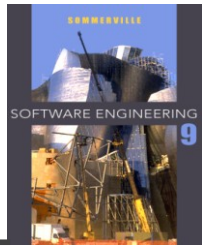
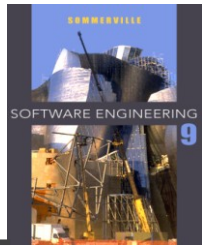


Chapter 6 – Mimari Tasarım

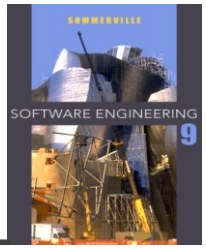
Lecture 1

Konular



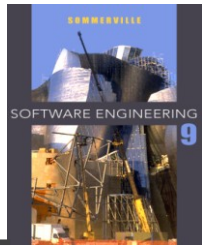
- ✧ Mimari Tasarım Kararları
- ✧ Mimari Bakış Açıları
- ✧ Mimari Desenler
- ✧ Uygulama Mimarileri

Yazılım Mimarisi



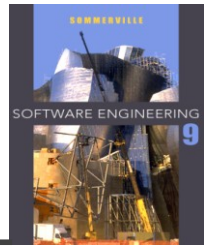
- Sistemi meydana getiren alt sistemlerin belirlenmesi için yapılan tasarım süreci ve alt sistemlerin kontrolü ve iletişimi için ortak çerçeve belirleme çalışması mimari tasarımdır.
- Sürecin çıktısı da yazılım mimarisidir.

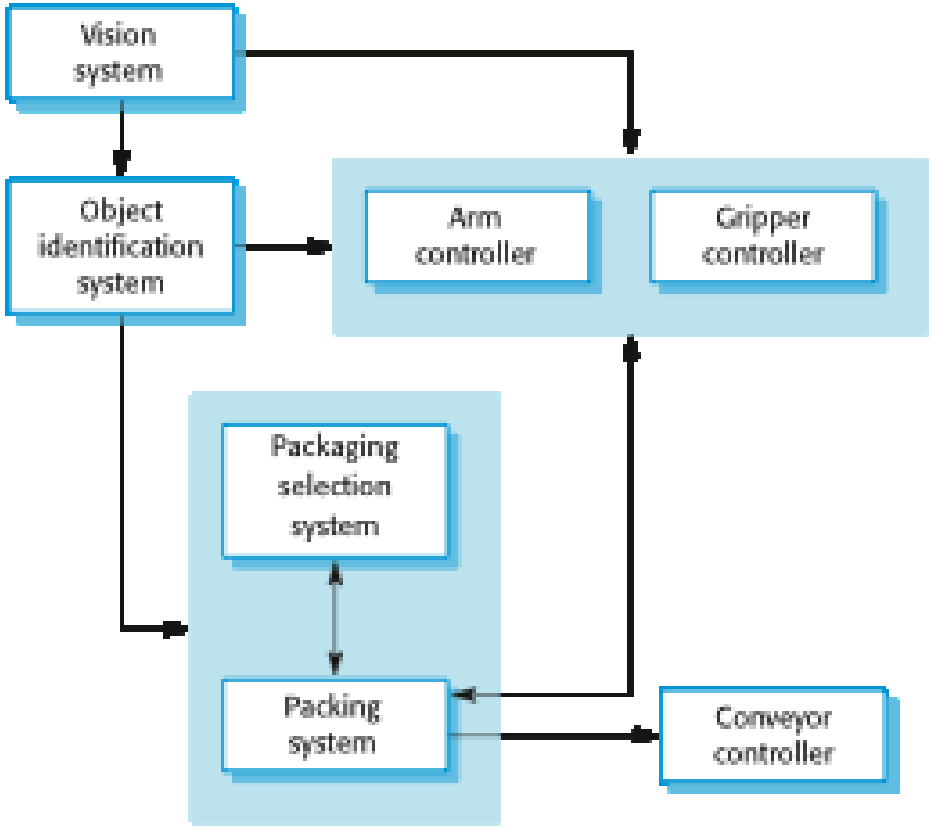
Mimari Tasarım



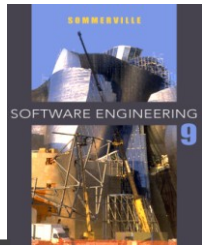
- Tasarım aşamasının ilk süreçlerinden biri
- Gereksinimlerle Tasarım arasında bağlantı görevi görür
- Genellikle bazı gereksinim analizleri ile birlikte yürütülür
- Sistemin büyük parçalarının ve ilişkilerinin belirlenmesini içerir

Paketleme Robotu Kontrol Sistemi için Bir Mimari Tasarım



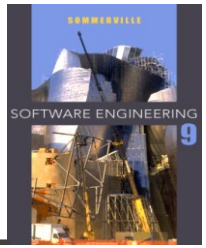


Mimari Soyutlama



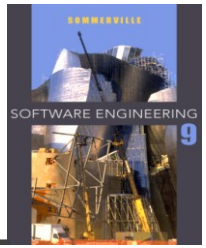
- Küçük çaplı mimari bireysel programlar ve bileşenleri ile ilgilenir
- Büyük çaplı mimari Diğer sistemleri, programları ve program bileşenlerini içeren kompleks kurumsal sistemlerle ilgilenir. Bu programlar farklı kuruluşların sahip olduğu farklı bilgisayarlara dağıtılmış olabilir.

Mimarinin açıkça ortaya konulmasının faydaları



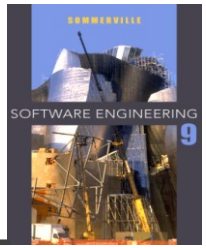
- ✧ Paydaş iletişimi
- ✧ Sistem analizi
  - Sistem fonksiyonel olmayan gereksinimleri karşılayabilir mi sorusunun cevabı.
- ✧ Büyük çaplı yeniden kullanım
  - Mimari, belirli bir gruptaki sistemler için yeniden kullanılabilir.

Mimari Tasarım Kararları



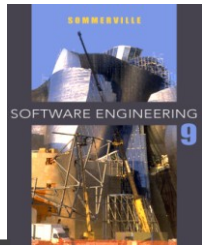
- Geliştirilmekte olan sisteme göre değişir
- Ancak bütün tasarım süreçleri için ortak noktalar vardır ve bu ortak noktalar fonksiyonel olmayan gereksinimleri etkiler

Mimari Tasarım Kararları



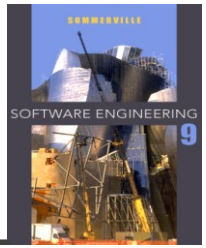
- ✧ Kullanılabilecek genel bir uygulama mimarisi var mı?
- ✧ Sistem dağıtık mı?
- ✧ Sistem modüllere nasıl bölünecek?
- ✧ Hangi kontrol stratejisi uygulanacak?
- ✧ Mimari tasarım nasıl değerlendirilecek?
- ✧ Mimari tasarım nasıl belgelenecek?

Mimarilerin yeniden kullanımı



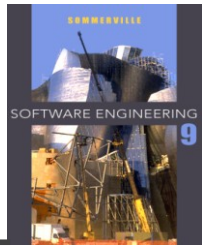
- Aynı alandaki sistemlerin gereksinimleri birbirine benzer.
- Belirli bir çekirdek mimari etrafında tasarlanan yazılımlar, kullanıcı gereksinimine göre değiştirilebilir.

Mimari ve sistem karakteristikleri



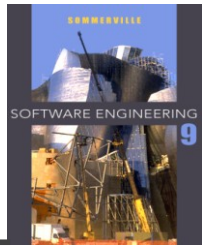
- Performans
  - Kritik işlemleri yerelleştir ve iletişimi en aza indir. Küçük bileşenler (fine-grain) yerine büyük bileşenler kullan.
- Güvenlik
  - Güvenliği önemli olan varlıkları daha iç katmanlarda tanımlamak şartı ile katmanlı yapı kullan
- Güvenilirlik
  - Güvenilirliği önemli olan özellikleri, sayıları az olan alt sistemlerde topla
- Erişilebilirlik
  - Hata toleransı için fazladan bileşenler ve mekanizmalar ekle
- Sürdürülebilirlik
  - Yenisi ile değiştirilebilir, küçük bileşenler kullan.

Yazılım Mimarisi Bakış Açıları



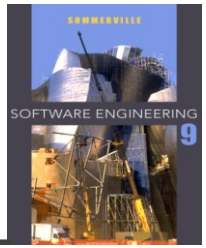
- Mantıksal bakış açısı, sistemin ana hatlarını nesneler ve sınıflar olarak özetler.
- İşleyiş bakış açısı, sistemin çalışma zamanını gösterir.
- Geliştirme bakış açısı, sistemin hangi parçalar halinde geliştirileceğini gösterir.
- Fiziksel bakış açısı, sistemi donanım olarak ve hangi yazılım parçalarının hangi işlemcilerde çalışacağını gösterir.

Mimari Desenler (Örüntüler)



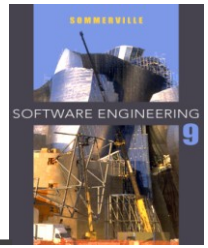
- Tecrübenin gösterimi, paylaşımı ve yeniden kullanımıdır.
- Bir mimari desen, farklı ortamlarda denenmiş iyi uygulamalardır.
- Desenler, ne zaman faydalı oldukları ve ne zaman olmadıkları ile ilgili bilgileri içermelidir.
- Sekmeli veya grafiksel olarak gösterilebilir.

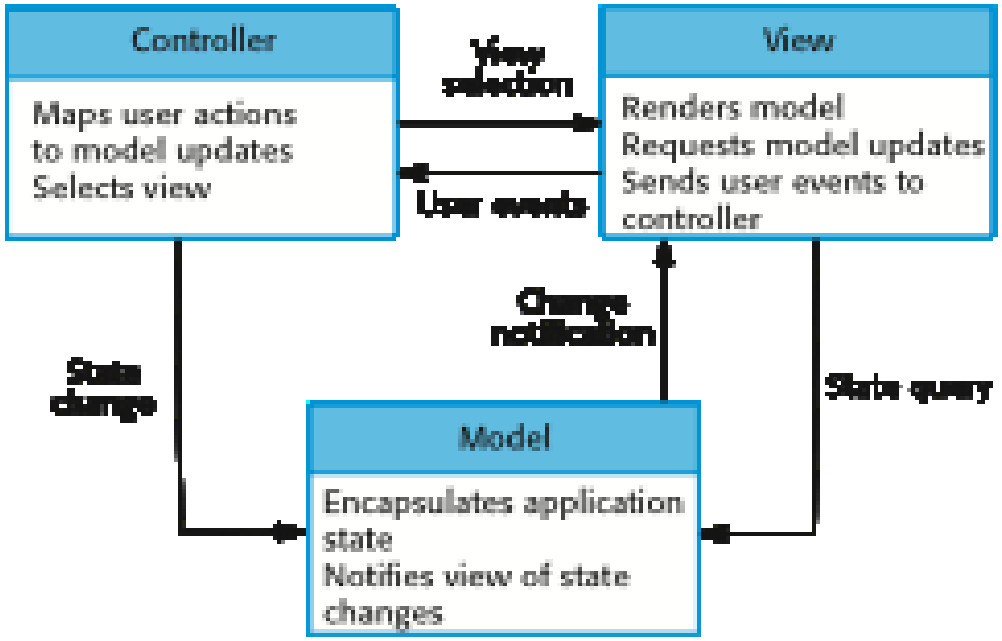
Model-View-Controller (MVC) deseni



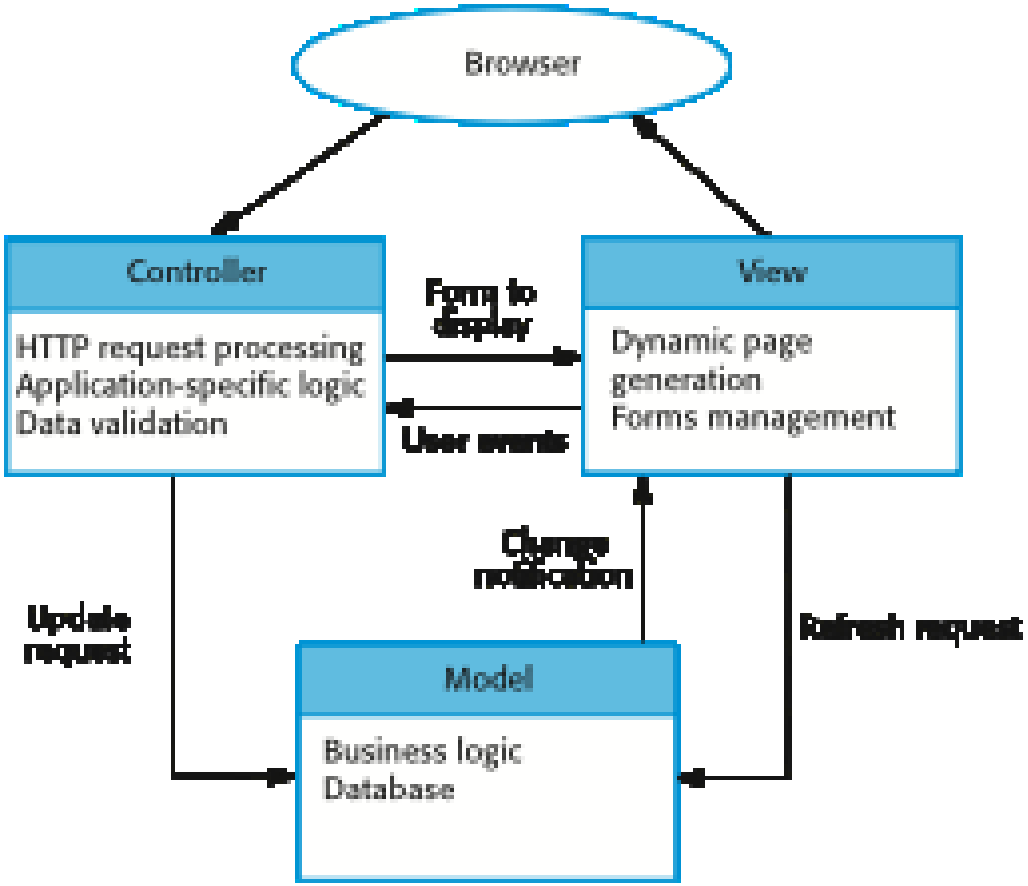
Desenin adı	MVC (Model-View-Controller)
Tanım	Sunum ve etkileşimi sistem verisinden ayırır. Sistem birbirleri ile ilişki içerisinde olan 3 mantıksal bileşen olarak yapılandırılır. Model bileşeni, sistem verisini ve veri üzerindeki ilgili işlemleri yönetir. View bileşeni, verinin kullanıcıya nasıl gösterileceğini tanımlar ve yönetir. Controller bileşeni, kullanıcı etkileşimlerini (ör: klavyeye basma, fareye tıklama) yönetir ve bu etkileşimleri View ve Model bileşenlerine iletir. (bkz. sonraki sayfadaki şekil.)
Örnek	(bkz. İki sonraki sayfadaki şekil.)
Ne zaman kullanılır	Veri ile etkileşimin ve veriyi gösterimin birçok yolu olduğu zaman kullanılır. Aynı zamanda, verinin gösterimi ve etkileşimle ilgili bilinmeyenlerin olduğu durumlarda kullanılır.
Avantajları	Verinin gösteriminin, verinin kendisinden bağımsız olarak değiştirilebilmesini sağlar.
Dezavantajları	Veri modeli ve etkileşimlerin basit olduğu durumlarda fazladan kod ve kod karmaşıklığı getirir.

Model-View-Controller deseninin organizasyonu

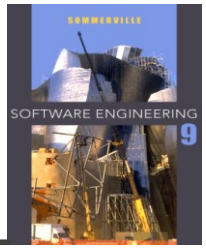




MVC desenini kullanan web uygulaması mimarisi

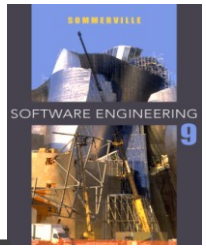


Katmanlı Mimari



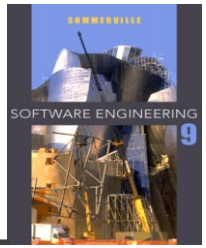
- Alt sistemler arasında interface oluşturmak için kullanılır.
- Sistemi katmanlar halinde tasarlar. Her bir katman farklı hizmetler sunar.
- Farklı katmanlardaki alt sistemlerin artırımlı olarak geliştirilmesini destekler. Bir katmanın arayüzü değiştiğinde yalnızca komşu katmanı ilgilendirir.

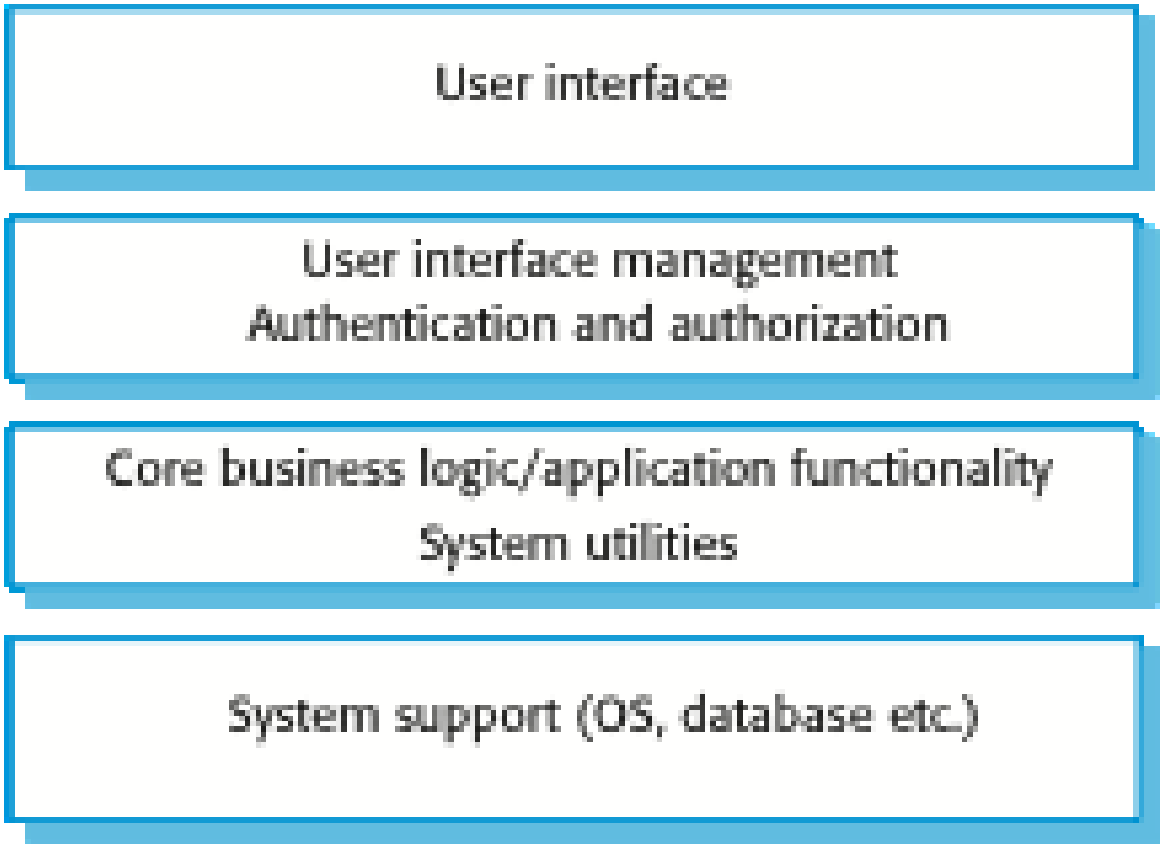
Katmanlı mimari deseni



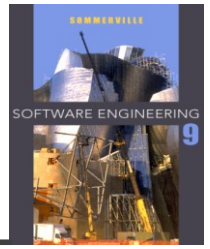
Desenin adı	Katmanlı mimari
Tanım	Sistemi, ilgili fonksiyonelliklerin aynı katmanda bulunduğu katmanlar yığını olarak organize eder. Her katman kendisinin üzerinde yer alan katmana hizmet veriri. Böylece, en alttaki katman çekirdek hizmetleri sunar.
Örnek	(bkz. İki sonraki sayfadaki şekil.)
Ne zaman kullanılır	Var olan sistemlerin üzerine yeni eklentiler yapmak için; geliştirme işleminin farklı takımlar arasında paylaştırıldığı durumlarda; çok katmanlı güvenlik ihtiyacının olması durumunda.
Avantajları	Bir katmanın tamamının, arayüze dokunmadan değiştirilebilmesini sağlar. Her katmanda fazladan sunulan “yetenekler” sistemin güvenilirliğini arttırır.
Dezavantajları	Pratikte, katmanlar arası keskin ayırımlar oluşturmak zordur ve en üstteki katmanın kendisinin altında yer alan katmanı atlayarak daha alt bir katmanla iletişime geçmesi gerekebilir. Birden çok katman performans problemi oluşturabilir.

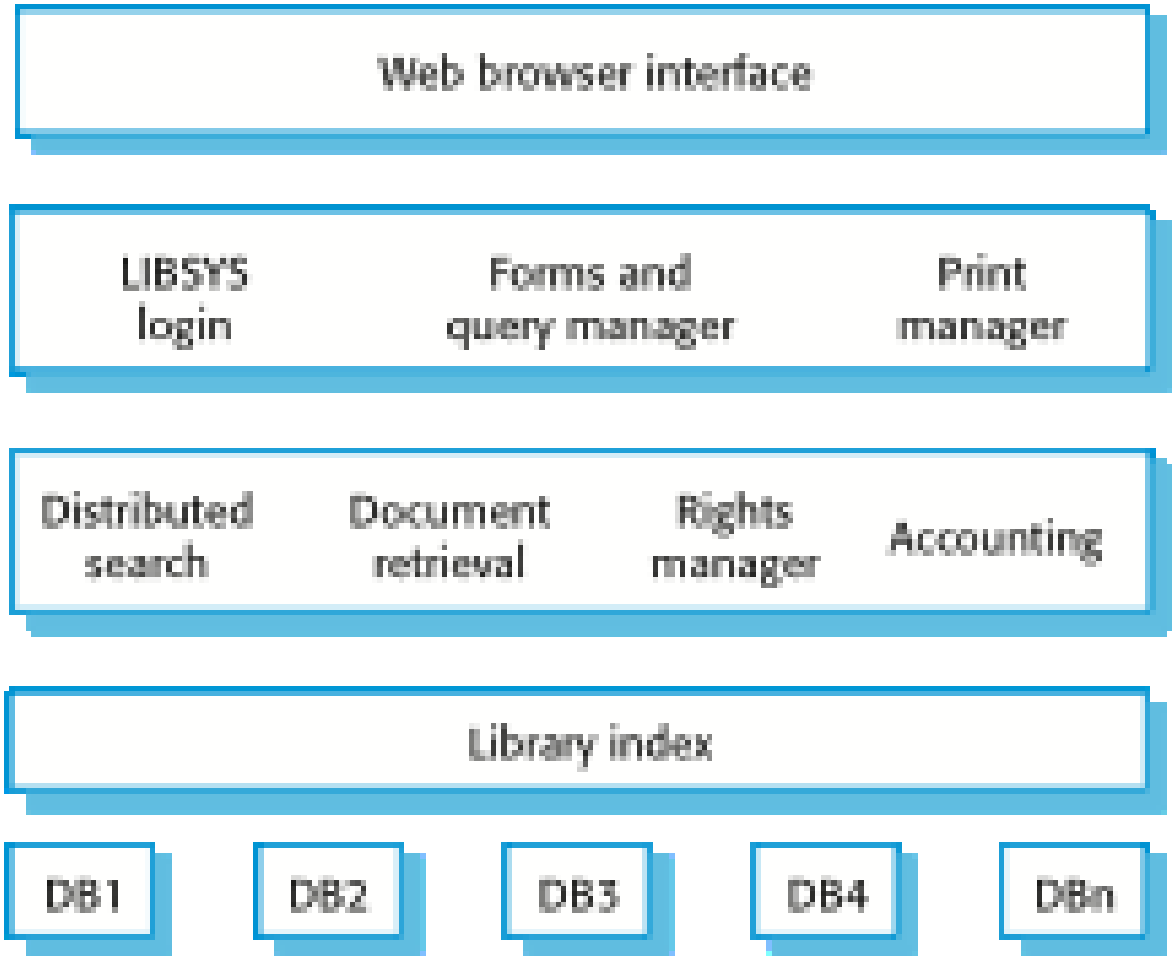
Genel bir katmanlı mimari

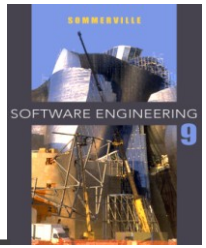




LIBSYS sisteminin mimarisi



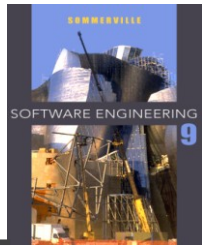




Chapter 6 – Mimari Tasarım

Lecture 2

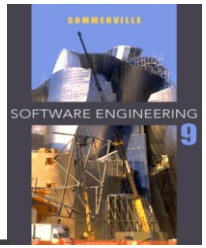
Repository (Depo) mimarisi



- Bir yazılımı oluşturan alt sistemlerin bilgi alışverişi gerekliyse bu iki şekilde olur:
  - Veriler merkezi bir veritabanında veya bir depoda paylaşılırlar ve alt sistemler bu verilere buralardan ulaşabilir.
  - Herbir alt sistem kendi veritabanını tutar ve diğer alt sistemlerle doğrudan veri alışverişinde bulunur.

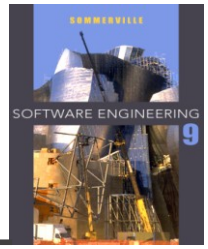
- Büyük miktarlarda verinin paylaşılması gerekiyorsa depo mimarisini kullanmak mantıklıdır.

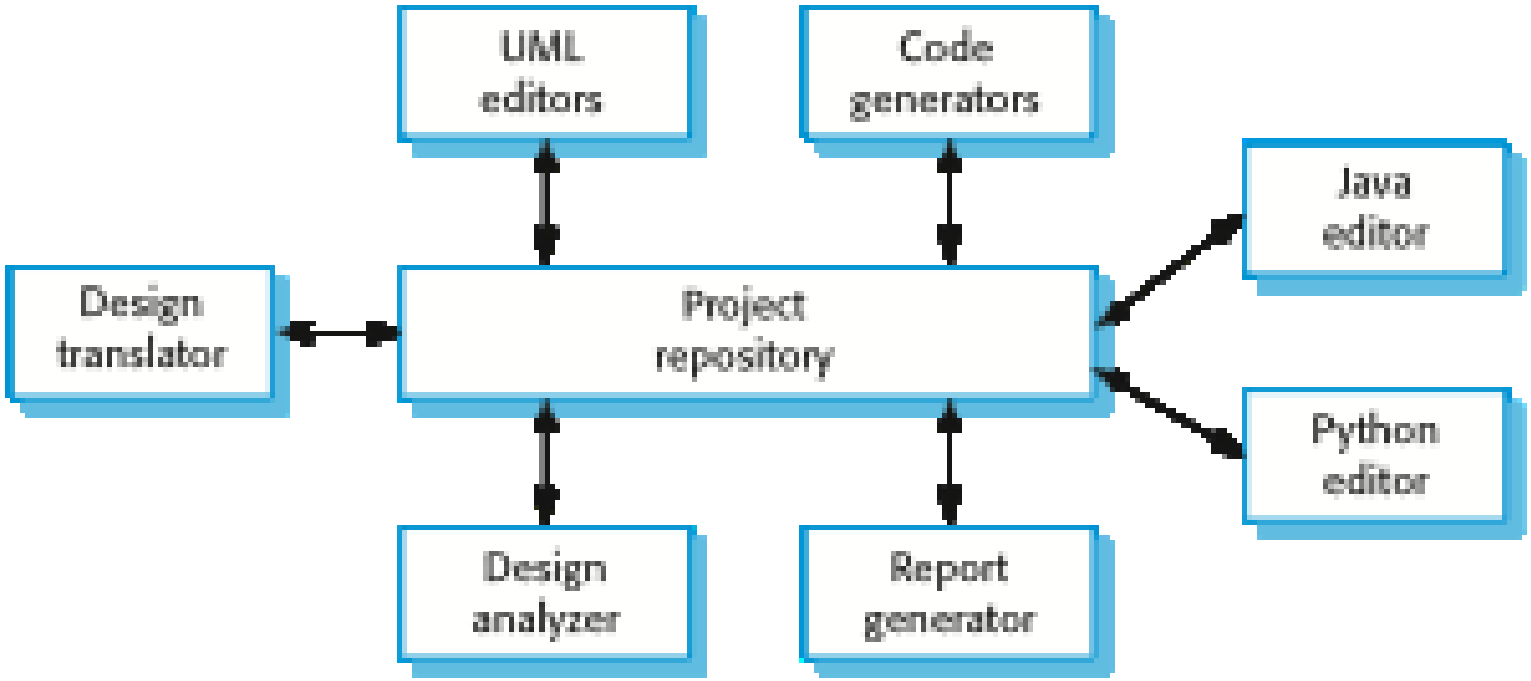
Repository (Depo) deseni



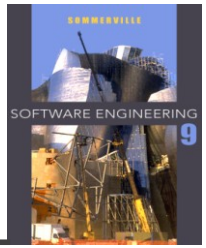
Desenin adı	Repository (Depo)
Tanım	Sistemdeki bütün veriler merkezi bir veri deposunda tutulur ve bütün sistem bileşenleri tarafından ulaşılabilir. Bileşenler doğrudan birbirleri ile etkileşime girmezler. Yalnızca veri deposunu üzerinden etkileşirler.
Örnek	(bkz. sonraki sayfadaki şekil.)
Ne zaman kullanılır	Uzun bir süre saklanacak çok miktarda veri üreten bir sistemde kullanılabilir. Ayrıca, verinin kontrol ettiği sistemlerde de kullanılabilir. Bu durumda depoya bir verinin dahil edilmesi bir işlemi tetikler.
Avantajları	Bileşenler bağımsız olabilir. Birbirlerinin varlıklarını bilmelerine bile ihtiyaçları yoktur. Bir bileşen tarafından yapılan bir değişiklik bütün bileşenler arasında yayılabilir. Bütün veriler tek merkezde oldukları için tutarlı bir biçimde yönetilebilirler.
Dezavantajları	Veri deposunda meydana gelecek bir hata bütün sistemi etkiler. Bütün iletişimi veri deposu üzerinden yapmak verimsiz olabilir. Veri deposunu birden fazla bilgisayar üzerine dağıtmak zor olabilir.

Bütünleşik Geliştirme Ortamı (IDE) İçin Bir Depo Mimarisi



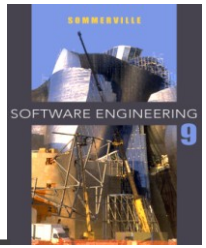


Client-server (İstemci-sunucu) mimarisi



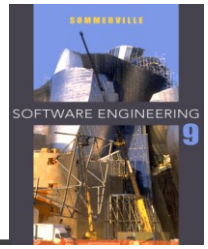
- Verinin ve işlem yükünün çeşitli bileşenler arasında nasıl dağıtılacağını gösteren dağıtık sistem modeli.
  - Bir bilgisayar üzerinde de gerçekleştirilebilir.
- Veri yönetimi, yazdırma hizmet gibi hizmetleri sunan bağımsız sunucular kümesi vb.
- Bu servisleri çağıran istemciler kümesi.
- İstemcilerin sunuculara erişmesini sağlayan bir ağ

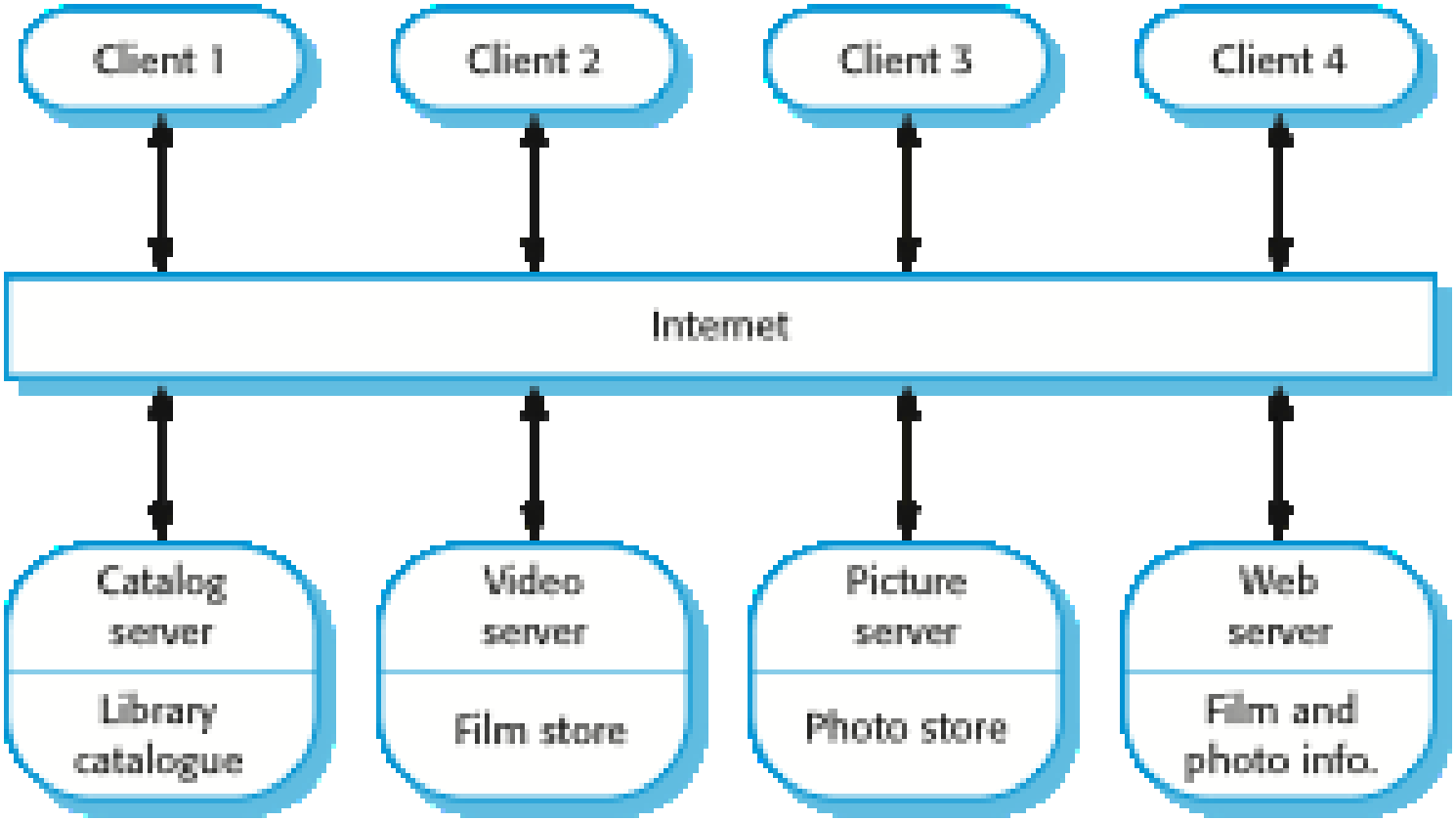
Client–server (İstemci-sunucu) deseni



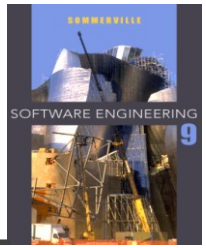
Desenin adı	İstemci-sunucu
Tanım	Bir istemci-sunucu mimarisinde sistemin fonksiyonellikleri servislere ayrılmıştır ve her bir servis farklı sunucular tarafından verilir. İstemciler bu servislerin kullanıcılarıdır ve bunları kullanma için sunuculara erişirler.
Örnek	(bkz. sonraki sayfadaki şekil.)
Ne zaman kullanılır	Paylaşılan bir veritabanındaki verilere farklı yerlerden erişme zorunluluğu olduğu zaman kullanılabilir. Aynı servisi veren birden fazla sunucu oluşturulabileceği için, sistem yükünün değişken olduğu durumlarda da kullanılabilir.
Avantajları	Bu modelin en temel avantajı sunucuların ağ üzerinde dağıtılmış olmasıdır. Genel bir fonksiyonellik (ör: yazdırma hizmeti) bütün istemciler tarafından erişilebilir durumdadır ve bütün servislerin bu servisi ayrıca barındırmasına gerek yoktur.
Dezavantajları	Her bir servis bir zafiyet noktası olarak düşünülebilir ve denial of service (DOS, hizmet aksatma) atakları sunucuyu başarısızlığa uğratabilir. Performans, ağa da bağlı olduğu için performansı tahmin etmek kolay olmaz. Eğer sunucular farklı kuruluşlar tarafından barındırılıyorsa yönetim sorun olabilir.

Film kütüphanesi için bir İstemci-sunucu mimarisi



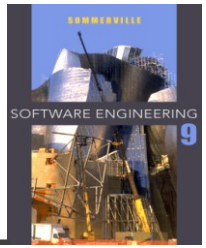


“Pipe and filter” mimarisi



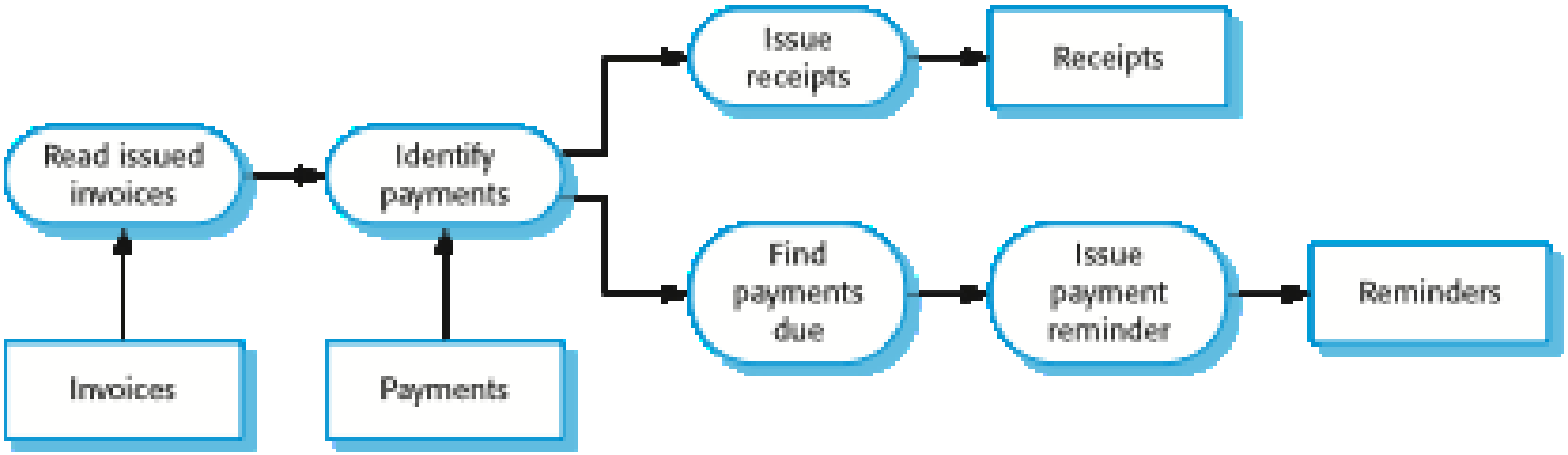
- Fonksiyonel yazılım birimleri kendilerine gelen girişleri işlerler ve çıkışlar üretirler.
- Bu yaklaşımın farklı türleri yaygın olarak kullanılır. Yazılım birimleri sıralı ise, bu tür bir yaklaşım veri işleme sistemlerinde sıklıkla kullanılan bir sıralı yığın veri işleme modelidir.
- Etkileşimli sistemler için uygun değildir.

pipe and filter deseni

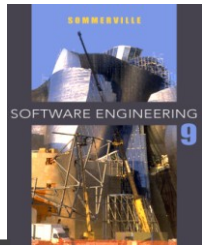


Desenin adı	Pipe and filter
Tanım	Bir sistemdeki verinin işlenme süreci organize bir biçimde yapılır ve her bir veri işleme birimi (filter) veri üzerinde belirli bir işlemi gerçekleştirir. Veri sanki bir boru içindeymiş gibi bir bileşenden diğerine akar.
Örnek	(bkz. sonraki sayfadaki şekil.)
Ne zaman kullanılır	Girişlerin ayrı aşamalarda işlendiği ve ilgili çıktıların elde edildiği veri işleme uygulamalarında (yığın ve transaction tabanlı) sıklıkla kullanılır.
Avantajları	Anlaması kolaydır ve bileşenlerin yeniden kullanımını destekler. İş akışı tarzı bir çok iş sürecine uygundur. Yeni bileşenlerin eklenmesi kolaydır. Sıralı veya eş zamanlı sistemler olarak geliştirilebilir.
Dezavantajları	Bileşenler arasında transfer edilen verinin formatı üzerinde anlaşmak gerekir. Her bir bileşen, üzerinde anlaşılan veri formatında giriş almalı ve çıkış üretmelidir. Bu, sistemin yükünü arttırır ve uyumsuz veri formatları ile bileşenlerin yeniden kullanılabilmesini olanaksız kılar.

pipe and filter mimarisine bir örnek

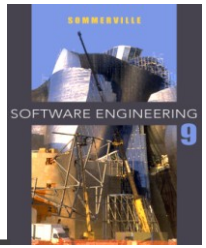


Uygulama mimarileri



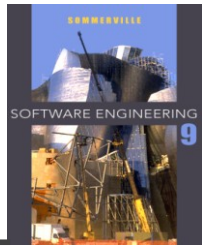
- Uygulama sistemleri kurumsal bir ihtiyacı karşılamak için geliştirilirler.
- İş dünyasının ortak ihtiyaçları olduğu için, bunların kullandığı uygulamalarının da ortak ihtiyaçları olma eğilimindedir.
- Genel bir uygulama mimarisi, düzenlenip belirli ihtiyaçları karşılayabilecek bir sistemin geliştirilmesi için için kullanılabilecek bir mimaridir.

Uygulama mimarilerinin kullanımı



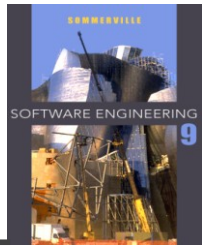
- Mimari tasarım için bir başlangıç noktasıdır.
- Tasarım aşaması için bir “yapılacak işler listesidir”
- Geliştirme takımının yapacağı işlerin organizasyonu için bir yoldur.
- Bileşenlerin yeniden kullanımını değerlendirmek için bir vasıtadır.
- Uygulama tipleri hakkında konuşmak için bir sözlüktür.

Uygulama tipleri için örnekler



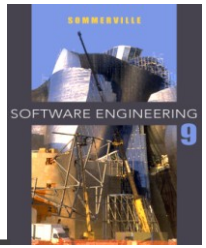
- ✧ Veri işleme uygulamaları
  - Verileri kullanıcı müdahalesi olmadan yığın olarak işleyen ve verinin kontrol ettiği uygulamalar
- ✧ Transaction (işlem/hareket/ticari işlem) işleme uygulamaları
  - Kullanıcı isteklerini işleyip bir veritabanı sistemindeki bilgiyi güncelleyen veri merkezli uygulamalar
- ✧ Olay işleme sistemleri
  - Sistemin çevresinde gerçekleşen olayların yorumlanmasına bağlı olarak harekete geçen uygulamalar.

Uygulama tipleri için örnekler



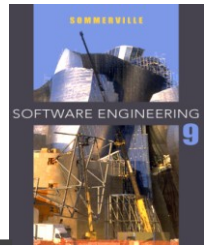
- Bizim ilgileneceğimiz sistemler Transaction (işlem/hareket/ticari işlem) uygulamaları olacak.
  - E-ticaret sistemleri
  - Rezervasyon sistemleri

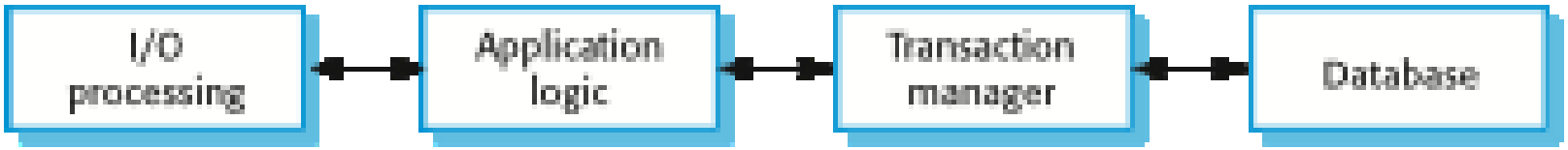
Transaction (işlem/hareket/ticari işlem) işleme uygulamaları



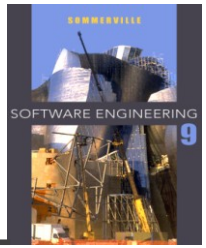
- Bir kullanıcı açısından bir transaction:
  - Belirli bir amaca yönelik uyumlu işlemler dizisi
  - Ör: Londra-Paris uçuşlarının zamanlarının bulunması
- Kullanıcı hizmet almak için asenkron olarak bir istekte bulunur ve bu istek “transaction yöneticisi” tarafından işlenir.

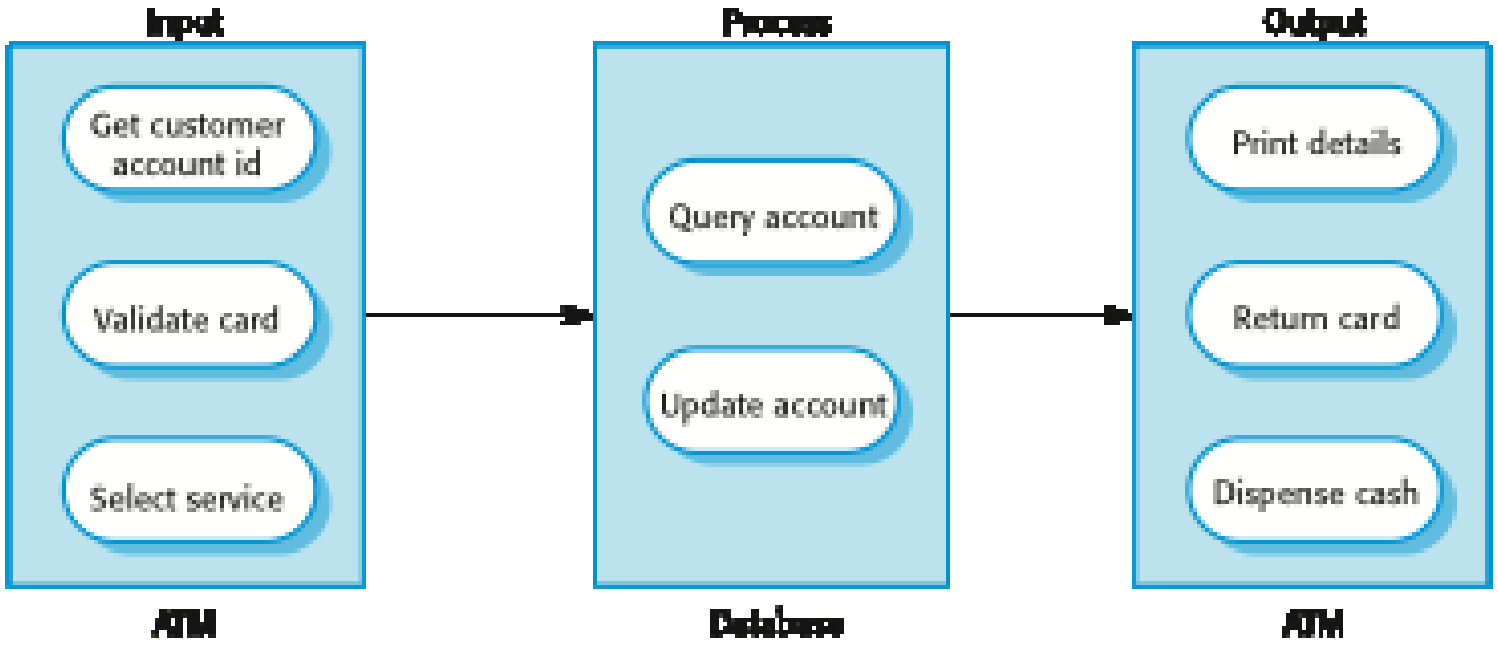
Transaction (işlem/hareket/ticari işlem) işleme uygulamalarının yapısı



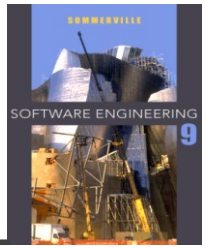


Bir ATM sistemi için yazılım mimarisi



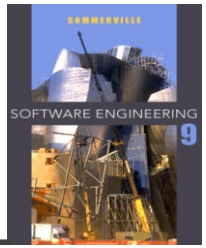


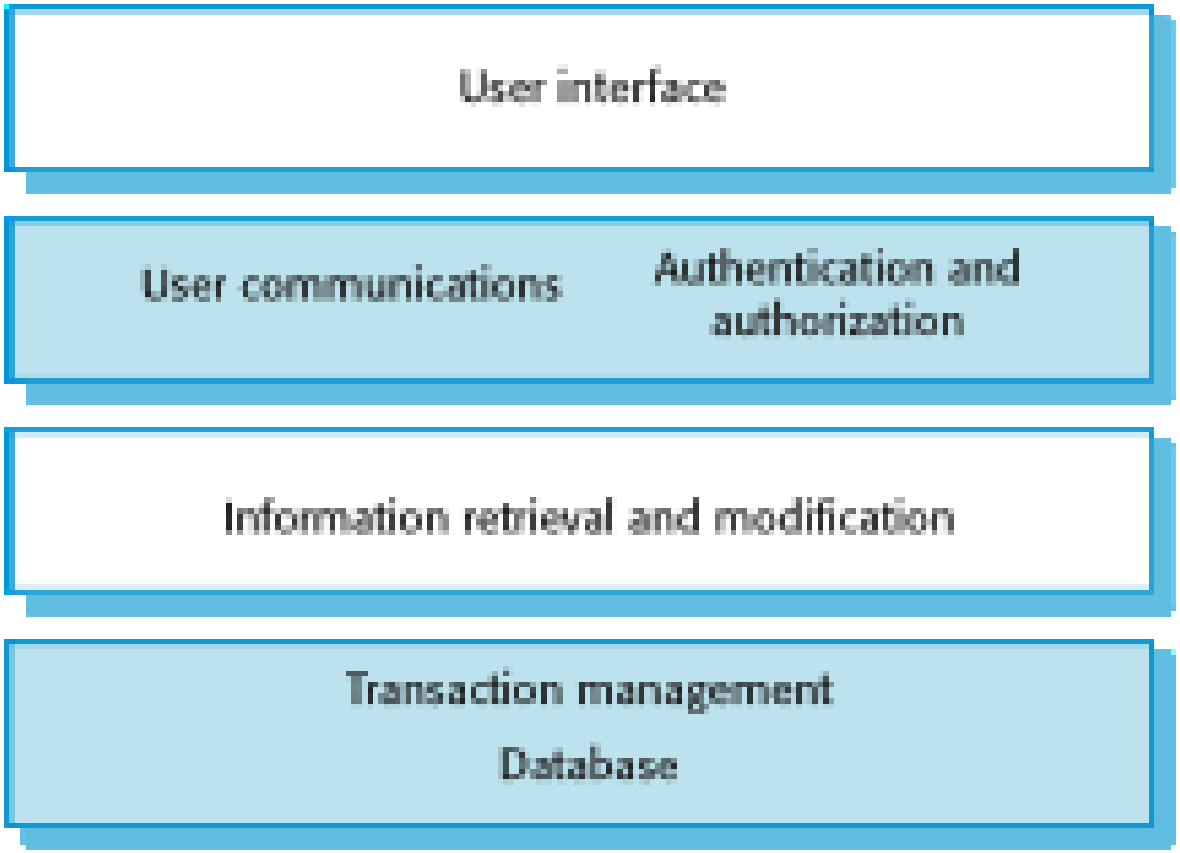
Bilgi sistemleri mimarisi



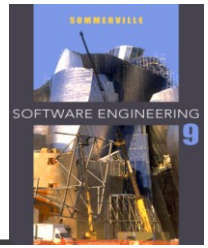
- Bilgi sistemleri, katmanlı bir yapıda organize edilebilecek genel bir mimariye sahiptir.
- Bunlar genellikle veritabanı transaction'ları içeren etkileşimli sistemlerdir.
- Katmanlar şunları içerir:
  - Kullanıcı arayüzü
  - Kullanıcı ile iletişim
  - Bilgiye erişim
  - Sistem veritabanı

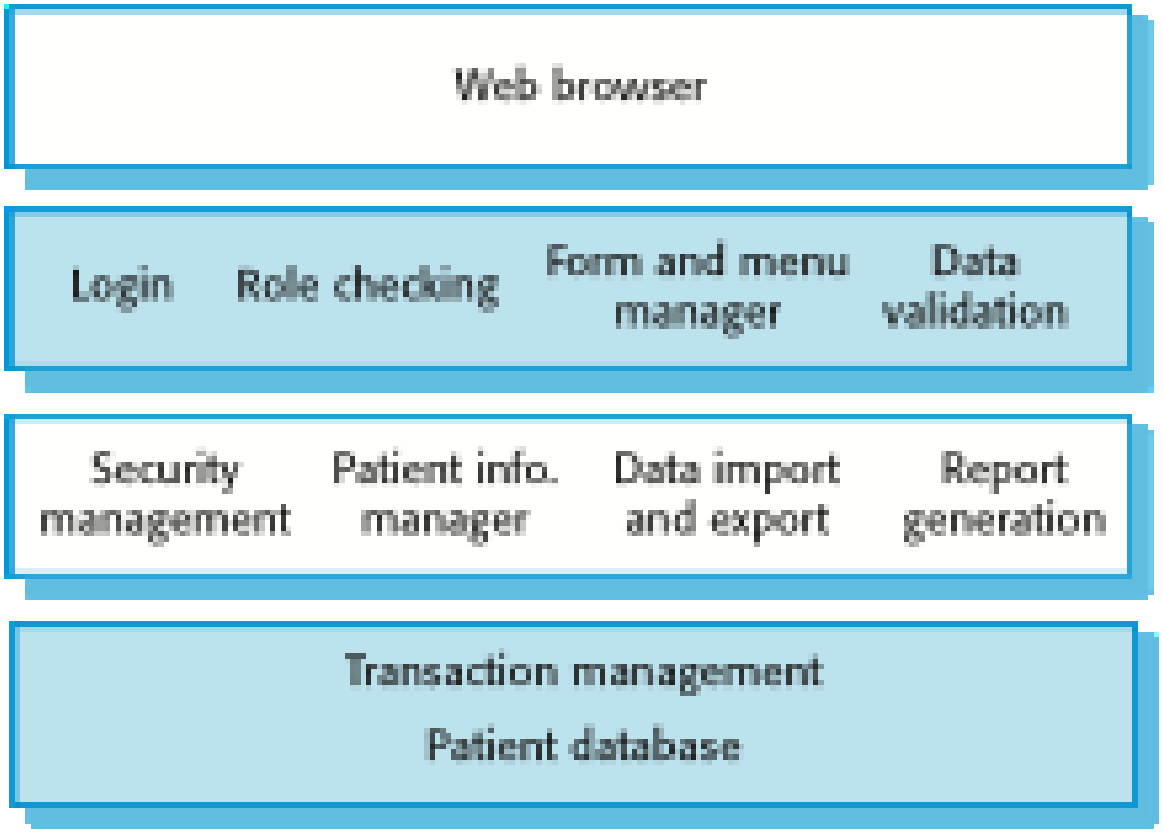
Katmanlı bilgi sistemi mimarisi



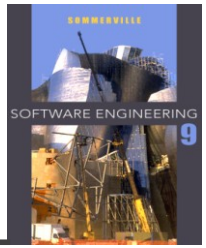


MHC-PMS'nin mimarisi





Web tabanlı bilgi sistemleri



- Bilgi ve kayna yönetim sistemleri günümüzde genellikle web tabanlı olarak geliştirilirler ve kullanıcı arayüzleri bir web tarayıcı kullanılarak gerçekleştirilir.
- Ör: e-ticaret sistemleri