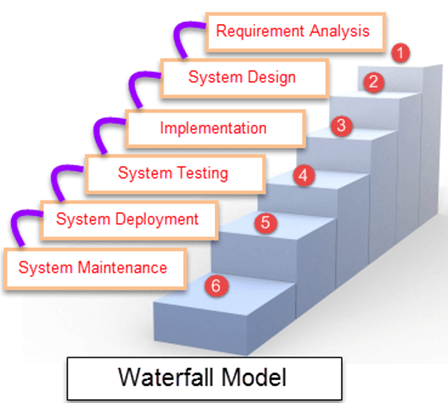
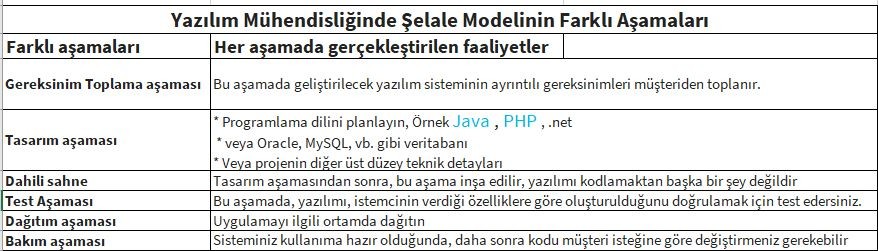
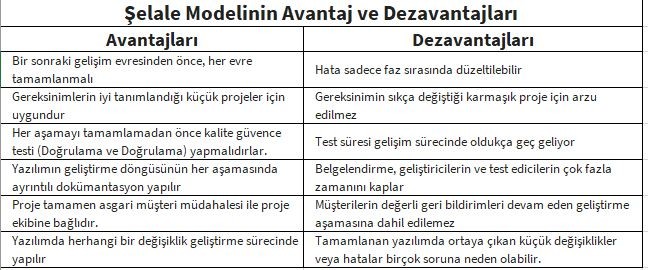
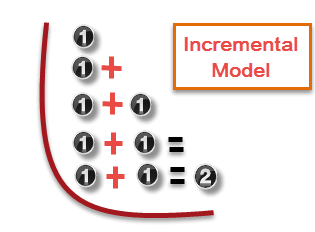
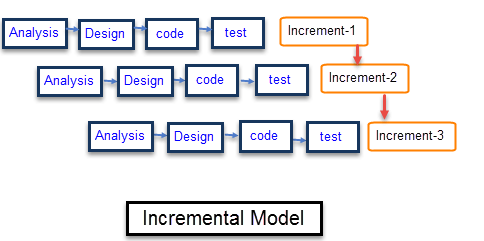
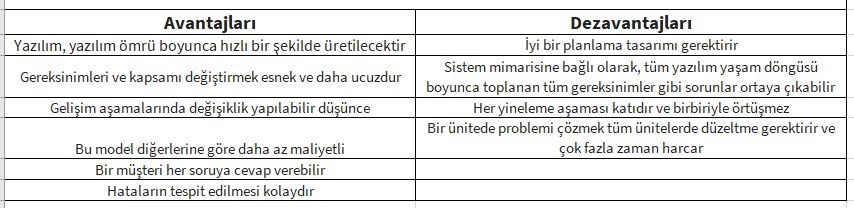
**SDLC (Yazılım Geliştirme Yaşam Döngüsü) Nedir, Aşamaları ve Modeli**  
  
**SDLC Nedir?**  
**Yazılım Geliştirme Yaşam Döngüsü, inşa edilen yazılımın kalitesini ve doğruluğunu sağlayan yazılım oluşturmak için sistematik bir işlemdir. SDLC süreci, müşteri beklentilerini karşılayan yüksek kaliteli yazılım üretmeyi amaçlar. Yazılım geliştirme önceden belirlenmiş zaman dilimi ve maliyetinde tamamlanmalıdır.  
  
SDLC, belirli yazılımların nasıl planlanacağını, oluşturulacağını ve korunacağını açıklayan ayrıntılı bir plandan oluşur. SDLC yaşam döngüsünün her aşamasının bir süreci ve bir sonraki aşamaya beslenen teslimatları vardır.  
  
Bu Yazılım Geliştirme Yaşam Döngüsünün Başlıkları Aşağıdaki Gibidir.  
  
\* SDLC Nedir  
\* SDLC‘de Şelale Modeli Nedir  
\* SDLC‘de Artımlı Model  
\* Spiral Model Nedir  
\* RAD Modeli Nedir  
\* Prototipleme Modeli Nedir  
\* Şelale vs Artıcı vs Spiral -Rad Modelleri  
\* Yetenek Olgunluk Modeli ( CMM )  
\* N Katmanlı ( Çok Katmanlı ) Mimari Modeli  
\* Tam Yığın Geliştirici Nedir  
\* İşlevsel Programlama Eğitimi**  
**\* Neden SDLC**  
İşte, bir yazılım sistemi geliştirmek için SDLC‘nin önemli olmasının başlıca nedenleri.  
  
\* Proje planlama, zamanlama ve tahmin için bir temel sunar  
\* Standart bir dizi faaliyet ve teslimat için bir çerçeve sağlar  
\* Proje takibi ve kontrolü için bir mekanizmadır  
\* Proje planlamasının geliştirme sürecindeki tüm paydaşlara görünürlüğünü arttırır  
\* Artırılmış ve geliştirme hızını arttırın  
\* Geliştirilmiş müşteri ilişkileri  
\* Proje riskini azaltmanıza ve genel olarak proje yönetim planını azaltmanıza yardımcı olur  
  
**\* SDLC Aşamaları**  
Tüm SDLC süreci aşağıdaki aşamalara ayrılmıştır:  
  
https://i.hizliresim.com/EOqg79.png  
  
\* Gereksinim Toplama ve Analiz  
\* Fizibilite Çalışması  
\* Tasarım  
\* Kodlama  
\* Test Etme  
\* Kurulum ve Yerleştirme  
\* Bakım  
  
**Gereksinim toplama ve analiz**  
Gereklilik, SDLC sürecindeki ilk aşamadır. Sektördeki tüm paydaşlardan ve alan uzmanlarından gelen girdilerle kıdemli ekip üyeleri tarafından yürütülmektedir. Bu aşamada kalite güvence gereklilikleri için planlama ve ilgili risklerin tanınması da yapılmaktadır.  
  
Bu aşama, tüm projenin kapsamı ve projeyi tetikleyen beklenen konular, fırsatlar ve direktiflerin daha net bir görüntüsünü sunar.  
  
Gereksinimler Toplama aşaması, ayrıntılı ve kesin gereksinimler elde etmek için ekiplere ihtiyaç duyar. Bu, şirketlerin bu sistemin çalışmasını bitirmek için gerekli zaman çizelgesini tamamlamalarına yardımcı olur.  
  
**Fizibilite çalışması**  
Gereksinim analizi aşaması tamamlandıktan sonra, bir sonraki adım, yazılım gereksinimlerini tanımlamak ve belgelendirmektir. Bu süreç, ‘SRS‘ belgesi olarak da bilinen ‘Yazılım Gereksinimi Spesifikasyonu‘ belgesinin yardımı ile gerçekleştirilmiştir. Proje yaşam döngüsü boyunca tasarlanması ve geliştirilmesi gereken her şeyi içerir.  
  
**Temel olarak beş tür fizibilite kontrolü vardır**  
**\* Ekonomik**:Projeyi bütçe dahilinde tamamlayabilir miyiz?  
**\* Yasal**:Bu projeyi siber hukuk ve diğer düzenleyici çerçeve / uygunluk olarak ele alabilir miyiz.  
**\* İşlem Yapılabilirliği**: Müşteriden beklenen işlemleri yaratabilir miyiz?  
**\* Teknik**:Geçerli bilgisayar sisteminin yazılımı destekleyip desteklemediğini kontrol etmeniz gerekiyor?  
**\* Program**:Projenin verilen program dahilinde tamamlanıp tamamlanamayacağına karar verin.  
  
**Tasarım**  
Bu aşamada, sistem ve yazılım tasarım belgeleri, gereksinim belgesine göre hazırlanır. Bu, genel sistem mimarisini tanımlamaya yardımcı olur.  
  
Bu tasarım aşaması, modelin bir sonraki aşaması için girdi görevi görür.  
  
Bu aşamada geliştirilen iki tür tasarım dokümanı vardır:  
  
Üst Düzey Tasarım (HLD)  
\* Her modülün kısa açıklaması ve adı  
\* Her modülün işlevselliği hakkında bir taslak  
\* Arayüz ilişkisi ve modüller arası bağımlılıklar  
\* Temel unsurlarıyla birlikte tanımlanan veritabanı tabloları  
\* Teknoloji detayları ile birlikte komple mimari şemaları  
  
Düşük Seviyeli Tasarım (LLD)  
\* Modüllerin fonksiyonel mantığı  
\* Tür ve boyut içeren veritabanı tabloları  
\* Arayüzün eksiksiz detayı  
\* Her tür bağımlılık sorununu giderir  
\* Hata mesajlarının listelenmesi  
\* Her modül için eksiksiz giriş ve çıkışlar  
  
**Kodlama**  
istem tasarım aşaması sona erdiğinde, bir sonraki aşama kodlanır. Bu aşamada, geliştiriciler seçilen programlama dilini kullanarak kod yazarak tüm sistemi oluşturmaya başlar. Kodlama aşamasında, görevler birimlere veya modüllere ayrılır ve çeşitli geliştiricilere atanır. Yazılım Geliştirme Yaşam Döngüsü sürecinin en uzun aşamasıdır.  
  
Bu aşamada, Geliştirici önceden tanımlanmış belirli kodlama kurallarına uymak zorundadır. Ayrıca, kodu oluşturmak ve uygulamak için derleyici, tercüman, hata ayıklayıcı gibi programlama araçlarını kullanmaları gerekir.  
  
**Test Etme**  
Yazılım tamamlandığında ve test ortamında dağıtılır. Test ekibi tüm sistemin işlevselliğini test etmeye başlar. Bu, tüm uygulamanın müşteri ihtiyacına göre çalıştığını doğrulamak için yapılır.  
  
Bu aşamada, KG ve test ekibi geliştiricilere bildirdikleri bazı hataları / kusurları bulabilir. Geliştirme ekibi hatayı düzeltir ve yeniden test için KG’ya geri gönderir. Bu işlem, yazılım hatasız, kararlı ve bu sistemin iş gereksinimlerine göre çalışana kadar devam eder.  
  
**Kurulum / Yerleştirme**  
Yazılım test aşaması sona erdiğinde ve sistemde herhangi bir hata veya hata kalmadığında, son dağıtım işlemi başlar. Proje yöneticisi tarafından verilen geri bildirimlere dayanarak, nihai yazılım piyasaya sürülür ve varsa dağıtım sorunları için kontrol edilir.  
  
**Bakım**  
Sistem dağıtıldıktan ve müşteriler gelişmiş sistemi kullanmaya başladıktan sonra, 3 etkinlik gerçekleştirilir  
  
\* Hata düzeltme - hiç test edilmemiş bazı senaryolar nedeniyle hatalar rapor edildi  
\* Yükseltme - Uygulamayı Yazılımın daha yeni sürümlerine yükseltme  
\* Geliştirme - Mevcut yazılıma yeni özellikler ekleme  
  
Bu SDLC safhasının ana odağı, ihtiyaçların karşılanmaya devam edilmesini ve sistemin birinci aşamada belirtilen şartnameye göre çalışmaya devam etmesini sağlamaktır.  
  
**Popüler SDLC modelleri**  
İşte, SDLC yaşam döngüsünün en önemli aşamalarından bazıları:  
  
**Şelale Modeli: Şelale yaygın olarak kabul edilen bir SDLC modelidir. Bu yaklaşımda, yazılım geliştirme sürecinin tamamı çeşitli aşamalara ayrılmıştır. Bu SDLC modelinde, bir fazın sonucu, bir sonraki fazın girişi olarak işlev görür.**  
Bu SDLC modeli, daha sonraki aşamalarda yapılması gerekenleri belgeleyen daha önceki aşamalarla belgelendirme amaçlıdır.  
  
**Artımlı Yaklaşım**:Artımlı model ayrı bir model değildir. Esasen bir dizi şelale döngüsüdür. Gereksinimler, projenin başlangıcında gruplara ayrılmıştır. Her grup için, yazılım geliştirmek üzere SDLC modeli izlenir. SDLC işlemi tekrarlanır, her sürüm tüm gereksinimler karşılanıncaya kadar daha fazla işlevsellik ekler.  
  
Bu yöntemde, her döngü önceki yazılım sürümü için bakım aşaması olarak işlev görür. Artımlı modelde değişiklik yapmak, geliştirme çevrimlerinin üst üste gelmesini sağlar. Bundan sonra bir sonraki döngü tamamlanmadan sonraki döngü başlayabilir.  
  
**V-Modeli**:Bu tip SDLC model testlerinde ve geliştirilmesinde faz paralel olarak planlanır. Bu nedenle, yan tarafta doğrulama aşamaları ve diğer tarafta doğrulama aşaması vardır. V-Model Kodlama aşamasına katılır.  
  
**Agile Model**:Çevik metodoloji, herhangi bir projenin SDLC sürecinde gelişim ve test arasındaki etkileşimin devam etmesini teşvik eden bir uygulamadır. Çevik yöntemde, tüm proje küçük artımlı yapılara ayrılmıştır. Bu yapıların tümü yinelemelerde sağlanmıştır ve her yineleme bir ila üç hafta sürer.  
  
**Spiral modeli**:Spiral model risk odaklı bir süreç modelidir. Bu SDLC modeli, ekibin bir şelale, artımlı, şelale gibi bir veya daha fazla işlem modelinin elemanlarını benimsemesine yardımcı olur.  
  
Bu model prototip modelinin ve şelale modelinin en iyi özelliklerini benimsemektedir. Spiral metodoloji, tasarım ve geliştirme faaliyetlerinde hızlı prototipleme ve eşzamanlılığın bir birleşimidir.  
  
**Big Bang modeli**:Big bang modeli, yazılım geliştirme ve kodlamadaki her türlü kaynağa ya da çok az planlama yapmadan odaklanmaktadır. Gereksinimler geldiğinde anlaşılır ve uygulanır.  
  
Bu model, birlikte çalışan daha küçük boyutlu geliştirme ekibiyle küçük projeler için en iyi sonucu verir. Ayrıca akademik yazılım geliştirme projeleri için de faydalıdır. İhtiyaçların bilinmediği veya son teslim tarihi verilmediği ideal bir modeldir.

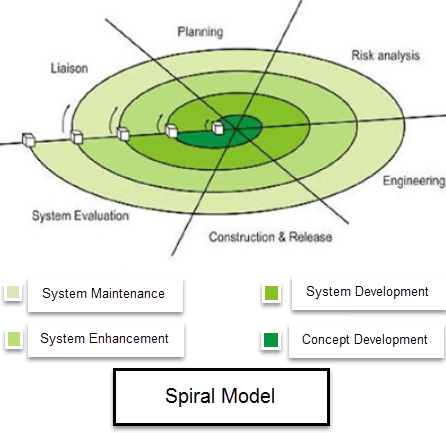
**Şelale Modeli Nedir**  
Şelale Modeli, yazılım gelişimini farklı aşamalara bölen sıralı bir modeldir. Her faz, SDLC fazı sırasında spesifik aktivite gerçekleştirmek için tasarlanmıştır. 1970 yılında Winston Royce tarafından tanıtıldı.  
  
  
  
  
  
  
  
  
**SDLC Şelale Modeli ne zaman kullanılır**  
Şelale modeli ne zaman kullanılabilir  
  
\* Gereksinimler sık #8203; #8203;sık değişmiyor  
\* Uygulama karmaşık ve büyük değildir  
\* Proje kısa  
\* Gereksinim açık  
\* Çevre kararlıdır  
\* Kullanılan teknoloji ve aletler dinamik değildir ve kararlıdır.  
\* Kaynaklar mevcut ve eğitilmiş

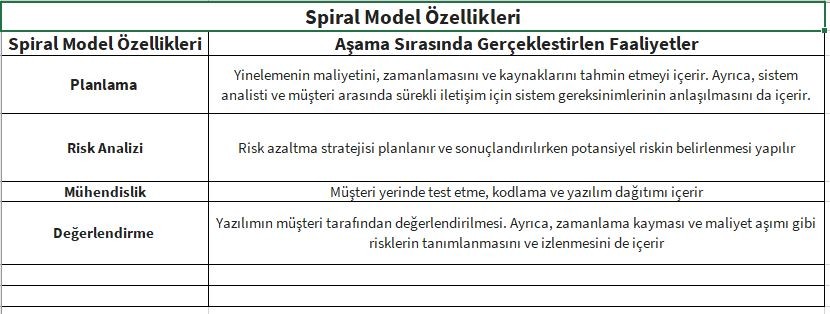


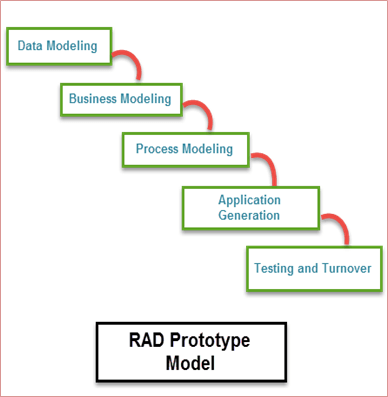
**SDLC‘de Artımlı Model**  
  
Artımlı Model, gereksinimlerin yazılım geliştirme döngüsünün çoklu bağımsız modüllerine bölündüğü bir yazılım geliştirme sürecidir. Artımlı geliştirme, analiz tasarımı, uygulama, test / doğrulama, bakım adımlarından yapılır.  
  


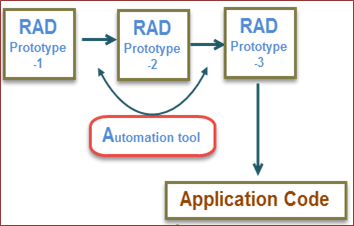
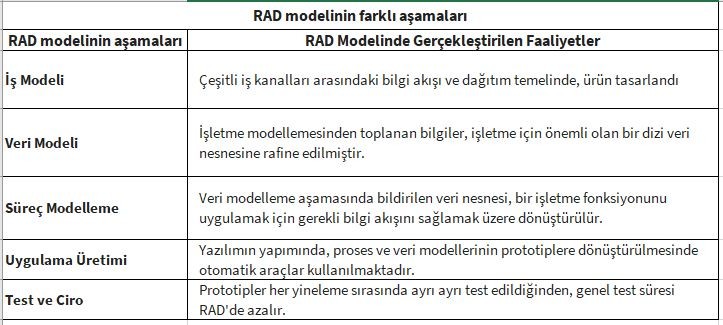
Her yineleme **gereksinimler, tasarım, kodlama ve test aşamalarından geçer** . Ve sistemin sonraki her bir sürümü, tüm tasarlanmış işlevler yerine getirilinceye kadar önceki sürüme işlev ekler.  
  


lk artış sağlandığında sistem üretime alınır. İlk artış genellikle temel gereksinimlerin ele alındığı bir temel üründür ve sonraki artışlara ek özellikler eklenir. Çekirdek ürün müşteri tarafından analiz edildikten sonra, bir sonraki artış için plan geliştirme söz konusudur.  
  
**Artımlı modülün özellikleri şunları içerir:**  
\* Sistem geliştirme birçok mini geliştirme projesine bölünmüştür  
\* Kısmi sistemler, son bir toplam sistemi üretmek için art arda inşa edilmiştir.  
\* İlk önce en yüksek öncelik gereksinimi aranır  
\* Gereksinim geliştirildikten sonra, bu artış için gereklilik dondurulur  
  
**Artımlı modeller ne zaman kullanılır**  
\* Sistemin gereksinimleri açıkça anlaşılıyor  
\* Bir ürünün erken tahliyesi talebi ortaya çıktığında  
\* Yazılım mühendisliği ekibi çok iyi vasıflı olmadığında veya eğitilmediğinde  
\* Yüksek riskli özellikler ve hedefler söz konusu olduğunda  
\* Bu tür bir metodoloji web uygulaması ve ürün bazlı şirketler için daha fazla kullanılmaktadır.  
  
**Artımlı Modelin Avantajları ve Dezavantajları**  


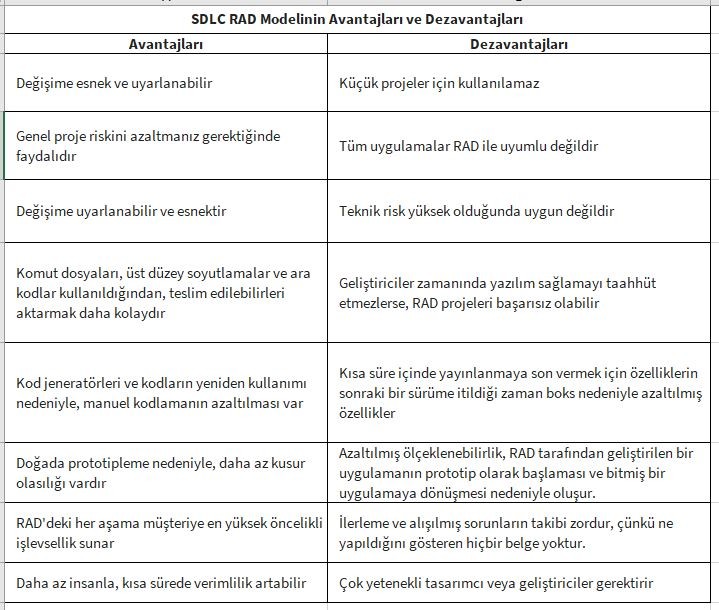
**Spiral Model Nedir**  
Spiral Model bir şelale modeli ve yinelemeli modelin bir birleşimidir. Spiral modeldeki her aşama bir tasarım hedefi ile başlar ve müşterinin ilerlemeyi gözden geçirmesiyle sona erer. Spiral model ilk olarak 1986 yılında Barry Boehm tarafından belirtildi.  
  
Spiral-SDLC modelindeki geliştirme ekibi, küçük bir gereksinim kümesiyle başlar ve bu gereksinimler kümesi için her geliştirme aşamasından geçer. Yazılım mühendisliği ekibi, uygulama üretim aşamasına hazır olana kadar her artan spiralde ek gereksinim için işlevsellik ekler.  
  


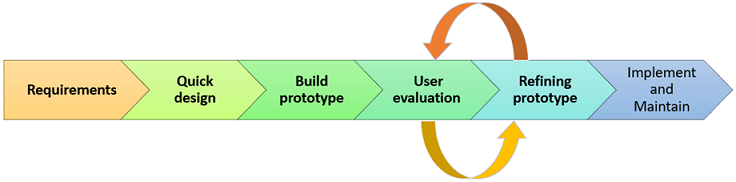
  
  
**Spiral Metodları ne zaman kullanılır**  
\* Proje büyük olduğunda  
\* Sürümlerin sık olması gerektiğinde  
\* Bir prototip oluşturulması uygulanabilir olduğunda  
\* Risk ve maliyet değerlendirmesi önemli olduğunda  
\* Orta ve yüksek riskli projeler için  
\* Gereksinimler belirsiz ve karmaşık olduğunda  
\* Değişiklikler herhangi bir zamanda gerekebilir  
\* Ekonomik önceliklerdeki değişiklikler nedeniyle uzun vadeli proje taahhüdü mümkün olmadığında  
  


**RAD (Hızlı Uygulama Geliştirme) Modeli Nedir**  
RAD veya Hızlı Uygulama Geliştirme süreci, şelale modelinin benimsenmesidir; kısa sürede yazılım geliştirmeyi hedefler. RAD yinelemeyi takip et  
  
SDLC RAD modeli aşağıdaki aşamalara sahiptir  
  
\* İş modeli  
\* Veri Modellemesi  
\* Süreç Modelleme  
\* Uygulama Üretimi  
\* Test ve Ciro  
  


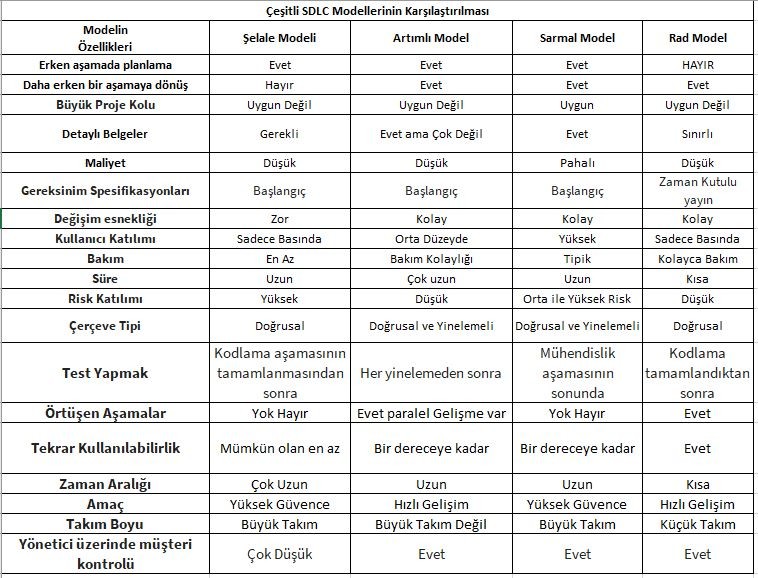
Girdi-çıktı kaynağına ve bilginin hedefine odaklanır. Küçük parçalar halinde projeler sunmayı vurguluyor; daha büyük projeler bir dizi daha küçük projeye bölünür. RAD modelinin temel özellikleri, şablonların, araçların, işlemlerin ve kodun yeniden kullanımına odaklanmasıdır.  
  
  
  


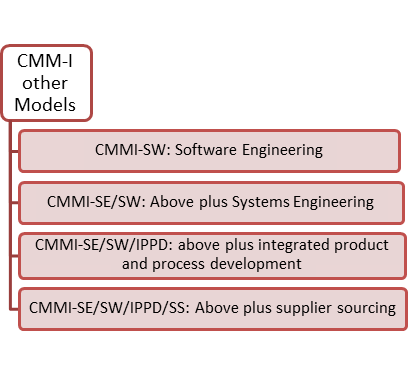
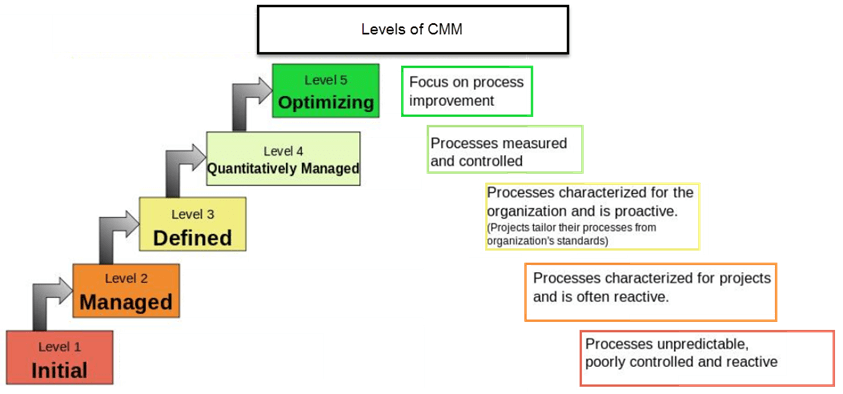
**RAD Metodolojisi ne zaman kullanılır**  
\* Bir sistemin kısa sürede üretilmesi gerektiğinde (2-3 ay)  
\* Gereksinimler bilindiğinde  
\* Kullanıcının tüm yaşam döngüsü boyunca ne zaman yer alacağı  
\* Teknik risk daha az olduğunda  
\* 2-3 ayda modüler hale olduğunda

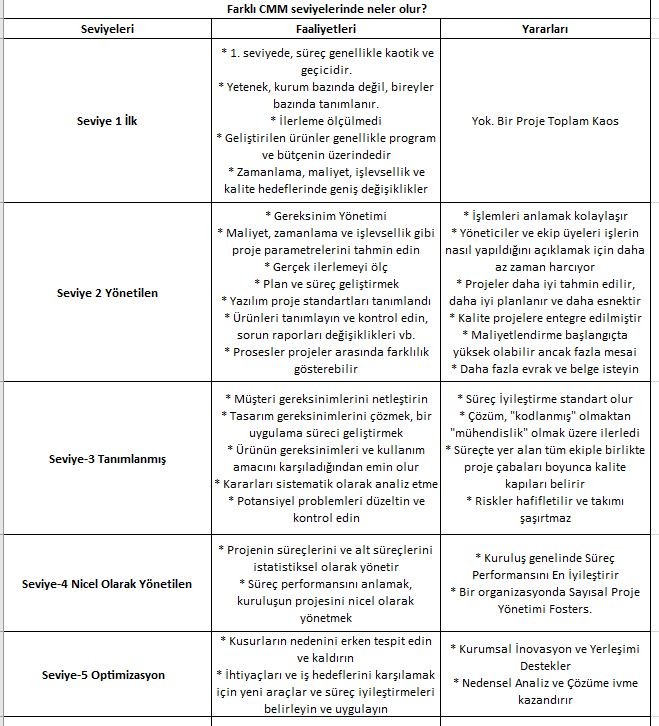
\* Bir bütçe, kod oluşturma için otomatik getirilebilecek bir sistem oluşturma zorunluluğu araçların maliyeti ile birlikte modelleme için tasarımcıları karşılayacak kadar yüksek olduğunda. 

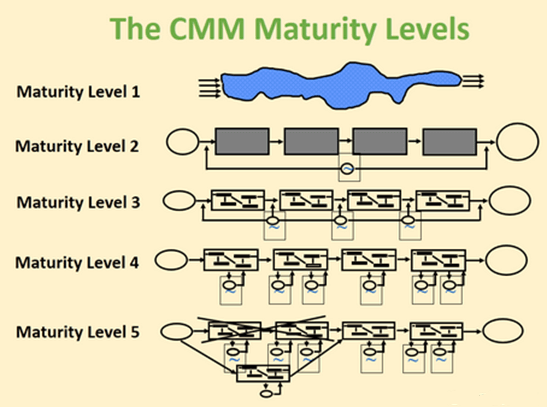
**Yazılım Prototipleme Modeli Nedir**  
Prototip metodolojisi, bir prototipin yapıldığı, test edildiği ve daha sonra gerektiğinde kabul edilebilir bir prototip elde edilinceye kadar yeniden yapıldığı bir Yazılım Geliştirme modeli olarak tanımlanır. Aynı zamanda son sistemi üretmek için bir üs oluşturur.  
  
Yazılım prototipleme modeli, projenin gereksiniminin bilinmediği senaryolarda en iyi sonucu verir. Geliştirici ve istemci arasında gerçekleşen yinelemeli, deneme ve hata yöntemidir.  
  
**Prototipleme Model Aşamaları**  
  


Prototipleme Modeli, aşağıdaki gibi altı SDLC sırası izlemiştir:  
  
Gereksinimler toplama ve analiz: Bir prototipleme modeli ihtiyaç analizi ile başlar. Bu aşamada, sistemin gereksinimleri ayrıntılı olarak tanımlanmaktadır. Süreç boyunca, sistemden beklentilerinin ne olduğunu bilmek için sistem kullanıcıları ile görüşülür.  
  
Hızlı tasarım :İkinci aşama bir ön tasarım veya hızlı bir tasarımdır. Bu aşamada sistemin basit bir tasarımı yaratılır. Ancak, tam bir tasarım değil. Kullanıcıya sistem hakkında kısa bir fikir verir. Hızlı tasarım prototipin geliştirilmesine yardımcı olur.  
  
Bir Prototip Oluşturma :Bu aşamada, hızlı tasarımdan toplanan bilgilere dayanarak gerçek bir prototip tasarlanır. Gerekli sistemin küçük bir çalışma modelidir.  
  
İlk kullanıcı değerlendirmesi :Bu aşamada, önerilen sistem ilk değerlendirme için müşteriye sunulur. Çalışma modelinin güçlü ve zayıf yönlerini bulmaya yardımcı olur. Yorum ve öneri müşteriden toplanır ve geliştiriciye sunulur.  
  
Prototipin Test edilmesi :Kullanıcı mevcut prototipten memnun değilse, prototipi kullanıcının geri bildirimlerine ve önerilerine göre hassaslaştırmanız gerekir.  
  
Bu aşama, kullanıcı tarafından belirtilen tüm gereksinimler karşılanıncaya kadar bitmeyecektir. Kullanıcı geliştirilen prototipten memnun kaldığında, onaylanmış final prototipine dayanan son bir sistem geliştirilir.  
  
Ürünü Uygulayın ve Bakımını Yapın :Son sistem son prototip bazında geliştirildikten sonra, tamamen test edilir ve üretime dağıtılır. Sistem, duruş sürelerini en aza indirmek ve büyük çapta arızaları önlemek için rutin bakımdan geçer.  
  
**Prototip Modelleri Türleri**  
Dört tip prototipleme modeli:  
  
\* Hızlı atın prototipleri  
\* Evrimsel prototip  
\* Artımlı prototip  
\* Aşırı prototip  
  
Hızlı Atış Prototipi  
Hızlı atış, ön gereksinime dayanır. Gereksinimin görsel olarak nasıl görüneceğini göstermek için hızlıca geliştirilmiştir. Müşterinin geri bildirimi, gereksinimdeki değişiklikleri yönlendirmeye yardımcı olur ve gereksinim temelleninceye kadar prototip yeniden oluşturulur.  
  
Bu yöntemde, geliştirilen bir prototip atılacak ve sonuçta kabul edilen prototipin bir parçası olmayacak. Bu teknik fikirleri keşfetmek ve müşteri istekleri için anında geri bildirim almak için kullanışlıdır.  
  
Evrimsel Prototipleme  
Burada geliştirilen prototip, son olarak kabul edilene kadar müşterinin geri bildirimlerine dayanan adım adım iyileştirilmektedir. Hem emek hem zamandan tasarruf etmenize yardımcı olur. Bunun nedeni, sürecin her etkileşimi için sıfırdan bir prototip geliştirmek, bazen çok sinir bozucu olabiliyor.  
  
Bu model iyi anlaşılmayan yeni bir teknolojiyi kullanan bir proje için faydalıdır. Ayrıca, her bir işlevin bir kez kontrol edilmesi gereken karmaşık bir proje için de kullanılır. Gereksinimin kararlı olmadığı ya da ilk aşamada açıkça anlaşılmadığı durumlarda faydalıdır.  
  
Artımlı Prototipleme  
Artımlı Prototiplemede, nihai ürün farklı küçük prototiplere bölünür ve bireysel olarak geliştirilir. Sonunda, farklı prototipler tek bir üründe birleştirilir. Bu yöntem, kullanıcı ile uygulama geliştirme ekibi arasındaki geri bildirim süresini azaltmada yardımcı olur.  
  
Aşırı Prototipleme  
Aşırı prototipleme yöntemi çoğunlukla web geliştirme için kullanılır. Üç ardışık aşamadan oluşur.  
  
\* Tüm mevcut sayfalara sahip temel prototip HTML biçiminde bulunur.  
\* Bir prototip servis katmanı kullanarak veri işlemini simüle edebilirsiniz.  
\* Hizmetler nihai prototipte uygulanır ve entegre edilir.  
  
**En iyi prototipleme uygulamaları**  
Prototip oluşturma sürecinde dikkat etmeniz gereken birkaç şey:  
  
\* Gereksinimler belirsiz olduğunda prototip kullanmalısınız.  
\* Planlı ve kontrollü Prototipleme yapmak önemlidir.  
\* Projeyi zamanında tutmak ve masraflı gecikmeleri önlemek için düzenli toplantılar çok önemlidir.  
\* Kullanıcılar ve tasarımcılar prototipleme sorunlarından ve tuzaklardan haberdar olmalıdırlar.  
\* Çok erken bir aşamada, bir prototipi onaylamanız ve ancak o zaman ekibin bir sonraki adıma geçmesine izin vermeniz gerekir.  
\* Yazılım prototipleme yönteminde, yeni fikirlerin uygulanması gerektiğinde önceki kararları değiştirmekten asla korkmamalısınız.  
\* Her sürüm için uygun adım boyutunu seçmelisiniz.  
\* Önemli özellikleri önceden uygulayın, böylece zamanınız tükenirse, yine de değerli bir sisteme sahip olursunuz.  
  
**Prototipleme Modelinin Avantajları**  
Prototipleme modellerini kullanmanın önemli avantajları / faydaları:  
  
\* Kullanıcılar aktif olarak gelişime katılıyor. Bu nedenle, yazılım geliştirme sürecinin ilk aşamasında hatalar tespit edilebilir.  
\* Prototipleme aynı zamanda bir risk azaltma faaliyeti olarak kabul edildiğinden başarısızlık riskini azaltmaya yardımcı olan eksik işlevler tanımlanabilir.  
\* Ekip üyesinin etkili iletişim kurmasına yardımcı olur  
\* Müşteri memnuniyeti var çünkü müşteri ürünü çok erken bir aşamada hissedebiliyor.  
\* Yazılım reddetme şansı neredeyse hiç olmayacak.  
\* Daha hızlı kullanıcı geri bildirimi, daha iyi yazılım geliştirme çözümleri elde etmenize yardımcı olur.  
\* İstemcinin, yazılım kodunun yazılım özelliği ile eşleşip eşleşmediğini karşılaştırmasını sağlar.  
\* Sistemde eksik olan işlevselliği bulmanıza yardımcı olur.  
\* Ayrıca karmaşık veya zor işlevleri de tanımlar.  
\* İnovasyonu ve esnek tasarımı teşvik eder.  
\* Basit bir modeldir, bu yüzden anlaşılması kolaydır.  
\* Model oluşturmak için uzman uzmanlara gerek yok  
\* Prototip, bir sistem spesifikasyonunun türetilmesi için bir temel oluşturur.  
\* Prototip, müşterinin ihtiyaçlarını daha iyi anlamaya yardımcı olur.  
\* Prototipler değiştirilebilir ve hatta atılabilir.  
\* Bir prototip ayrıca operasyonel şartnamelerin temelini oluşturur.  
\* Prototipler gelecekteki yazılım kullanıcıları için erken eğitim sunabilir.  
  
**Prototipleme Modelinin Dezavantajları**  
İşte prototip modelinin önemli eksileri / sakıncaları:  
  
\* Prototipleme yavaş ve zaman alıcı bir süreçtir.  
\* Bir prototip geliştirmenin maliyeti prototipin en sonunda atılmasından dolayı toplam bir atıktır.  
\* Prototipleme aşırı değişim taleplerini teşvik edebilir.  
\* Bazı zamanlar müşteriler uzun süre boyunca yineleme döngüsüne katılmak istemeyebilirler.  
\* Her prototip müşteri tarafından değerlendirildiğinde, yazılım gereksinimlerinde çok fazla değişiklik olabilir.  
\* Kötü dokümantasyon çünkü müşterilerin ihtiyaçları değişiyor.  
\* Yazılım geliştiricilerin, müşterilerin talep ettiği tüm değişiklikleri karşılaması çok zordur.  
\* Müşteriler erken prototip bir model gördükten sonra, asıl ürünün yakında kendisine ulaşacağını düşünebilirler.  
\* Müşteri, ilk prototipten memnun olmadığında nihai ürüne olan ilgisini kaybedebilir.  
\* Prototipleri hızlı bir şekilde oluşturmak isteyen geliştiriciler, alt standart geliştirme çözümleri oluşturmaya başlayabilir.

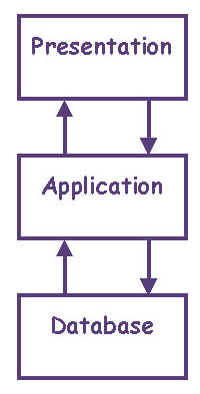
**elale vs. Artımlı vs. Spiral vs. Rad Model: Temel Farkları**  
  
Yazılım geliştirme döngüsü boyunca karmaşıklık seviyesini yönetmek için, yazılım şirketleri tarafından çeşitli SDLC modelleri uygulanmaktadır. Amaçları kaliteli bir yazılım ürünü sunmaktır, ancak her model kendi yazılım geliştirme yaklaşımları açısından benzersizdir. SDLC modellerini etkili bir şekilde öğrenmek için çeşitli Yazılım Mühendisliği modellerini karşılaştıracağız. Burada her bir modelin maliyet, zaman süresi, gereksinimler, bakım vb. Açısından diğerlerinden farklı olduğunu göreceğiz.  
  


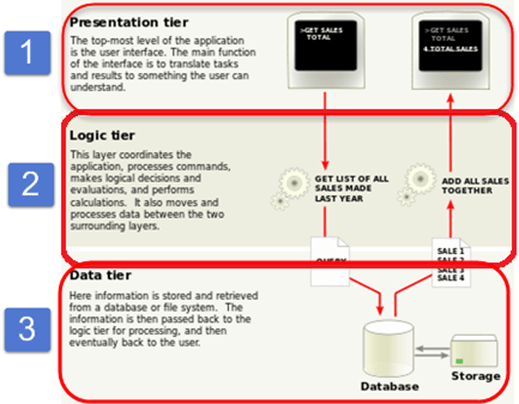
**CMM nedir**  
Yetenek Olgunluk Modeli, bir kurumun yazılım sürecinin vadesini ölçmede bir ölçüt olarak kullanılır.  
  
CMM, 80‘li yılların sonlarında Yazılım Mühendisliği enstitüsünde geliştirildi. Taşeronların çalışmalarını değerlendirmenin bir yolu olarak ABD Hava Kuvvetleri tarafından finanse edilen bir çalışma sonucunda geliştirilmiştir. Daha sonra 1991‘de yazılım geliştirmenin olgunluğunu değerlendirmek için oluşturulan CMM-SW modeline dayanarak, diğer birçok model de CMM-I ile bütünleştirilir.  
  
**Yetenek Olgunluk Modeli (CMM) Seviyeleri Nedir**  
\* ilk  
\* Tekrarlanabilir / Yönetilen  
\* Tanımlı  
\* Nicel Olarak Yönetilen  
\* Optimize  
  


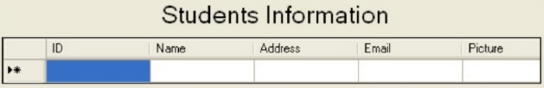


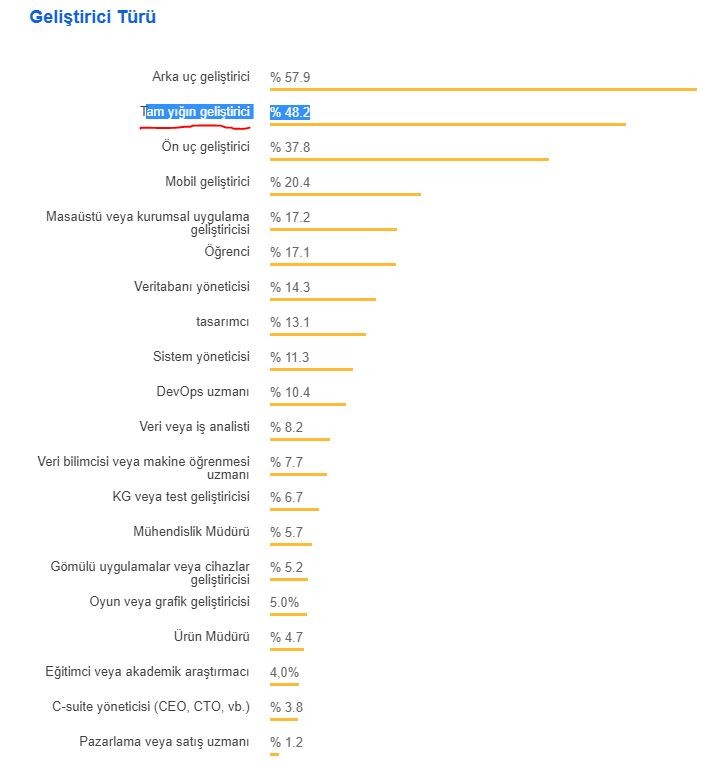
Aşağıdaki şemada, farklı CMM seviyesinde olanların resimli bir gösterimi verilmektedir.  
  


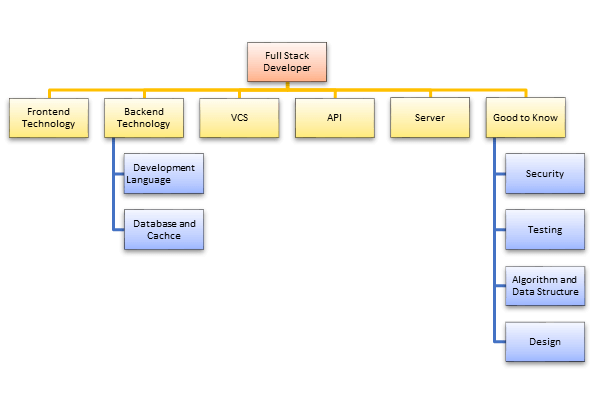
**CMM‘nin uygulanması ne kadar sürer**  
CMM, herhangi bir yazılım geliştirme şirketi için ürünün kalitesini korumak için en çok istenen işlemdir, ancak uygulanması beklenenden biraz daha uzun sürer.  
  
\* CMM uygulaması bir gecede gerçekleşmez  
\* Bu sadece bir "evrak" değil.  
\* Uygulama için tipik zamanlar  
- 3-6 ay hazırlık için  
- 6-12 ay uygulama için  
- 3 ay değerlendirme hazırlığı için  
- 12 ay - her yeni seviye için gt; b  
  
**CMM‘nin İç Yapısı**  
CMM‘deki her seviye, seviye-1 hariç, önemli işlem alanına veya KPA‘ya tanımlanır . Her bir KPA, toplu olarak gerçekleştirildiğinde yazılım kapasitesini geliştirmek için hayati önem taşıyan bir takım hedeflere ulaşan ilgili faaliyetler kümesini tanımlar.  
  
Farklı CMM seviyeleri için, KPA‘lar vardır, örneğin CMM model-2, KPA  
  
\* Reqm - İhtiyaç Yönetimi  
\* PP-Proje Planlama  
\* PMC - Proje İzleme ve Kontrol  
\* SAM- Tedarikçi Anlaşması Yönetimi  
\* PPQA Süreci ve Kalite Güvencesi  
\* CM-Yapılandırma Yönetimi  
  
Aynı şekilde, diğer CMM modelleri için de belirli KPA’lara sahipsiniz. Bir KPA‘nın uygulanmasının etkili, kalıcı ve tekrarlanabilir olup olmadığını bilmek, aşağıdaki esaslara göre haritalandırılır.  
  
**CMM Modellerinin Sınırlamaları**  
\* CMM, bir sürecin nasıl uygulanması gerektiği yerine nelerin ele alınması gerektiğini belirler.  
\* Her yazılım süreci iyileştirme olasılığını açıklamıyor  
\* Yazılım konularına odaklanır, ancak stratejik iş planlaması, teknolojilerin benimsenmesi, ürün yelpazesi oluşturulması ve insan kaynaklarını yönetmeyi düşünmez  
\* Bir örgütün ne tür bir iş yapması gerektiği hakkında bir şey söylemez  
\* Şu anda bir kriz geçiren projede CMM faydalı olmayacak  
  
**Neden CMM Kullanılmalıdır**  
Bugün CMM, yazılım endüstrisinde bir "onay damgası" olarak hareket etmektedir. Yazılım kalitesini yükseltmek için çeşitli şekillerde yardımcı olur.  
  
\* Tekrarlanabilir standart sürece doğru yönlendirir ve bu sayede işlerin nasıl yapılacağına dair öğrenme süresini kısaltır  
\* CMM uygulamak, geliştirme için standart protokol uygulamak anlamına gelir; bu, yalnızca ekibin zamandan tasarruf etmesine yardımcı olmakla kalmaz, aynı zamanda ne yapılacağı ve ne bekleneceği konusunda da net bir görüş sağlar.  
\* Kalite aktiviteleri, ayrı bir olay olarak düşünülmek yerine, proje ile iyi bir şekilde birleşiyor  
\* Proje ile ekip arasında bir banliyö görevi görür.  
\* CMM çabaları her zaman sürecin iyileştirilmesine yöneliktir.

**N-Tier Nedir**  
Bir **N-katmanı uygulama** programı, dağıtılmış ağ içinde üç ya da daha fazla ayrı bilgisayar arasında dağıtılır biridir.  
  
N-katmanlı en yaygın şekli, 3-katmanlı Uygulama‘dır ve üç kategoriye ayrılır.  
  
Kullanıcı bilgisayarında kullanıcı arayüzü programlama  
Daha merkezi bir bilgisayarda iş mantığı ve  
Veritabanını yöneten bir bilgisayarda gerekli veriler.  
Bu mimari model, Yazılım Geliştiricilere, maksimum esnekliğe sahip Yeniden Kullanılabilir uygulama / sistemler oluşturma olanağı sağlar.  
  
Olarak **N-katmanlı, "N",** - katman veya katmanların sayısı anlamına gelir gibi kullanılmaktadır **2 katmanlı, 3 katmanlı ya da 4-katmanlı, vb**. Aynı zamanda **“ Çok Katmanlı Mimari”** olarak da adlandırılır .  
  
**Çok katmanlı mimari** , endüstride kanıtlanmış bir yazılım mimarisi modelidir. Ölçeklenebilirlik, güvenlik, hata toleransı, yeniden kullanılabilirlik ve bakım kolaylığı için çözümler sunarak kurumsal düzeyde istemci-sunucu uygulamalarını desteklemek uygundur. Geliştiricilerin esnek ve yeniden kullanılabilir uygulamalar oluşturmasına yardımcı olur.  
  
**N-Tier Mimarisi**  
Sunum, uygulama ve veri tabanı katmanları - n-katmanlı bir sistemin şematik temsili burada gösterilmektedir.  
  


N Katmanlı Mimari Şeması  
Bu üç katman, gereksinimlere bağlı olarak farklı alt katmanlara ayrılabilir.  
  
**\* Dağıtılmış Ağ**:Ağ bilgisayarlarında bulunan bileşenlerin eylemlerini yalnızca iletiler ileterek koordine ettiği ve ilettiği bir ağ mimarisidir. Farklı düğümlerde bulunan birden fazla sistemden oluşan bir koleksiyondur, ancak kullanıcıya tek bir sistem olarak görünür.  
\* Farklı ağlar tarafından ayrı ayrı yönetilebilen tek bir veri iletişim ağı sağlar.  
\* Farklı istemcilerin bir tarafta LAN mimarisi içinde, diğer tarafında ise hizmet düğümleri içeren bir sunucu rafıyla birlikte yüksek hızlı anahtarlara bağlandıkları Dağıtılmış Ağ örneği.  
  
**\* İstemci-Sunucu Mimarisi:**İstemcinin (bir programın) bir sunucudan (başka bir program) hizmet talep ettiği bir mimari modeldir, yani internet üzerinden veya bir intranet aracılığıyla sağlanan bir istek yanıt hizmetidir.  
Bu modelde, Müşteri ağ üzerinden bir dizi eylem yürüten tek bir program / kod seti olarak görev yapacaktır. İken Sunucusu , diğer taraftan, istendiği gibi istemci sistemine sonuç kümelerini gönderir başka bir program, bir dizi.  
  
\* Bu durumda, istemci bilgisayar bir sunucudan bir servis veya bir kaynak talep etmek için bir son kullanıcıya bir arayüz sağlar ve diğer yandan sunucu daha sonra isteği işler ve sonucu son kullanıcıya görüntüler.  
\* İstemci-Sunucu Modeli örneği - bir ATM makinesi. Bir banka, uygulamayı büyük müşteri veritabanlarında işlemek için kullanılan sunucudur ve ATM makinesi, bazı basit uygulama işlemleriyle kullanıcı arayüzüne sahip olan müşteridir.  
  
**\* Platform:** Bilgisayar bilimleri veya yazılım endüstrisinde, bir platform, uygulama programının çalışabileceği bir sistemdir. İşlemcilerin / mikroişlemcilerin belirli işlemleri gerçekleştirmeleri için yerleşik bir yönergeye sahip bir donanım ve yazılım kombinasyonundan oluşur.  
\* Daha basit bir deyişle, platform herhangi bir uygulamanın belirli bir görevi elde etmek için çalıştırıp çalıştırabileceği bir sistem veya temeldir.  
\* Platform Örneği - 2 farklı platform örneği olarak Windows 2000 veya Mac OS X ile yüklenen kişisel bir makine.  
  
**\* Veri Tabanı:**Kolayca erişilebilmesi, yönetilebilmesi ve güncellenebilmesi için düzenli bir bilgi topluluğudur.  
Veritabanına Örnekler - MySQL, SQL Server ve Oracle Database bazı yaygın Db‘lerdir.  
  
**N-Tier Mimarisi Türleri**  
3 katmanlı Mimari, 2 katmanlı Mimari ve 1 katmanlı Mimari gibi farklı N-Katman Mimarisi türleri vardır .  
  
İlk önce, çok önemli olan 3 katmanlı Mimariyi göreceğiz.  
  
**3 Katmanlı Mimari**  
Aşağıdaki şemaya bakarak, 3 katmanlı mimarinin üç farklı katmana sahip olduğunu kolayca belirleyebilirsiniz .  
  
\*Sunum katmanı  
\*İş Mantığı katmanı  
\*Veritabanı katmanı  
  


İşte bu üç katmanı anlamak için basit bir öğrenci formu örneği aldık. İsim, Adres, E-posta ve Resim gibi bir öğrenci hakkında bilgi sahibidir.  
  
**Kullanıcı Arabirimi Katmanı veya Sunum Katmanı**  
  
  
  
**Sunum Katmanı**  
  
**private void DataGrid1\_SelectedIndexChanged (nesne gönderen,System.EventArgs e)**  
**{**  
**// Özellik katmanının**  
**nesnesi clsStudent objproperty = new clsStudent ();**  
**// İş katmanının**  
**nesnesi clsStudentInfo objbs = new clsStudentInfo ();**  
**// İş katmanı tarafından gönderilen verileri aldığımız veri kümesinin nesnesi**  
**DataSet ds = new DataSet ();**  
**// burada**  
**// özellik katmanının nesnesini kullanarak özelliğe değer koyuyoruz**  
**objproperty.id = int.Parse(DataGridl.SelectedItem.Cells [1] .Text.ToString ());**  
  
**// Bu kodda, iş katmanından bir fonksiyon çağırıyoruz ve**  
**// ID‘yi veritabanına taşıyacak olan özellik katmanınınnesnesini geçiyoruz.**  
**DS = objbs.GetAllStudentBsIDWise (objproperty);**  
  
**// Veriler yukarıdaki işlev tarafından veri setine geri döndürülen veriler**  
**// sunum katmanına yerleştiriliyor.**  
**txtId.Text = ds.Tables [0] .Rows [0] [0] .ToString ();**  
**txtFname.Text = ds.Tables [0] .Rows [0] [1] .ToString ();**  
**txtAddress.Text = ds.Tables [0] .Rows [0] [2] .ToString ();**  
**txtemail.Text = ds.Tables [0] .Rows [0] [3] .ToString ();**  
  
**Kod Açıklaması**  
Yukarıdaki kod, uygulamaların ön uç görünümünün temel tasarımını ve ayrıca diğer katmanların fonksiyonlarının çağrılmasını ve böylece birbirleriyle bütünleştirilebilmelerini tanımlar.  
  
**İşletme Erişim Katmanı**  
Bu, uygulama katmanındaki verileri kabul eden ve veri katmanına ileten iş katmanının işlevidir.  
  
\* İş mantığı, İstemci katmanı ve Veri Erişim Katmanı arasında bir arayüz görevi görür.  
\* Tüm iş mantığı - veri doğrulama, hesaplamalar, veri ekleme / değiştirme gibi işlemler iş mantığı katmanına yazılmıştır.  
\* İstemci ve veri katmanı arasında iletişimi hızlandırır ve kolaylaştırır  
\* Bir görevi tamamlamak için gereken uygun bir iş akışı etkinliği tanımlar.  
  
**// bu**  
**// uygulama katmanından verileri kabul eden ve veri katmanına ileten iş katmanının işlevidir**  
**genel sınıf clsStudentInfo**  
**{**  
**genel DataSet GetAllStudentBsIDWise (clsStudent obj)**  
**{**  
**DataSet ds = yeni DataSet ();**  
**ds = objdt.getdata\_dtIDWise (obj);**  
**// Veri katmanı fonksiyonunun çağrılması**  
**ds;**  
**}**  
**}**  
  
**Kod açıklaması**  
Kod, uygulama katmanının verilerini kabul edip veri katmanına ileten iş katmanının işlevini kullanıyor. İş katmanı kodları, sunum katmanında tanımlanan fonksiyonlar ile veri katmanında arabulucu olarak görev yapar ve işlevleri tam tersini yapar.  
  
**Veri erişim katmanı**  
Bu, veriyi iş katmanından alan ve gerekli işlemi veri tabanına yapan veri katmanı işlevidir.

**Tam Yığın Geliştirici Nedir**  
  
**Full Stack Geliştiricisi Nedir**  
Tam destekli bir web geliştiricisi, hem uygulamanın hem de ön ucunda çalışabilecek bir teknoloji uzmanıdır. Kişi, 3 aşamalı bir modelin her katmanına aşina olmalıdır. 3 katmanlı oluşur.  
  
\* Sunum katmanı (Kullanıcı arayüzü ile ilgilenen ana ön uç kısım),  
\* İş Mantığı Katmanı (veri doğrulama ile ilgilenen herhangi bir uygulamanın arka uç kısmı)  
\* Veri Tabanı Katmanı  
  
Bir Full-Stack Developer, tüm teknolojilerde uzman değildir. Ancak, profesyonelin sunucu tarafında olduğu gibi müşteri üzerinde de çalışması ve bir uygulama geliştirirken neler olup bittiğini anlaması beklenir. Tüm yazılım teknolojilerine gerçekten ilgi duymalı.  
  
Geliştirici profillerinin [Stackoverflow Anketi](https://insights.stackoverflow.com/survey/2018/)  


**Full-Stack Developer‘a Neden ihtiyacınız var**  
Tam yığın geliştirme uzmanı tutmanızın bazı önemli nedenleri aşağıda verilmiştir:  
  
\* Tam yığın geliştirici, sistemin her parçasının sorunsuz çalışmasını sağlamanıza yardımcı olur  
\* Tam yığın geliştirici, ekipteki herkese yardım sağlayabilir ve ekip iletişiminin zaman ve teknik maliyetlerini büyük ölçüde azaltır  
\* Bir kişi farklı roller oynarsa, şirketinizin personelini, altyapısını ve işletme maliyetini düşürür.  
  
**Tam Yığın Geliştirici olmak için gereken beceri kümeleri**  
  


Ön uç teknolojisi  
Tam yığın geliştirici HTML5, CSS3, JavaScript gibi temel ön uç teknolojilerin ustası olu. JQuery, LESS, Angular ve React Js gibi üçüncü parti kütüphanelerin bilgisine ihtiyaç duyar.  
  
Gelişim Dilleri  
Tam yığın enginer Java, Python, Ruby, gibi en az bir sunucu tarafı programlama dillerini bilmelidir.  
  
Veritabanı ve önbellek  
Çeşitli DBMS teknolojisi bilgisi, tam yığın geliştiricinin bir diğer önemli ihtiyacıdır. MySQL, MongoDB, Oracle, SQLServer bu amaç için yaygın olarak kullanılmaktadır. Vernik, Memcached, Redis gibi önbellekleme mekanizmalarının bilgisi bir artıdır.  
  
Temel tasarım yeteneği  
Başarılı bir tam yığın geliştirici olmak için, tasarım bilgisi de tavsiye edilir. Ayrıca, kişi temel prototip tasarımı ve UI / UX tasarımı prensibini bilmesi gerekmektedir.  
  
Sunucu  
Apache veya nginx sunucularını kullanılmalı. Linux‘taki iyi bir geçmiş, sunucuların yönetiminde muazzam bir şekilde yardımcı olur.  
  
Sürüm kontrol sistemi (VCS)  
Bir sürüm kontrol sistemi, tam yığın geliştiricilerin kod tabanında yapılan tüm değişiklikleri takip etmelerini sağlar. Git‘in bilgisi, tam yığın geliştiricilerin en son kodun nasıl alınacağını anlamalarına, kodun bölümlerini güncelleştirmelerine, diğer geliştiricilerin kodlarında herhangi bir şeyi bozmadan değişiklik yapmalarına yardımcı olur.  
  
API ile çalışma (REST ve SOAP)  
Web servisleri veya API bilgisi de tam yığın geliştiriciler için önemlidir. REST ve SOAP servislerinin kreasyonları ve tüketimi bilgisi arzu edilir.  
  
**Yazılım Yığını Nedir. Hangi yığını öğrenmeliyiz**  
Yazılım yığını, belirli bir sonuç üretmek için birlikte kullanılan programların toplamıdır. Bir işletim sistemi ve onun uygulamasını içerir. Örneğin, bir akıllı telefon yazılımı yığını, telefon uygulaması, web tarayıcıları ve varsayılan uygulamalarla birlikte işletim sistemi içerir.  
  
Tam bir yığın mühendisi için yukarıdaki beceri setleri listesi göz korkutucu olabilir. Kariyer hedeflerinize, projenize ve şirket gereksinimlerinize göre bir yazılım yığında ustalaşmanız gerekir. Aşağıda popüler yazılım yığınlarının bir listesi bulunmaktadır.  
  
Seçtiğiniz yığının ne olduğuna bakılmaksızın, mimaride benzerlik ve tasarım yığınlarını farklı yığınlarda bulabilirsiniz  
  
LAMP Yığını  
LAMP web servis yığınları için yaygın olarak kullanılan bir modeldir. "LAMP" adı dört açık kaynaklı bileşenin kısaltmasıdır.  
  
L = Linux: Açık kaynaklı bir işletim sistemi  
A = Apache: Yaygın olarak kullanılan web sunucusu yazılımı  
M = MySQL: Popüler açık kaynak veritabanı  
P = PHP: Sunucu tarafı açık kaynaklı kodlama dili  
  
Yukarıdaki bileşenler, birbirlerini desteklemektedir. Birçok popüler web sitesi ve web uygulaması LAMP yığında çalışır, Örnek: Facebook.  
  
MERN Yığını  
MERN, JavaScript tabanlı teknolojilerin bir koleksiyonudur:  
  
M = MongoDB: Popüler NoSQL veritabanı  
E = Express: Hafif ve taşınabilir web programı çerçevesi  
R = React: Kullanıcı arayüzleri oluşturmak için bir javascr\*ipt kütüphanesi  
N = Node.js: Sunucu tarafı JavaScript çalışma zamanı  
  
Bu uygulama şu anda büyük miktarda talep görmekte ve web uygulamaları geliştirmek için yaygın olarak kullanılmaktadır.  
  
MEAN Yığını  
MEAN Yığın Uygulama Geliştirme kullanımda artan bir eğilim görüyor. MEAN, kısaltmasıdır:  
  
M = MongoDB: nosql Veri Tabanı  
E = Express: Hafif ve taşınabilir web programı çerçevesi kullanımı kolay  
A = Angular.js: HTML5 ve JavaScript-web programlarını geliştirmek için güçlü bir çerçeve  
N = Node.is: sunucu tarafı JavaScript çalışma zamanı