabilir bir model yoktur. Bu modelde dokümantasyon güncel modellerden farklı olarak ayrı bir süreç olarak değerlendirilir. Bu model günümüzde kullanılmamaktadır ve güncelliğini yitirmiştir.

**Çağlayan (Waterfall) Modeli:** Şelale modeli bundan sonra bahsedeceğimiz diğer güncel yazılım yaşam döngüsü modellerinin babasıdır diyebiliriz. Bu yüzden de geleneksel bir model olarak karşımıza çıkar. Haliyle geçmişte en popüler döngü modeliydi. Bu model barok modelinden farklı olarak dokümantasyonu ayrı bir süreç olarak değil de üretimin doğal bir parçası olarak ele alır. Bu modelin en önemli aşaması analiz aşamasıdır. Bu aşamada müşteri ve sistem gereksinimlerinin en ince ayrıntısında kadar belirlenebilmesi gerekmektedir.



Bu modelde yapılacak işler aşama aşama yapılır yani bir aşamayı bitirmeden diğer aşamaya geçilemez. Ayrıca her aşamada dokümantasyon olmalıdır eğer bir aşamada dokümantasyon ve test yoksa o aşama tamamlanmış sayılmaz. Tasarım aşaması da analiz aşaması gibi çok detaylı bir çalışma gerektirir bundan dolayı bu modelin en önemli 2 aşaması tasarım ve analiz aşamalarıdır. Eğer çalışılan proje uzun soluklu bir projeyse bahsettiğimiz tasarım ve analiz aşamasındaki titiz çalışmaya rağmen gereksinimlerin değişmesi kaçınılmazdır. Gereksinimler değişirse kodlama veya test aşamasında ağır maliyet yükü ortaya çıkarırlar. Bu yüzden çağlayan modeli küçük, gereksinimleri belli ve iyi anlaşılmış projeler için daha uygundur. Yazılımın kullanıcıya ulaşma zamanı uzundur. Gereksinimlerin tanımlamaları birçok kez net bir şekilde yapılamadığından az önce bahsettiğimiz uzun soluklu projelerdeki gibi düzeltme işlemlerinin maliyet yükü fazladır. Yazılım ekipleri olabildiğince hızlı kod yazıp, çalıştırıp, sonucu görme düşüncesinde olduğu için bu modelle çalışan ekiplerde mutsuzluk oluşabilir ve kod yazmak dışındaki işlere önem vermez. Kullanıcı bu sürecin içerisinde yer almadığı için yazılım piyasaya sürüldükten sonra geri bildirimler fazlalaşabileceği için bu geri bildirimler de fazladan maliyet yüküne sebep olurlar.

**V Süreç Modeli**



V süreç modeli belirsizliklerin az olduğu ve iş tanımlamalarının belirgin olduğu bilgi teknolojileri projeleri için uygun bir modeldir. V süreç modeli çağlayan modeline karşın kullanıcının projeye katkısını arttırır ve çağlayan modelinden daha gelişmiş bir yazılım geliştirme süreci vaat eder. Bu yüzden V süreç modeli çağlayan modelinin gelişmiş hali olarak düşünülebilir. V süreç modeli temel olarak 3’e ayrılır: Kullanıcı Modeli, Mimari Model, Gerçekleştirim Modeli.

**a-) Kullanıcı Modeli:** Kullanıcıların proje hakkındaki tanımları, fikirleri ve istekleri kullanılır ve tamamlanmış halinin teslim edilmesinden oluşur.

**b-) Mimari Model:** Projenin tasarımı ve tasarımların deneme işlemlerinden oluşur.

**c-) Gerçekleştirim Modeli:** Yazılım modüllerinin kodlanması ve bunların denenmesinden oluşur.

**Helezonik (Spiral) Model**



Helezonik model risk analizi yapma ve prototip üretme üzerine kurulu bir yazılım yaşam döngüsü modelidir. Bu modelde her döngü öncesinde içinde bulunulan fazın risk analizi yapılır ve içine bulunulan faz için planlanmış prototip geliştirilmeye başlanır. Her döngü bittiğinde yeniden planlama yapılarak ulaşılması planlanan hedefler, yararlı olabilecek alternatifler ve kısıtlamalar belirlenir. Bu model önceden geliştirilmiş yazılım projelerinin yeniden kullanıldığı projeler için gayet uygun bir modeldir. Helezonik modelin en önemli getirisi, her döngünün başında risk analizi yapıldığı için maliyet ve zaman ögelerinin kolay tahmin edilebilir olmasıdır. Helezonik model küçük yazılım projelerine uygun değildir. Helezonik modeli uygulayan kişilerin tecrübeli kişiler olması gereklidir. Helezonik model üretim süresi boyunca ürün üretebilme ve üretilen ara ürünün kullanıcı tarafından denenmesi temeline dayanır. Helezonik model projedeki yazılımı kullanacak personelin bu sürece erken katılması gelecekte oluşacak istenmeyen durumları engeller. Helezonik model proje sahibine ve yatırımcıya üretilen yazılımla proje boyunca karşılaştıkları için kolay izleme imkanı sunar. Helezonik modelde yazılımın kodlanması ve denenmesi erken başlar. Helezonik model 4 temel aşamadan oluşur: Planlama, Risk Analizi, Üretim, Kullanıcı Değerlendirmesi.

**a-) Planlama:** Üretilecek ara ürün için işin planlanması, amaç, kısıt ve alternatiflerin belirlenmesi, bir önceki adımda üretilmiş ürünle bütünleştirme yapılmasıdır.

**b-) Risk Analizi:** Alternatifler değerlendirilir ve risk analizi yapılır.

**c-) Üretim:** Planlanmış ara ürün üretilir.

**d-) Kullanıcı Değerlendirmesi:** Ara ürün hakkında kullanıcıların test ve değerlendirmeleri yapılır.

**Artımsal Geliştirme Süreç Modeli**

Diğer yazılım yaşam döngüsü modellerine nazaran burada sistemi tek seferde teslim etmek yerine geliştirme ve teslim aşamaları parçalara bölünür. Her teslim parçası yazılım ekibinden istenen işlevselliğin bir parçasıdır. Teslimler kullanıcı gereksinimlerinin önceliğine göre ayrılır. Öncelikli gereksinime sahip teslimler erken teslimlere eklenir. Eğer bir parça geliştirilmeye başladıysa artık o parçanın gereksinimleri değiştirilemez. Eğer ortaya olası değişiklikler çıkarsa sonraki teslimlerde dikkate alınırlar. Üretilen her yazılım parçası birbirini kapsayacak şekilde ve gittikçe artan işlev içerecek şekilde geliştirilir. Uzun soluklu ve sistemin eksik işlevsellikle çalışabileceği nitelikteki projeler bu modele uygundur. Bir taraftan kullanım yapılırken diğer taraftan üretim yapılır. Her teslim edilen parça ile müşteriye görünen bir değer döndüğünden sistemin işlevselliği erken safhalarda ortaya çıkar. Erken teslimler sonraki teslimler için tespit edilecek gereksinimleri ortaya çıkartmakta prototip görevi görür. Artımsal geliştirme süreç modeli projenin tamamen batma riskini azaltır. Bu model doğal olarak yinelemelidir. Yeniden kullanılabilir bir ürün, işlevsellik sağlamış bir şekilde tüm aşamaların sonunda ortaya çıkar.



**Kodla ve Düzelt Yaşam-Döngü Modeli**

Az satır koddan oluşan programlar için kullanılabilir bir modeldir. Direkt ilk aşamadan yazılımın ilk sürümünü geliştirmeyle başlar. Yazılım istenen düzeye gelene kadar geliştirilir. Bakım aşaması var ama çok zor çünkü planlı programlı bir olay ve dokümantasyon yok. Emeklilik aşaması vardır yani ürünü hazır hale getirip piyasaya sürersiniz ve bakım aşaması da zor olduğu için program piyasaya sürüldüğü an bitmiş olur.



**Çevik Modeller**

Çevik modeller yazılım projelerindeki başarı oranını, ekip içi iletişimi ve hızlıca yazılım üretmek amaçlarıyla ortaya çıkmıştır. Şimdi bazı çevik modelleri inceleyelim.

**Extreme Programming (XP)**

XP, Kent Beck tarafından 1999 yılında bir yazılım geliştirme disiplini olarak ortaya çıkmıştır. XP kolay, grup içi iletişime önem veren, geri dönüşlerin daha fazla olmasını sağlayan bir yazılım geliştirme yöntemidir. XP’nin 4 temel değeri vardır: İletişim, Basitlik, Geri Bildirim, Cesaret.

**İletişim:** Yazılım ekibindeki insanlar birbirleriyle anlaşamayabilirler. XP de bunu aşıp iletişimi arttırmayı amaçlar.

**Basitlik:** Basitlik sağlanması zor bir konudur ama XP buna karşın esnek bir yapı kurup bunu sağlamak istemektedir.

**Geri Bildirim:** Geri bildirim, sistemler için çok önemli bir fırsattır. Onun sayesinde ortaya çıkabilecek hatalar ortadan kaldırılır.

**Cesaret:** Cesaretin olmadığı projelerde korku gelişir ve bu korku projeyi başarısızlığa iter ve bu başarısızlık yazılımın çöpe gitmesi demektir.

**SCRUM**

Jeff Sutjerland ve Ken Schawaber tarafından 1990’ların ortalarında geliştirilen, çevik yazılım geliştirme metodolojileriyle uygulanabilecek bir proje yönetim yaklaşımıdır. Karmaşık yazılım işlerini küçük birimlere (sprint) bölerek geliştirmeyi öngörür. Karmaşık ortamlarda adım adım yazılım geliştiren ekipler için uygundur.



Scrum 3 temel kavramdan oluşur: Roller, Toplantılar, Bileşenler/Araçlar.

**Roller:** Ürün sahibi, scrum yöneticisi ve scrum takımından oluşur. Ürün sahibi scrum’ın başındaki kişidir.

Scrum yöneticisi takımın scrum etiklerine bağlı kalmasını, takımı ve organizasyonu scrum’a adapte eder.

Scrum takımı devamlı iletişim halinde olan ve tek bir hedefe ulaşmak için mücadele eden kişilerden oluşur, kişi sayısı 5-9 olarak değişir.

**Toplantılar:** Scrum toplantıları her gün iş günündeki belirlenen bir saatte olur ve tüm takım bu toplantılara katılır. Her sprint öncesinde yapılır. Bu toplantılar takımın ilerleyişini ve karşılaştıkları engelleri görmeleri için çok iyi bir fırsattır. Tüm ekip üyeleri önceki gün ne yaptıklarını, bugün ne yapacaklarını ve karşılaştıkları engelleri söylerler.

**Bileşenler/Araçlar:** Ürün gereksinim dokümanı ve sprint dokümanı oluşturulur. Ürün gereksinim dokümanı, proje boyunca yapılması gerek işlerin bir listesidir, dinamik olmalıdır çünkü görevler ve kişiler değişebilir. Sprint dokümanı ürün gereksinim dokümanından elde edilen işleri kapsar, işlerin amacı sprint sonunda son ürünün bir parçası olan bir işlevselliği ya da çalışabilir bir parçayı elde etmektir, sprint dokümanı sadece takım tarafından değiştirilebilir.

**Scrum Neden Bu Kadar Popüler ?**

- Zamandan ve paradan tasarruf edilebilmesi.

- Ekip içi iletişimin üst düzeyde olması ve bundan dolayı hataların erken fark edilip düzeltilmesi.

- Değişen gereksinimlere anında cevap verebilmesi.

- Değişen teknolojilere ayak uydurabilmesi.

**Yazılım Yaşam Döngüsü Modellerinin Avantajları ve Dezavantajları**

Gelişigüzel model ve barok modeli güncelliğini yitirmiş ve tarihe karışmışlardır. Çünkü gelişigüzel modelde bir yöntem ya da model yoktur. Kişiye özgü olduğu için takip edilebilmesi ve uyum sağlamak oldukça zordur ve barok modelde ortada kullanılan ele alınabilir bir model yoktur. Bu modelde dokümantasyon güncel modellerden farklı olarak ayrı bir süreç olarak değerlendirilir.

Çağlayan (Waterfall) modelini kullanması ve anlaması basittir, yönetimi kolaydır. Projenin aşamaları ayrı olduğundan dolayı iş bölümü ve planı ilk başta çok net bir şekilde bellidir. Bu durum da projenin yönetimini basit hale getirir. Çağlayan modeli çok küçük ve gereksinimleri iyi anlaşılmış projelerde iyi çalışır. Avantajlarına baktık, bir de dezavantajlarına bakalım. Komplike ve nesne yönelimli programlama (OOP) için uygun değildir, devam eden ve uzun soluklu projeler için uygun değildir, değişikliğe açık olmayan bir modeldir eğer değişiklik yapılmak istenirse maliyet büyük oranda artar, müşteri memnuniyetini sağlamak zordur çünkü gelişim ve değişikliğe açık bir model değildir, bu model aşamalardan oluştuğu ayrıca ürün son aşamada hazır hale geldiği için gereksinimler iyi tanımlanmazsa proje bittikten sonra projenin iptaline sebep olur.

V modeli proje yönetimi konusunda kolaylık sağlar, kullanımı kolaydır. Avantajları bunlardı bir de dezavantajlarına bakalım. Aynı zamanda gerçekleştirilebilecek olaylara olanak tanımaz, fazlar arasında tekrarlamaları kullanmaz, risk çözümleme yoktur, yazılım da diğer sistemler gibi zamanla gelişir, geliştirme devam ederken gereksinimler değişkenlik gösterebilir, son ürüne ulaşmada istikrarsız bir grafik çizer.

Helezonik modelde kullanıcılar sistemi erkenden görebilir, geliştirme küçük parçalara bölünür ve en riskli kısımlar önce yapılır, çeşitli yazılım modelini içinde bulundurur, riske duyarlı yaklaşımıyla olası zorlukları engeller, opsiyonlara daha erken odaklanır, hataları erken gidermeye odaklanır, yazılım-donanım sistemi gerçekleştirme için bir çerçeve sağlar. Dezavantajlarına da bakalım. Küçük ve düşük riskli projeler için maliyetli bir modeldir, komplekstir, spiral sonsuza kadar gidebilir, ara adımların fazlalığı yüzünden çok fazla dokümantasyon gerektirir, kontrat tabanlı yazılıma uymaz, yazılımın içten geliştirileceğine inanır, öznel risk değerlendirmelerine dayanır.

Artımsal geliştirme süreç modelinin avantajları, sistem için gerekli olan gereksinimler müşteriler ile belirlenir, gereksinimlerin önemine göre teslim edilecek artımlar belirlenir, öncelikle en önemli gereksinimleri karşılayan çekirdek bir sistem geliştirilir, erken artımlar prototip gibi davranarak, gereksinimlerin daha iyi anlaşılmasını sağlar, tüm projenin başarısız olma ihtimalini azaltır, en önemli sistem özellikleri fazla denenme imkanı bulur, böl ve yönet yaklaşımıdır. Dezavantajlarıysa, artımları tanımlamak için tüm sistemin tanımlanmasına gerek vardır, gereksinimleri doğru boyuttaki artımlara atamak zor olabilir, tecrübeli personel gerektirir, artımların kendi içlerinde tekrarlamasına izin vermez.

Kodla ve düzelt yaşam-döngü modeli hiçbir planlamaya ihtiyaç duymaz, çok küçük projelerde veya kısa ömürlü prototipler için kullanılabilir, program aşamaları çabuk geçilir, uzman görüşüne ihtiyaç yoktur çünkü herkes bu modeli kolaylıkla kullanabilir. Dezavantajlarına da bakalım. Kontrollü değildir, kaynak planlaması yoktur, bitiş süresi belli değildir, hataların bulunması ve doğrulanması zordur, kodları düzeltmek ekstra maliyete sebep olabilir, kodlar kullanıcının ihtiyacını karşılayamayabilir, kodlar esnek değildir ve değiştirilmesi zordur.

Çevik modellerin avantajları insanın doğal eğilimine çok yatkındır öğrenim gerektirmez adaptasyon hızlıdır, kısa döngüler dolayısıyla yazılımın ekibinin morali yüksektir, verim artar, sık çıkı üretip geri besleme aldığından kaynağı müşteri ihtiyaçlarına ve sonuca yönlendirmeye etmeye odaklanır, plan aşamasında ayrıntılı plan yerine iterasyonun planı yapılır, değişime açıklık ve esneklik en üst düzeydedir, sürdürülebilir kalite sağlar, projeyi planlama ve yürütmeyi aynı anda yapmaya imkan sağlar, takım ruhunu ortaya çıkarır. Dezavantajlarıysa, kurumsal yapılarda uygulanması çok zordur, sürekli değişen ihtiyaçlar yüzünden aşırı çalışma, ürün başarası = proje başarısı olduğu için kariyer riski taşır, yazılım ekibi üzerinde hedef baskısı oluşturur.

**Hangi Projede Hangi Modeli Kullanmalıyız ?**

Küçük ve gereksinimleri iyi belirlenmiş projeler için Çağlayan (Waterfall) modeli kullanılabilir.

Belirsizliklerin az olduğu ve iş tanımlamalarının belirgin olduğu bilgi teknolojileri projeleri için V modeli kullanılabilir.

Uzun soluklu, büyük ve maliyetli projelerde Helezonik model veya Artımsal Geliştirme Süreç modeli kullanılabilir.

Zaman sıkıntısı olmayan ve tek kişi çalışılan küçük programlarda Kodla ve Düzelt Yaşam-Döngü modeli kullanılabilir.

Orta, küçük ve uzun soluklu olmayan projelerde çevik modeller kullanılabilir.

**Kaynakça**

[**https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/**](https://fikirjeneratoru.com/yazilim-proje-yonetimi-yontemleri/)

[**https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8**](https://medium.com/@secilcor/scrum-nedi%CC%87r-6a4326951dd8)

[**http://www.aspmvcnet.com/tr/m/yazilim-muhendisligi/helezonik-tasarim-spiral-tasarim.html**](http://www.aspmvcnet.com/tr/m/yazilim-muhendisligi/helezonik-tasarim-spiral-tasarim.html)

[**https://caglartelef.com/yazilim-yasam-dongusu/**](https://caglartelef.com/yazilim-yasam-dongusu/)

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