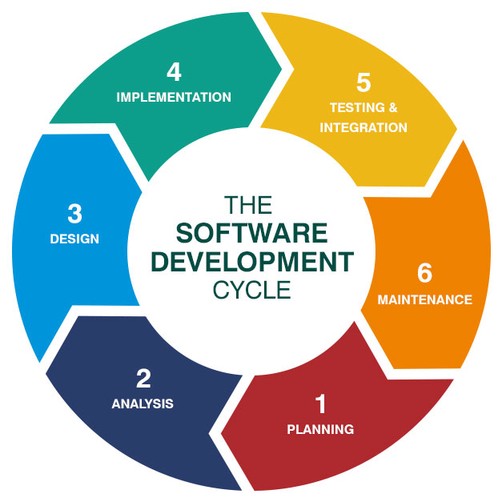
**Yazılım Yaşam Döngüsü Nedir?**

****

Yazılım ürünü canlı bir olgudur. Bizimle birlikte yaşayan, zaman içerisinde geliştirilmeye, eksikliklerinin giderilmesine, güçlendirilmesine ihtiyaç duyan bir kavramdır. Üretiminden müşterinin elindeki son kullanım anına kadar gerekli işlemlerin yapılması gerekir. Birkaç adımda uygulanıp bitirilebilecek bir şey değildir. **Süreç** işidir. Geriye dönüşler yapılabilir. Bu yüzden yazılımın bir yaşam **döngüsü** vardır. Buna yazılım yaşam döngüsü **(Software Development Life Cycle - SDLC)** denir. Sadece yazılım için değil sistemlerin geliştirilmesi içinde bu döngü adımları takip edilebilir.(System Development Life Cycle) Daha nitelikli ürünlerin ortaya konmasına, karmaşık ve büyük projelerde sürecin adım adım, aşama aşama planlı ve sistemli bir şekilde yürütülerek yükün hafiflemesine, projenin en iyi şekilde gerçekleştirilmesine yardımcı olur. Yazılım yaşam döngüsü birkaç temel aşamadan meydana gelmektedir. Bu aşamalar literatürde farklı sayılarda olabilir. Ancak nihayetinde yapılmak istenen şeyler aynıdır. Bu aşamalar aşağıdaki gibidir;

**YAZILIM YAŞAM DÖNGÜSÜ TEMEL ADIMLARI**

**Planlama:** Proje için tüm ihtiyaçların, kullanılacak kaynakların, izlenecek yol ve yöntemlerin, projeye  ayırılacak zamanın belirlendiği, görevlerin belirlenip dağılımının yapıldığı, fizibilite çalışmalarının yapıldığı kısaca projenin baştan sona detaylı bir şekilde planlandığı aşamadır diyebiliriz.

**Analiz:**Bu aşamayaTanımlama aşaması da denilebilir. Mevcut Sistemin tanımı yapılır. Teknoloji gereksinimlerinin belirlendiği ve alternatiflerin değerlendirildiği aşamadır. Gereksinimler ayrıntılı ve net bir şekilde belirlenir ve analiz edilir. Sistemin detaylı analizinin yapıldığı aşamadır. Yapılan analizlerin belgelendirme işlemleri yapılır. Sistemin etkilediği, etkilendiği durumlar vs.  analiz edilir. Müşteri, Sistem analisti, İş analisti, yazılım mühendisi, ürün yöneticisi gibi konumlardaki kişiler bir araya gelir, toplantı yapar. İhtiyaçların tam olarak anlaşılıp anlaşılmadığı kontrol edilir. Çeşitli diyagramlar bu aşamada çizilmeye başlar ve Tasarım aşamasında devam eder. ( UML, Use Case vs.) Proje ile ilgili raporlar hazırlanır.

**Tasarım:**Yazılımın veya sistemin ilk ve temel halinin oluşturulduğu aşamadır. Sistemin genel yapısı tasarlanır. Bir sonraki aşamaya kadar yazılımın veya sistemin son hali belli olur, son şekiller verilir, kararlar verilir ve her şey netleştirilir. Ara yüzler ve Sistem belirlenir. 2 çeşit tasarım yapılır. Bunlar;

**Üst Seviye ve Mimari Tasarım (Architectural Design):**Kabataslak plan, modüller, akış şemaları Mimari tasarım dokümanları oluşturulur.

**Detaylı Tasarım (Detailed Design):**Veri tabanı tasarımı, kullanıcı ekranları, veri akış yolları…

**Fiziksel Tasarım**

**Mantıksal tasarım**

**Gerçekleştirme (Kodlama):**Tasarımı yapılan projenin artık kodlanmaya başlandığı, yazılım halini aldığı aşamadır. Önceki aşamalarda belirlenen programlama dili ve teknolojik alt yapının kullanıldığı aşamadır. Bu aşamanın temel **taşları Kodlama, Test ve Kurulum**dur. Temiz ve okunabilir kod oluşturma amaçtır. İyi kod; okunabilirliği ve bakımı kolay olan koddur. Modüller birleştirilir, entegrasyon testi yapılır. Her bir Modül birbirinden bağımsız olarak test edilir. Analiz  aşamasından itibaren her aşamanın sonunda   test  yapılır. Bu aşamada  da test yapılmalıdır. Testlerin önemi büyüktür. Hataları erken fark edip zaman ve para kaybı yaşanmasına engel olur.

**Test:**Bu aşamada bahsedilen Test Sistemin genel problemlerini tespit etmek amacıyla yapılan Sistem Testidir. Daha önce de bahsettiğimiz gibi analiz aşamasından başlayarak her aşamanın sonunda gereken testler yapılır. Çeşitli Test türleri bulunmaktadır. Bu aşamadan sonra ürün teslim edilir. Ürünle birlikte kullanma kılavuzu ve gerekli dokümanlar da teslim edilmelidir.

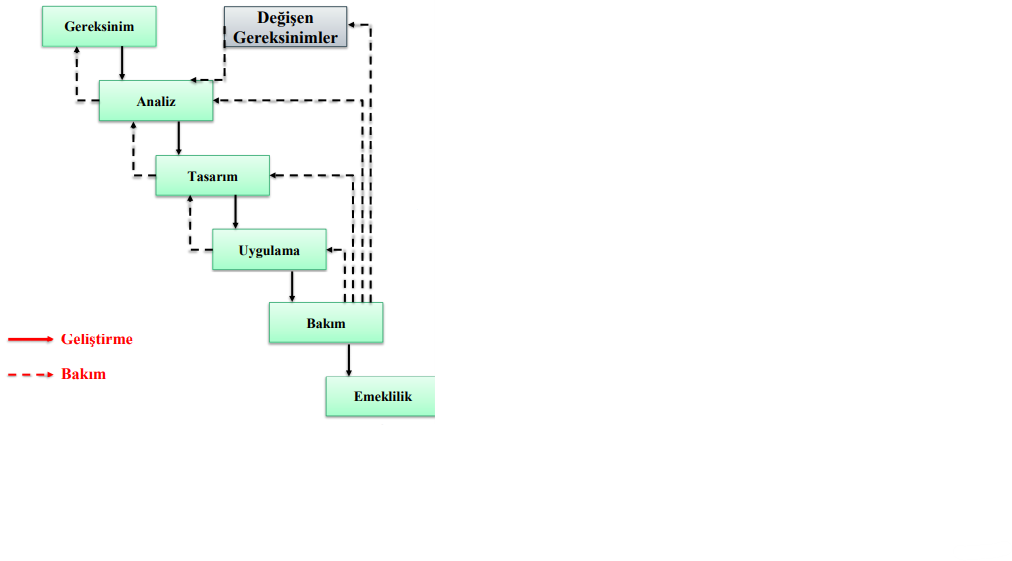
**Teslim sonrası Bakım** **(Post-delivery maintenance):**Bakım aşaması yazılımın ömrünün sonuna kadar sürer. Hata gidermeler, güncellemeler ve gerekli eklemeler yapılır. Ortaya çıkan gereksinimlere  göre yeni analizler, tasarımlar, uygulamalar yapılır ve bu süreç bir **döngü** halinde devam eder.

Yazılım yaşam döngüsü yazılımların, projelerin büyüklüğüne, hitap ettiği kesime, içerdiği aşamaların sayısına ve niteliğine göre farklı modellere ayrılmıştır. Yazılım yaşam döngüsünün sağlıklı ve verimli şekilde gerçekleşmesi için kullanılacak modelin o yazılımın karakterine, yapısına uygun nitelikte olması gerekmektedir.

**YAZILIM YAŞAM DÖNGÜSÜ MODELLERİ**  **(SOFTWARE DEVELOPMENT LİFE CYCLE (SDLC))**

* **Şelale, Çağlayan Modeli (Waterfall Model)**
* **Kodla ve Düzelt (Code and Fix)**
* **Barok Modeli**
* **Gelişi güzel Model**
* **V süreç Modeli (V process Model)**
* **Prototipleme (Prototyping)**
* **Artımlı Geliştirme (Incremental Development)**
* **Helezonik (Spiral) Model**
* **Formal Sistem Geliştirme (Formal System Development)**
* **Araştırma Tabanlı Model**
* **Evrimsel Geliştirme (Evolutionary Development)**
* **Çevik Modeller (Agile models: XP,Scrum)**

**1. Şelale, Çağlayan Modeli (Waterfall Life Cycle Model)**



Basit ve diğer modellere temel teşkil eden bir modeldir. Geleneksel ve geçmişte kullanılan en popüler modeldir ancak günümüzde kullanımı azalmıştır. Bu model küçük, gereksinimleri çok iyi anlaşılmış, gereksinimleri daha sonra değişmeyecek, yeniliğe açık olmayan kısa süreli projeler için uygundur. Karmaşık ve uzun süren projeler için elverişsizdir. Çünkü süreç içerisinde gereksinimler değişecektir ve bulunduğumuz aşamadan başka bir aşamaya müdahale edemeyiz. Bu yüzden işler çok zorlaşır, zaman ve maliyet kaybına neden olur. Her safhada dokümantasyon ve test işlemi yapılır bunlar yapılmadan aşama bitmiş kabul edilemez. Aşamalar en az birer kez tekrar edilir. Bir aşama bitmeden diğer aşamaya geçilemez. İlk iki aşama çok önemlidir. Bu aşamalarda büyük bir titizlikle çalışılmalıdır. Bu aşamaların eksik tamamlanması dâhilinde diğer süreçler de bu durumdan etkileneceği için proje istenildiği gibi olmayacak, büyük kayıplar yaşanacaktır.

**1.1 Gereksinim Safhası (Requirement Phase):** Burada devreye iş analisti girer. Müşteriden doğru ve eksiksiz bir şekilde gereksinimleri toplar, sonrasında bu gereksinimleri düzenler ve çalışacak olan ekibe, kişiye liste veya rapor halinde verir. Müşterinin ihtiyaçları belirlenir. Burada önemli nokta müşterinin istekleri değil ihtiyaçlarıdır. İstenilen her şey gerçekleştirilemez. Önemli olan gerçek ihtiyaçlardır. Bu aşamanın önemi büyüktür. Çünkü Gereksinimler doğru belirlenmezse bundan sonraki aşamaları da etkileyecek olup ortaya istenmeyen olumsuz sonuçlar çıkacaktır. Yapılan hatalar ne kadar erken fark edilirse zaman ve maliyet kaybı o kadar az olur.

**1.2Analiz Safhası (Analysis Phase) :** Gereksinimler analiz edilir.Genelde2 tür analiz yapılır. Durum ve istek analizi. **“Geliştirilecek yazılım ürününün ne yapmasını istiyoruz**?” sorusuna yanıt aranır. Analizin sonucu şartname dokümanı olarak yazılır.

**1.3 Tasarım Safhası (Design Phase) :** Bu safhada elimizdeki analizlere ürünün yapısal ve detay tasarımları ya- pılır. **“Ürün beklenilen işlevselliği nasıl sağlayacak?”** sorusuna yanıt aranır. Çeşitli diyagramlar (Class.Use-Case UML vs.) çizilir. Bu diyagramları hazırlamamızın sebebi şudur: Kodlanacak verileri öncesinde bir bütün halinde görsel olarak görmek işe daha çok hâkim olmamızı, işi daha iyi anlamamızı sağlar. Yani işimizi kolaylaştırır.

Yazılımlardaki hataları azaltmamıza yardımcı olur. Genelde 2 tür Tasarım vardır.

**1.3.1 Üst Seviye ve Mimari Tasarım (Architectural Design)**

Kabataslak plan, Modüller…

**1.3.2Ayrıntılı Tasarım (Detailed Design)**

Veri tabanı tasarımı, kullanılacak algoritmalar

Sınıf, aktivite, etkileşim diyagramları

**1.4 Uygulama Safhası (Implementation Phase):** Geliştirme ( Development) safhasıdır. Birden çok alt safhası vardır. Tasarımı yapılan projenin kodlanma (Coding) safhasıdır. Modüller kodlanmaya başlanır. Birim testi yapılır. Sonra bu Modüller birleştirilirip entegrasyon testi yapılır. Sonrasında Genel sistem testi yapılır. Son olarak **Müşteri** tarafından kabul testi (acceptance testing) yapılır.

**1.5 Teslim sonrası Bakım Safhası (Post-Delivery Maintenance):** Müşteriye teslim edilen ürünün kullanımında gerçekleşen ihtiyaçlara ve müşterinin taleplerine göre güncellemeler, iyileştirmeler yapılır. 2 çeşittir:

**1.5.1 Düzeltici Bakım (Corrective Maintenance),**

Sistemdeki hataları düzeltmek için yapılan bakım (Software Repair)

**1.5.2 Özelliklerin Arttırılması (Enhancement) – Yazılım Güncellemesi ( Software Update)**

**1.5.3 Mükemmelleştirici Bakım (Perfective Maintenance)**

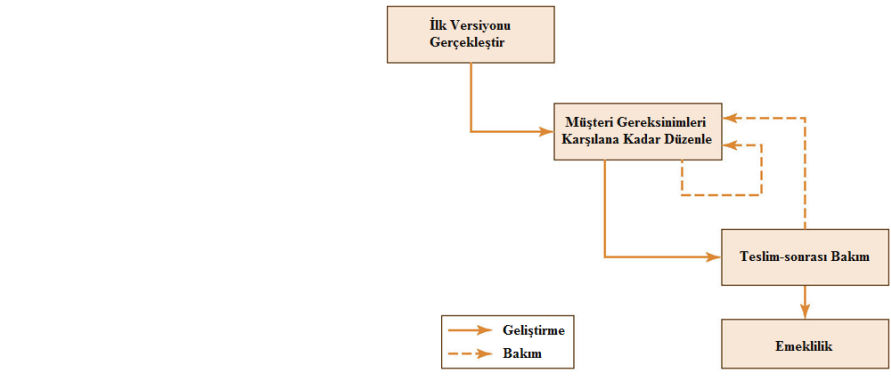
Müşterinin talepleri doğrultusunda ürün memnuniyetini arttırmak için yapılan bakımdır.

**1.5.4 Uyarlanabilir Bakım (Adaptive Maintenance)**

Ürünün değişen şartlara uyum sağlaması için yapılan bakımdır.

**1.6 Emeklilik Safhası (Retirement Phase) :** Yazılımın kullanım süresi dolduğunda durdurulur ve yazılım emekliye ayrılmış olur.

**2. Kodla ve Düzelt (Code and Fix)**



Kodlama, Bakım ve Emeklilik Safhası vardır. Planlamaya gerek duyulmaz. Kaynak planlaması da yoktur. Şartname ve tasarım dokümantasyonu olmadığı için bakımı çok zordur. Kodlar sonradan değiştirilmek için yazılmadığından esnek değildirler ve değiştirilmesi zordur. Direkt yazılım ürünü üretilir bu yüzden en kolay modeldir ancak yazılımda yapılan değişikliklerin maliyeti yüksek olduğu için pahalı bir modeldir. Küçük, kişiye özel projelerde ve kısa ömürlü prototiplerde kullanılabilir. Bitiş süresi belli değildir. Müşteri sürece dâhil edilmediği için memnuniyetsizlikler olabilir. Ekip işinden ziyade bireysel çalışmaya elverişlidir.

**3. Barok Modeli**

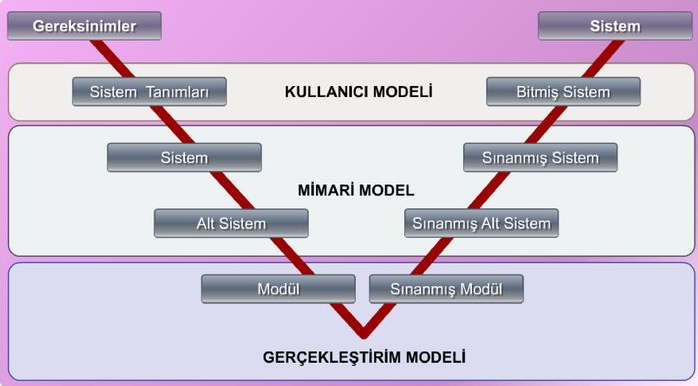


70’li yılların ortalarından itibaren kullanılmaya başlanmıştır. Temel adımları lineer bir şekilde geliştirilmiştir. Bel- gelemeyi (dokümantasyonu) ayrı bir süreç olarak ele alır. Ancak günümüzde belgeleme işlemi her safhadan sonra yapılır. Bu dokümantasyon süreci yazılımın test ve geliştirme aşamalarından sonra gerçekleştirilir. Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlanmamıştır. Gerçekleştirim aşamasına daha çok ağırlık veren bir model olduğu için günümüzde kullanılmamaktadır ve kullanımı tavsiye edilmemektedir.

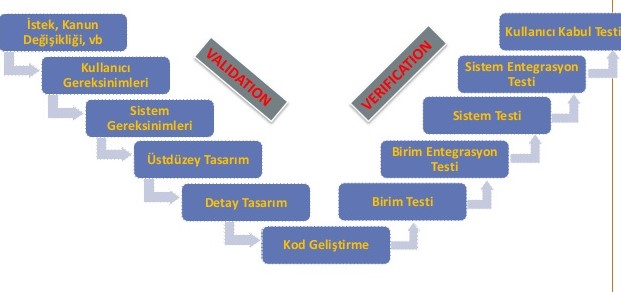
**4. Gelişi güzel Model**

Aslında bir Model değildir. Belli bir yöntemi de yoktur. 1960’lı yıllarda kullanılmıştır. Kişiye özel yazılım geliştir- me olduğu için izlenebilirliği ve bakımı çok zordur. Tek kişilik üretim ortamı vardır ve çok basit programlamaya sahiptir.

**5.** **V süreç Modeli (V process Model)**



Şelale (Waterfall) modeline benzer. Ancak gelişmiş halidir. Kodlama aşamasından sonra yukarı eğim gösterir. Karşı karşıya olan her adım birbiriyle ilgilidir. Mesela alt sistem(detaylı tasarım) karşısında olan sınanmış alt sistem(birim testleri) adımıyla sınanır. Gereksinimlerin kabul testleri yapılır. Sol taraf üretim sağ taraf ise sına-imadır (Test). Belirsizliklerin az iş tanımlarının ise belirgin olduğu büyük projeler için uygundur.(Bilgi teknolojileri Projeleri gibi) Bu modelde kullanıcı projeye daha çok dahil edilir. Risk çözümle aktiviteleri bulunmaz. Verification ve Validation planları uygulanır ve sadece son üründe değil teslim edilebilir tüm ürünlerde uygulanır. Takibi ve kullanımı kolay bir modeldir.



3 tane temel çıktıları bulunur;

**Kullanıcı Modeli:** Kullanıcının isteklerinin belirlendiği, tanımların yapıldığı ve bunların sınanıp (kabul testleri) projenin teslim edildiği aşamadır.

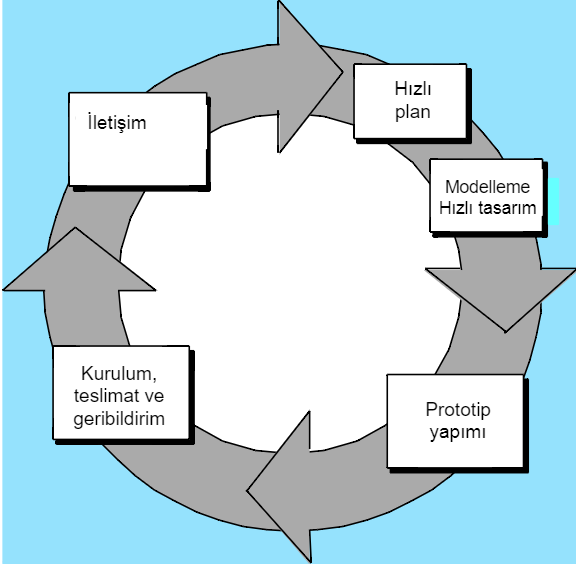
**Mimari Model:** Üstve alt (detaylı) tasarımların yapıldığı ve bunların sınandığı, test edildiği aşamadır.

**Gerçekleştirim Modeli:** Yazılım modüllerinin kodlandığı ve testedildiği aşamadır.

**6. Prototipleme (Prototyping)**

Bu modelde önce gereksinimler toplanır. Geliştiriciler ve müşteri bir araya gelerek yazılımdan elde edilecek bütün çıktılara, bu çıktılar için gerekli girdilerin nasıl sağlanacağına, nasıl korunacağına ve hangi işlemlere uğrayacağına karar verirler. Daha sonra hızlıca tasarım yaptıktan sonra örnek ürün (prototip) üretilir ve müşteriye kullanması ve değerlendirmesi için sunulur. Amaç: Alınan geri bildirimlere göre müşterinin isteğine en yakın ürünü üretmektir.

Müşterinin değerlendirmelerine, taleplerine göre ürün üzerinde değişiklikler yapılır ve yine müşteriye sunulur. Bir döngü halinde bu süreç işler. Müşteri ile ürün fonksiyonları hakkında anlaşmaya varılır. Sınırlı sayıda tekrarlardan sonra ürün müşteriye teslim edilir. Bu modelde geliştiriciler ve müşteri sürekli iletişim halindedir. En büyük avantajı yanlış anlaşılmaların engellenmiş olması ve yeni gereksinimlerin net bir şekilde anlaşılmasıdır. Böylece riskler kontrol altında tutulmuş olur. Dezavantajı ise dokümantasyonu olmayan hızlı ve kirli ( Quick and Dirty) prototiplerin oluşmasıdır.

****

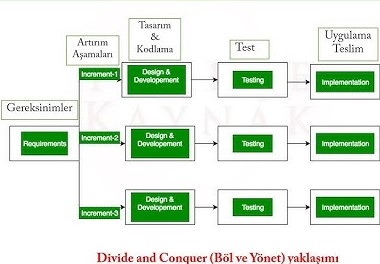
**Gereksinim Analizi**

**Prototipleme için Tasarım**

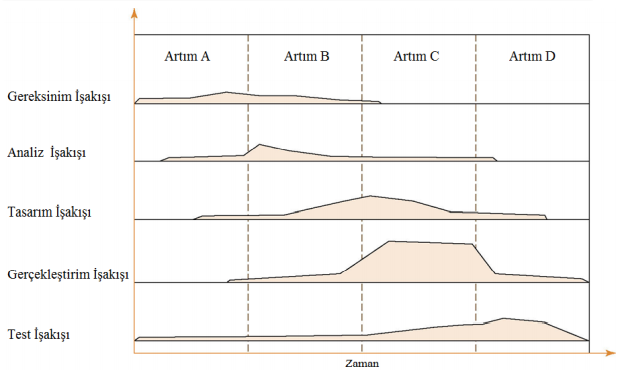
**Geliştirme**

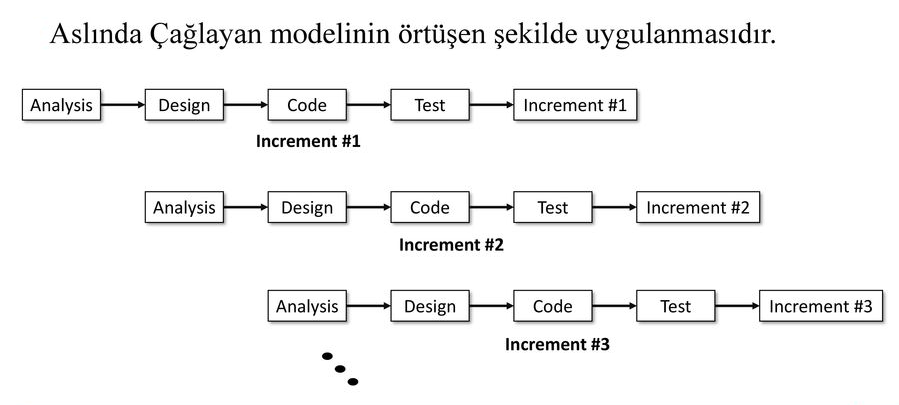
**Kurulum,Teslimat,GeriBildirim İletişim**

**7. Artımlı Geliştirme Süreç Modeli (Incremental Development Process Model)**

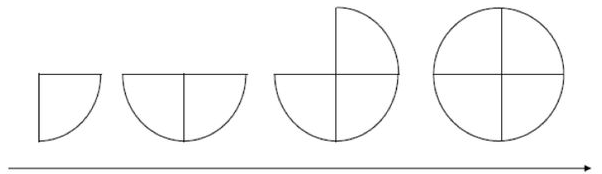
****

Proje tek seferde teslim edilmez. Proje parçalara bölünür. Bu parçaların işlemleri bittiğinde ara ürünler oluşur. Ara ürünler her defasında gelişerek, üzerine eklemeler yapılarak oluşturulur ve ara ara teslimler yapılır. Ara ürünler sürekli geliştirildiği için yapılan her döngü bakım görevi görmüş olur. Öncelikli gereksinimleri karşılayan ürünler ünce teslim edilir. Bir taraftan üretim bir taraftan da kullanım yapılır. Uzun zaman alabilecek, maliyetli, büyük sistemin eksik işlevlikle çalışabileceği, daha sonra ekleme yapılabilecek olan projeler için uygundur. Projenin tamamen başarısız olma olasılığını azaltır. Bir önceki modelde de olduğu gibi ara ürünler meydana geldiği için son üründen önce prototip meydana getirilmiş olur. Prototipler gereksinimlerin daha net anlaşılmasını sağlamış olur. Önemli özellikler çok kez test edilmiş olur. Artımların kendi içlerinde tekrarlamalara izin vermez. Tecrübeli personel gerektirir. **Böl, parçala, fethet (divide and conquer)** yaklaşımı vardır. Uzun zaman alabilecek ve sistemin eksik işlevlikle çalışabileceği türdeki projeler bu modele uygundur. Ayrı bir model değildir. Değişen ihtiyaçlara göre artırımlı bir şekilde ilerlenir. **İteratif** bir modeldir.



****

**Şelale modeli ve Evrimsel geliştirme modeli arası bir modeldir.**



**8. Helezonik (Spiral) Model**

**Planlama - Risk Analizi – Üretim – Kullanıcı Değerlendirmesi**

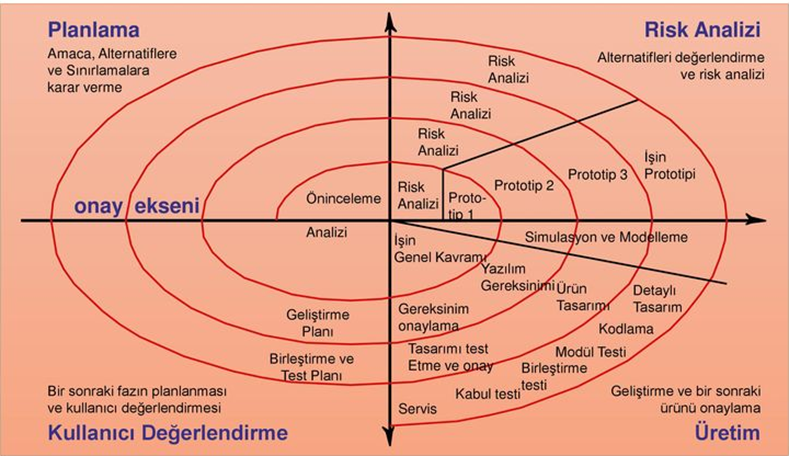
**Planlama:** Üretilecek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme yapılır.

**Risk Analizi:** Risk seçeneklerinin araştırılması ve risklerin belirlenmesi

**Üretim:** Ara ürünün üretilmesi

**Kullanıcı Değerlendirmesi:** Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınama ve değerlendirmeler

Doğrudan gereksinim, analiz, tasarım gibi kesin aşamalar yoktur. Neye ihtiyaç duyulursa o adım gerçekleştirilir. Her halka bir **fazı** gösterir. Bu modelde **Risk Analiz Olgusu** ön plana çıkmıştır. Riskler belirlenip çözülür. En riskli kısımlar önce gerçekleştirilir. Spiral metodoloji, tasarım ve geliştirme faaliyetlerinde hızlı prototip oluşturma ve eşzamanlılığın bir kombinasyonudur. Olumlu yanları: Kullanıcılar sistemi erkenden görebilirler. Hatalar erken giderilir. Kullanıcıya prototip sunulduğu için kullanıcı değerlendirmeleri sonucunda son ürün talebe uygun hale gelecektir. Hata yapma olasılığı azalacaktır. Yinelemeli artımsal ve prototip yaklaşımı vardır. Büyük, maliyetli ve uzun süren projeler için uygundur. Olumsuz yanları şunlardır: Spiral komplekstir ve sonsuza kadar gidebilir (Uzun sürer). Dokümantasyon işlemi fazladır. Kontrat tabanlı yazılıma uymaz. Öznel risk değerlendirme deneyimine dayanır.



**9.** **Formal Sistem Geliştirme (Formal System Development)**

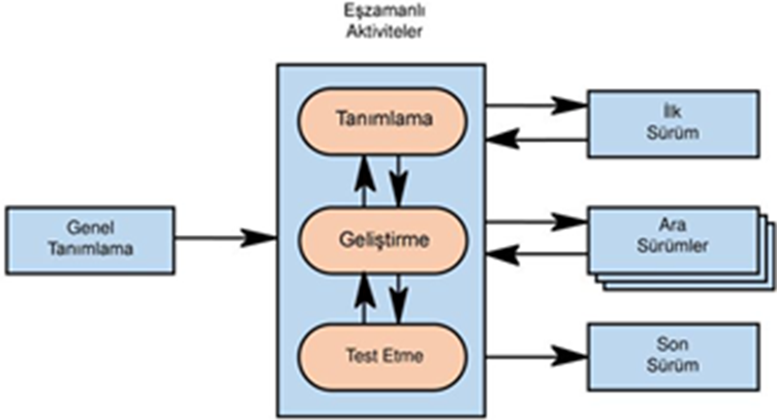
Matematiksel bir tekniktir. Karmaşık programları geliştirmeyi hedefler. Pahalı hata ayıklama işlemini önlemek amacıyla kodu ilk yazarken doğru yazmak ve doğruluğunu test etmek bu modelin esaslarıdır. Cleanroom yazılım geliştirme esaslıdır. Program geliştirmeye destek olur. Yazılım artırımlarla geliştirilir. Sürekli tümleştirme vardır ve fonksiyonellik tümleştirilen yazılım artımları ile artar. Formal belirtim, tasarım ve geçerleme işlemlerini kullanır. Bu model kullanıcı sistemi kullanmaya başladığında karşısına çıkan belirtim hatalarını en aza indirir. Eğitim ve özel beceri gerektirir.

**10. Araştırma Tabanlı Model**

Araştırma ortamları tamamıyla belirsizlik üzerindir. Ortaya çıkacak sonuçlar belirgin değildir. Geliştirilen yazılımlar genellikle sınırlı sayıda kullanılır ve kullanım bittikten sonra işe yaramaz hale gelir ve atılır. Bu yüzden Yap-At prototipi olarak bilinir. Model-zaman-fiyat belli olmadığı için sabit fiyat sözleşmeleri için uygun değildir. Örnek olarak: En hızlı çalışan asal sayı test programı, en büyük asal sayıyı bulma programı, satranç programı.

**11. Evrimsel Geliştirme (Evolutionary Development)**

İlk tam ölçekli modeldir. Geniş alana yayılmış, çok birimli organizasyonlar için önerilmektedir (Çok birimli banka uygulamaları). Her aşamada üretilen ürünler,  üretildikleri alan için tam fonksiyonelliği içermektedirler. Belirtim (specification), geliştirme ve geçerleme (validation) aktivitleri koşut zamanlı yürütülür. Olumlu yanları: Kullanıcıların kendi gereksinimlerini daha iyi anlamalarını sağlar. Olumsuz yanları: değişiklik denetimine sahip değildir. Modelin yapılandırma, kalite, sürüm ve değişiklik yönetiminde sıkıntılar vardır. Sürekli değişiklik yazılımın yapısına zarar verir. Düzenli teslim edilebilir ürün yoktur. **Pilot uygulama kullan, test et, güncelle, diğer birimlere taşı.** Modelin başarısı ilk evrimin başarısına bağımlıdır. Küçük ve orta boyutlu etkileşimli sistemlere, kısa süreli kullanılacak sistemlere ve büyük kitlelere hitap edecek projelere uygulanabilir.



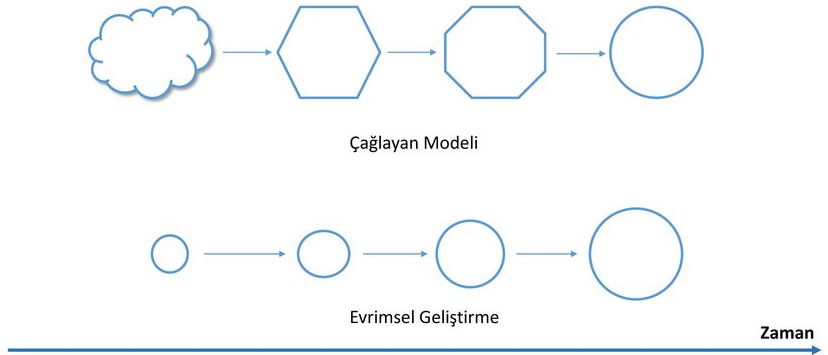
2 çeşittir;

**- Keşifçi geliştirme (Exploratory Development)​​**

Amacı müşterinin gereksinimlerini incelemek için müşteri ile çalışıp son sistemi teslim etmektir. İyi anlaşılan gereksinimlerle başlanmalıdır**. “ Ne istediğimi sana söyleyemem ama onu gördüğümde bilirim”**

**-Atılacak prototipleme (throw-away prototyping)**

Amacı sistemin gereksinimlerini anlamaktır. Tam anlaşılmamış gereksinimlerle başlar.

****

**12. Çevik Modeller (Agile models: XP, Scrum)**

1990’lı yılların sonlarına doğru geliştirilmiştir. Çevik metodoloji müşteriye ürünü zamanında teslim edememe, ürünün değişen isteklere hızlı bir şekilde cevap verememesi, değişen koşullara adapte olamaması, hataların erken tespit edilememesi gibi sorunları ortadan kaldırmayı ve daha başarılı projeler geliştirmeyi amaçlamaktadır. Dolayısıyla Çevik metodoloji verimliliği yüksek, esnek, güçlü, hata oranı düşük, hızlı ve ucuz çözümler sağlamaktadırlar. Ekip çalışmasını destekleyen bir metodolojidir. Ekip sürekli iletişim halindedir. Çevik metotlar bir yazılım geliştirme yaklaşımı değil, geliştirme süreçleri topluluğudur. Projenin boyutu fark etmeksizin küçük yinelemelere ayrılır.(İteratif temellidir) Her yinelemeye bir proje gözüyle bakıldığı için her yinelemenin sonunda elde edilen çalışan ürün müşteriye sunulur ve müşteri süreci yakından gördüğü ve eline çalışan sistem geçtiği için memnuniyeti artar. Her bir yineleme 1-3 hafta arası sürer. Geniş çaplı şirketlerin kompleks ve uzun projeleri için son derece uygundur.

**Çevik Yazılım Geliştirme Manifestosu**

2001 yılında Kent Beck ve 16 arkadaşı tarafından çevik manifesto oluşturulmuştur. Bu manifesto içerdiği değerler, prensipler ve pratikler, somut yazılım metotları ile geleneksel modellemeye göre daha esnek ve kullanışlı yazılımlar üretilmesin sağlar. Bu manifestoda;

Süreçler ve Araçlar yerine **Bireyler ve Etkileşimler**

Kapsamlı Belgeler yerine **Çalışan Yazılım**

Sözleşme Görüşmeleri yerine **Müşteri İlişkileri**

Plan İzleme yerine **Değişikliğe Açıklığın**, daha önemli ve öncelikli olduğu belirtilmektedir.

**Değerler:**

Süreç ve araçlar arası etkileşim

Kapsamlı dokümantasyon üzerinden yazılım geliştirme

Uzlaşılan taahhüt üzerinden müşteriyle birlikte çalışma

Değişiklik yönetimini plan üzerinden yönetmek

**Çevik Yazılım Geliştirme Prensipleri**

* Basit olma
* Yüz yüze iletişim
* Değişen gereksinimlere göre ilerlenmiş kodlama aşamasında değişikliğe gidilebilme (Esneklik)
* Başarılı projeler üretmek ekibe de bağlıdır. Bu yüzden ekibin motivasyonu yüksek olmalıdır. Ekibe gerekli imkânlar sunulmalıdır, her türlü destek verilmelidir ve ekip güvendiğimiz kişilerden oluşmalıdır.
* Süreç içerisinde eksiklikleri belirleyip bunları minimum hale getirmek
* Güçlü teknik altyapı ve tasarımın modelin başarısını etkilemesi
* Çalışan yazılımlara göre sürecin ilerleyebilmesi
* **Sürdürülebilir geliştirmeyi** desteklemek
* Kısa aralıklarla kaliteli ürünler ortaya çıkarmak
* Değişen koşullara hızlı bir şekilde adapta olabilme

**Avantajları**

Ekibin motivasyonuna önem verildiği için verimlilik artar.

Kaliteli ve başarılı projeler üretilir.

Kısa sürede teslimatlar yapıldığı için hem üretkenlik artmış olur hem de müşteri memnuniyeti.

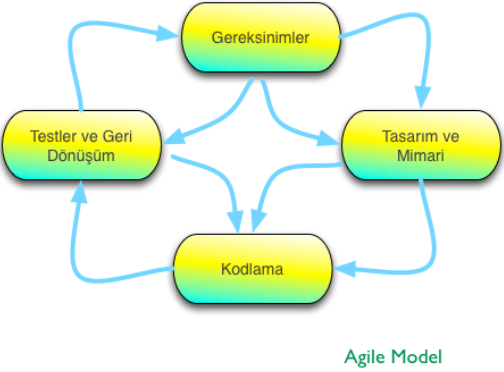
Zamandan ve paradan kazanılır.

Değişime açık olmak, esnek olmak.

**Dezavantajları**

Ekip üzerinde ister istemez bir baskı oluşur.

Değişikliğe açık olduğu için ekstra yorucu bir çalışma temposu ekibi mutsuz eder.



**Çevik Metodolojiler**

**- Uç değer Programlama (Extreme Programming – XP)**

Kent Beck ve arkadaşları tarafından 1996 yılında ortaya çıkarılmıştır. Ekip içi iletişime önem verir. Yazılım geliştirmeden kolaylığı ve esnekliği sağlamak için, **XP 12 farklı pratiği** ön görür.

**4 temel değeri vardır;**

**Basitlik:** Karmaşıklık XP de olmaması gereken bir durumdur. Kodların olabildiğince temiz, sade ve anlaşılabilir olması gerekir.(Clean Code) Bu basitliği sağlamak aslında zordur. Basit değildir. Anlık ihtiyaçlara basit çözümler üretilir.

**Cesaret:** En zor değerdir. Herkesin yapabileceği bir şey değildir. (Yapılmak istenmez) Başarısızlıktan korkan değil başarısızlığı meydana getirecek etkenleri ortadan kaldırabilen biri olmak gerekir. Gerektiğinde çok uzun zaman alan ancak başarısız olan kodları çöpe atıp yeniden başlamayı bilmek gerekir. En önemli sloganı **Elini korkak alıştırma** olmalıdır.

**Geri Bildirim (Feedback):** Geri bildirimler sayesinde hatalar minimum hale gelir. Eğer geri bildirimler olmasaydı yazılım ürünü teslim edildiğinde müşteri ürünü beğenmediğinde “ isteklerime uygun bir ürün olmamış ben böyle istememiştim” diyebilir ve bu durum haliyle birçok açıdan olumsuzluklar doğuracaktır. Müşteri doğrudan projeye dahil edilir ve ondan geri bildirimler alınır. Sadece müşteri değil ekip personelleri ve proje yöneticisinin de geri bildirimlerinin alınması son derece önemlidir

.**İletişim:** Ekibin iletişim eksikliği, yanlış anlaşılmalar projeyi olumsuz etkiler. XP bunu ortadan kaldırmayı amaçlar. En iyi iletişim yolu olan yüz yüze iletişime önem verir.

**XP 12 farklı pratiği**

**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.**

**-SCRUM**

Scrum, Jeff Sutjerland ve Kin Schawaber tarafından 1990’ların ortalarında geliştirilmiştir. Günümüzde en çok kullanılan yazılım geliştirme metodudur. Yalnız yazılım sektöründe değil başka alanlarda başka sistemlerin geliştirilmesinde de kullanılır. Bir proje yönetim yaklaşımıdır. Dolayısıyla sadece yazılım sektöründe değil diğer sektörlerde, alanlarda uygulanabilir. Kompleks projeler küçük projelere bölünür ve bunlara **sprint** adı verilir. İhtiyaçların net bir şekilde tanımlanamadığı, karmaşık projeler için uygundur.



**Scrum’da 3 temel kavram bulunmaktadır:**

**- Roller (Roles)**

**Ürün Sahibi (Product Owner) :** Projenin en büyük yardımcısıdır denilebilir.

**Scrum Yöneticisi (Scrum Master) :** 5-9 arası kişiden meydana gelir. Sürekli iletişim halinde olmaları gerekir.

**Scrum Takımı (Scrum Team) :** Takımın Scrum’ın kurallarına, değerlerine uygun bir şekilde davranmasını, iş yapmasını sağlamakla yükümlüdür.

**- Toplantılar (Meetings)**

**Sprint (Koşu) Planlama (Sprint Planning):** Gereksinim listeleri tanımlanır ve Dağıtım gereksinimleri belirlenir. Dağıtım için takımlar belirlenir. Risk kontrol ve analizleri yapılır vs.

**Sprint (Koşu) Gözden Geçirme (Sprint Review)**

**Günlük Scrum Toplantısı** **(Daily Scrum) :** Scrum’ın spesifik özelliğidir ve mutlaka olmalıdır. Günlük 15 dakikalık kısa toplantılar yapılır. Tüm ekip katılır. Karşılaşılan sorunlar tartışılır, çözüme gidilir. Dünün analizi bugünün işleri ve gelecek günlerdeki oluşacak problemlere çözümler aranır.

**-Bileşenler/Araçlar (Artifacts)**

**Ürün Gereksinim Dokümanı (Product Bacaklar) :** Projesüresince yapılması gerekenler bu dokümana yazılır. Bu doküman sürekli değiştirilebilir. Kullanıcı hikâyelerinden (Uzer story) oluşur ve kullanıcı bakış açısından bakılır.

**Sprint (Koşu) Dokümanı (Sprint Backlog) :** Ürün gereksinim dokümanının devamıdır. Sadece takım tarafından değiştirilebilir. Sprint sonunda çalışabilen parça elde etmek amaçtır.

**Sprint Kalan Zaman Grafiği (Burndown Chart):** Projenin tamamında nereye gelindiğini gösterir.

**SCRUM GÜNÜMÜZDE NEDEN POPÜLER ?**

* Oldukça kompleks ve ihtiyaçları net tanımlanmamış, tanımlanamayan projeler için ideal olması
* Değişen koşulları hızlı bir şekilde adapte olması (Değişen Teknoloji vs.)
* Zamandan ve paradan kaybın minimum düzeyde olması
* Yapılan günlük, ayaküstü, kısa toplantılarda geçmişin analiz edilip problemlerin çözülmesi, hataların erken fark edilip iyileştirilmesi, gelecekte karşılaşabilecek engeller için çalışmaların yapılması doğrudan projenin başarısına etki eder.
* Müşterinin de sürece dâhil edilmesiyle müşterinin taleplerine en uygun ürün üretilmiş olur ve müşteri memnuniyeti sağlanmış olur.
* Basitliğin önem görmesi

Gibi etkenlerden dolayı Scrum günümüzde bu kadar popüler durumda.

**MODELLERİN KARŞILAŞTIRILMASI**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model/Özellik** | **Şelale Modeli** | **V Süreç**  **Modeli** | **Helezonik**  **(Spiral)**  **Model** | **Artımsal Model** | **Çevik Model** | **Kodla ve Düzelt** | **Evrimsel Geliştirme** | **Prototipleme** |
| **Zaman** | Çok uzun | Uzun | Uzun | Uzun | Kısa | Çok uzun | Uzun | Uzun |
| **Basitlik** | Basit | Orta | Karmaşık | Orta | Karmaşık | basit | Karmaşık | Orta |
| **Risk Duyarlılığı** | Yüksek | Yüksek değil | düşük | Düşük | Azaltılmış | Yüksek | Yüksek | Düşük |
| **Esneklik** | Katı | Düşük | Çok Esnek | Çok Esnek | Çok Esnek | Katı | Düşük | Çok Esnek |
| **Bakım** | Düşük Bakımlı | Düşük Bakımlı | Var | Var | Var | Düşük | Düşük | Var |
| **Dokümantasyon ve Eğitim gerekliliği** | Zorunluluk | Evet | Evet ama çok değil | Evet ama çok değil | Evet | Evet | Evet | Evet ama çok değil |
| **Değişiklik yapma** | Zor | Zor | Kolay | Kolay | Zor | Kolay | Zor | Kolay |
| **Maliyet** | Maliyetli | Maliyetli | Maliyetli | Düşük | Çok yüksek | Düşük | Düşük | Düşük |
| **Yeniden Kullanılabilirlik** | Düşük İhtimal | Düşük İhtimal | Mümkün | Mümkün | Evet | Düşük | Düşük | Mümkün |
| **Uygulama** | Kolay | Kolay | Zor | Kolay | Kolay | Kolay | Kolay | Kolay |
| **Başarı garantisi** | Düşük | Orta | Yüksek | Yüksek | Çok Yüksek | Düşük | - | Orta |

**KAYNAKÇA**

[**https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742**](https://medium.com/@omerharuncetin/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-543c7879a742)

[**https://medium.com/@batincetin44/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-2342547d0840**](https://medium.com/@batincetin44/yaz%C4%B1l%C4%B1m-ya%C5%9Fam-d%C3%B6ng%C3%BC-modelleri-2342547d0840)

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

[**https://www.codex.com.tr/yazilim-gelistirme-modelleri**](https://www.codex.com.tr/yazilim-gelistirme-modelleri)

[**https://slideplayer.biz.tr/slide/15310022/**](https://slideplayer.biz.tr/slide/15310022/)

[**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://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/@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**](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)

[**https://furkanalniak.com/yazilim-muhendisligi-yazilim-yasam-donguleri/**](https://furkanalniak.com/yazilim-muhendisligi-yazilim-yasam-donguleri/)

**Doç. Dr. Deniz Kılınç, Bakırçay Üniversitesi Yazılım Mühendisliğine Giriş Dersi 2, 3 ve 4. Hafta Ders Sunumları**