

Yazılım Mimarisi ve Tasarımı

- Soyutlama, insanlar olarak karmaşıklıkla başa çıkmanın temel yollarından biridir.
- Dahl, Dijkstra ve Hoare şunu öneriyor:
 - Soyutlama, gerçek dünyadaki belirli nesneler, durumlar veya süreçler arasında ki benzerliklerin tanınmasından ve bu benzerliklere yoğunlaşma ve şimdilik farkları görmezden gelme kararından kaynaklanır.

Soyutlama (Abstraction)

- Bir şeyin en önemli, temel veya ayırt edici taraflarını öne çıkarıp daha az önemli, önemsiz veya saptırıcı ayrıntılarını bastıran veya görmezden gelen herhangi bir model.
- Ortak noktaları vurgulamak için farkların kaldırılmasının neticesi.
- Soyutlama, bir modülün amacını uygulamasından ayırır.
- Öncelikli hedefi çıkarım yapılarının geliştirilmesi olan matematik için enformasyonu dikkate almamaktır.
- Öncelikli hedefi etkileşim kalıplarını (interaction patterns) geliştirilmesi olan bilgisayar biliminde ise enformasyonu kapamadır(örtmedir). (information hiding)
- Bilgisayar bilimin temel faaliyeti yazılım üretimidir ve bu faaliyet esas olarak çıkarım yapılarının yaratılması ve kullanılmasıyla değil etkileşimin modellemesiyle kendini...
- Geleneksel - deneysel bilimler bu nedenle hem deneysel cihaz şeklindeki somut modellerle hem de matematiksek soyut modellerle ilgilenir.
- Bilgisayar bilimi, şimdiye kadar yazılımla ilgilendi, modellerin fiziki (somut) olmamasıyla deneyle bilimlerden ayrılır.
- Özetle, soyutlama felsefe, matematik ve mantıkta belirli özellikleri gözardı ederek özgeleği ortadan kaldırma süreci olarak nitelendirilmiştir.
- Aho ve Ullman, bilgisayar bilimini bir soyutlama bilimi olarak tanımladılar:
 - Bir problem hakkında düşünmek için doğru modeli yaratmak ve

- onu çözmek için uygun teknikleri tasarlamak.
- Wing'e göre:
 - Bir bilgisayar bilimcisi gibi düşünmek ... birden fazla soyutlama düzeyinde düşünmeyi gerektirir.
- Veri soyutlama, onların nasıl uyguladığınıza değil, işlemlerin veri toplama ile ne yaptığına odaklanır. Verilerin nasıl saklandıklarını bilmezler.
- Berzins, Gray ve Naumann bunu tavsiye ediyor:
 - Bir kavram, ancak sonunda onu gerçekleştirmek için kullanılacak mekanizmadan bağımsız olarak tanımlanabildiği, anlaşılabilir ve analiz edilebildiği taktirde bir soyutlama olarak nitelendirilir.
- Bir dosya nesnesi, belirli bir bellek aygıtında belirli bir yer kaplar; bir adı ve içeriği var. Bunların hepsi statik özelliklerdir.
- Bu özelliklerin her birinin değeri, nesnenin yaşam süresine göre dinamikdir: Bir dosya büyüyebilir veya küçülebilir, adı değişebilir. "Procedure-oriented" bir programlama tarzında, nesnelerin dinamik değerini değiştiren etkinlik, tüm programların merkezi parçasıdır; alt programlar çağrıldığında ve komutlar çalıştırıldığında işler olur.
- Bağımsız soyutlamaların yazılım onarımı ve geliştirmesi üzerinde etkisi vardır.
- Doğruluk ispatında yeri vardır..

Ayrıştırma

- Bir görevi, onu iki veya daha fazla ayrılabilir alt göreve bölerek ayrıştırır.
- Ne yazık ki, birçok sorun için ayrılabilir alt görevler, bütünüyle çekip çevrilme bakımından hala çok karmaşıktır.

Sarmalayıcı

- Soyutlama ve sarmalama tamamlayıcı kavramlardır:
 - Soyutlama, bir nesnenin gözlemlenebilir davranışına odaklanır

- Sarmalama, bu davranışa yol açan uygulamaya odaklanır.
- Soyutlama, insanların ne yaptıklarına dair düşünmelerine yardımcı olurken,
- sarmalama, program değişikliklerinin sınırlı çabayla güvenilir bir şekilde yapılmasına izin verir.
- Sarmalama, farklı soyutlamalar arasında açık engeller sağlar ve böylece ilgilerin açık bir şekilde ayrılmasına götürür.

Soyutlama Bize Ne Sunar?

- Soyutlama, bir varlığın diğer tüm varlık türlerinden ayıran temel özelliklerine odaklanarak karmaşıklığı yönetmemizi sağlar.
- Bir soyutlama, alana ve perspektife bağlıdır.
 - Bu bağlamda önemli olan başka bir bağlamda önemsiz olabilir.
- O, soyutlamalar kullanarak, problem alanından (örneğin; sınıflar ve nesneler) sistemimizi modellememize izin verir.

Soyutlama Örnekleri

- Öğrenci, üniversitedeki sınıflara kayıtlı kimse.
- Profesör: üniversitede ders veren kimsedir.
- Ders: üniversite tarafından sunulan bir ders.
- Ders programı: haftanın günleri ve saatleri dahil olmak üzere kurs için özel bir zamandır.

1- Soyutlama Nedir?

▼ Cevap

Soyutlama, bir varlığın diğer tüm varlık türlerinden ayıran temel özelliklerine odaklanarak karmaşıklığı yönetmemizi sağlar.

2- Matematik ve diğer bilim dalları ile bilgisayar bilimleri arasında ki fark öncelikli hedefleri bakımından nedir?

▼ Cevap

Matematikte amaç ispat yapmaktır, bir şeyi açıklama ve uygulama söz konusudur.

Soyutlama felsefe, matematik ve mantıkta belirli özellikleri gözardı ederek özgeleği ortadan kaldırma süreci olarak nitelendirilmiştir.

3- Bu fark soyutlamada hangi farkı ortaya çıkarmaktadır?

▼ Cevap

Konuyla ilgili bilgisi olmayan bilgileri görmezden gelirler, bilgisayar bilimlerinde bu bilgileri görmezden gelmiyoruz, üstünü kapatıyoruz.

4- Soyutlama ve sarmalama arasında ki fark nedir?

▼ Cevap

- Soyutlama, bir nesnenin gözlemlenebilir davranışına odaklanır
- Sarmalama, bu davranışa yol açan uygulamaya odaklanır.
- Soyutlama, insanların ne yaptıklarına dair düşünmelerine yardımcı olurken,
- sarmalama, program değişikliklerinin sınırlı çabayla güvenilir bir şekilde yapılmasına izin verir.
- Sarmalama, farklı soyutlamalar arasında açık engeller sağlar ve böylece ilgilerin açık bir şekilde ayrılmasına götürür.

5- Hangi güç(yetenek), bazı bilgisayar bilimcileri tarafından, yetkin bir programcının hayati faaliyetlerinden biri olarak görülmektedir.

▼ Cevap

soyutlama

6- Büyük yazılım sistemlerinin tasarımı ve onaylanmasında kilit sorun nedir ve nasıl aşılar?

▼ Cevap

Karmaşıklık temel sorun olabilir. Çözümde iki türdür: soyutlama ve ayırıştırma.



Doğru karar, genellikle tecrübenin sonucudur ve tecrübe çoğu zaman kötü kararların sonucudur. Fakat başkalarının tecrübelerinden öğrenmek, tecrübeli olanların bilgiyi takip edenlerle paylaşmasını gerektirir.

Barry LePatner

Yazılım Mimarisi Nedir?

Bir sistemin yazılım mimarisi,

- sistem hakkında
- mantık yürütmek için gerekli olan ve
- yazılım öğelerini
- onların aralarında ki ilişkileri ve
- her ikisinin özelliklerini içeren

yapılar kümesidir.

- Bu tanım, sistemin "erken" veya "büyük" tasarım kararlarından bahseden diğer tanımlamalarla çelişir.
- Pek çok mimari kararların çoğu mimari değildir.
- Bir karara bakıp "esas" olup olmadığını söylemek zordur.
- Öte yandan, yapıların yazılımda tanımlanması oldukça kolaydır ve sistem tasarımı için güçlü bir araç oluştururlar.

Mimari, bir dizi yazılım yapısıdır

- Yapı, bir ilişki tarafından bir arada tutulan bir dizi unsurdur.

- Yazılım sistemleri birçok yapıdan oluşur ve hiçbir yapı tek başına mimari olma iddiasında değildir.
- Mimari yapıların üç önemli kategorisi vardır:
 - Modül
 - Bileşen ve bağlayıcı
 - Tahsis

Modül Yapısı

- Bazı yapılar sistemleri modül dediğimiz uygulama birimlerine dönüştürür.
- Modüllere belirli hesaplamalı sorumluluklar atanır ve programlama ekipleri için iş atamalarının temellerini oluşturur.
- Büyük projelerde, bu öğeler alt ekipler atanmak üzere alt bölümlere ayrılır.
- Örneğin; büyük bir kurumsal kaynak planlama (ERP) uygulaması için veri tabanı o kadar karmaşık olabilir ki, uygulama birçok parçaya bölünür. Bu ayrışmayı yakalayan yapı bir çeşit modül yapısıdır, aslında modül ayrıştırma yapısıdır.
- Başka bir tür modül yapısı, nesne yönelimli analiz ve tasarım sınıfı diyagramlarının bir çıktısı olarak ortaya çıkar.
- Modüllerinizi katmanlar halinde bir araya getirirseniz, başka bir modül yapısı oluşturmuş olursunuz.
- Modül yapıları statik yapılardır; sistem işlevlerinin bölünüp uygulama ekiplerine atanmasına odaklanır.

Bileşen ve Bağlayıcı Yapılar

- Diğer yapılar dinamiktir, öğelerin sistemin işlevlerini yerine getirmek için çalışma esnasında hangi yolla birbirleriyle etkileşimde bulunduğu odaklanır.
 - Sistemin bir dizi hizmet olarak kurulacağını varsayalım.

- Hizmetler, etkileşimde bulundukları altyapı, ve aralarında ki senkronizasyon ve etkileşim ilişkileri, genellikle bir sistemi tanımlamak için kullanılan başka bir tür yapıyı şekillendirir.
- Bu hizmetler, çeşitli uygulama birimlerindeki modüller programlardan oluşur.
- Runtime yapılarına bileşen ve bağlayıcı (C&C) yapıları diyoruz.
- Bizim kullanımımızda bir bileşen her zaman runtime bir varlıktır.

Tahsis Yapıları

- Bir üçüncü yapı, tahsis yapıları, yazılım yapılarını sistem ortamlarıyla örtüştürmeyi tanımlar.
 - Örgütsel
 - Gelişimsel
 - Kurulum
 - İcra
- Örneğin,
 - Modüller, geliştirilmesi için ekiplere atanır ve geliştirme, entegrasyon ve test için dosya yapısında bir yere atanır.
 - Bileşenler, yürütülmek üzere donanıma yerleştirilir.

Hangi Yapılar Mimaridir?

- Bir yapı, sistem ve sistemin özellikleri hakkında akıl yürütmeyi destekliyorsa mimardır.
- Muhakeme, sistemin bazı "stakeholder" larca önemsenen bir niteliği olmalıdır.
- Bunlar arasında:
 - sistem tarafından sağlanan işlevsellik
 - arıza durumunda sistemin kullanılabilirliği

- sistemde belirli deęişikler yapmanın zorluğu
- sistemin kullanıcı taleplerine cevap verme kabiliyeti

Mimarlık Bir Soyutlamadır

- Bir mimari, yazılım öğelerini ve bunların birbiriyle ilişki şeklini içerir.
 - Bir mimari, sistem hakkında mantık yürütmek için bir faydası olmayan unsurlar hakkındaki belirli bilgileri bilhassa gözardı eder.
 - Mesela, içinde "z" harfi olan kaynak kodu satırlarının harf sayısına göre kısıdan uzuna sıralaması bir yapı oluşturur ama bu ne ilginç ne de mimari bir yapıdır.
- Bir mimari bir öğenin dışarısında hiçbir sonucu olmayan belgiler atlar.
- Bir mimari, belirli ayrıntıları seçer ve diğerlerini bastırır.
- Öğelerin özel ayrıntıları - yalnızca iç uygulamayla ilgileri ayrıntılar- mimari değildir.
- Mimari soyutlama, sisteme:
 - öğeleri
 - nasıl düzenlendikleri
 - nasıl etkileşime girdikleri
 - nasıl bir araya getirildikleri
 - sistem mantığımızı destekleyen özelliklerinin neler olduğunu vb. açısından bakmamızı sağlar.
- Bir soyutlama, bir mimarinin karmaşıklığını uysallaştırmak için gereklidir.
- Biz karmaşıklığın hepsiyle ve sürekli uğraşamayız, uğraşmak da istemeyiz.

Her sistemin bir yazılım mimarisi vardır

- Her sistem, bir tür muhakemeyi desteklemek için öğeleri ve aralarında ki ilişkileri kapsar.

- Ancak mimari kimse tarafından bilinmeyebilir.
 - Sistemi tasarlayanlar gitmiş olabilir
 - Bilgiler kaybolmuş olabilir
 - Kodlar silinmiş olabilir
- Bir mimari, tanımından veya şartnamesinden bağımsız olarak var olabilir.
- Belgelendirme hayatidir.

Mimari Davranışı İçerir

- Her bir öğenin davranışı,
 - bu davranışın sistem hakkında mantık yürütmek için kullanılabildiği ölçüde mimarinin bir parçasıdır.
- Bu davranış, unsurların birbirleriyle nasıl etkileşime girdiğini somutlaştırır ve bu açıkça mimari tanımının bir parçasıdır.
- Mimari olarak aktarılan kutu ve çizgi çizimler, mimari değildir.
- Bu, her unsurun tam davranışının ve performansının her koşulda belgelenmesi gerektiği anlamına gelmez.
- Bu unsurun davranışının:
 - başka bir unsuru etkilediği veya
 - bir bütün olarak sistemin kabul edilebilirliğini etkilediği ölçüde,
 - bu davranışın yazılım mimarisinin bir parçası olarak değerlendirilmeli ve belgelenmelidir.

Yapılar ve Görünümler

- Bir view:
 - Sistem stakeholderları tarafından yazılan ve okunan şekliyle,

- coherent bir mimari unsurlar kümesinin temsilidir.
- Bir "view", bir dizi öğenin ve aralarındaki ilişkilerin temsilini ihtiva eder.
- Bir yapı yazılımda ve donanımda var olduğu haliyle, bir unsurlar kümesidir.
- Kısaca bir "view" bir yapının temsilidir.
 - Mesela bir modül yapısı sistem modüllerinin ve onların örgütlerinin kümesidir.
 - Bir modül "view" seçilen notasyondaki şablona göre dökümente edilen ve bazı sistem stakeholderları tarafından kullanılan bu yapının temsilidir.
- Mimarlar yapıyı tasarlar. Bu yapının "view"larını dökümate ederler.

Module Structures

- Modül yapıları sistemin inşa edilmesi veya tedarik edilmesi gereken bir dizi kod veya veri bilimleri olarak nasıl yapılandırılacağını ilişkin kararları içerir.
- Herhangi bir modül yapısında , öğeler bir tür modellerdir.(belki hepsi uygulama (implementation) birimi olan sınıflar veya tabakalar veya yalnızca işlevsellik bölümleri).
- Modüller tayin edilmiş işlevsel sorumluluk alanlarıdır, bu yapılarda , yazılımın koşma (run) esnasında nasıl tezahür ettiğine dair vurgu azdır.(Modülün kim ve ne yapacağına dair atama vardır.)
- ▼ Modül yapıları , aşağıdakine benzer sorulara cevap vermemize imkan verir:
 - ▼ Her bir modüle atanan birincil işlevsel sorumluluk nedir?



Modülleştirmenin genel ilkeleri vardır . Coherent(birim içerisindeki irtibatının sıklığından)

- Başka hangi yazılımı gerçekte kullanır ve bunlara bağlıdır?
- Başka hangi yazılım öğelerinin kullanmasına izin verilir?
- Hangi modüller diğer modüllerle genelleştirme veya özelleştirme(yani kalıtım) yoluyla irtibatlandırılır?



aşağıdan yukarıya genelleştirme , yukarıdan aşağı özelleştirme. Alt eleman üst elemanın özelliklerini miras olarak alıyor .

Component and Connector Structures (Bileşen ve Bağlayan Yapıları)

Bileşen ve bağlayan yapıları, sistemin koşu (run) davranışı (bileşenler) ve etkileşimlere (bağlayanlar) sahip bir ögeler kümesi olarak nasıl yapılandırılacağına ilişkin kararları içerir.

▼ Ögeler(bileşen) ,

- hizmetler , akranlar , istemciler , sunucular, filtreler veya diğer birçok koşu (run) ögesi türü gibi koşu ögeleridir

▼ Bağlayanlar ,

- çağrı dönüşü , süreç senkronizasyon operatörleri , kanallar ve sair gibi
- bileşenler arasındaki iletişim araçlarıdır.

▼ Bileşen ve bağlayan görünüşleri , aşağıdakine benzer soruları cevaplamamızı yardımcı olur:

- Başlıca yürütme bileşenleri nelerdir ve run edilirken nasıl etkileşirler?
- Başlıca paylaşılan veri depoları nelerdir?
- Sistemin hangi bölümleri kopyalanır(ikilenir)? (kopyalama=replikasyon)
- Veriler sistemde nasıl ilerler?
- Sistemin hangi bölümleri paralel olarak çalışabilir?
- Sistemin yapısı , yürürken değişebilir mi ; değişirse nasıl değişir?

▼ Bileşen ve bağlayan görünüşleri ,sistemin

- koşu (run) performansı , güvenlik , erişilebilirlik ve dahası hakkında soru sormak için hayati önemdir

Allocation Structures (Tahsis Yapıları)

- Tahsis yapıları, yazılımın oluşturulduğu ve yürütüldüğü bir veya daha fazla harici ortamdaki ögeler ile yazılım ögeleri arasındaki ilişkiyi gösterir.

Structures Provide Insight (Kavrama ve içgörü)

- Yapılar sahip oldukları analitik ve mühendislik gücü nedeniyle yazılım mimarisine bakış açımızda çok önemli bir rol oynamaktadır.
- Her yapı, ilgili kalite özelliklerinden bazıları hakkında mantık yürütmek için bir perspektif sağlar



Örneğin: Hangi modüllerin diğer hangi modülleri kullandığını içeren **modül yapısı** , bir sistemin genişletilmesi veya daraltılmasının **kolaylığına** sıkı sıkıya bağlıdır.



Örneğin: Paralelliği sistem bünyesinde barındıran **eşanlılık yapısı** bir sistemin kilitlenme ve performans darboğazlarından kurtarılma kolaylığına sıkı sıkıya bağlıdır



Bir sistemde önemli olan kaç kullanıcıyı desteklemesi değil , aynı anda iş anlamlı kaç kullanıcıyı desteklemesi önemli. Bunun ayırt edilmesi lazım .



Örneğin: **İntikal yapısı** , performans , erişebilirlik ve güvenlik hedeflerine ulaşılmasına sıkı sıkıya bağlıdır.

Some Useful Module Structures (Bazı Kullanışlı (Faydalı) Modül Yapıları)

- **Ayrıştırma yapısı**,

- Birimleri , "bir alt modüldür" ilişkisi ile birbirleriyle irtibatlı modüllerdir.
- Modüllerin, kolayca anlaşılacak kadar daha küçük modüllere tekrarlı olarak nasıl ayrıştırıldığını gösterir. (Kalite unsurlarını ihlal etmeyene kadar ayrıştırabiliriz.)
- Modüller genellikle kendileriyle eşlenik ürünlere (arayüz şartnameleri , kod, test planları vb.) sahiptir.
- Ayrıştırma yapısı büyük ölçüde olası değişikliklerin yerleştirilmesini temin ederek sistemin değiştirilebilirliğini belirler.
- Ayrıştırma yapısı, genellikle , geliştirme projesinin , dokümantasyon yapısı , proje entegrasyonu ve test planlarını ihtiva eden organizasyonu için temel olarak kullanılır.
- Ayrıştırma yapısındaki birimler "segment" veya "alt sistem" gibi kuruluşa has adlara sahip olma eğilimindedir.
- **Kullanır yapıları,**
 - Buradaki birimler aynı zamanda **modüller** belki de **sınıflar**dır.
 - Bu birimler bağımlılığın özel bir biçimi olan "**kullanır**" ilişkisi ile irtibatlıdır.
 - Bir yazılım birimi ilkinin doğruluğu ikincisinin doğru çalışan bir sürümünün (bir saplamanın aksine) varlığını gerektiriyorsa , bir diğerini kullanır.
 - Bu yapı işlevsellik eklenebilecek veya faydalı işlevsel alt kümeler çıkarabilecek sistemler yapabilmek için kullanılır.
 - Bir sistemin bir alt kümesini kolayca oluşturma yeteneği , **artırımlı geliştirmeye** izin verir.
- **Tabaka(soyut bir kavram) yapısı,**
 - Bu yapıdaki modüllere tabakalar adı verilir.
 - Bir tabaka , yönetilen bir arabirim vasıtasıyla sıkı ve türdeş (cohesive) bir hizmet kümesi sağlayan soyut bir "sanal makinedir".
 - Tabakaların diğer tabakaları sıkı bir yönetim tarzıyla kullanmasına izin verilir.
 - Tabakaların katı tabakalı sistemlerde , bir tabakanın yalnızca tek bir başka tabaka kullanmasına izin verilir.

- Bu yapı bir sisteme taşınabilirlik yani dayandığı bilgisayar zeminini değiştirme yeteneği verir.
- Sınıf(veya genelleştirme) yapısı,
 - Bu yapıdaki modül birimlerine sınıflar denir.
 - İlişki , "-den mirastır" veya "-in bir oluşudur" dur.
 - Bu yapı benzer davranış veya yetenek hakkında muhakeme yürütmeyi destekler. Diğer sınıfların miras aldığı ve parametreleştirildiği farklılıklar sınıfı.
 - Sınıf yapısı birine , yeniden kullanım ve işlevselliğin artırımı eklenişi üzerine akıl yürütme izni verir.
 - Nesneye dönük analiz ve tasarım sürecini takip eden bir projede herhangi bir belge mevcutsa o , genellikle bu yapıdadır.
- Veri Modeli
 - Statik bilgi yapısını , veri varlıkları(entity) ve ilişkileri(relation) açısından tarif eder.



Mesela , bir bankacılık sisteminde varlıklar, tipik olarak hesap ,müşteri ve krediyi içerir. Hesap , hesap numarası ,tür(tasarruf veya çek) , durum ve cari bakiye gibi çeşitli niteliklere sahiptir.

Some Useful C&C Structures()

- Tüm bileşen ve bağlayan yapılarındaki ilişki, bileşen ve bağlayanların birbirlerine nasıl bağlandığını gösteren "ek" tir.
- Bağlayanlar , "çağırır" gibi aşına yapılar olabilir.
- Hizmet Yapısı
 - Birimler , SOAP gibi hizmet koordinasyon mekanizmaları sayesinde bir arada çalışan servislerdir.

- Bu yapı anonim ve birbirinden bağımsız olarak geliştirilmiş olabilecek bileşenlerden oluşan bir sistem mühendisliğine yardımcı olur.
- **Eşanlılık Yapısı**
 - Bu yapı , paralellik fırsatlarını ve kaynak çekişmesinin ortaya çıkabileceği yerleri tespit etmeye yardımcı olur.
 - Birimler bileşenlerdir.
 - Bağlayanlar onların iletişim mekanizmalarıdır.
 - Bileşenler mantıksal **silsile**de (thread=belirlenen bir yol) düzenlenmiştir.

Mimari Şablonlar (Architectural Patterns)

- Elemanların kompozite edilmesi bir araya getirilmesi sonra dökümanite edilip yayınlanmasıdır.
- **mimari şablonlar**, problemin çözümünde kullanılan öge türlerini ve bunların etkileşim biçimlerini tanımlar.
- Burada elemanların birbiriyle etkileşiminden bahsediyor. (şablon)
- Kompozisyonların zaman içinde ve birçok farklı alanda faydalı olduğu bulunmuştur.
- **Mimari şablonlarda bahsettiğimiz 3 temel tip vardır:**
 - 1-Yapılar (modüller)
 - 2- Bağlayan ve bileşen
 - 3-Allocation (tahsis)
- **Yaygın bir modül tipi modeli, Katmanlı modeldir (tabakalaşma).**
 - Yazılım öğeleri arasındaki kullanım ilişkisi kesinlikle tek yönlü olduğunda, bir tabaka sistemi ortaya çıkar.
 - Tabaka, tutarlı bir ilgili işlevsellik kümesidir. (coherent=elemanların birbiri ile sıkı ve türdeş bir ilişkide bulunması)
 - Yapısal kısıtlamayı azaltan bu modelin birçok varyasyonu pratikte ortaya çıkar.

- **Yaygın bileşen ve bağlayıcı türü desenleri:**

1. Paylaşılan veri (veya havuz) kalıbı:

- Bu model, kalıcı verileri oluşturan, (component and connectors)depolayan ve bunlara erişen bileşenleri ve bağlayıcıları içerir.
- Depo genellikle bir (ticari) veritabanı biçimini alır.
- Konektörler, SQL gibi verileri yönetmek için kullanılan protokollerdir.

2. İstemci sunucu tabakası (client -server pattern)

- Bileşenler, istemciler ve sunuculardır.
- Konektörler (connector), sistemin çalışmasını yürütmek için birbirleriyle paylaştıkları protokoller ve mesajlardır.

- **Ortak tahsis modelleri:**

1. Multi-tier Pattern (çok katmanlı desen)

- Bir sistemin bileşenlerinin, bazı iletişim araçlarıyla birbirine bağlanan farklı donanım ve yazılım alt kümelerinde nasıl dağıtılacağını ve tahsis edileceğini açıklar.
- Bu model, genel dağıtım (yazılımdan donanıma ayırma) yapısını uzmanlaştırır.

2. Yetkinlik merkezi düzeni ve platform düzeni

- Bu modeller, bir yazılım sisteminin iş atama yapısını uzmanlaştırır.
- Yetkinlik merkezinde, bir sahada bulunan teknik veya alan uzmanlığına bağlı olarak iş sahalarına tahsis edilir.
- Platformda bir site, bir yazılım ürün hattının yeniden kullanılabilir temel varlıklarını geliştirmekle görevlidir ve diğer siteler, temel varlıkları kullanan uygulamalar geliştirir.

Mimariyi "iyi" yapan nedir?

- Doğası gereği iyi veya kötü mimari diye bir şey yoktur.
- Mimari değerlendirilebilir, ancak yalnızca belirtilen belirli hedefler bağlamında.
- Bununla birlikte, iyi pratik kurallar vardır.

1. Process "Rules of Thumb"

▼ Mimari, tek bir mimarın veya belirli bir teknik liderin bulunduğu küçük bir mimarlar grubunun ürünü olmalıdır.

- Bu yaklaşım mimariye kavramsal bütünlüğünü ve teknik tutarlılığını verir.
- Bu öneri, Agile ve açık kaynaklı projeler ile "geleneksel" projeler için de geçerlidir.
- Mimar (lar) ile geliştirme ekibi arasında güçlü bir bağlantı olmalıdır.
- Mimar (veya mimar ekibi), mimariyi, iyi belirlenmiş kalite özellik gereksinimlerinin öncelikli bir listesine dayandırmalıdır.
- Mimari, görünümüler kullanılarak belgelenmelidir. Görüşler, proje zaman çizelgesini destekleyen en önemli paydaşların endişelerini ele almalıdır.

▼ Mimari, sistemin önemli kalite özelliklerini sunma kabiliyeti açısından değerlendirilmelidir.

- Bu, yaşam döngüsünün başlarında gerçekleşmeli ve uygun şekilde tekrarlanmalıdır.

▼ Mimari, artımlı uygulamaya uygun olmalıdır.

- İletişim yollarının uygulandığı ancak ilk başta minimum işlevselliğe sahip olan bir "iskelet" sistemi oluşturun.

2. Structural (Yapı) "Rules of Thumb"

▼ Mimari, işlevsel sorumlulukları bilgi gizleme ve işlevlerin ayrılması ilkelerine atanan iyi tanımlanmış modüller içermelidir.

- Bilgi gizleme modülleri, değişme olasılığı olan şeyleri kapsamalıdır.
- Her modül, diğer yazılımlardan değiştirilebilir yönleri "gizleyen" iyi tanımlanmış bir arayüze sahip olmalıdır.

- Gereksinimleriniz emsalsiz olmadıkça, kalite özellikleriniz her bir özelliğe özgü iyi bilinen mimari kalıplar ve taktikler kullanılarak elde edilmelidir.
- Mimari hiçbir zaman ticari bir ürünün veya aracın belirli bir sürümüne bağlı olmamalıdır. Gerekirse, farklı bir sürümüne geçiş basit ve ucuz olacak şekilde yapılandırılmalıdır.
- ▼ Veri üreten(produce) modüller, veri tüketen(consume) modüllerden ayrı olmalıdır.
 - Bu, değiştirilebilirliği artırma eğilimindedir
 - Değişiklikler sıklıkla verinin üretim veya tüketim tarafıyla sınırlıdır.
- Modüller ve bileşenler arasında bire bir yazışma beklemeyin.
- Her işlem, belirli bir işlemciye atanması, belki çalışma zamanında bile kolayca değiştirilebilecek şekilde yazılmalıdır.
- ▼ Mimari, bileşenlerin etkileşime girmesi için az sayıda yol içermelidir.
 - Sistemler baştan sona aynı şeyleri aynı şekilde yapmalıdır.
 - Bu anlaşılabilirliğe yardımcı olacak, geliştirme süresini azaltacak, güvenilirliği artıracak ve değiştirilebilirliği artıracaktır.
- Mimari, belirli (ve küçük) bir dizi kaynak çekişme alanı içermeli, bunların çözümü açıkça belirtilmiş ve korunmuş olmalıdır.

(özet slayt sayfaları)



Summary

- The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.
- A structure is a set of elements and the relations among them.
- A view is a representation of a coherent set of architectural elements. A view is a representation of one or more structures.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License



Summary

- There are three categories of structures:
 - Module structures show how a system is to be structured as a set of code or data units that have to be constructed or procured.
 - Component-and-connector structures show how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).
 - Allocation structures show how the system will relate to nonsoftware structures in its environment (such as CPUs, file systems, networks, development teams, etc.).
- Structures represent the primary engineering leverage points of an architecture.
- Every system has a software architecture, but this architecture may be documented and disseminated, or it may not be.
- There is no such thing as an inherently good or bad architecture. Architectures are either more or less fit for some purpose.

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Software Quality Attributes

(Yazılım Kalitesi Nitelikleri)

Bir yazılım tasarımı oluşturmanın iki yolu vardır:

- Bir yol, bunu o kadar basit hale getirmektir ki, açıkça hiçbir eksiklik yoktur,
- Diğer bir yol, onu o kadar karmaşık hale getirmektir ki, hiçbir belirgin eksiklik yoktur.
- İlk yöntem çok daha zordur.

Quality Attributes (Kalite Özellikleri):

- ▼ Kalite özelliği gereksinimleri, aşağıdakilerin bir parçasıdır;
 - bir uygulamanın işlevsel olmayan gereksinimleri

- birçok yönünü yakalayan
- bir uygulamanın işlevsel gereksinimlerinin nasıl elde edildiği
- ("Uygulama ölçeklenebilir olmalıdır") Bu çok kesin değildir ve hiç kimsenin işine yaramaz.
- ("Kurulum ve yapılandırma için çaba / maliyette bir artış olmaksızın dağıtımı coğrafi olarak dağınık ilk 100 kullanıcı masaüstünden 10.000'e ölçeklendirmek mümkün olmalıdır. ") Bu kesin ve anlamlıdır.
- Bununla birlikte, birçok kalite özelliğini doğrulamanın ve test etmenin aslında biraz zor olduğunu unutmayın.

Kalite Nitelikleri

1. Tasarım Kaliteleri
2. Run kaliteleri (koşu)
3. Sistem Kaliteleri
4. Kullanıcı Kaliteleri

1-Tasarım Kaliteleri:

▼ Özellikler

- Bir arada çalışabilirlik
- Kavramsal Bütünlük
- Erişilebilirlik
- Kavramsal bütünlük , genel tasarımın tutarlılığını ve sıklığı ve türdeşliğini (coherent) tanımlar. Bu , bileşenlerin veya modüllerin tasarlanma yolunun yanı sıra kodlama tarzı ve değişken adlandırma gibi faktörleri içerir.
- Maintainability
- Müdahale edilebilirlik (idame edilebilirlik), sistemin ne derece kolaylıkla değiştirilebileceğidir. Bu değişiklikler , işlevsellik eklendiği veya değiştirildiğinde ,

hataları düzeltildiğinde ve yeni iş ihtiyaçlarını giderildiğinde , bileşenleri , hizmetleri, özellikleri ve arabirimleri etkileyebilir.

- Modifiability
- Reusability
- Tekrar kullanılabilirlik , bileşenlerin ve alt sistemlerin diğer uygulamalarda ve diğer senaryolarda kullanıma müsaitliğini tanımlar. Tekrar kullanılabilirlik, bileşenlerin ikilenmesini ve ayrıca devreye alma süresini en aza indirir.

2-Koşu (run) Kaliteleri:

▼ Özellikler

- Güvenlik
- Ölçeklenebilirlik
- Tekrar Kullanılabilirlik
- Güvenilirlik
- Performans
- Yönetilebilirlik
- Müdahale edilebilirlik
- Availability (Ben 7/24 sistemi ayakta görmeliyim (erişilebilir istiyor))
- Erişilebilirlik , sistemin işlevsel ve çalışır durumda olduğu zaman nisbetini tanımlar. Önceden tanımlanmış bir süre boyunca sistemin toplam kapalı kalma süresinin yüzdesi olarak ölçülebilir. Erişilebilirlik , sistem hatalarından , altyapı sorunlarından , kötü niyetli saldırılardan ve sistem yükünden etkilenecektir.
- Failures (başarısızlık)
- MTBF (Arızalar arasında kalan süre) (meantime between failure)
- MTTR (Sistemin ayağa kalkma süresi)
- Recoverability
- Replication (sistemi başka tarafta aynalamak / yani sistem iki makinede paralel olarak çalışması)

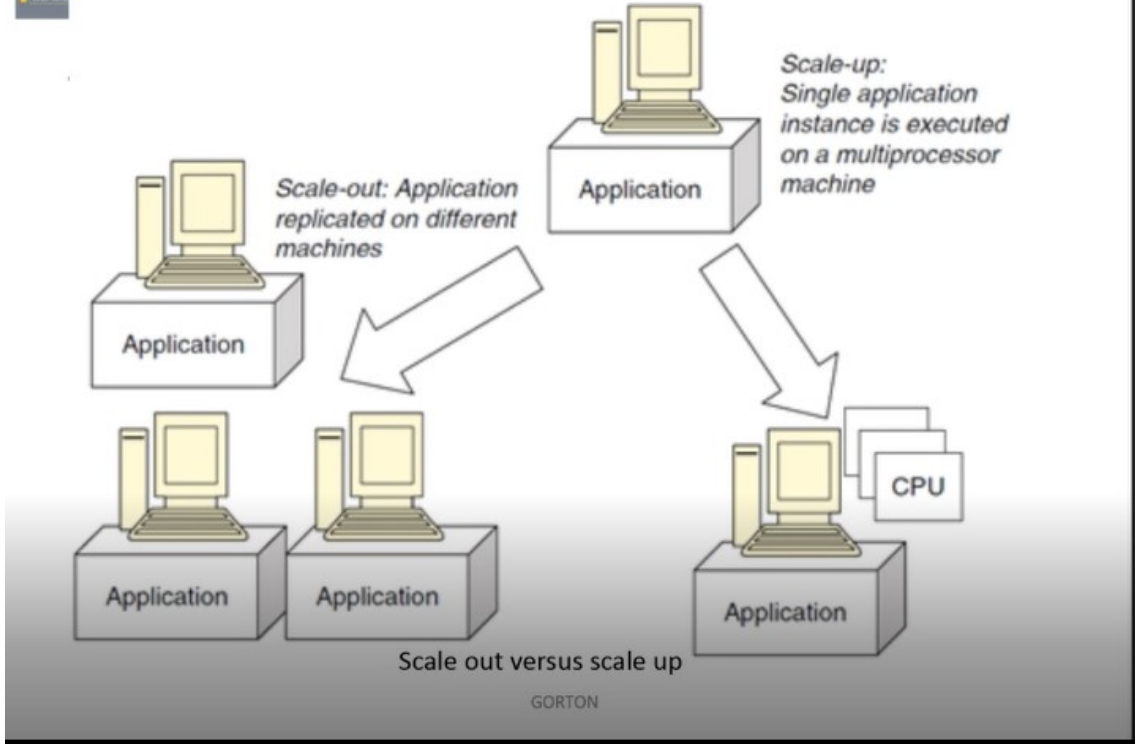


mtbf / (mtbf+mttr)

- Interoperability (Birlikte çalışabilirlik) (açık sistemin özelliği)
- Bir arada çalışabilirlik, bir sistemin veya farklı sistemlerin , harici taraflarca yazılan ve çalıştırılan diğer harici sistemlerle iletişim kurarak ve bilgi alışverişinde bulunarak başarılı bir şekilde çalışabilmesidir. Bir arada çalışabilir bir sistem , bilgilerin harici olduğu kadar dahili olarak değiştirilmesini ve yeniden kullanılmasını kolaylaştırır.
- Manageability
- Yönetilebilirlik , sistem yöneticilerinin , genellikle gözetleme sistemlerindeki kullanım ile hata bulma ve performans ayarı için açılmış yeterli ve kullanışlı araçlar vasıtasıyla , uygulamayı yönetmesinin kolaylık seviyesini tanımlar.
- Performans , belirli bir zaman aralığında herhangi bir eylemi gerçekleştirmek için bir sistemin tepki verme gücünün bir göstergesidir.
 - Gecikme(latency)veya throughput olarak ölçülebilir.
 - Gecikme , herhangi bir olaya tepki vermek için geçen süredir.
Throughput(average pick) belirli bir süre içinde gerçekleşen olay sayısıdır.
 - Response Time (guaranteed average), (ortalama)
 - Deadlines (süre sonu)
 - Güvenilirlik (Reliability) , bir sistemin zaman içinde çalışır durumda kalma yeteneğidir. Güvenilirlik bir sistemin belirli bir zaman aralığında niyetlenen işlevlerini yerine getirmede başarısız olmama ihtimali olarak ölçülür.
 - Scalability (Ölçeklenebilirlik) , sistemin yükteki artışları sistem performansı etkilemeden çekip çevirmesi veya kolaylıkla genişletebilmesidir.
 - Request Load (Yük talep et)
 - Simultaneous Connections (Eşzamanlı Bağlantılar)
 - Data Size (Veri Boyutu)
 - Deployment (Dağıtım)



2 Kösü Kaliteleri



- Security (güvenlik) , bir sistemin tasarlanan kullanım dışında kötü niyetli veya kazara eylemleri önleme , bilgi ifşasını veya kaybını önleme yeteneğidir. Güvenli bir sistem , varlıkları korumayı ve bilgilerin yetkisiz değıştirilmesini önlemeyi amaçlar.
- Authentication (Doğrulama ve kimliklendirme)
 - Uygulamalar, kullanıcılarının ve iletişim kurdukları diğer uygulamaların kimliğini doğrulayabilir. Bilgileri oluşturan sistem kullanıcısının kimliğini doğrular.
 - Bir sistem varlığının (bir sistem, bir ağ düğümü) kimliğini ve varlığın bilgisayarlı bilgilere erişme uygunluğunu doğrulama eylemi.
 - Sahte oturum açma etkinliklerine karşı koruma sağlamak için tasarlanmıştır.

- Kimlik doğrulama teknikleri,
- Confidentiality (Gizlilik)
 - Yetkili kurumların beklediği bilgileri anlaşılmaz hale getirir.
- Encryption (Şifreleme)
 - Uygulamaya / uygulamadan gönderilen mesajlar şifrelenir.
- Nonrepudiation (Reddedilmeme)
 - Bir mesajın göndericisinin teslim kanıtı vardır ve alıcı, gönderenin kimliğinden emin olur. Bu, mesaj alışverişine katılımlarını sonradan reddedemeyeceği anlamına gelir.
 - Gönderenin mesajı göndermeyi reddedememesini sağlar.
- SSL (Secure Socket Layer) ⇒ Güvenli Yuva Katmanı (şifreleme yöntemi)
- PKI (Public Key Infrastructure) ⇒ Açık Anahtar Altyapısı

▼ **Mimari şablon veya kalıplar nedir?**

Mimari unsurların kompozisyonu

▼ **Mimari yapı neyi ortaya koyar?**

Birbirleriyle tiplerini ve etkileşimini koyarız.

▼ **Modül tip kalıbının örnekleri nelerdir?**

Tabaka şablonları

▼ **Tabakadan ne kastedilir?**

Tek yönlü çalışıyorlar

İlgili işlevlerin sıkı ve türdeş olması

▼ **Bileşen ve bağlayan kalıbına örnek veriniz?**

Client Server

▼ **Tahsis tip nedir (şablon örneği veriniz)?**

Uzmanlık merkezi şablonu ve Platform şablonu

▼ **İyi mimari için birtakım kurallar gerekir ve bu kurallar hangi iki ana başlığın altında toplanır?**

Process(İşlem el yordamı) ve Structures(Yapı el yordamı)

▼ **İşlem ve yapı el yordamı kuralları**

Mimari sağlaması gereken önemli sistem kalite nitelikleri bakımından irdelenmelidir ⇒ İşlem el yordamı (Process)

Talep ve ihtiyaçlar hiç örneği görülmemiş olmadıkça kalite nitelikleri her niteliğe özel çok iyi bilinen mimari şablon ve taktiklerle erişilebilir.⇒ Yapı el yordamı (Structures)

Tek bir mimar veya belirlenmiş küçük bir mimar grubu tarafından ortaya konulmalıdır. ⇒ İşlem el yordamı (Process)

▼ **"Tek bir mimar veya belirlenmiş küçük bir mimar grubu tarafından ortaya konulmalıdır " neden böyle diyoruz?**

Tutarlılığın bozulmaması için

Kavramsal bütünlüğün bozulmaması için

▼ **Functional requirement(işlevsel olan ihtiyaçlar)**

Doğrudan kullanıcının işine yarayacak

▼ **nonfunctional requirement(işlevsel olmayan ihtiyaçlar)**

Tüm bunları sağlamaya yarayanlar

▼ **Kalite nitelikleri 4 ana gruba ayrılmıştır?**

1. Tasarım

2. Koşu

3. Sistem

4. Kullanıcı

▼ **Kavramsal bütünlük neyi ifade eder?**

Coherent (sıkı ve türdeş)

Temel tasarımın tutarlılığını

▼ **Performans kalite niteliklerini neyle ölçeriz?**

Deadline

Response time

Gecikme

▼ **Aşağıdaki söylenen niteliklerin gruplarını söyleyiniz?**

- Gizlilik
- Yetkilendirme
- İnkâr edilememe
- Kimliklendirme

Security

3-Sistem Kaliteleri:

▼ Özellikler

- Desteklenebilirlik
- Test edilebilirlik
- Supportability (Desteklenebilirlik)
 - Sistemin düzgün çalışmadığında sorunları tespit etmek ve çözmek için yararlı bilgiler sağlama yeteneğidir.
- Testability (Test edilebilirlik)
 - Sistem ve bileşenleri için test kriterleri oluşturmanın ve kriterlerin karşılanıp karşılanmadığını belirlemek için bu testleri gerçekleştirmenin ne kadar kolay olduğunun bir ölçüsüdür.
 - İyi test edilebilirlik , bir sistemdeki arızaların zamanında ve etkili bir şekilde izole edilebilmesini daha mümkün kılar. (Sistemin kalitesine de bağlıdır.)

4-Kullanıcı Kaliteleri:

- Usability (Kullanılabilirlik)

- Uygulamanın , sezgiye ve elverişliliği , yerelleştirme ve küreselleştirme kolaylığı , engelli kullanıcılar için iyi erişi sağlayışı ve genel olarak iyi bir kullanıcı tecrübesi sayesinde , kullanıcı ve müşterinin talep ve ihtiyaçlarını ne çapta karşıladığını tanımlar.
- ISO 3241 spesifikasyonu, benzer bir kullanılabilirlik tanımı sağlar:
 - Belirli kullanıcıların belirli ortamlarda belirli hedeflere ulaşma etkinliği, verimliliği ve memnuniyeti.

5-Diğer Kaliteler:

- **Portabilty(Taşınabilirlik)**

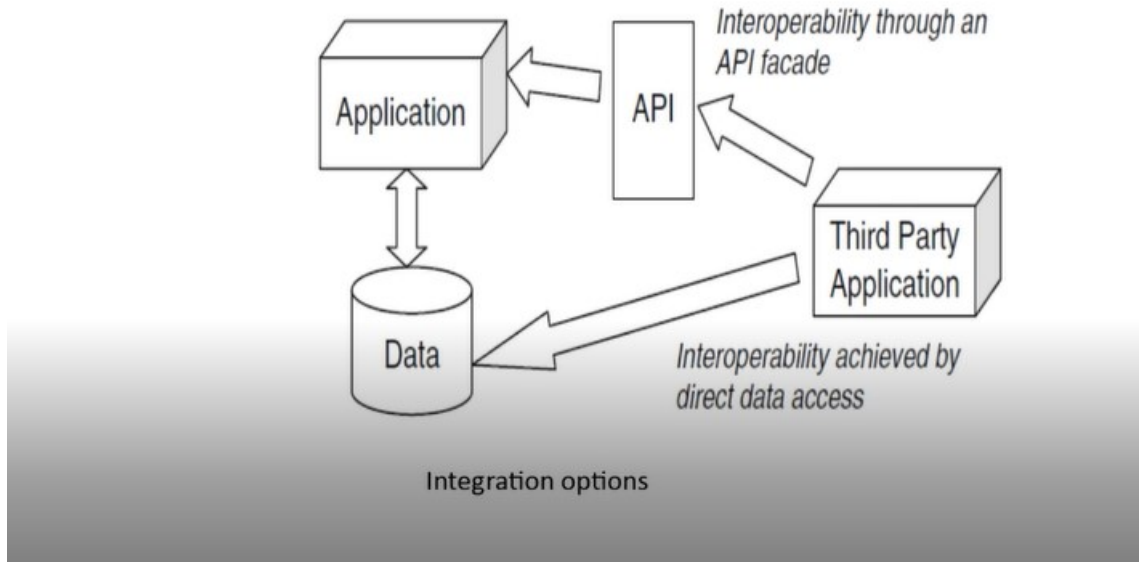
- Bir uygulama , geliştirildiğinden farklı bir yazılım / donanım platformunda kolaylıkla yürütülebilir mi?
- Taşınabilirlik , uygulamayı gerçekleştirmek için kullanılan yazılım teknolojisi tercihlerine ve üzerinde çalışması gereken platformların özelliklerine bağlıdır?
- Kolayca taşınabilir kod tabanları , uygulamanın geri kalanını etkilemeden değiştirilebilen izole edilmiş ve küçük bir bileşen setinde kapsüllenmiş platform bağımlılıklarına sahip olacaktır.

- **Integration (Bütünleştirme)**

- Bir uygulamanın daha geniş bir uygulama bağlamına yararlı bir şekilde dahil edilebilmesinin kolaylığı ile ilgilidir.
 - veri entegrasyonu veya API
- Veri integrasyonu , bir uygulamanın işlediği verileri diğer uygulamaların erişebileceği şekilde depolamayı içerir.
- Bu , veri depolama için standart bir ilişkisel veritabanı kullanmak kadar basit olabilir veya verileri XML gibi bilinen bir biçime veya diğer uygulamaların alabileceği virgülle ayrılmış bir metin dosyasına çıkarmak için mekanizmalar uygulamak kadar basit olabilir.
- Alternatif , birlikte çalışabilirliğin bir API aracılığıyla elde edilmesidir.

- Bu durumda , uygulamamamının sahip olduđu ham veriler , verilere kontrollü harici erişimi kolaylaştıran bir dizi işlemin arkasında gizlenir.
- Bu şekilde ,API uygulamasında iş kuralları ve güvenlik uygulanabilir.
- Verilere erişmenin ve uygulama ile entegre olmanın tek yolu , sağlanan API yı kullanmaktır.

• Integration



Modülerlik Kalitesi

- Bir coherent (sıkı ve türdeş) modülü coupling (artçılama) çalışmak okunurluđu azaltır.



Coupling

programın içerisindeki modüllerden bahseder. Modüllerin arasındaki irtibat ve bağlantı hangi seviyede kurulmalıdır onu belirler. Modüllerin diğer modüllerden bağımsız olması (amaç). Ve bu amacı gerçekleştirirse modüllerde daha basit değişiklikler yapılır ya da daha az riskli değişim yapılır. Çünkü bu değişiklik diğer modülü etkilemez.

- Ne kadar az coupling o kadar iyi kalitedir.

- Sistem içerisinde coherent in yüksek olması lazım.
- Modülerite:
 - Bir dizi standart parça ya da ayrık birimin daha girift bir yapıda kullanılmasıdır.

Program Modülü Modeli

3 parçadan oluşur:

- Inputlar
- Outputlar
- Processler

Coupling ve coherent programın giriftliğini ve karmaşıklığını etkiler.

Coupling ne kadar artarsa karmaşıklık veya giriftlik o kadar artar.

Coherent ise bir modüldeki işlerin diğerlerine taşmaması için olmalı.

Coupling

Coupling in etkilendiği 3 faktör var:

- Modüller arasında geçen **veri birim sayısı** (sayısı ne kadar fazlaysa coupling o kadar kuvvetli ve yüksek) → ama olumsuz bir durum bu
- Modüller arasında geçen **kontrol verisi sayısı** (sayısı ne kadar fazlaysa coupling o kadar kuvvetli ve yüksek) → ama olumsuz bir durum bu
- Modüllerin ortak kullandığı **veri havuzu** (**paylaşılan veri sayısı** ne kadar fazlaysa coupling o kadar kuvvetli ve yüksek) → ama olumsuz bir durum bu

Coupling Tipleri (Yukarıdan aşağıya olumsuzluk derecesi artıyor)

- Data coupling (veri bağılılığı)
- Stemp coupling (mühür , damga)
- Control coupling (kontrol)

- Common coupling (genel, ortak)
- Content coupling (içerik)

Bizim açımızdan en mantıklısı data coupling tir.

Bizim açımızdan an makbul olmayanı ise content coupling tir.

Data Coupling (Veri bağıllığını) düşürmek için ne yapmalıyız?

- Veri modüller arasında geçerken seri halinde atlama ile başka bir modüle gitmemeli. Doğrudan doğruya geçiş yapacaksa bu bağlantılı olduğu modüle geçsin.
- İki modül birbiriyle doğrudan bir parametre olarak geçirilen bir değişken veya dizi (tablo) aracılığıyla iletişim kurarlarsa , iki modül veribağılıdır.
- Veriler, program kontrol amaçları için değil problem ilişkili işlemede kullanılır.

Stemp Couplingi (Birleşik veri aktarımı) düşürmemiz için ne yapmalıyız?

- Bir array ya da tablo şeklinde verimiz var. Bunu bütünüyle diğer modüle aktardığımızda aslında birden fazla lüzumsuz bilgiyi de gönderiyoruz.
- İki modül birleşik (composite) veri ögesi (örneğin kayıt) aracılığıyla iletişim kurarlarsa , iki modül damga bağılıdır.
- Bileşik veri ögesi , bir modül tarafından kendisine iletilse bile kullanılmayan veri parçalarını içerebilir.

Control Coupling

- Çalışmasında birtakım göstergeler kullanırız.
- Bir modülün gelen veriler , diğerinin talimat icra sırasını yönlendirmek için kullanılıyorsa , iki modül kontrol bağılıdır (örneğin bir modülde bir "flag" başka bir modülde karşılaştırma için kullanılır.)

Common Coupling

- Veri havuzuna ortak olarak birden fazla modül buraya erişebiliyor. Bunları kontrol etmek zor.

- İki modül **aynı global veri alanlarını** paylaşıyorlarsa iki modül ortak bağlıdır.

Content Coupling

- Bu içerik bağıllığında bir modül diğer bir modülün elemanına müdahale ediyor. Bu hiç istenmeyen bir şey.
- Bir modül başka bir modüle atıfta bulunuyorsa veya başka bir modülü değiştiriyorsa (örneğin , bir modül kodundan başka bir modüle dallanma veya dönüşme) iki modül içerik bağlıdır.

DeCoupling Kılavuzu

- En iyi , program tasarımı esnasında yapılır.
- Modülleri bağımsız yapma çalışması
- **Data couplingi azaltmak için :**
 - Sadece olan veriyi geçir.
- **Stamp couplingi azaltmak için**
 - Bileşik veri elemanları sadece kullanılacak verileri ihtiva etsin
- **Control couplingi azaltmak için**
 - Mümkün olduğunca sayısını azalt
- **Common Couplingi azaltmak için**
 - Ortak veri alanı kullanmaktan kaçın
- **Content couplingi hepten kaldır**

Cohesion

Cohesion bir modül içindeki öğelerin ne kadar güçlü bağa sahip olduğunu ölçer.

Modül elemanlarının içerisindeki elemanların arasında kuvvetli bir bağ olması lazım ve türdeş olması lazım.

Coherent tipleri:

- Functional coherent

- Temporal coherent
- Sequential coherent
- Logical coherent
- Communicational coherent
- Coincidental coherent
- Procedural coherent

Functional coherent:

İşlevsel olarak bir modülün sıkı ve türdeş olduğunu söylememiz için elemanların vazgeçilmez temel olması lazım

Bir modüldeki her öge, bir ve yalnızca bir işlevin gerekli ve temel bir parçasıdır.

Modülün her parçası diğeriyle ilişkilidir ve modül , çalışması için gerekli olan her şeyi içerir.

- Örnek: Modül : ödemeyi bulmak;
 1. ödemeyi hesaplıyor
 2. brütü hesaplıyor
 3. net i hesaplıyor

Sequential coherent

Bir modülün elemanlar arasında çalışma sırasında bir ardışıklık vardır. Ama burada önemli olan birinin input u diğeriinin output una teşkil ediyor. Henüz modül dışına çıkmadan içeride de çalışma devam ediyor.

- Örnek: Modül : Stokları update etmek ;
 1. stokun ne olduğuna dair veriyi alıyoruz
 2. birim olarak çeviriyoruz
 3. update işlemini yapıyoruz

Logical coherent:

Modül yapılarının yaptıkları işlerinin aralarında mantıksal bir ilişki vardır. Soyut bir ilişki

İşlemler aynı, veriler farklı.

Communicational coherent:

Bir modülün elemanları belirli bir veri üzerinde çalışıyor. Belli bir sıra içinde yapılması

- Örnek: Modül : Hareketi işle ;
 1. hareketi oku
 2. hareketin üzerinde oynama yap
 3. hareketi üzerine işle

Temporal coherent:

Zaman itibariyle aynı anda yapıyoruz fakat onların birbirleriyle ilişkili değiller.

- Örnek: Modül :başlat;
 1. sayaçları bul
 2. tabloları sıfırla
 3. tabloları başlat

Coincidental coherent

Tamamen tesadüfi bir şekilde işlemleri yapıyor. Oldukça zayıf bir bağ vardır.

Bir modülün öğeleri, temelde herhangi bir ortak işlev , prosedür , veri veya herhangi başka bir şeyle ilişkili değildir.

- Örnek: Modül : ;
 1. dönüştürme oku
 2. baskı hatası yapar
 3. read old master
 4. sayacı sıfırla

Procedural coherent

Modülün tüm parçaları bir prosedürün belirli bir sırayla yapılması gereken belirli bir sıra adımlarının tamamıdır.

- Örnek: Modül : siparişleri işle; (döngü halinde)
 1. siparişleri oku
 2. siparişleri işle

En makbulu **Functional coherent**:

En makbulsuzu **Concidental**

Modül nedir?

- Modül bir sistemin daha küçük parçalara ayrılarak karmaşıklığın azaltılması.

Modül kalitesi ile yazılım kalitesi karıştırılmamalı

- Modül kalitesinde 2 temel ölçü var
 - coupling
 - coheision

Düşük coupling yüksek coheision olmalı (ideal)

Coupling nedir?

- Bir modül içerisindeki elemanların birbirleriyle sıkı ve türdeş olması

Coupling türleri nelerdir?

- Data coupling
- Control coupling
- Content coupling

Kalite açısından en mantıklı olanı data coupling tir.

En mantıksız olanı ise content coupling tir.

Coheision türleri?

- En mantıklı olan fonksiyonel
- En mantıksız olan cohesidental

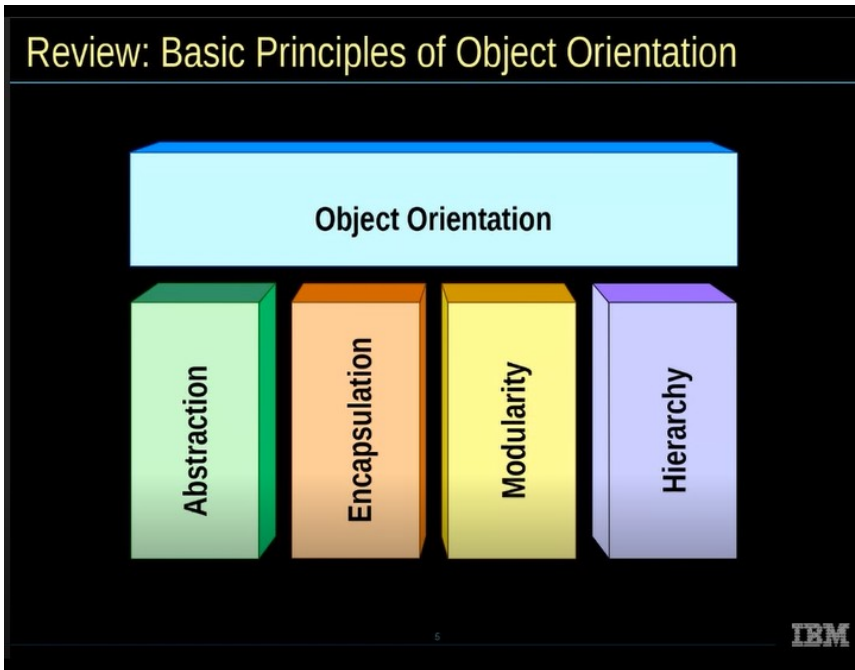
OBJECT ORIENTATION

- Notasyon olarak genelde UML kullanılır.
- Modelleme dört amaca ulaşır:
 - Bir sistemi olmasını istediğiniz gibi görselleştirmenize yardımcı olur.
 - Bir sistemin yapısını veya davranışını belirlemenize izin verir.
 - Size bir sistem oluştururken rehberlik edecek bir şablon verir.
 - Verdiğiniz kararları belgeler.
- Böyle bir sistemi bütünüyle kavrayamadığınız için karmaşık sistemlerin modellerini oluşturuyorsunuz
- Geliştirdiğiniz sistemi daha iyi anlamak için modeller oluşturursunuz.

Modellemenin Dört İlkesi

- Oluşturduğunuz model, soruna nasıl saldırılacağını etkiler.
- Her model, farklı hassasiyet seviyelerinde ifade edilebilir.
- En iyi modeller gerçeklikle bağlantılıdır.
- Tek bir model yeterli değildir.

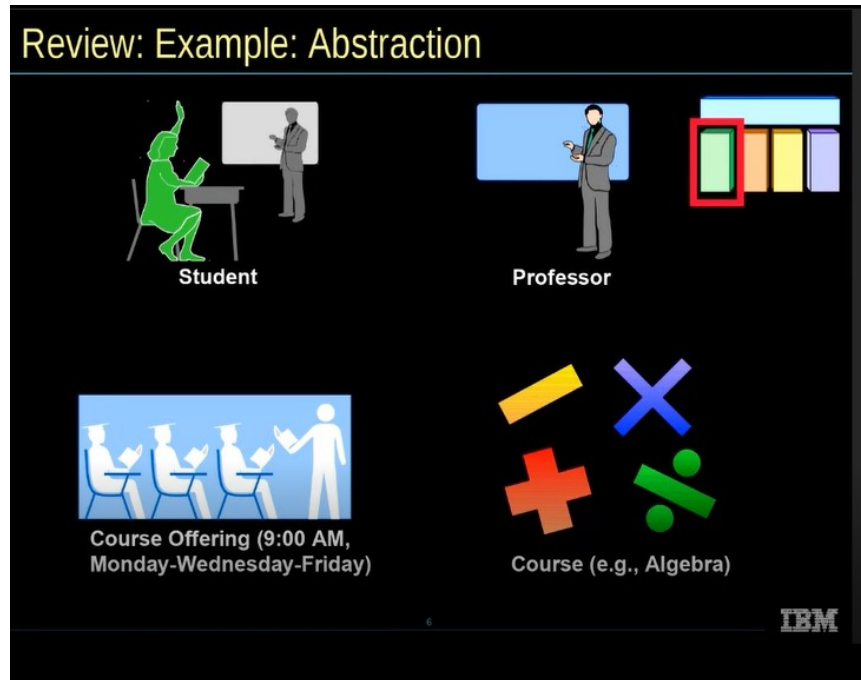
Gözden Geçirme: Nesne Yöneliminin Temel İlkeleri



Abstraction

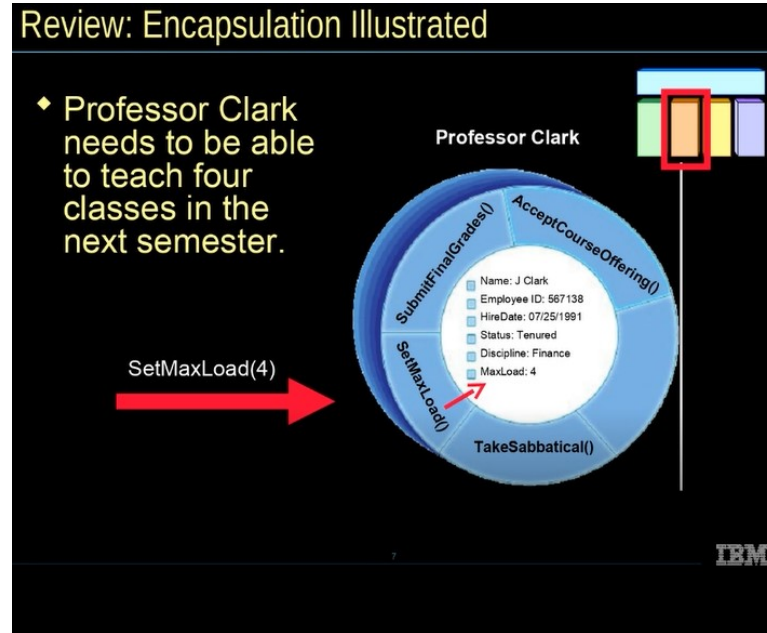
Soyutlama

- Burada öğrenci derken soyutlama yapıyoruz çünkü hangi öğrenci olduğunu bildirmiyoruz.



Encapsulation

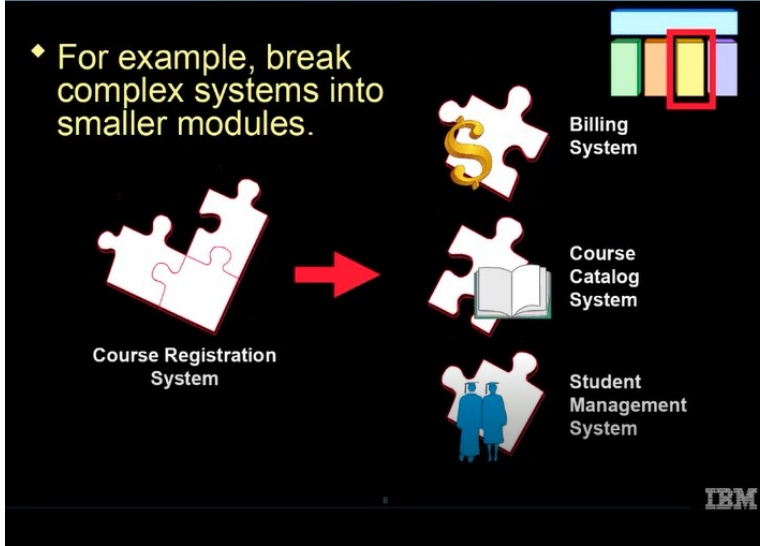
- Aynı özellikleri taşıyan varlıkların bir araya getirilmesi ama bizi kısıtlamaması gerekir.
- Sınıf içerisindeki muamelelere veya objenin işlerine bir başkasının karışmaması



Modularity

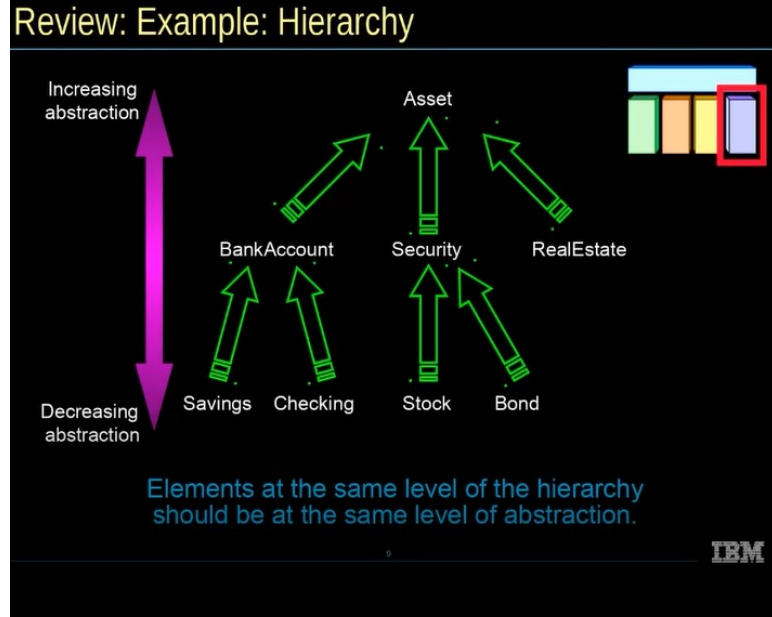
- Bir büyük işi daha küçük parçalar halinde devreye sokmak. Bunu yaparken modülerasyon ilkelerine uyarak yapmak önemli yani coupling ve coheision ilkelerine uyarak yapmalıyız.

Review: Example: Modularity



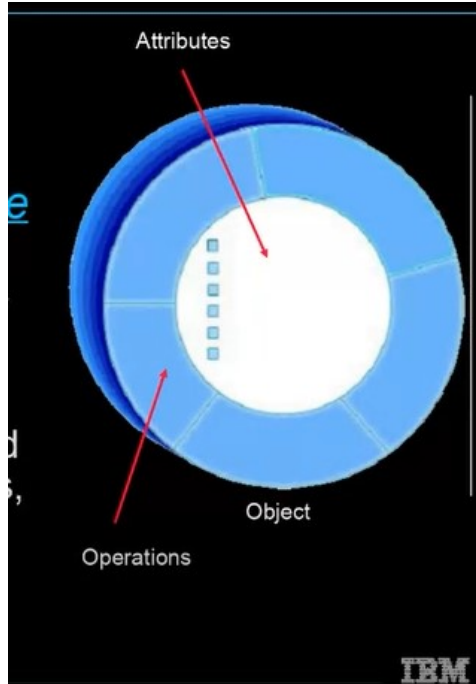
Hierarchy

- Kademelendirme
- Aynı hiyerarşi düzeyindeki öğeler, aynı soyutlama düzeyinde olmalıdır.
- Sınıf objelerin topluluğu değildir. Burada önemli olan aynı attributelara aynı davranışlara sahip olan objelerin bir araya gelmesidir.
- Asset: varlık
- Aynı seviyede olması lazım



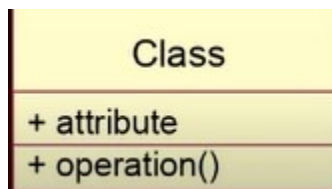
Gözden Geçirme: Nesne

- Bir nesne, iyi tanımlanmış bir sınıra ve durumu ve davranışı kapsayan bir kimliğe sahip bir varlıktır.
 - Devlet, nitelikler ve ilişkilerle temsil edilir.
 - Davranış, işlemler, yöntemler ve durum makineleri ile temsil edilir.



Gözden Geçirme: Sınıf

- Sınıf, aynı öznitelikleri, işlemleri, ilişkileri ve anlambilimini paylaşan bir dizi nesnenin açıklamasıdır.
 - Nesne, bir sınıfın örneğidir.
- Bir sınıf, içinde bir soyutlamadır.
 - İlgili özellikleri vurgular.
 - Diğer özellikleri bastırır.



Class ın ortaya çıkış şekli \Rightarrow object



entity nin ortaya çıkış şekli \Rightarrow couple veya row

Attribute nedir

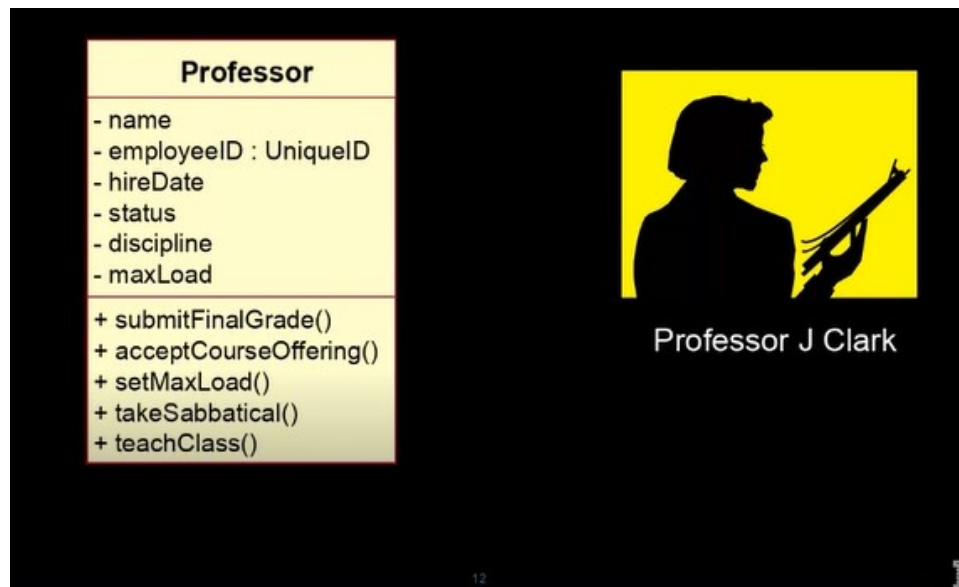
- Bir nitelik , bir sınıfın değerler aralığını tarif eden adlandırılmış bir özelliğidir ki özelliklerin oluşlarını tutabilir.
- Bir sınıfın herhangi bir sayıda niteliği olabilir veya hiç olmayabilir.

Operation (operasyon) nedir

- Bir nesneden , davranışı etkilemek için talep edilebilecek bir hizmet . Bie eylem mümkün olan gerçek parametrelerin kısıtlayabilecek bir imzaya sahiptir.
- Bir sınıfın herhangi bir sayıda eylemi olabilir veya hiç olmayabilir.

Gözden geçirme: UML'de Sınıfları Temsil Etme

Bir sınıf, bölmeli bir dikdörtgen kullanılarak temsil edilir.



1.bölümde itentity var kimliği

- 2. bölümde attribute ler var.
- 3. bölümde de operations var.

Gözden geçirme: Sınıflar ve nesneler arasındaki ilişki

- Bir sınıf, bir nesnenin soyut bir tanımıdır
 - Oluşturmak için bir şablon görevi görür.
 - Yapıyı ve davranışı tanımlar.
- Sınıflar, nesne koleksiyonları değildir.

Sınıf İlişkileri

- Sınıflar arasındaki anlamsal bağlantı
- Sınıf diyagramları aşağıdakileri içerebilir



Normal Ok → Normal Bağlantı

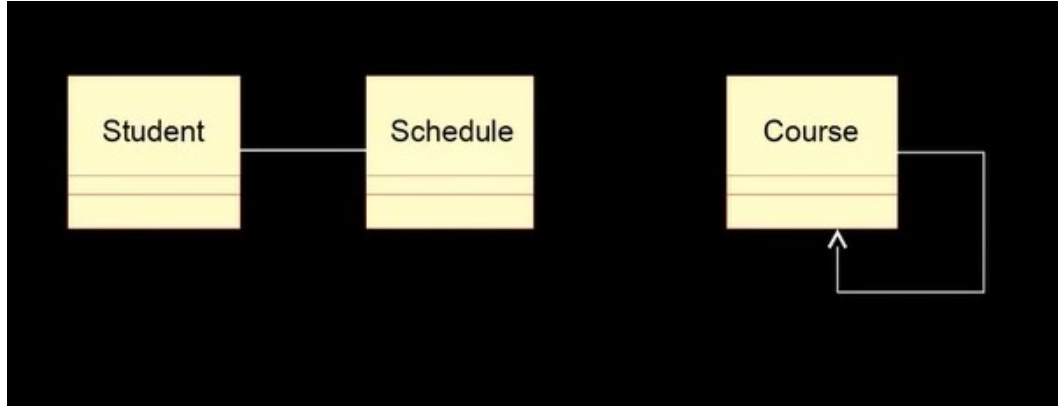
Kesikli Ok → Arayüz Bağlantısı

Aggregation → Zayıf İlişki, örn: bilgisayar - mouse

Composition → Güçlü İlişki, örn: insan - kalp

Bağlantı, İrtibat (Association)

- Örnekleri arasındaki bağlantıları belirleyen iki veya daha fazla sınıflandırıcı arasındaki anlamsal ilişki.
 - Bir sınıfın nesnelerinin ile diğerinin nesnelerine bağlı olduğunu belirten yapısal bir ilişki.



Multiplicity (Çokluk)

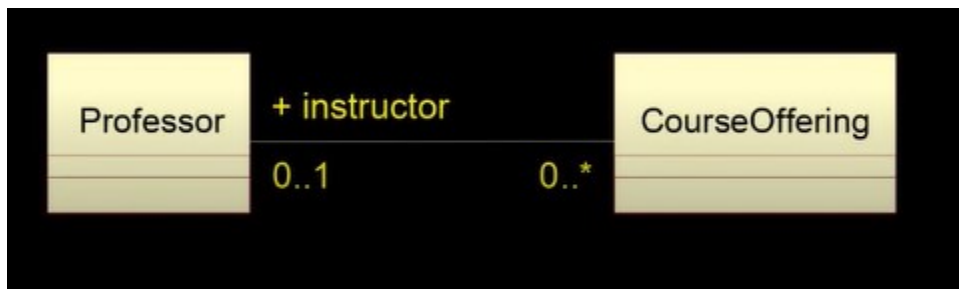
- Çokluk, bir sınıfın başka bir sınıfın bir örneğiyle ilgili olduğu örneklerin sayısıdır.
- Her bağlantı(association) için, her bir bağlantı (association) sonu için bir tane olmak üzere, alınacak iki çokluk kararı vardır.
 - Profesör'un her bir örneği için birçok Kurs Teklifi öğretilebilir.
 - Kurs Tekliflerinin her bir örneği için, eğitmen olarak bir veya sıfır Profesör olabilir.

Review: Multiplicity Indicators

Unspecified	
Exactly One	1
Zero or More	0..*
Zero or More	*
One or More	1..*
Zero or One (optional scalar role)	0..1
Specified Range	2..4
Multiple, Disjoint Ranges	2, 4..6

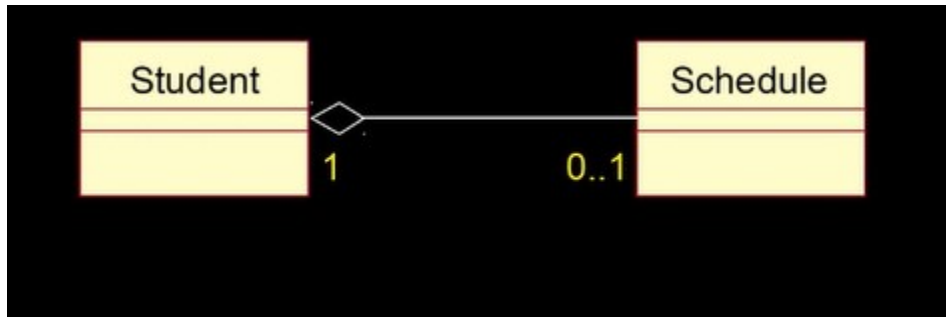
18

IBM



İçerme, Bütünleşme (Aggregations)

- Birden fazla parçadan oluşur
- Bir bütün ilişkisi vardır.
- İçi boş baklava dilimleri ile birbirine bağlanır.
- Örneğin: TV'nin bileşenleri ekran, kumanda, devre; devrenin bileşenleri: pil tuş takımı, ışık lambasıdır.



- Bu ilişkinin daha kuvvetli formu, bileşik (Composition) ilişkisidir.
- Diğer bağlantı(associations) gibi çokluk temsil edilmektedir.
- Bu ilişki logic ilişkidir çünkü çizelge öğrenciyi oluşturmuyor.(fiziksel olarak bir parçası değil)

Composition (Bileşik)

- Kompozisyon, kompozitin kendi bölümlerini - ayırma ve serbest bırakma gibi yönetmekle sorumlu olduğu daha güçlü bir ilişkilendirme biçimidir.
- Karşı ucunda elmas dolgulu bir süsleme ile gösterilmiştir.



Generalization (genelleme)

- Bir sınıfın bir veya daha fazla sınıfın yapısını ve / veya davranışını paylaştığı sınıflar arasındaki ilişki.
- Gruplama yapılıyor.
- Bir alt sınıfın bir veya daha fazla üst sınıftan miras aldığı bir soyutlama hiyerarşisini tanımlar.
 - Çoklu miras (Multiple)
 - Tek miras (inheritance)
- "Bir tür" ilişkidir
- inheritance generalization ile de örtüşüyor. Inheritance bir mekanizmadır.
- bir child birden fazla parent a bağlanabilir

Kalıtım

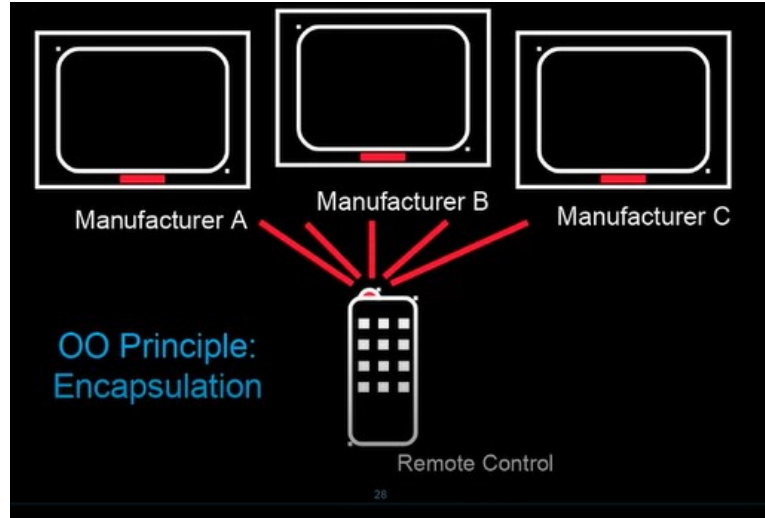
- Ortak özelliklere bağlıdır.
- İçi boş oklarla birbirine bağlanır.

..



Polymorphism nedir

- Tek bir arayüzün arkasına birçok farklı uygulamayı gizleyebilme yeteneği
- Yine bir soyutlama yapılıyor.



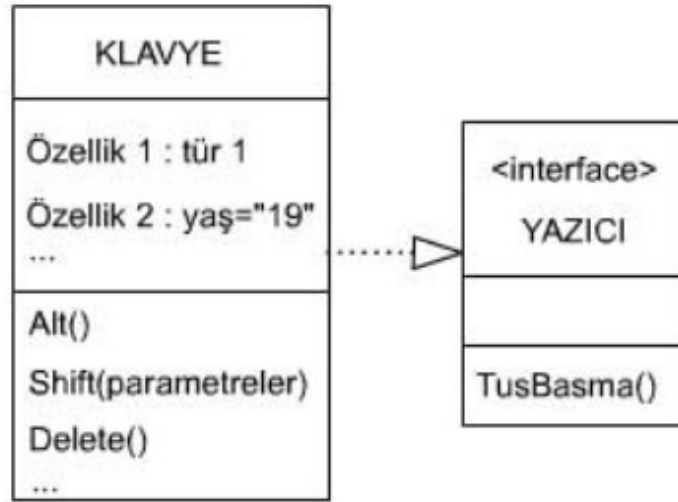
Interface

- Tutarlı (coherent) bir kamusal özellikler ve yükümlülükler kümesi beyanı.
- Hizmet sağlayıcıları ve tüketicileri arasında bir sözleşme. Arayüz örnekleri şunlardır
 - Sağlanan (provided) arayüz \Rightarrow Elemanın çevresine maruz kaldığı arayüzler
 - Gerekli (required) arabirimler \Rightarrow Öğenin, sağlanan işlevselliğin tam setini sunabilmesi için ortamındaki diğer öğelerden gerektirdiği arabirimler.

Arayüz (Interface)

- Bir takım işleri yerine getirmek için başka sınıflar tarafından kullanılırlar.
- Arayüzlerin özellikleri yoktur.

anabiliriz.


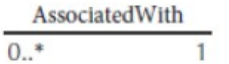
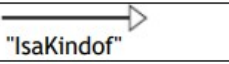




Bileşen Diyagramları

- Bileşen Diyagramları yazılım sistemine daha yüksek bir seviyeden bileşenler seviyesinden bakabilmeyi sağlar.
- Bunlar sistemdeki sınıflar, çalıştırılabilen program parçaları kütüphane bileşenleri veya veritabanı tabloları olabilir.
- Bu gösterim bileşenler, arayüzler ve bağımlılık ilişkilerinden oluşur ve sistemin fiziksel gösterimini sağlar.

Dağılım Diyagramları

- Sistemin çalışma platformundaki durumlarını gösterirler.
- Sistemdeki yürütülebilen parçalar, kütüphaneler, tablolar ve dosyalar gibi bileşenlerin dağılımını belirler.
- Bu diyagramlar, sistem donanım mimarisinin gösterilmesi, gerekli donanım bileşenlerinin tanımlanması amacıyla kullanılabilirler.

A class: <ul style="list-style-type: none"> • Kalın yazılmış ve üst bölümünde ortalanmış bir adı vardır. • Orta bölümünde bir nitelikler listesi vardır. • Alt bölümünde eylemlerin bir listesi vardır. • Sistemin haklarındaki enformasyonu yakalaması ve saklaması gereken bir tür kişiyi, yeri veya şeyi temsil eder. • Mevcut olan eylemleri bütün sınıflar için açıkça göstermez. 	 <pre> classDiagram class Class1 { -Attribute-1 +Operation-1 () } </pre>
An attribute (nitelik): <ul style="list-style-type: none"> • Bir nesnenin durumunu tanımlayan özellikleri temsil eder. • Adının önüne eğik çizgi koyarak gösterilen diğer niteliklerden türetilir. 	attribute name / derived attribute name
An operation (eylem): <ul style="list-style-type: none"> • Bir sınıfın gerçekleştirebileceği eylemleri veya işlevleri gösterir. • Yapıcı, sorgu veya güncelleme eylemi olarak sınıflandırılabilir. • Eylemi gerçekleştirmek için gerekli parametreleri veya bilgileri ihtiva eden parantezler içerir 	Operation name()
An association(irtibat): <ul style="list-style-type: none"> • Bir sınıfın kendisi ile veya birden çok sınıf arasındaki ilişkiyi temsil eder. • İlişkiyi daha iyi temsil eden bir fiil cümlesi veya rol adı kullanılarak etiketlenir. • Bir veya daha fazla sınıf arasında olabilir. • Bir sınıf oluşunun ilgili diğer bir sınıf oluşuyla irtibatlandırılabilceği minimum ve maksimum sayıları temsil eden çokluk sembolleri içerir. 	 <pre> classDiagram class0 "0..*" -- "1" class1 : AssociatedWith </pre>
A generalization(genelleştirme): Birden fazla sınıf arasındaki "-inBirTürüdür" ilişkisini gösterir.	 <pre> classDiagram class0 -- > class1 : "IsaKindof" </pre>
An aggregation: <ul style="list-style-type: none"> • Birden çok sınıf veya bir sınıf ile kendisi arasındaki mantıksal "-inBirParçasıdır" ilişkisini gösterir. • İrtibatın özel bir biçimidir. 	 <pre> classDiagram class0 "0..*" -- "1" class1 : IsPartOf </pre>
A composition: <ul style="list-style-type: none"> • Birden çok sınıf veya bir sınıf ile kendisi arasındaki fiziksel "-inBirParçasıdır" ilişkisini gösterir. • İrtibatın özel bir biçimidir. 	 <pre> classDiagram class0 "1..*" -- "1" class1 : IsPartOf </pre>

Sınıflar nesne koleksiyonları değildir.

- Sınavda association kelimesi yerine irtibat kullanılmış
- Modüller arası ve modüllerin öğeleri arası farklı konu (sınavda dikkat et)
- Bağlantı (Association) (SINAV)
- Sınıf ilişkileri
 - Dependency ve Realization ilişkileri sınava dahil değil
 - Composition da bütünü bozduğumuz zaman parçada bozulur fakat aggregation da bütünü yok edersek parça müstakil olarak devam eder.
 - Aggregation ve Composition bütün parça ilişkisidir.

Data Connection

Modüller arası veri geçişi,

- Eğer modüller arasında parametre vasıtasıyla ise,
 - açık veri irtibatı (explicit)
- Eğer modüller aynı veriyi referans alıyor ise
 - örtük veri irtibatı (implicit)

var demektir.

Data Connection complexity (Veri Bağlantısı karmaşıklığı)

- Modüller arası irtibatın giriftliği kontrol edilmeli ve azaltılmalıdır.
- Bu irtibatın (bağlantının) irdelenmesi için iki temel ölçü var
 - Coupling
 - Cohesion

Bağımsızlık Ölçüsü

- Modüller arasındaki bağımsızlık düzeyini **coupling** ile ölçüyoruz.
- Bir modülün ögeleri arasındaki ilişkinin gücünü (seviyesini) **cohesion** ile ölçüyoruz.

Modül Kalitesinin İki Esası

- Loosely Coupled (gevşek bağ)
 - Modüller arasında küçük etkileşim
 - Ters: Tightly coupled (kuvvetli bağ)
- Highly Cohesive (Yüksek sıklık-türdeşlik)
 - Modül ögeleri arasında güçlü etkileşim

Couplingi Etkileyen Faktörler

- Modüller arasında geçen veri elemanı sayısı
- Modüller arasında geçen kontrol verisi miktarı
- Modüller tarafından kullanılan ortak global veri elemanlarının sayısı

Veri elemanı sayısı nedir?

- Bir modül çağrısında yer alan parametre veya argüman sayısı
- Örneğin: 100 parametre içeren bir prosedür çağrısı 1 parametre içerenden daha giriftir(karmaşıktır)

Kontrol verisi nedir?

- Program talimatlarının (instruction) ve modül çağrılarının icra sırasını yönlendirir.
- Örneğin: dosya sonu , hareket kodu

PATTERNS AND TACTICS (Kalıplar ve Taktikler)

- What is a Pattern?
- Pattern Catalogue
 - Module patterns
 - Component and Connector Patterns
 - Allocation Patterns
- Relation Between Tactics and Patterns
- Using tactics together
- Summary

Pattern (Kalıp) Nedir?

- Bir mimari kalıp aşağıdakiler arasında bir ilişki kurar
- Bir bağlam: Dünyada bir soruna yol açan , tekrarlayan , yaygın bir durum
- Bir sorun: Verilen bağlamda ortaya çıkan , uygun bir durum
- Bir sorun : Verilen bağlamda ortaya çıkan , uygun şekilde genelleştirilmiş sorun.
- Bir çözüm : Uygun şekilde soyutlanmış probleme başarılı bir mimari çözüm. Bir kalıp için çözüm şu şekilde belirlenir ve açıklanır:
 - Bir dizi öge türü (örneğin , veri havuzları , işlemler ve nesneler)
 - Bir dizi etkileşim mekanizması veya bağlayıcılar (örneğin , method çağrılar, olaylar veya mesaj veri yolu)
 - Bileşenlerin topolojik deseni
 - Topolojiyi , öge davranışını ve etkileşim mekanizmalarını kapsayan bir dizi anlamsal kısıtlama

Genel Bakış-1

- Faydalı ve yaygın olarak kullanılan kalıplar
- Bu kataloğun kapsamlı olması amaçlanmıştır.
 - Temsili olması amaçlanmıştır.
- Run (koşma) zamanı ögelerinin kalıpları (aracı veya istemci-sunucu gibi)
- Tasarım zamanı ögeleri (tabakalar/(katmanlar gibi))
- Her kalıp için bağlamı , sorunu ve çözümü listeliyoruz. Çözümün bir parçası olarak, her kalıbın unsurlarını , ilişkilerini ve kısıtlarını kısaca açıklıyoruz.

Genel Bakış-2

- Bir kalıp uygulaması "ya hep ya hiç" önerisi değildir.
- Uygulamada mimarlar, iyi bir tasarım ödeshimi olduğunda bunları küçük şekillerde ihlal etmeyi seçebilirler.
- Kalıplar , gösterdikleri baskın öge türlerine göre kategorize edilebilir:
- modül kalıpları modülleri gösterir

- bileşen ve bağlayan (C&C) kalıpları bileşenleri ve bağlayanları gösterir ve
- tahsis kalıpları , yazılım öğelerinin (modüller , bileşenler, bağlayıcılar) ve yazılım dışı öğelerin bir kombinasyonunu gösterir.

LAYER PATTERN-1 (Tabaka Kalıbı)

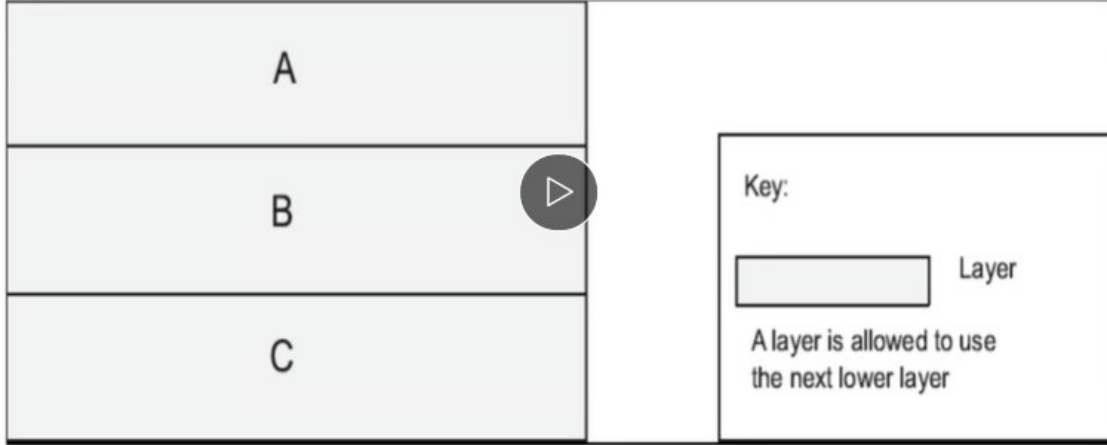
BAĞLAM: Tüm karışık sistemler, bölümlerin bağımsız olarak geliştirilme ve evrilmesine ihtiyaç duyar. Bu nedenle , sistemin geliştiricileri , sistem modüllerinin bağımsız olarak geliştirilip sürdürülebilmesi için açık ve iyi belgelenmiş bir **ilgiler ayırımına** ihtiyaç duyar.

SORUN: Yazılımın , **bölmömlere ayrılmaya ihtiyacı** vardır. Öyle ki **modüller** , taşınabilirliği , değıştirilebilirliği ve yeniden kullanımı destekleyecek şekilde , **parçalar arasında çok az etkileşimle ayrı ayrı geliştirilebilecek ve evrilebilecek** olsun.

ÇÖZÜM: Bu **ilgi ayırımına** ulaşmak için **tabakalı kalıp** , yazılımı tabaka adı verilen birimlere ayırır. Her tabaka , birleşik bir hizmet kümesi sunan bir modül grubudur. Kullanım tek yönlü olmalıdır. Tabakalar bir yazılım kümesini tamamen bölümlere ayırır ve her bölüm bir açık-genel arabirim aracılığıyla sergilenir.



Layer Pattern Example



© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

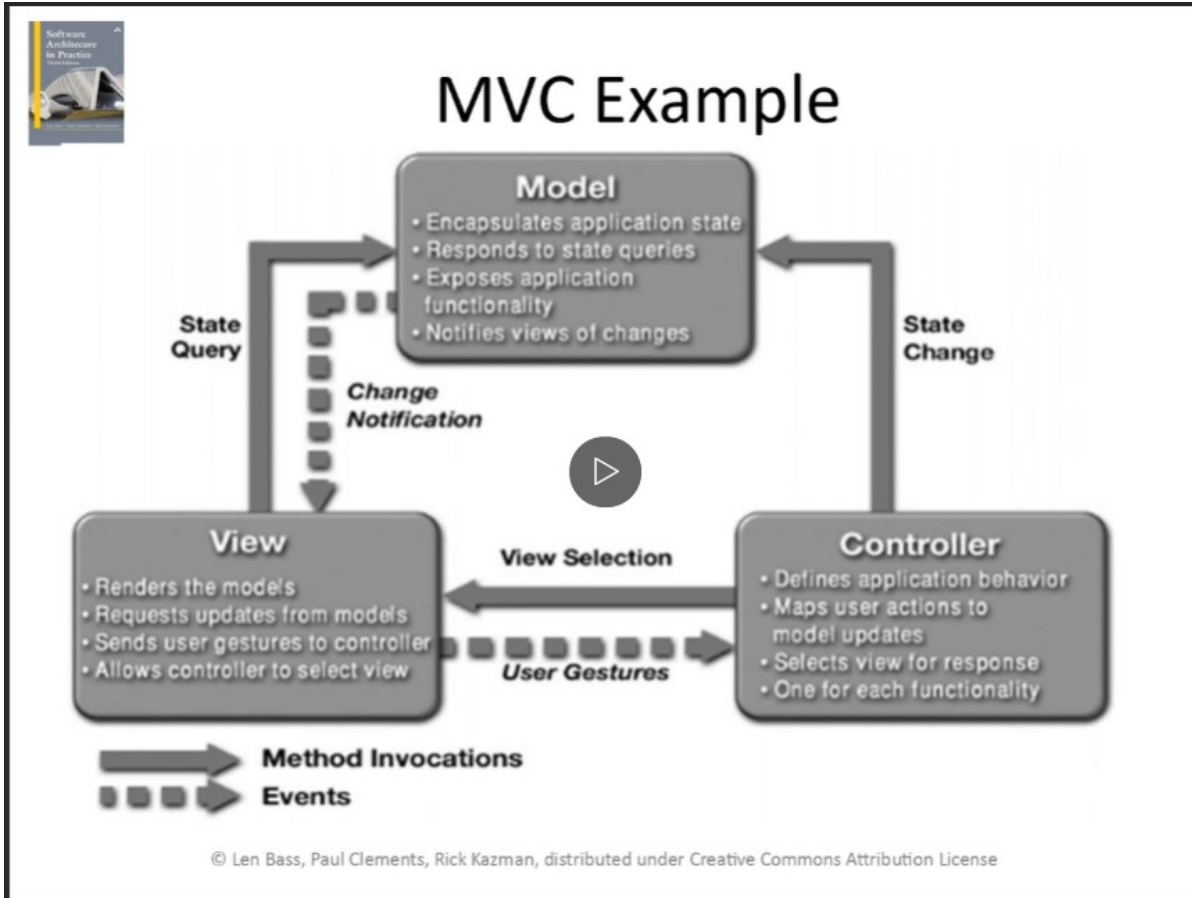
LAYER PATTERN SOLUTION (Tabaka Kalıbı Çözümleri)

- **GENEL BAKIŞ** : Tabakalı desen , tabakaları (birleşik hizmet kümesi sunan modül gruplamaları) ve tabakalar arasında tek yönlü bir kullanıma izin verilen ilişkiyi tanımlar.
- **ÖGELER**: Tabaka , bir tür model .Bir tabakanın açıklaması , tabakanın hangi modüller içerdiğini tanımlamalıdır.
- **İLİŞKİLER**: Kullanmasına imkan verilir. Tasarım , tabaka kullanım kurallarının ne olduğunu ve imkan verilen istisnaları tanımlamalıdır.
- **KISITLAR**:
 - Her yazılım parçası tam olarak bir tabakaya tahsis edilmiştir.
 - En az iki tabaka vardır (ancak genellikle üç veya daha fazla vardır.).

- **Kullanmasına imkan verilir** ilişkileri dairesel olmamalıdır. (yani , daha düşük bir tabaka yukarıdaki bir tabakayı kullanamaz).
- **ZAYIF YÖNLERİ:**
 - Tabakaların eklenmesi , bir sisteme ön maliyet ve karmaşıklık ekler.
 - Tabakalar performans kaybını artırır.

MODEL-VIEW-CONTROLLER PATTERN (MVC KALIBI)

- **BAĞLAM:** Kullanıcı arayüz yazılımı , genellikle etkileşimli bir uygulamanın en sık değiştirilen kısmıdır. Kullanıcılar genellikle verilere **farklı açılardan** bakmak isterler (çubuk grafik veya pasta grafik gb.). Bu gösterimlerin her biri verilerin mevcut durumunu yansıtmalıdır.
- **SORUN:** Kullanıcı arayüz **işlevselliği** uygulamanın işlevinden **nasıl ayrı tutulabilir** ve bu durumda kullanıcı girdisini veya temel alınan uygulamanın verilerindeki **değişikliklere nasıl tepki** verebilir? Ve temeldeki uygulamanın verileri değiştiğinde , kullanıcı arayüzünün **birden çok görünümü** nasıl oluşturulabilir , idame ettirilebilir ve koordine edilebilir ?
- **ÇÖZÜM :** Model-Görünüm-Denetleyici (MVC) kalıbı , **uygulama işlevselliğini** üç tür bileşene ayırır:
 - bir **model** , uygulamanın verilerini içerir
 - bir **görünüm** , temel verilerin bir kısmını görüntüler ve kullanıcıya etkileşim kurar.
 - bir **denetleyici** , model ve görünüm arasında aracılık eder ve durum değişikliklerinin bildirimlerini yönetir.



⇒ Çağrılar: kesiksiz çizgi

⇒ Olaylar: kesikli çizgi

MODEL

- uygulama durumunu sarmalar dışarıya karşı
- o haldeki sorgulamalara cevap dönüyor
- uygulama işlevlerini ortaya koyuyor (sergiliyor)
- view değişiklikleri bildirimde bulunuyor

VIEW

- modeli çalıştırıyor
- modelden gelecek güncellemeleri alır
- sistemle ilgili hareketleri controller a iletiyor.
- controller ın cevap vermesine imkan sağlıyor

CONTROLLER

- uygulama davranışını tanımlıyor
- view ile model arasındaki değişimlerin örtüştürülmesini sağlıyor (her bir iş için)

MVC SOLUTION (MVC Çözümü)

GENEL BAKIŞ: MVC kalıbı , sistem işlevselliğini üç bileşene ayırır : model , görünüm ve model ile görünüm arasında aracılık eden bir denetleyici

UNSURLAR:

- Model , uygulama verilerinin veya durumunun bir temsilidir ve uygulama mantığını içerir. (veya buna bir arayüz sağlar)
- Görünüm , kullanıcı için modelin bir temsilini üreten veya bazı kullanıcı girdi biçimlerine veya her ikisine imkan veren bir kullanıcı arayüzü bileşenidir.
- Denetleyici , kullanıcı eylemlerini modeldeki değişikliklere veya görünümdeki değişikliklere dönüştürerek model ve görünüm arasındaki etkileşimi yönetir.

İLİŞKİLER:

- Bildirir ilişkisi , model , görünüm ve denetleyici oluşlarını birbirine bağlayıp ilgili durum değişiklik öğelerini bildirir.

KISITLAR:

- Model , görünüm ve denetleyicinin her birinin en az bir oluşu olmalıdır.
- Model bileşeni , kontrolör ile doğrudan etkileşime girmemelidir.

ZAYIF YÖNLER:

- Basit kullanıcı arayüzleri için karmaşıklık buna değmeyebilir
- Model , görünüm ve denetleyici soyutlamaları , bazı kullanıcı arayüzü araç takımları için uygun olmayabilir.

CLIENT-SERVER PATTERN

BAĞLAM:

- Çok sayıda dağıtık kullanıcının (client) **erişmek** istediği ve erişim veya hizmet **kalitesini kontrol etmek** istediği **ortak kaynaklar ve hizmetler** vardır.

SORUN:

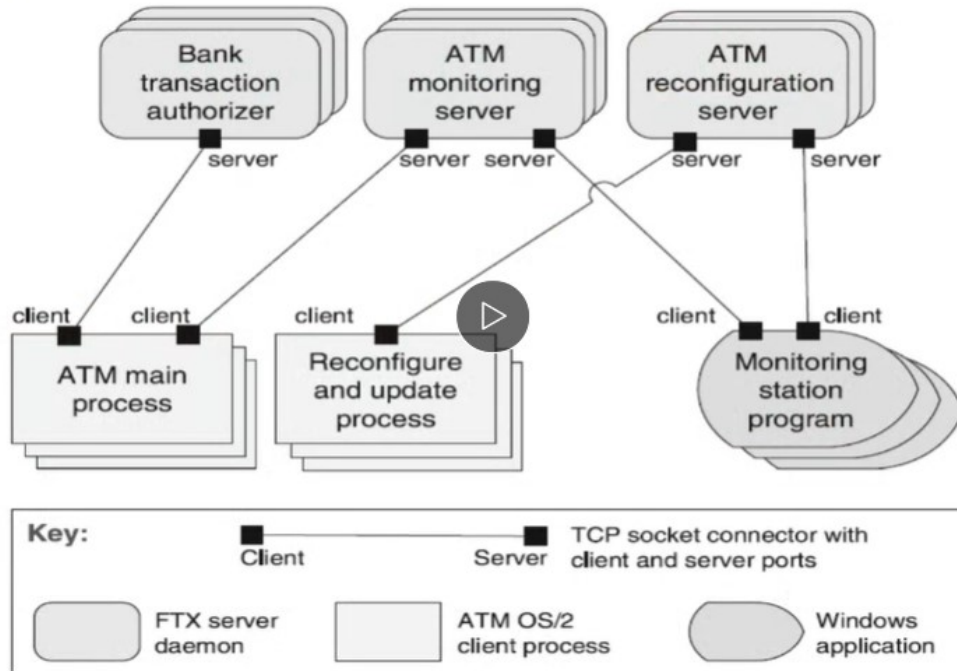
- Bir dizi ortak kaynak ve hizmeti **yönetebilir** ; ortak hizmetleri dışarıda bırakarak ve bunların tek bir yerde veya az sayıda yerde değiştirilmesini mümkün kılarak **değiştirilebilirliğini ve yeniden kullanımı** teşvik edebiliriz.
- Kaynakları birden çok fiziksel sunucuya dağıtırken , bu kaynakların ve hizmetlerin kontrolünü merkezleştirerek **ölçeklenebilirliği ve kullanılabilirliği** iyileştirmek istiyoruz

ÇÖZÜM:

- İstemciler, bir dizi hizmet sağlayan sunucudan hizmet talep ederek etkileşimde bulunur.
- Bazı bileşenler hem istemci hem de sunucu görevi görebilir.
- Bir merkezi sunucu veya birden fazla dağıtık sunucu olabilir



Client-Server Example



© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

CLIENT-SERVER SOLUTION

GENEL BAKIŞ:

- İstemciler, ihtiyaç duyulan hizmetleri sunuculardan isteyerek ve bu isteklerin sonuçlarını bekleyerek sunucularla etkileşimi başlatır.

UNSURLAR:

- İstemci, sunucu bileşeninin hizmetlerini çağıran bir bileşen . İstemcilerin ihtiyaç duydukları hizmetleri tanımlayan bağlantı noktaları (port) vardır
- Sunucu , istemcilere hizmet sağlayan bir bileşen. Sunucular, sağladıkları hizmetleri tanımlayan bağlantı noktalarına sahiptir.

İSTEK / CEVAP BAĞLAYICISI:

- İstemci tarafından bir sunucudaki hizmetleri çağırmak için kullanılan bir istek / cevap protokolü kullanan bir veri bağlayıcısı . Önemli özellikler , aramaların yerel mi yoksa uzak mı olduğunu ve verilerin şifreli olup olmadığını

İLİŞKİLER:

- Ek ilişkisi , istemcileri sunucularla ilişkilendirir.

KISITLAR:

- İstemciler, istek/ cevap bağlayıcıları aracılığıyla sunuculara bağlanır.
- Sunucu bileşenleri , diğer sunuculara istemci olabilir.

ZAYIF YÖNLER:

- Sunucuda bir performans darboğazı olabilir.
- Sunucu , tek bir hata noktası olabilir.
- İşlevselliğin (istemci veya sunucuda) nerede bulunacağına ilişkin kararlar genellikle karmaşıktır ve bir sistem kurulduktan sonra değiştirilmesi maliyetlidir.

SOA KALIBI (HİZMET ODAKLI MİMARİ)

BAĞLAM:

- Bir dizi **hizmet** , hizmet **sağlayıcıları** tarafından sunulur (ve tanımlanır) ve hizmet **tüketicileri** tarafından tüketilir.
- Hizmet tüketicileri , bunların uygulanmasına ilişkin ayrıntılı bilgi sahibi olmadan bu hizmetleri anlayabilmeye ve kullanabilmeye muhtaçtır.

SORUN:

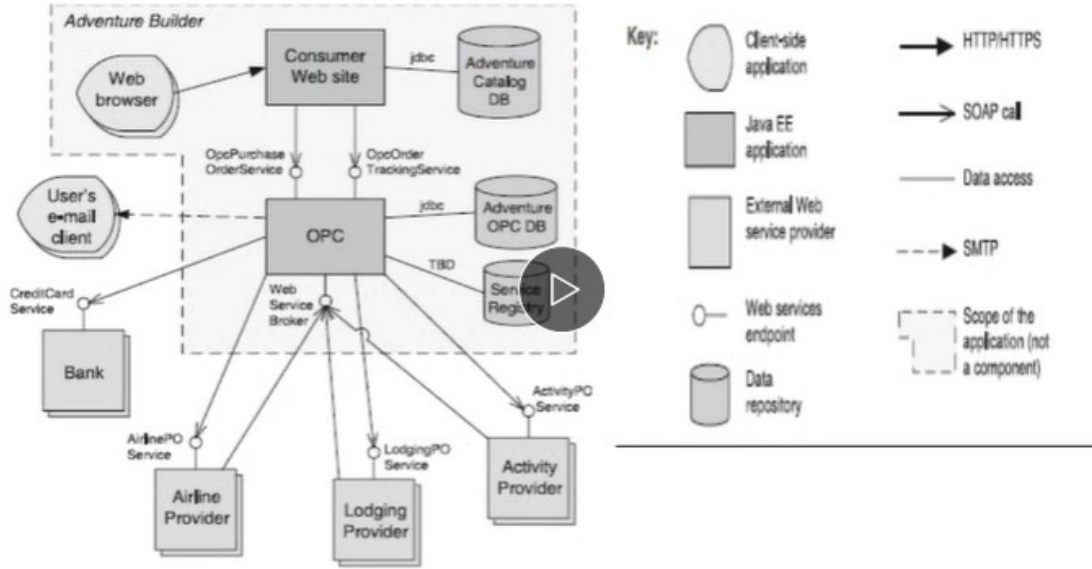
- Farklı platformlarda çalışan ve farklı uygulama dillerinde yazılmış , farklı kuruluşlar tarafından sağlanan ve internet üzerinden dağıtılan dağıtık bileşenlerin bir arada çalışabilirliğini nasıl **destekleyebiliriz**?

ÇÖZÜM:

- Hizmet odaklı mimari (SOA) modeli , hizmetleri sağlayan ve/veya tüketen dağıtık bileşenlerin bir koleksiyonunu tanımlar.



Service Oriented Architecture Example



© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License

Bu OPC ye farklı açılardan bakmak gerekiyor

1. Organizasyon açısından
2. Uygulama açısından
3. Kullanılan teknolojiler açısından

bunların hepsi bir arada olması lazım

SERVICE ORIENTED ARCHITECTURE SOLUTION (SOA ÇÖZÜMÜ)

GENEL BAKIŞ:

- Hesaplama (computation) , bir ağ üzerinden hizmetleri sağlayan ve /veya tüketen bir dizi **işbirliği bileşeniyle** elde edilir.

BİLEŞENLER:

- Yayınlanmış arayüzler aracılığıyla bir veya daha fazla hizmet sağlayan **hizmet sağlayıcıları**
- Hizmetleri doğrudan veya bir aracı aralığıyla başlatan **hizmet tüketicileri**
- **Hizmet sağlayıcıları** , (aynı zamanda hizmet tüketicileri de olabilir).
- Hizmet sağlayıcıları ile tüketiciler arasında mesajları yönlendirebilen ve dönüştürebilen bir aracı unsur olan **ESB(enterprise service bus)**
- Sağlayıcılar tarafından hizmetlerini kaydetmek için ve tüketiciler tarafından ise çalışan hizmetleri keşfetmek için kullanılabilen **hizmetler sicili**
- Hizmet tüketicileri ve sağlayıcılar arasındaki etkileşimleri , iş süreçleri ve iş akışları dillerine göre koordine eden **düzenleme (ahenk) sunucusu**

SINAV:

- 1- SOA pattern ile CLİENT SERVER pattern arasında ne gibi farklar vardır
- 2- SOA da öne çıkan en önemli şeyler nelerdir
- 3- SOA da giderilmesi gereken ihtiyaç nedir
- 4- SOA hangi ihtiyaca cevap vermek üzere gelişmiştir
- 5- CLİENT SERVER da esas ağırlık noktası nedir
- 6- LAYER bizim için ne çözüm üretmiştir
- 7- MVC neye çözüm üretmiştir
- 8- Sınavda farklar ve benzerlikler önemli

BAĞLAYICILAR:

- Tipik olarak HTTP üzerinden , web servisleri arasındaki senkron iletişim için SOAP protokolünü kullanan **SOAP bağlayıcı**
- HTTP protokolünün temel istek/cevap işlemlerine dayanan **REST bağlayıcı**
- Noktadan noktaya veya yayın aboneliğine asenkron mesaj alışverişi sunmak için bir mesajlaşma sistemi kullanan **asekron mesajlaşma bağlayıcısı**

NOT: Asenkron da mesajı iletiyor karşı taraf bunu aldı mı almadı mı veya nasıl cevap döndürdüğü ile uğraşmıyor

İLİŞKİLER:

- Mevcut olan farklı bileşen türlerinin ilgili bağlayıcılara iliştilmesi

KISITLAR

- Hizmet tüketicileri hizmet sağlayıcılarla irtibatlıdır , ancak aracı bileşenler (örneğin ESB, kayıt defteri , düzenleme sunucusu) kullanılabilir.

ZAYIF YÖNLER

- SOA tabanlı sistemlerin oluşturulması genellikle karmaşıktır.
- Bağımsız hizmetlerin gelişimini kontrol edemezsiniz.
- Middleware ilişkili bir performans yükü vardır ve hizmetlerde performans darboğazları olabilir ve tipik olarak performans garantisi sağlamazlar.

SORU: Her yerde bu yöntem kullanılabilir mi ?

⇒ karmaşıklığı fazla

⇒ tüketilen her hizmet için irtibata girmek çok zahmetli olur

⇒ Her zaman kullanılması gereken bir pattern değildir

MAP-REDUCE PATTERN

BAĞLAM:

- İşletmelerin , petabayt ölçeğinde ürettikleri veya eriştikleri muazzam hacimli verileri hızlı bir şekilde analiz etme ihtiyacı vardır.

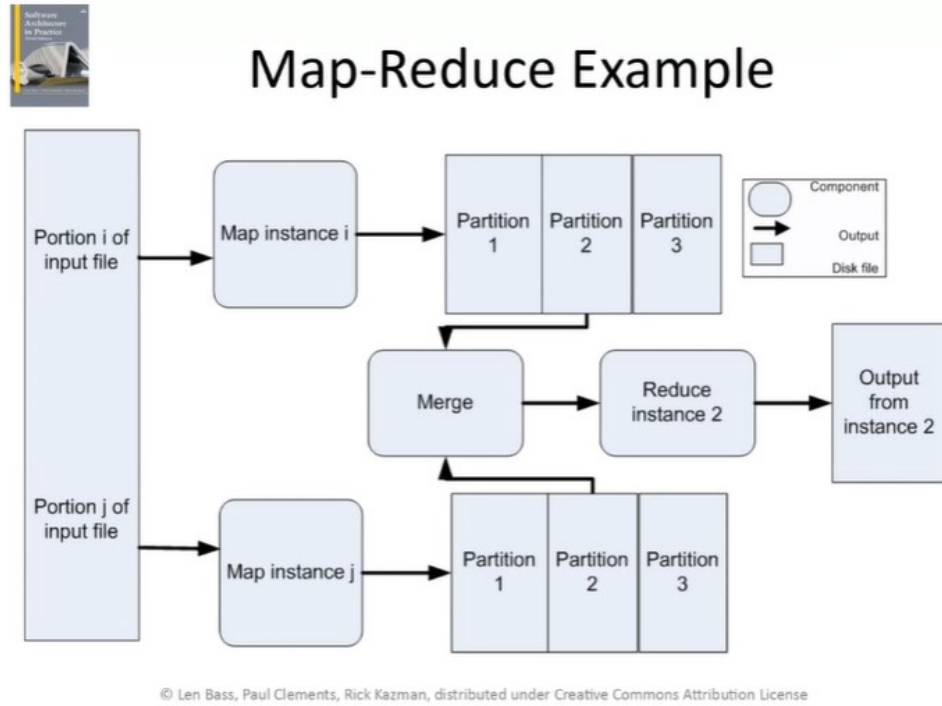
SORUN:

- Çok büyük veri kümeleri olan birçok uygulama için , verilerin sıralanması ve ardından gruplandırılmış verilerin analiz edilmesi yeterlidir.
- Map reduce kalıbının çözdüğü problem , dağıtık ve paralel türde büyük bir veri kümesini verimli bir **şekilde gerçekleştirmek** ve yapılacak analizi belirlemesi için programcıya basit bir araç sağlamaktır.

ÇÖZÜM:

- Map reduce kalıbı 3 bölüm gerektirir:

1. **Özellikli bir altyapı** , büyük ölçüde paralel bir bilgi işlem ortamında yazılımın donanım düğümlerinde tahsis edilmesiyle meşgul olur ve verilerin gerektiği gibi sıralanmasını çekip çevirir.
2. Bir programcı tarafından tanımlanan ve birleştirilecek öğeleri çekmek için verileri filtreleyen bileşene **map** denir.
3. Bir programcı tarafından tanımlanan ve **Map** ile **sonuçlandırılan verileri birleştiren** bileşene **reduce** denir.



GENEL BAKIŞ:

- Map reduce kalıbı , bir dizi işlemci üzerinde paralel olarak yürütülecek geniş bir dağıtık veri kümesini analiz etmek için bir çerçeve sağlar. Bu paralelleştirme , düşük gecikme ve yüksek erişilebilirlik sağlar . Map , analizin çekme ve dönüştürme kısımlarını ifa eder; reduce , sonuçların yüklenmesi işini yapar.

ELEMNETLER:

- Map , analizin çekme ve dönüştürme kısımlarını ifa eden , birden çok işlemciye dağıtılmış birden çok oluşa sahip bir işlevdir.

- Reduce , çekme-dönüştürme-yükleme işleminin yükleme bölümünü gerçekleştirmek için işlemciler arasında tek bir veya birden çok oluş olarak yerleştirilen bir işlemdir.
- Altyapı , map ve reduce oluşlarını yerleştirme , aralarındaki verileri yönlendirme ve arızayı tespit edip kurtarmadan sorumlu bir çerçevedir.

İLİŞKİLER:

- İntikal bir Map veya Reduce işlevinin bir tezahürü (oluşu) ile onun üzerinde kurulduğu işlemci arasındaki ilişkidir.
- Somut örnekleme , gözetleme ve kontrol , altyapı ile Map ve Reduce tezahürleri (oluşları) arasındaki ilişkidir.

KISITLAMALAR:

- Analiz edilecek verilerin bir dizi dosya olarak var olması gerekir
- Map işlevleri uyruksuzdur ve birbirleriyle iletişim kurmaz
- Map reduce oluşları arasındaki tek iletişim , map oluşlarından <anahtar, değer> (key , value) çifti olarak salınan verilerdir.

ZAYIF YÖNLER:

- Büyük veri kümeleriniz yoksa , "map reduce" ek yükü haklı değildir.
- Veri kümenizi benzer boyutlu alt kümelere bölemezsiniz , paralelliğin avantajları kaybolur.
- Birden çok "reduce" gerektiren operasyonların düzenlenmesi karmaşıktır.

ANALYSIS AND DESIGN OVERVIEW (ANALİZ VE TASARIM GENEL BAKIŞ)

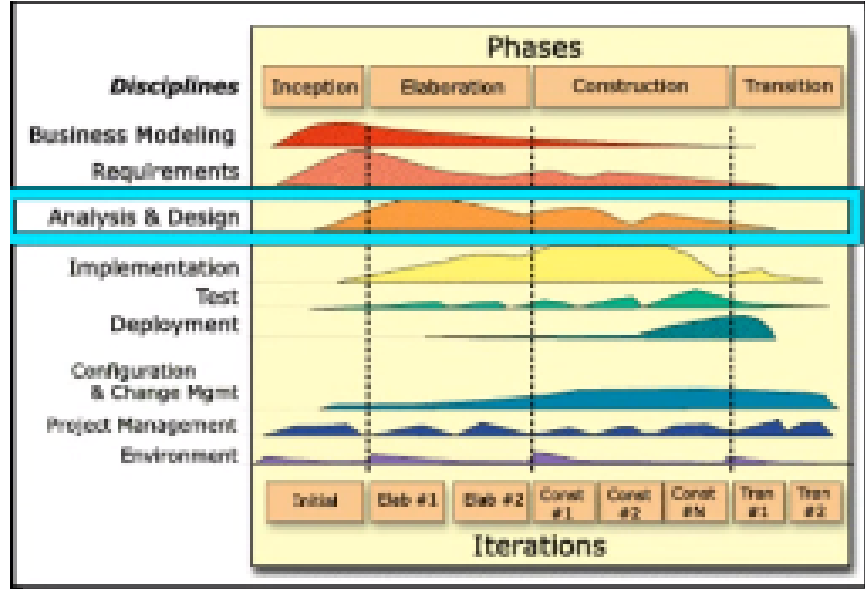
Analiz ve tasarım arasındaki fark nedir?

Analiz problem alanında çalışır onu tarif eder , tasarım ise çözüm alanında çalışır onu çözer.

Analiz işlevsel gereksinim ,tasarım işlevsel olmayan gereksinimlerdir.

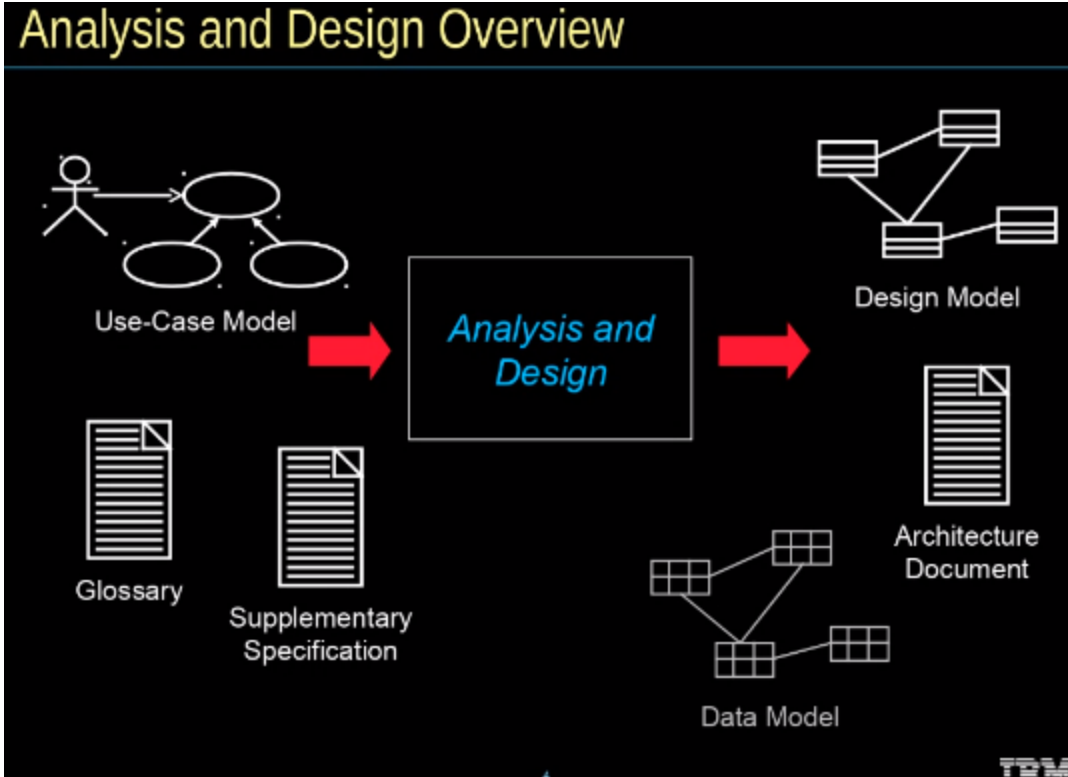
Analiz NE? sorusuna cevap verir , tasarım ise NASIL? sorusuna cevap verir.

Analysis and Design in Context (Bağlam İçinde Analiz ve Tasarım)



Analiz ve Tasarımın amaçları:

- Gereksinimleri sisteme uygun bir tasarıma dönüştürün (system-to-be)
- Sistem için sağlam bir mimari geliştirin
- Tasarımı, uygulama ortamına uyacak şekilde uyarlayın ve performans için tasarlayın



Bu üçü analiz ve tasarım için girdi oluşturluyorlar:

- Use-case bize requirement i sağlıyor.
- Glossary esasında sözlük gibi bir döküman
- Supplementary Specification



Analiz ve tasarımın sonucu ise kaynak kodun bir soyutlaması olarak hizmet eden bir tasarım modelidir.

Koda başladığım anda kafamızda tasarım belli değilse sırasıyla

- İlk kodu sonuçlandırırız.
- Diyelim ki hata veya eksiklikler çıktı sonra onları düzeltiyoruz.
- Sonra tekrar hata çıkıyor onu tekrar düzeltiyoruz
- Böylece yamalı bohçaya döndü (hocanın tabiriyle)
- Ondan sonra en baştan alıp tasarım hazırlıyorsunuz

Analysis Design Overview Topics (Analiz Tasarımına Genel Bakış Konular)

Anahtar Kavramlar

- Analiz tasarımında iş akışı
-

Mimarlık nedir?

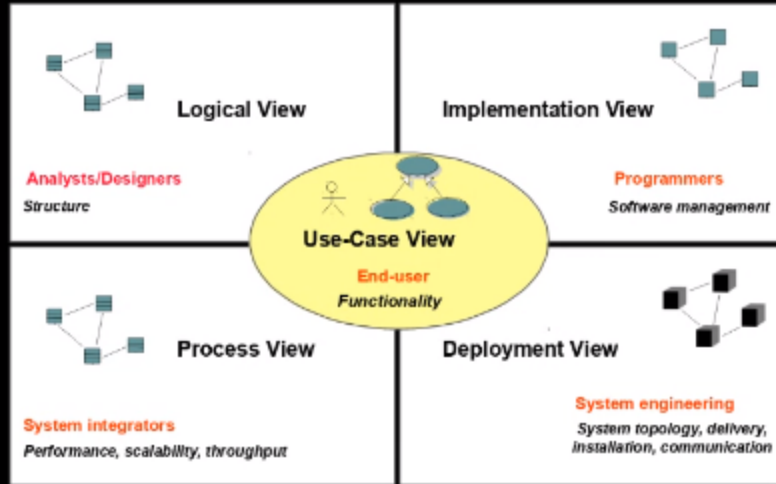
- Yazılım mimarisi, bir yazılım sisteminin organizasyonu hakkında bir dizi önemli kararı kapsar.
 - Bir sistemin oluşturulduğu yapısal elemanların ve bunların arayüzlerinin seçimi.
 - Bu yapısal ve davranışsal unsurların alt sistemler sırasına göre bileşimi.
 - Bu organizasyona rehberlik eden mimari tarz.

Mimari Kısıtlar Tasarım ve Uygulama

- Mimari, tasarımı ve inşayı kısıtlayan bir dizi stratejik tasarım kararını, kuralı veya modeli içerir.
- Mimari kararlar en temel kararlardır ve bunları değiştirmenin önemli etkileri olacaktır.

Yazılım Mimarisi : "4 + 1 görüntüleme" Modeli

Software Architecture: The “4+1 View” Model



1. Logical View

- Mimari analiz ve tasarım öğelerini tanımlama

2. Process View

- Eş zamanlılık tanımlanıyor

3. Deployment View

- Yerleştirme / Sahaya yerleştirme

4. Implementation View

- development ile ilgili

İş Akışı: Gözlemlenebilir veya sonucu üreten faaliyetler dizisidir.

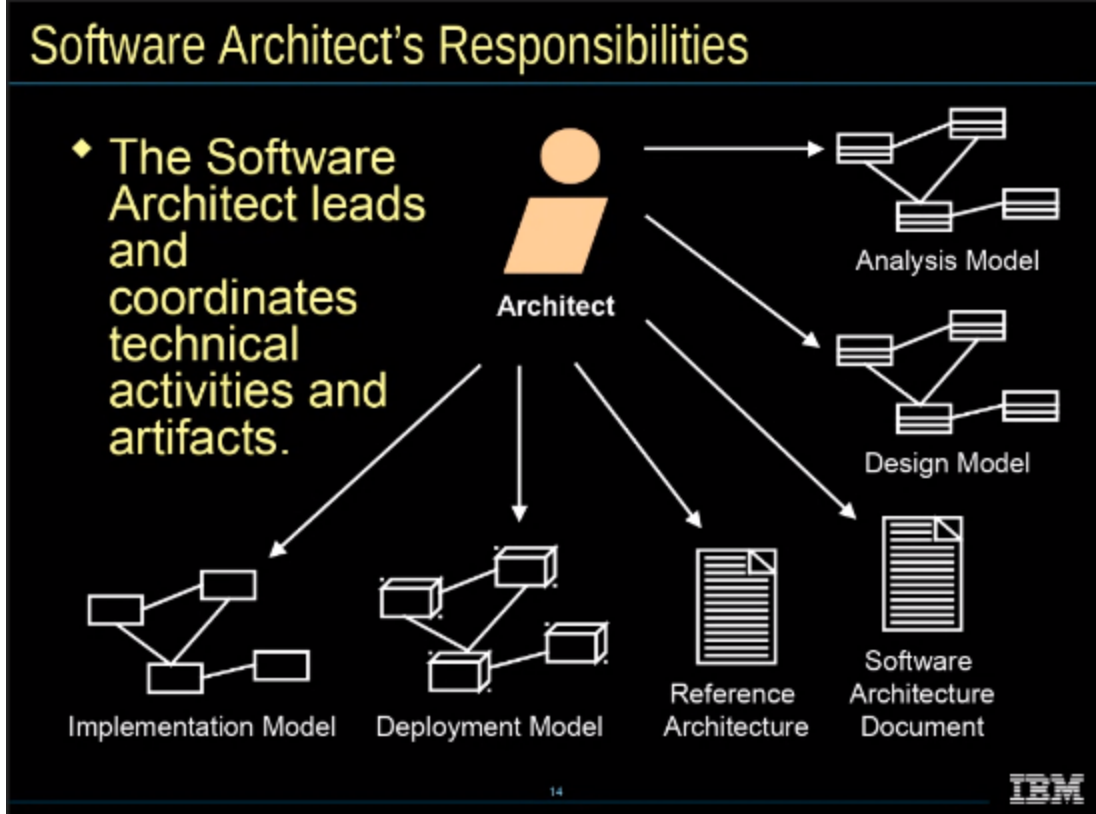
Sonuçta bizim analiz ve tasarım çalışmamız use-case modeli ile requirements disiplininin gelenle birlikte başlar kaynak kodun bir soyutlaması olarak hizmet eden tasarım modeli son buluyor.

Tasarım faaliyetleri mimari kavramı etrafında şekillenmiştir.

Bir mimarinin üretimi ve doğrulanması erken tasarım yinelemelerinin ana odak noktasıdır.

kesintisizlik==izlenebilirlik

Yazılım Mimarisinin Sorumlulukları



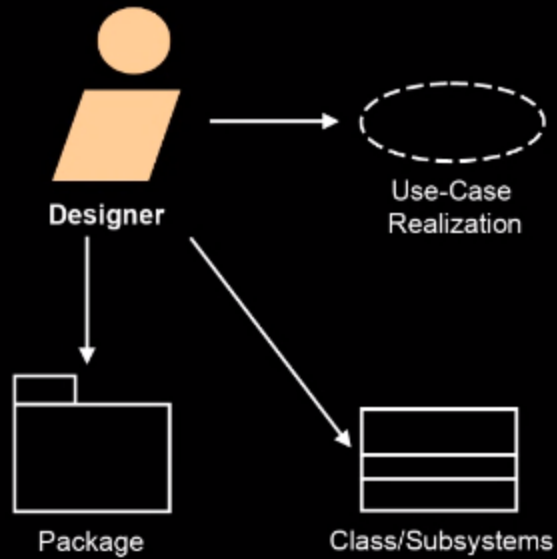
- Yazılım mimarı, teknik faaliyetleri ve eserleri koordine eden bir koordinasyona liderlik eder.
- Architect ten beklenen (mimariden)
 - Her alanda bilgi sahibi olsun ama bunun en detayına kadar değil.
- Tasarımcı ise
 - Teknik detayda da bilgi olacak ve kendi işiyle ilgili uzman olacak

Tasarımcının Sorumlulukları

- Tasarımcı, use-case modelleme tekniklerini, sistem gereksinimlerini ve yazılım tasarım tekniklerini bilmelidir.

Designer's Responsibilities

- ♦ The designer must know use-case modeling techniques, system requirements, and software design techniques.



15

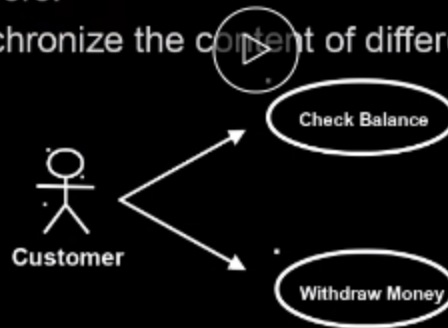
IBM

Analiz ve Tasarım Use-case Odaklı

- Bir sistem için tanımlanan Use-case, tüm geliştirme sürecinin temelini oluşturur
- Use-case in faydaları
 - Kısa, basit ve çok çeşitli paydaşlar tarafından anlaşılabilir.
 - Farklı modellerin içeriğini senkronize etmeye yardımcı olun.

Review: Analysis and Design Is Use-Case Driven

- ♦ Use cases defined for a system are the basis for the entire development process.
- ♦ Benefits of use cases:
 - Concise, simple, and understandable by a wide range of stakeholders.
 - Help synchronize the content of different models.



Module 3a VM Quiz Show (pg 102)

Questions (pg 103) (3a-2) (16 question).

DEV275 Essentials of Visual Modeling with UML 2.0 (231 sayfa).

Principles of Visual Modeling

Student Guide

Object technology is . . . ?

- A. A set of principles guiding software construction.
- B. A new theory striving to gain acceptance.
- C. A dynamic new language by Grady Booch.
- D. Based on the principles of abstraction and modularity.

Answer:

A model . . . ?

- A. Is not necessary when team members understand their job.
- B. Has to be structural AND behavioral.
- C. Is a simplification of reality.
- D. Is an excuse for building an elaborate plan.

Answer:

Why do we model?

- A. Helps to visualize a system
- B. Gives us a template for constructing a system
- C. Documents our decisions
- D. All of the above

Answer:

The best models are connected to . . . ?

- A. Java-script code
- B. Reality
- C. C ++
- D. Issues that tie it to an objectoriented developer

Answer:

Which project would be least likely to require a model?



Answer: B

Which principles of modeling are correct?

- A. The model you create, influences how the problem is attacked.
- B. The best kinds of models are those that let you chose your degree of detail.
- C. The best models are connected to reality.
- D. Create models that are built and studied separately.

Answer:

1. SORU: A
2. SORU: C
3. SORU: D
4. SORU: B
5. SORU: B
6. SORU: C

Views are "slices" of architecture. Which view focuses on structural issues?

- A. Use case
- B. Process
- C. Implementation
- D. Logical

Answer:

Which process characteristic is not essential to working with the UML?

- A. Iterative and incremental
- B. Use-case driven
- C. Resilient
- D. Architecture-centric

Answer:

The state of an object . . . ?

- A. Is defined by a "state" attribute or set of attributes.
- B. Does not normally change over time.
- C. Is defined by an object's attributes and relationships.
- D. Is the only condition in which an object may exist.

Answer:

State of an object is defined by the total of an object's attributes and links. For example, if Professor Clark's status changed from tenured to retired, the state of the Professor Clark object changes.

The visible behavior of an object is modeled by its . . . ?

- A. Attributes
- B. Responsibilities
- C. Operations
- D. Methods

Answer:

Objects are intended to mirror the concepts that they are modeled after, including behavior

Encapsulation . . . ?

- A. Allows direct manipulation of things that have been encapsulated.
- B. Is often referred to as information hiding.
- C. Causes costly and extensive maintenance.
- D. Causes changes to affect clients during implementation.

Answer

What happens when you incorporate modularity into your plan?

- A. It reduces something complex into manageable pieces.
- B. It builds modules that talk to each other.
- C. Creates systems too large to understand.
- D. Parts of your system cannot be independently developed.

Answer

A class . . . ?

- A. Is an encapsulation of an object.
- B. Represents the hierarchy of an object.
- C. Is an instance of an object.

1. SORU: D

2. SORU: C
3. SORU: C
4. SORU: C
5. SORU: B
6. SORU: A
7. SORU D

D. Is an abstract definition of an object.

Answer:

A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

A class is not an object. It is an abstract definition of an object. It defines the structure and behavior of each object in the class.

Polymorphism can be described as?

A. Hiding many different implementations behind one interface

B. Inheritance

C. Information placing

D. Generalization

Answer:

What phrase best represents a **generalization** relationship?

A. "Is a part of"

B. "Is a kind of"

C. "Is a replica of"

D. "Is an inheritance of"

Answer:

Which of the following would you use to organize elements into groups?

A. **Package**

B. Class

C. Encapsulation

D. Generalization

Answer:

1. SORU: A
2. SORU: B
3. SORU: A

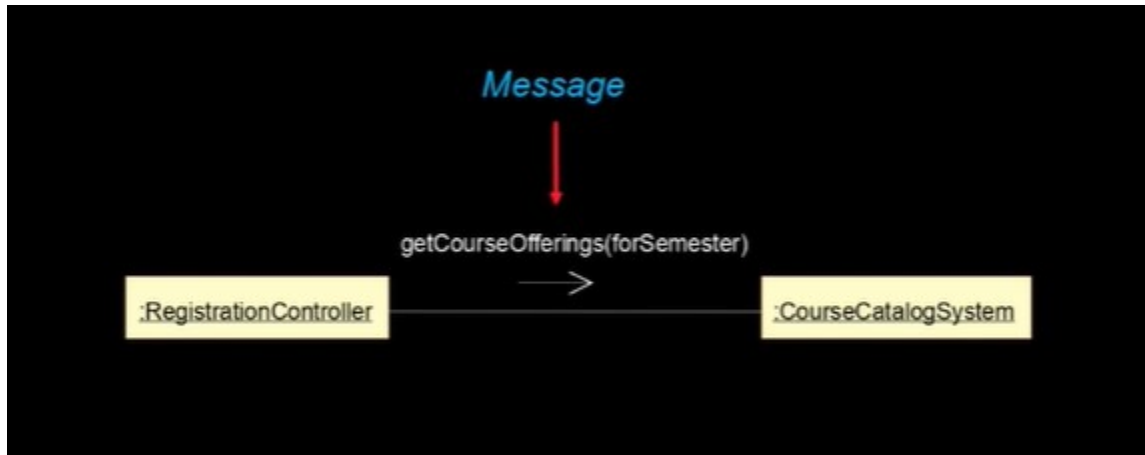
INTERACTION DIAGRAMS (Etkileşim Diyagramları)

OBJECTİVES (HEDEFLER)

- Dinamik davranışı tanımlayın ve onu bir modelde nasıl yakalayacağınızı gösterin.
- Nasıl okuyacağınızı ve yorumlayacağınızı gösterin:
 - Bir dizi diyagramı
 - Bir iletişim şeması
- İletişim ve sıra diyagramları arasındaki benzerlikleri ve farklılıkları açıklayın.

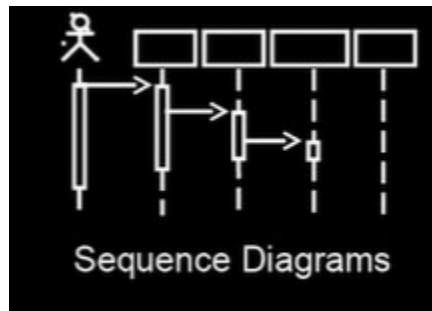
Mesajlarla Nesneler Etkileşimi (Objects interact with messages)

- Bir mesaj, bir nesnenin başka bir nesneden bir etkinlik gerçekleştirmesini nasıl istediğini gösterir.

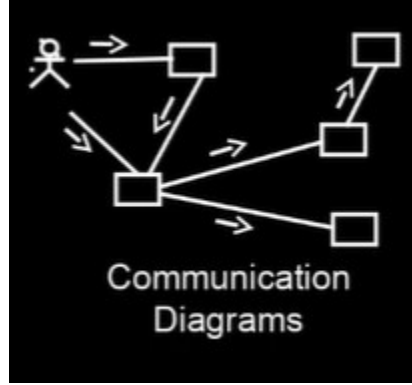


What is an Interaction Diagram? (Etkileşim Şeması nedir?)

- Nesne etkileşimlerini vurgulayan çeşitli diyagramlar için geçerli olan genel terim
 - Sıra diyagramı: (Sequence diagrams)
Nesne etkileşiminin zaman odaklı görünümü.



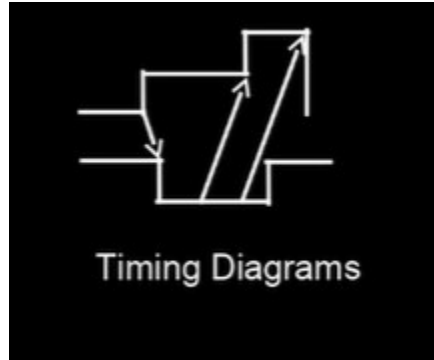
- İletişim diyagramı: (Communication diagrams)
Mesaj nesnelerinin yapısal görünümü



- Özel varyantlar

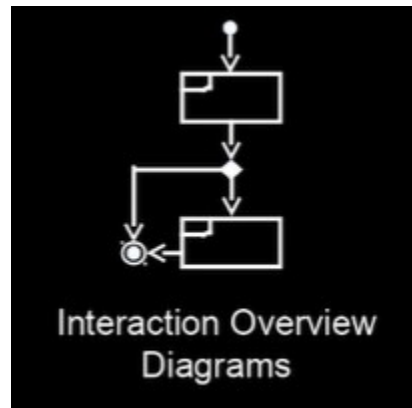
- Zamanlama diyagramı: (Timing diagrams)

Bir etkileşimde yer alan mesajların zaman kısıtlaması görünümü.



- Etkileşime genel bakış diyagramı: (Interaction overview diagrams)

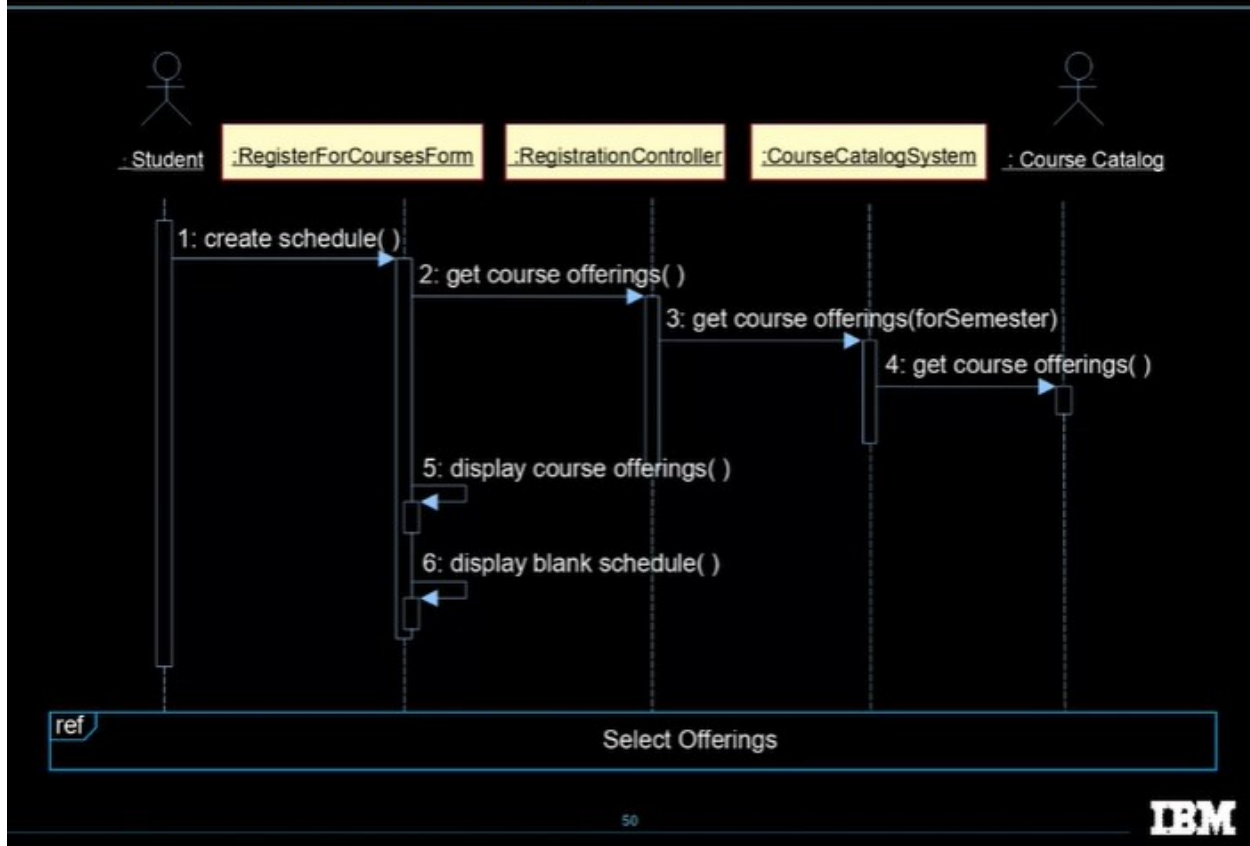
Mantık dizisinde birleştirilmiş etkileşim kümelerinin yüksek seviyeli görünümü



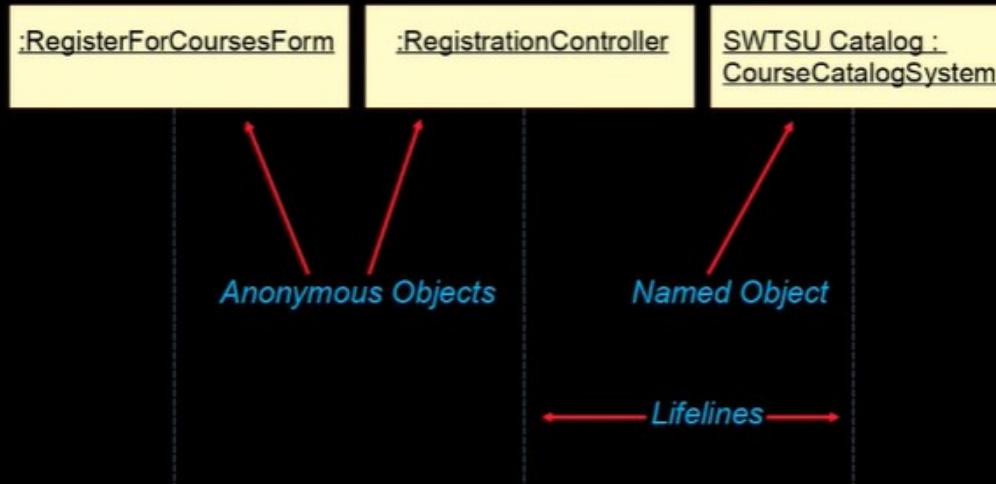
Sıra diyagramı (Sequence diagrams)

- Bir sıra diyagramı, mesajların zaman sırasını vurgulayan bir etkileşim diyagramıdır
- Diyagram şunları gösterir:
 - Etkileşime katılan nesneler.
 - Alınıp verilen mesajların sırası

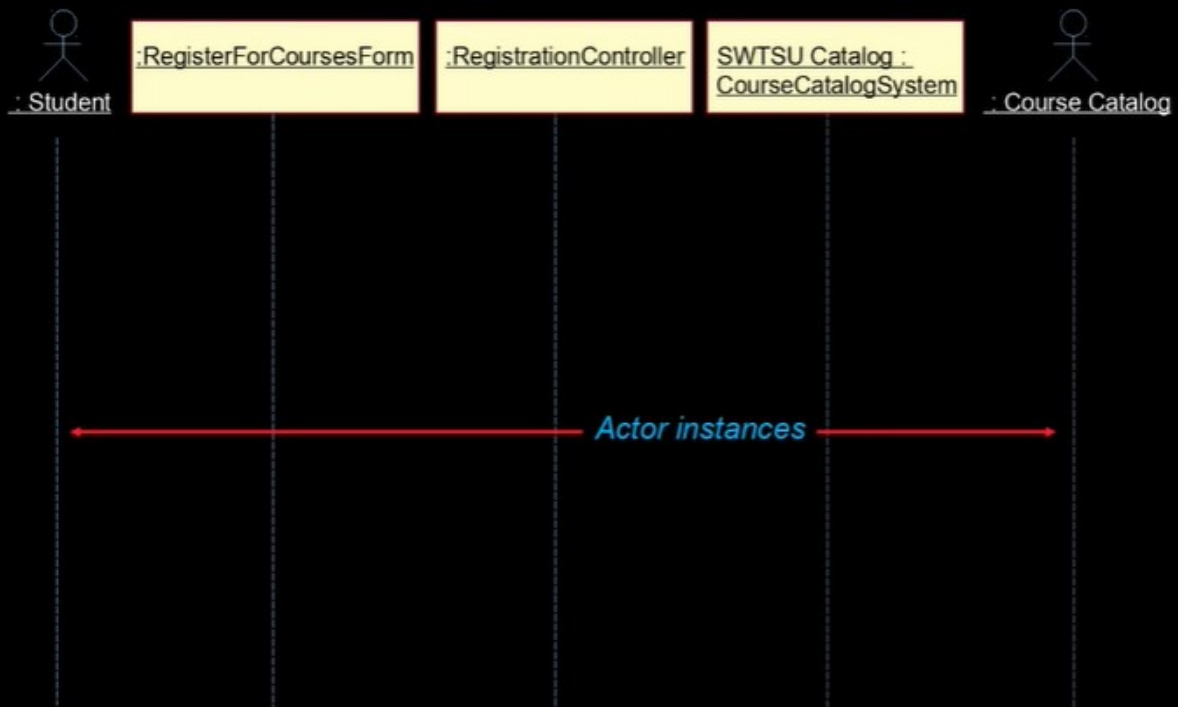
Example: Sequence Diagram



Sequence Diagram Contents: Objects



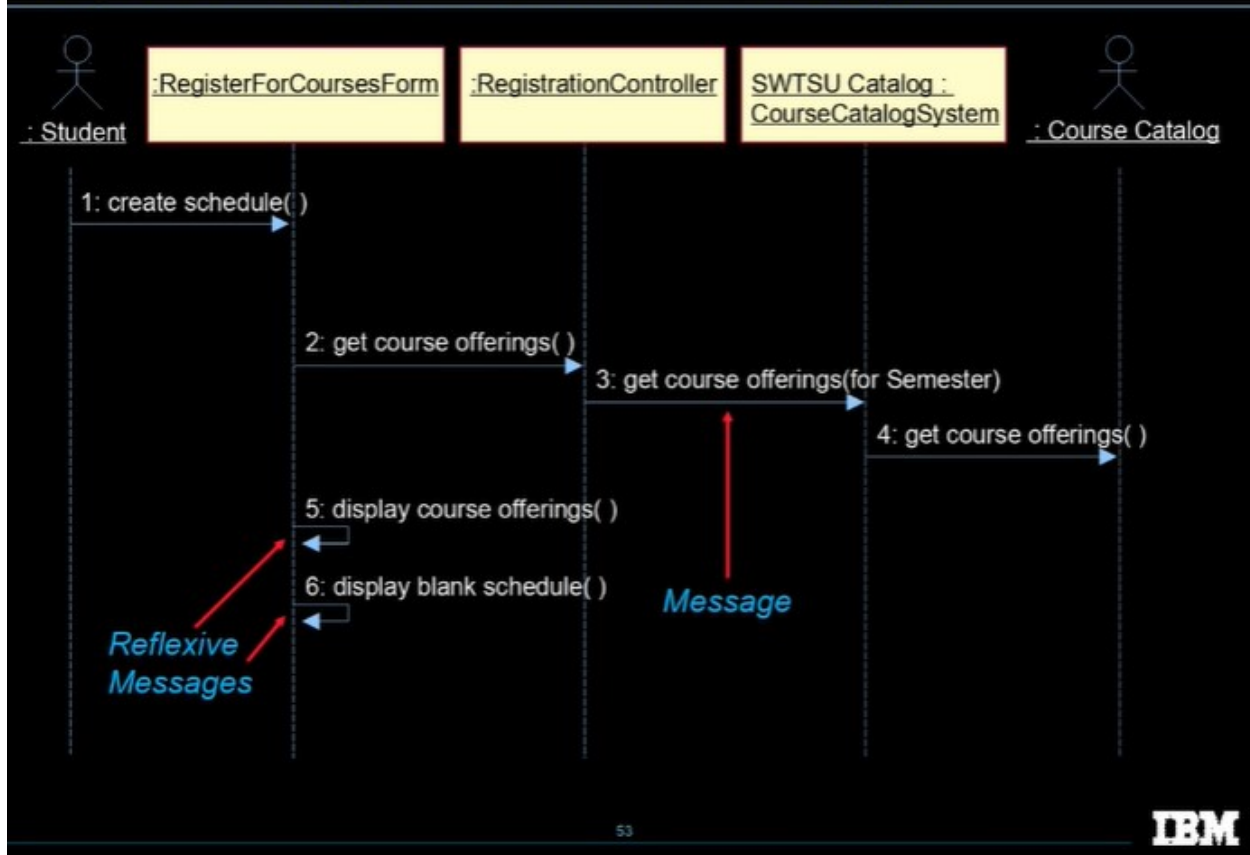
Sequence Diagram Contents: Actor



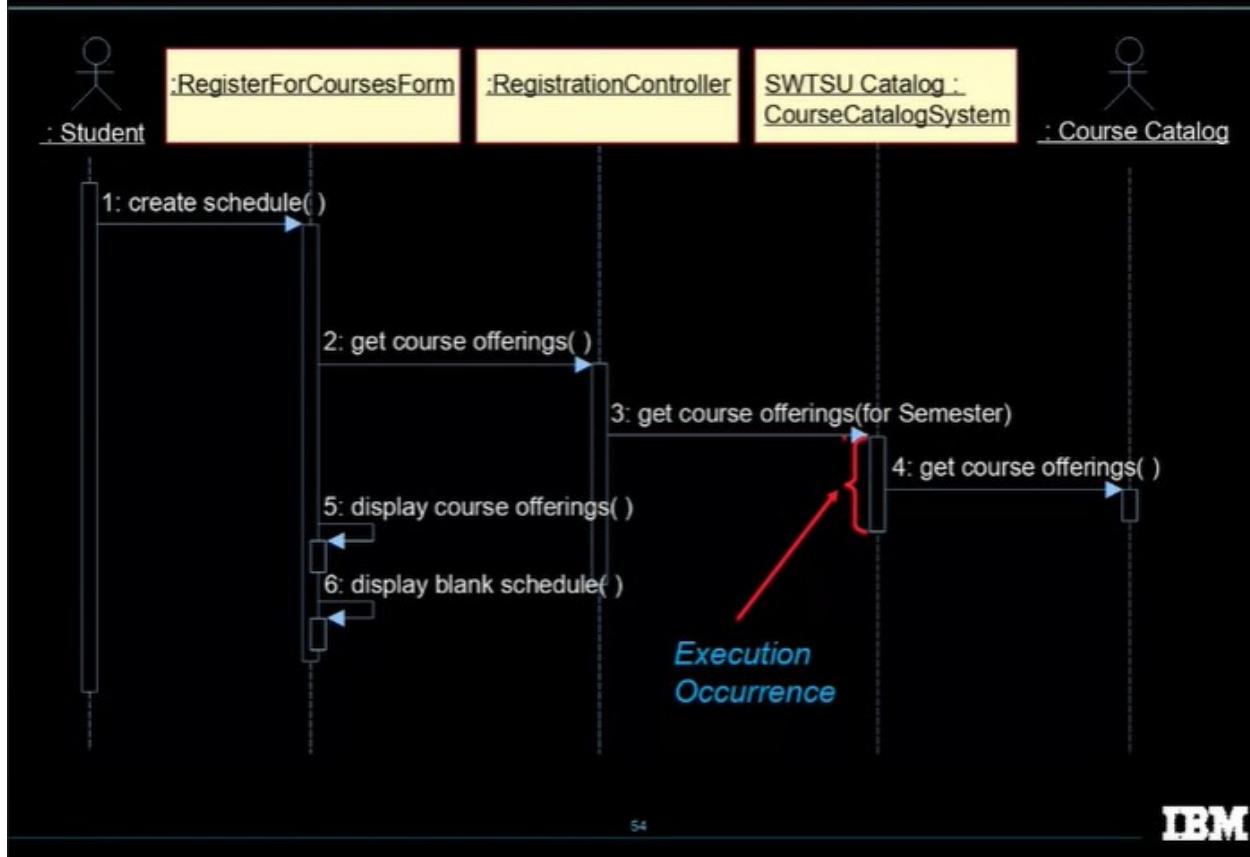
52

IBM

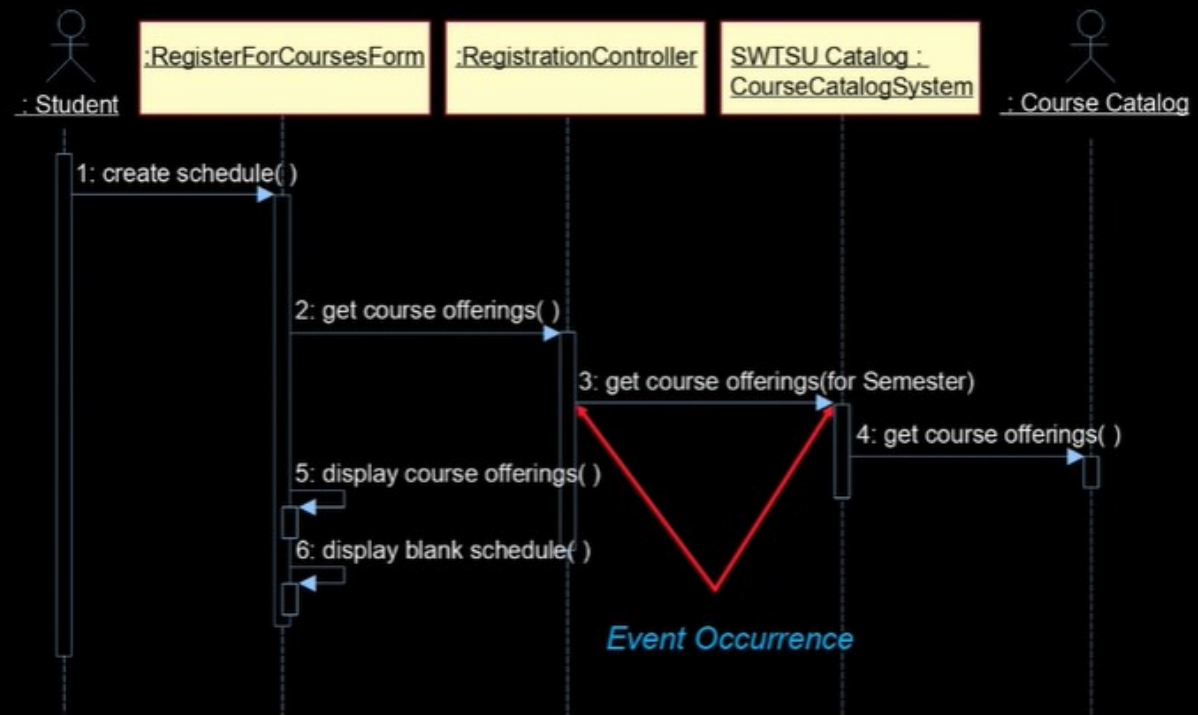
Sequence Diagram Contents: Messages



Sequence Diagram Contents: Execution Occurrence



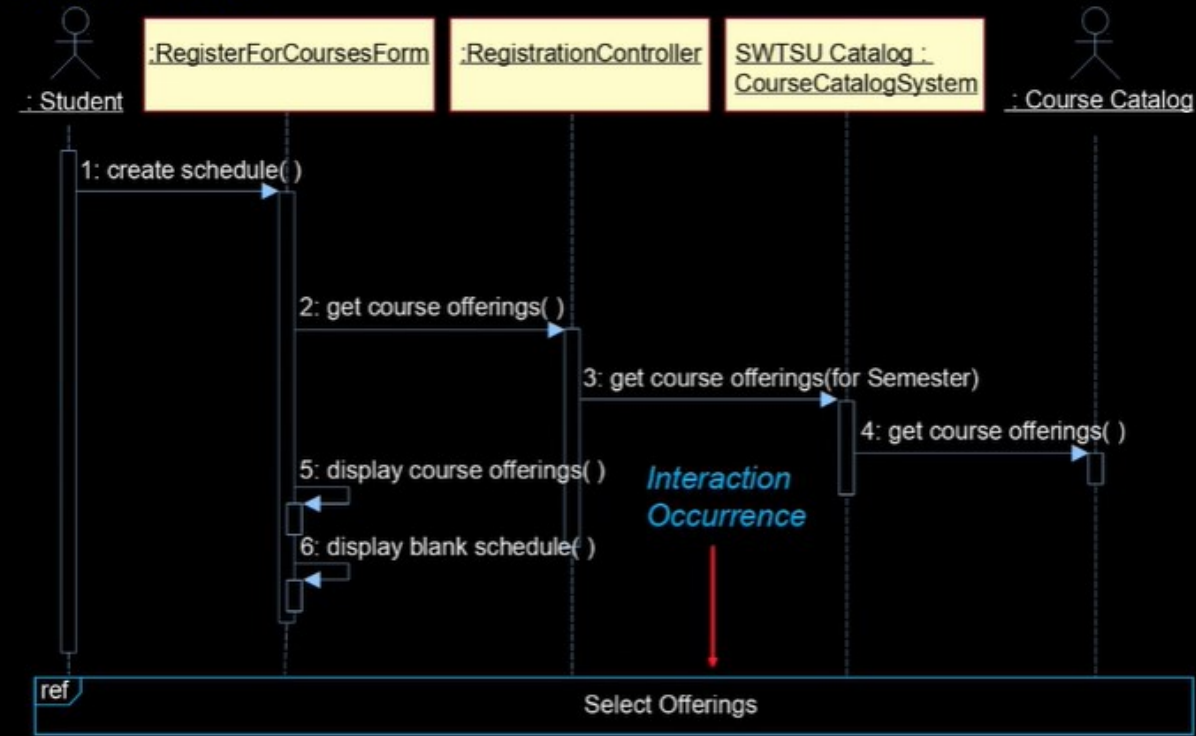
Sequence Diagram Contents: Event Occurrence



55

IBM

Sequence Diagram Contents: Interaction Occurrence



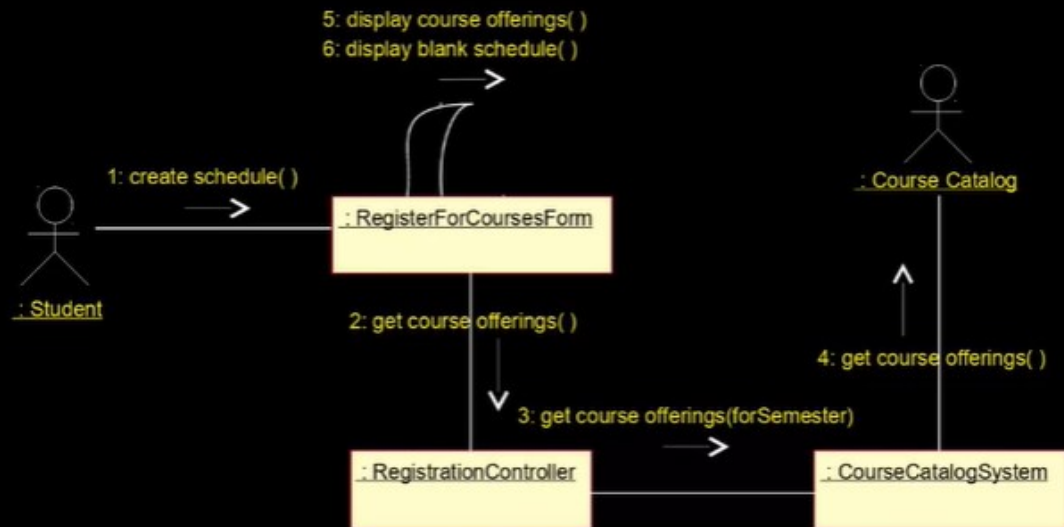
56

IBM

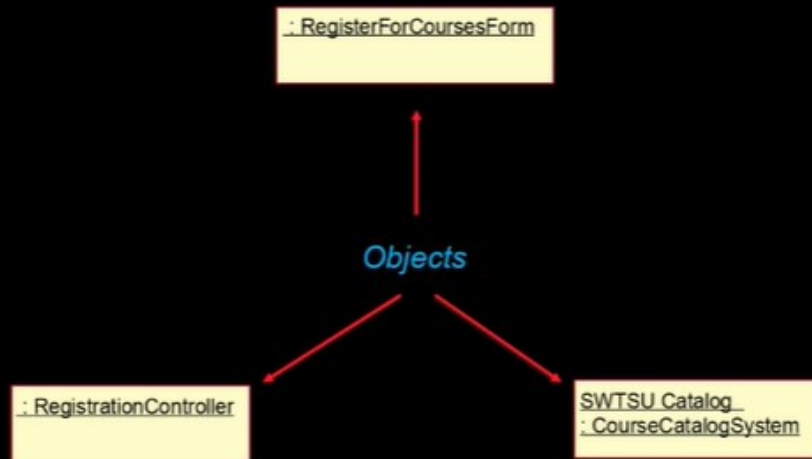
İletişim diyagramı: Communication diagrams

- Bir iletişim diyagramı, bir etkileşime katılan nesnelerin organizasyonunu vurgular.
- İletişim şeması gösterir:
 - Etkileşime katılan nesneler
 - Nesneler arasındaki bağlantılar
 - Nesneler arasında geçen mesajlar

Example: Communication Diagram



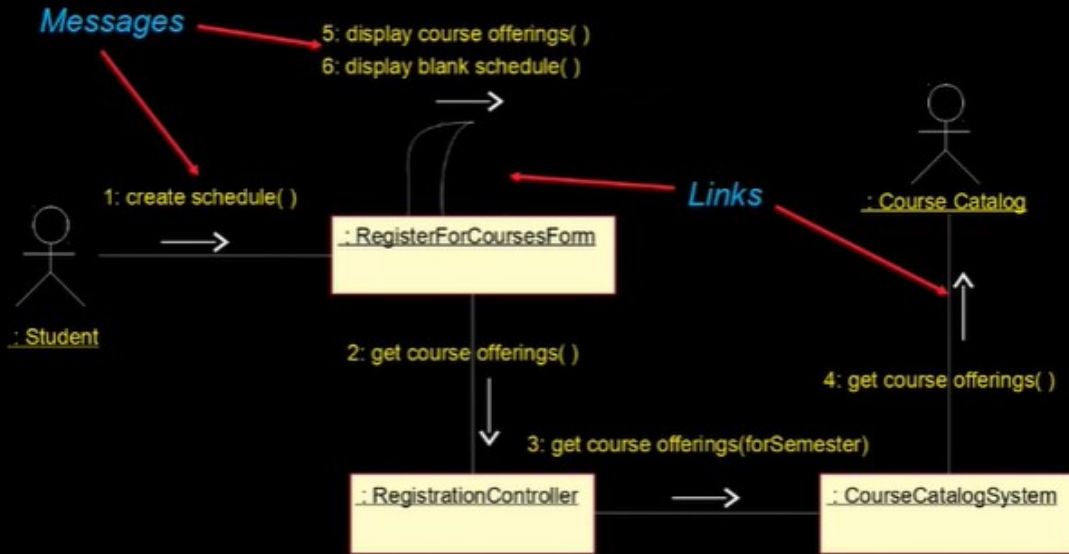
Communication Diagrams Contents: Objects



60

IBM

Communication Diagram Contents: Links and Messages



62

IBM

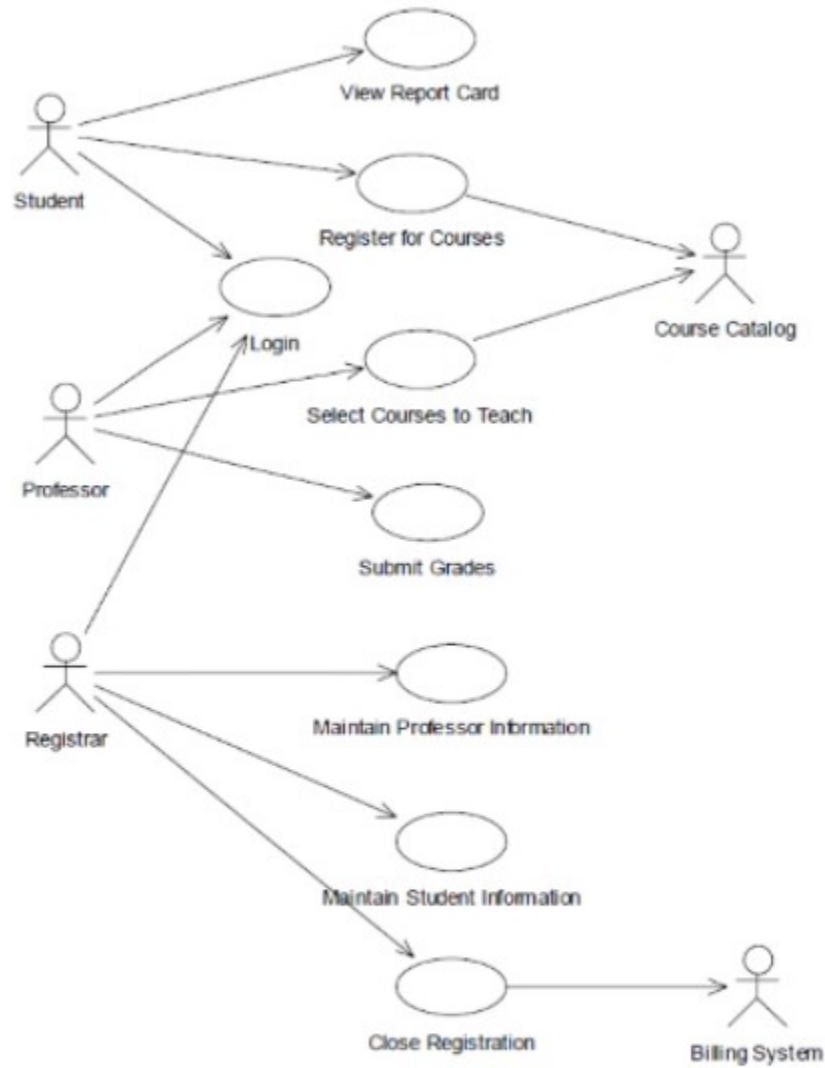
Sıra diyagramı (Sequence diagrams) ve İletişim diyagramı (Communication diagrams) BENZERLİKLERİ

- Anlamsal olarak eşdeğer
 - Herhangi bir bilgi kaybetmeden bir diyagramı diğerine dönüştürebilir
- Bir sistemin dinamik yönlerini modelleyin
- Bir kullanım senaryosu modelleyin

Sıra diyagramı (Sequence diagrams) ve İletişim diyagramı (Communication diagrams) FARKLILIKLARI

Aa Sıra Diyagramı	≡ İletişim Diyagramı
<u>Açık mesaj dizisini göster</u>	Etkileşimlere ek olarak ilişkileri göster
<u>Yürütme oluşumunu göster</u>	İletişim kalıplarını görselleştirmek için daha iyi
<u>Genel akışı görselleştirmek için daha iyi</u>	Belirli bir nesne üzerindeki tüm efektleri görselleştirmek için daha iyi
<u>gerçek zamanlı özellikler ve karmaşık senaryolar için daha iyi</u>	Beyin fırtınası oturumları için kullanımı daha kolay

Course Registration System Use-Case Model



Sorular

▼ Use Case gerçekleştirme adımında hangi çalışmaları yapıyoruz hangi diyagramları hazırlıyoruz

- Sequence diagram (koda en yakın diyagram)
- communication diagram
- timing diagram
- interaction overview diagram

▼ Sequence diyagram zaman sıralamasını dikkate alır mı
dikkate alır

▼ Communication diyagram zaman sıralamasını dikkate alır mı
dikkate almaz

▼ Sequence diyagramının bileşenleri nelerdir

- actor
- nesneler
- mesajlar
- hayat çizgisi
- event occurrence
- interaction occurrence

▼ Case nedir

computer aided software engineering / vaka

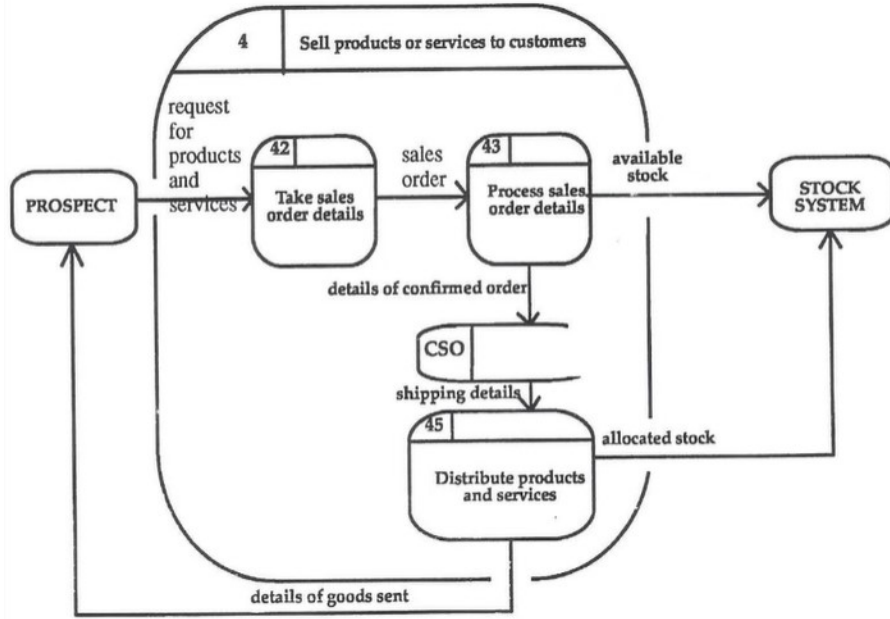
DATA FLOW MODELLING (DATA AKIŞ MODELİ)

Objectives of Data Flow Modelling (Veri Akışı Modellemesinin Amaçları)

- Niyetlenilen enformasyonu elde etmek için işlevlerin veya süreçlerin sağlanmasını gerekli veri ile garanti etmek
- Talep edilen verilerin kaynaklarını ve elde edilen enformasyonun varış yerlerini tanımlamak

Data Flow Modelling Basic Terms (Veri Akışı Modellemesi Temel Terimler)

- DFD
- Data akışı
- bilgi deposu - data store
- işlev / süreç - function/levelling
- Ayırıştırma / tesviye - Decomposition / levelling
- external entity - dış varlık
- context diagram - bağlam diyagramı
- elementary function / elementary process / functional primitive - temel işlev / temel süreç / işlevsel ilkel



Tekniğin Bileşenleri

- Bir veri akış modeli şunlardan oluşur:
 - bir dizi veri akış şeması
 - diyagramlarda resmedilen öğelerin her birini tanımlayan destek belgeleri
- Ayrıştırma (seviyelendirme) tekniği
 - yinelemeli olarak ve ayrıntı düzeylerini artırarak belirlemek için kullanılır
 - iş hedeflerini karşılamak için
 - verilerin mantıksal işlevler arasında , mantıksal işlevlerden ve mantıksal işlevlere doğru nasıl aktığını
 - işlev modellemelerde olduğu gibi
- Bu işlev işlev çizilmiştir

Aşağıdakileri Teşhis Eden ve Tanımlayan Teknik

- Projenin kapsamı ve sınırı
- girdi sağlayan ve çıktığı kabul eden harici varlıklar
- sistem sınırı boyunca girdi ve çıktı taşıyan veri akışları
- sistem sınırı içindeki veri akışları
- gelip geçici olanlar dahil bütün sistem veri depoları
- veriler üzerinde işleyen ve onun dönüştürülmesine ve iletilmesine yol açan süreçler

Yukarıdan aşağıya ve Aşağıdan yukarıya Metodolojiler

- Veri akışı modelleri , yukarıdan aşağıya ve yukarıdan aşağıya metodolojilerde farklı şekilde kullanılmaktadır
- "Mevcut sistemin" derinlemesine analizine odaklanan aşağıdan yukarı metodolojiler , DFM leri yaygın olarak kullanır.
 - mevcut fiziki sistem modellenir ve ortaya çıkan fiziksel DFM , genellikle mevcut işlevselliği tanımlayan mantıksal bir DFM sağlamak için
 - bir dizi biçimsel kural mantıksal hale getirilir.
- Bu daha sonra gelecekteki işlevsel talep ve ihtiyaçları içerecek şekle dönüştürülebilir ve bir enformasyon sistemi olarak hayata geçirilir.
- Yukarıdan aşağıya metodolojiler , DFM leri bir çıktı olarak veya diğer modellerin çapraz kontrolünde seçici olarak kullanma eğiliminde değildir.
- Mantıksal sistem talep ve ihtiyaçlarını belirlemek için temel bir araç olarak yukarıdan aşağıya yaklaşımıyla rahat bir şekilde örtüşmezler. Yine de , kullanıcı geri bildirimi ve çalışma aracı olarak çok yararlı olabilirler.

Data Flow Model Analizinde Nasıl Kullanılır

DFM analiste şöyle yardımcı olur:

1. iletişim
2. anlama
3. standartlaşma

İletişim

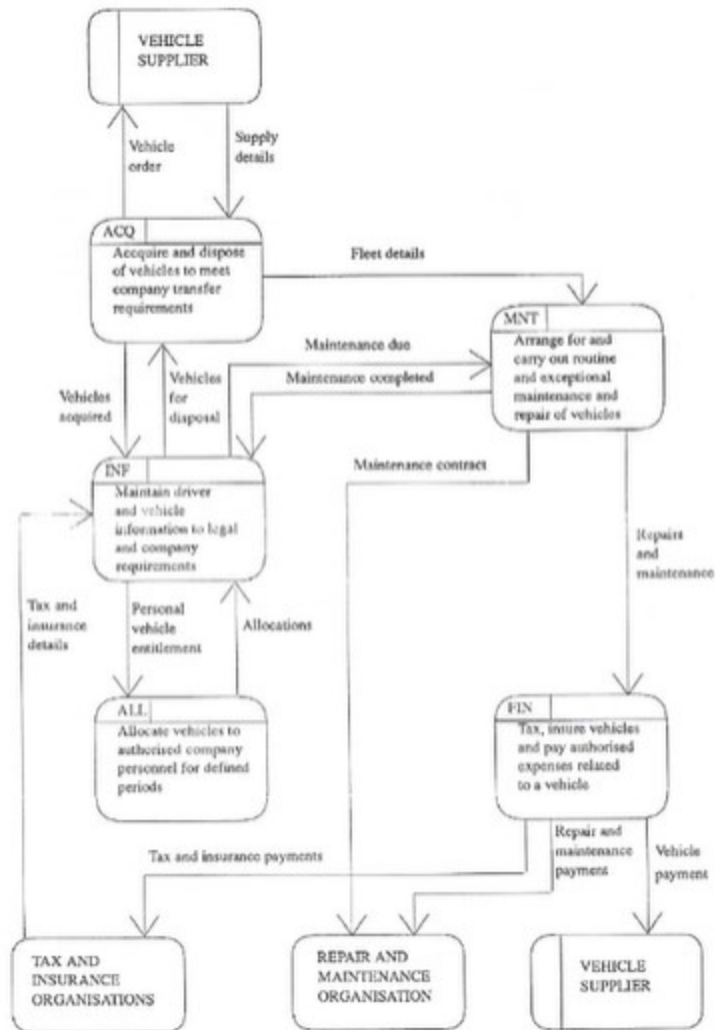
- araştırma çalışması esnasında kullanıcı iletişimine yardım eder
- DFM nin basit unsurları ve mantıksal yapısı kullanıcıların değerlendirmesi için kolaydır.
- Graphical DFD
 - Mülakat esnasında verilen enformasyonu mülakat yapılarının anlayabileceği ve yorum yapabileceği bir biçimde gösterme bakımından analisti yetkin kılar.
- Bu veri toplamaya destek olur ve geliştirme sürecini açıklığa kavuşturmaya yardım eder.

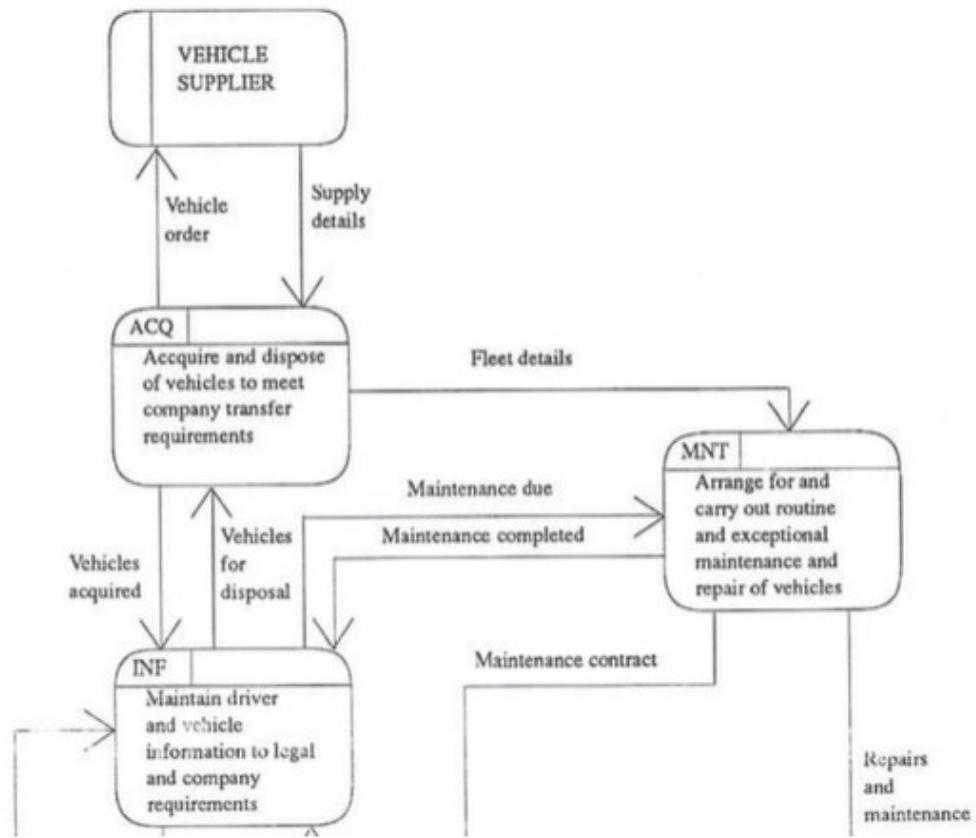
Anlama

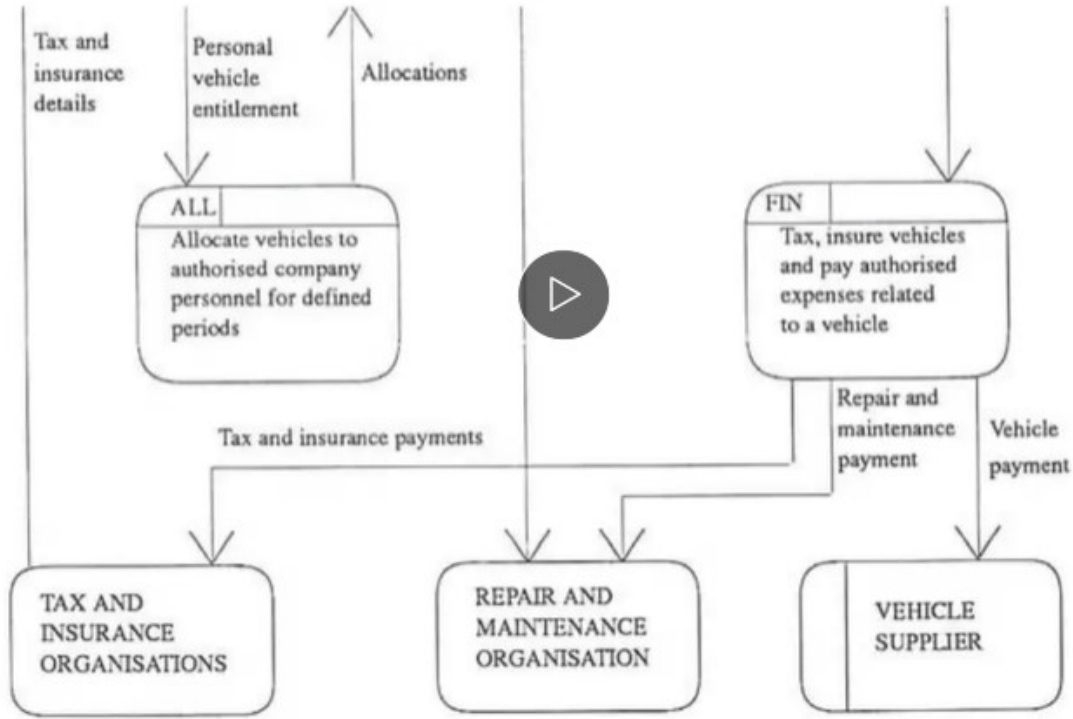
- DFM nin kademeli yapısı , bütün bir DFM nin ortaya çıkışının , işin analistler takımı içerisinde parçalanması ile kolayca halledilmesini garanti eder.
- Fiziki sistem içindeki enformasyon akışını belgeleyen DFM ler bilhassa mevcut bir sistemin bütün girift enformasyon akışı depolanması ile belgelenmesinde faydalıdır.

Standartlaşma

- DFM bölümsü süreçleri bölümlerarası politikalara girmeden standart bir yolla belgeleyebilir
- DFMLer analistler tarafından oldukça fazla kullanılır
- Onların kullanması , daha ilk günden itibaren belgeleri anlamaya muktedir kılarak , yeni proje üyelerinin daha kısa öğrenme eğrisine sahip olması demektir.







DFM ne zaman kullanılır

- DFM proje ömre çevrimi boyunca kullanılabilir
- Seçim şunlar tarafından etkilenir
 - projenin doğuşu
 - metodoloji
 - kuruluşun türü
 - analistin yetenek ve tecrübesi

DFM nin En Genel Kullanımı Şunlar içindir

Üst düzey kapsama

- Enformasyon sistemleri strateji planlaması esnasında daha ileri geliştirme için , belirli alanları ve bunların arayüzlerini tanımak ve belgelemek için bir "bağlam" seviyesinde DFM kullanılabilir.
- Uygulama seviyesinde üst düzey kapsama ,sistem sınırlarını çizerek ve belli başlı işlev alanlarına enformasyon akışını tanıyarak proje içindeki ve dışındakileri tanır
- üst düzey kapsama teknoloji sınırını (insan bilgisayar etkileşimi) ve sonra bir sistem çözümünün teknolojisini desteklemesi gereken el işi süreçlerin kalite güvencesini araştırmaya yardım eder.
- Sistem sınırlarına odaklanma bir proje başında gerçek dünya olaylarını tanır.

Belgeleme

- DFM talep ve ihtiyaçların ve üst seviyede detaya ihtiyaç duyulan fiziki sistemleri belgelemek için en verimli araçlardan biridir.

İletişim

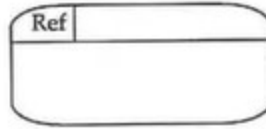
- DFM mülakatlar tartışmalar/ beyin fırtınaları esnasında enformasyonu göstermek için etkileşimli olarak kullanılabilir.
- Bunlar geribildirim ve yönetim sunumlarında kullanıcıya enformasyon sunmak için değerlidir. Kullanıcılar bunları anlamak ve sıklıkla DFM oluştururken katılmak için kolay ve doğal bulurlar.

Kavramlaştırma

- DFM, ilk önce meydana getirilen fiziki olandan mantıksal sistem tanımını çıkarmaya yardım eder.
- Mantıksallaştırma fiziki olanı mantıksal olana dönüştürür. Projede daha sonra mantıksal DFM den işlevsel gruplamalar meydana getirilebilir.

Data Flow Diyagram Gelenekleri

- DFD sistem geliřtirmede kullanılan en eski yapısal tekniklerden biridir. Bu gibi gelenekler diğerk modelleme tekniklerinden daha fazla insanın standartlařtırmama sürecine maruz kalmıřtır.
- Farklı metodolojilerin , herbiri geđerli sebeplere dayalı olarak , DFD kavramlarını göstermek için değıřik sembolleri vardır.
- Daha önemlisi bu farklı metodolojiler sembollerin /kavramların nasıl yönlendirileceğini dair farklı kuralları ve kısıtlar koyma eğilimindedirler.
- Bu bölüm , bütün varyasyonları kapsamaz ve CASE methodunun sembollerini ve kurallarını kullanır.
- Bununla beraber , řu her zaman hatırlanmalıdır ki metodolojiler de bir yere kadardır , kendileri bizatihi bir son değıřillerdir.
- Onları bir köle gibi değıřil akıllıca kullan
- İřLEM/İřLEV
 -



- Verinin dönüşümü
- Bir işleme giren bir veri kullanılır ve bu işlemde farklı bir veri akışı verecek şekilde değıřir.
 - ayrıştırılmış işlemler kutunun sağı üst köşesinde üç nokta ile tanınır.
- Mantıksal bir DFD de bir işlem veriyi çekip çevirmeli veya dönüřtürmeli , nihayetinde işlemde çıkan veri giren veriden farklı olmalı ve farklı adlandırılmalıdır.
- Fiziki bir DFD de kattığı enformasyon değıřeri ya çok az olan veya hiç olmayan ve bu yüzden aynı girdi ve çıktı veri akışına sahip bir işlem olabilir.
- Her işlem biricik bir tanımlayıcıya ve tarife sahip olmalıdır.

- Fiziki süreçler ayrıca yer veya işlemi ifa eden iş rolü ile şerh edilmelidir.
- İşlem en azından giren ve çıkan bir veri akışına sahip olmalıdır.
- HARİCİ VARLIKLAR

-



- Modellenen iş alanının dışında olan bir veri akışının menşei veya hedefi; bazen sırasıyla kaynak veya kap olarak adlandırılır.
- Sınırımızın dışında olduğu için DFD üzerinde detaylı olarak modellenmemiştir.
- CASE methodta harici varlıklar daha fazla ayrıştırılamaz. Diğer metodolojilerde izin verildiğinde ,ayrıştırma, işlevler için olduğu gibi üç noktayla gösterilebilir.
- ERD de harici bir varlık temsil edilemez . Hedef uygulama içinde bilgi tuttuğumuz bir konu değildir.

◦ VERİ AKIŞI

-

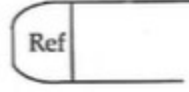


- Yönü gösteren başlı ok ile kaynaktan hedefe veri akışı . Bir işlev bir kaynak veya bir hedef veya her ikisidir. Bazı metodolojilerde veri dış varlıklar arasında akabilir, kesikli çizgi ile diyagramlardır.
- CASE method da veri akışı ayrıştırılamaz. İzin verilen diğer metodolojilerde ayrıştırma üç çizgi ile gösterilir veri akışı adı ile

- Veri akışı verinin akışıdır , fizik mal akışı değildir. İletişim maksadıyla bununla beraber malların akışını göstermek de faydalı olabilir . Böyle bir akışı açık ok ile gösteriniz malların adı ok içinde olarak.

◦ VERİ DEPOSU

-

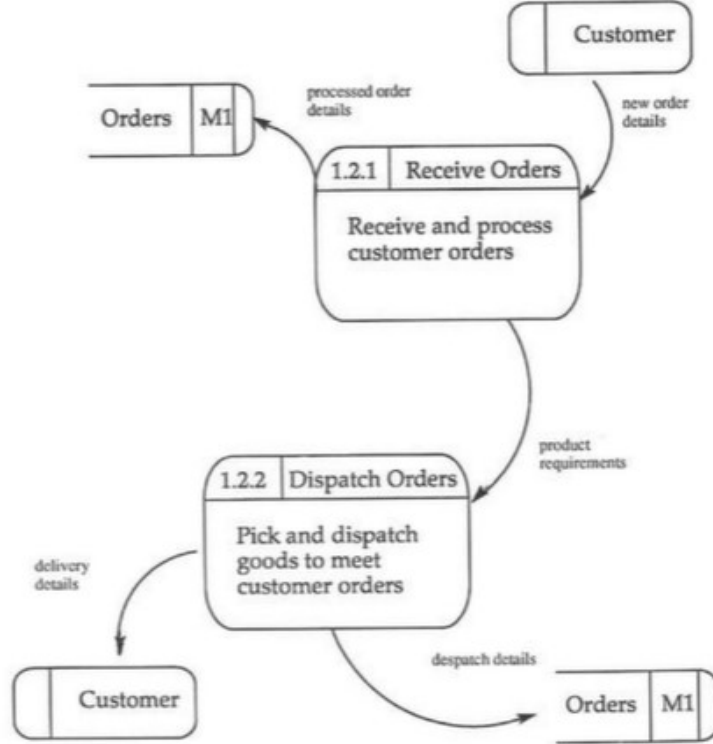


- Hareketsiz veri. Mantıksal DFM de veri deposu ERD den gelen varlık gruplarını gösterir
- Fiziki DFM de depo bir bilgisayar dosyası , bir dosya dolabı , bir evrak sepeti veya bir kimsenin hafızası olabilir.
- Genellikle veri deposu dosyaları , karmaşık yapıları veya veri setlerini gösterir; halbuki veri akışı münferit veri elemanlarını veya kayıt gibi basit yapıları gösterir.
- İzin verildiğinde , ayrıştırma işlevler için olduğu gibi üç nokta ile gösterilebilir. (CASE method dda izin verilmez)
- Veri depoları adlandırılır ve tanımlayıcı verilir. Tanımlama standardı şöyledir : örnek D sistem veri depoları için M el işi veri depoları için , T geçici veri depoları için.
- Okunabilirliğe yardımcı olmak için veri depoları ve harici varlık kutularının diyagramın farklı bölümlerinde tekrarlanması yaygın bir uygulamadır. Böyle bir tekrar , her seferinde dikey bir çubukla temsil edilir. Aynı DFD içindeki işlemlerin tekrarı mantıksal olarak tutarsız olacaktır.
- DFD ler zaman sıralaması göstermez ve bir DFD deki her işlevin aynı anda çalışması mümkündür. DFD ler karar kutuları kullanmazlar ve kontrol enformasyonu göstermezler.
- Veri depoları harici veya dahili olabilir. Harici veri depoları , mevcut hedef işlemin ayrışmasını biçimlendirenler dışındaki işlemler için erişilebilir.

- Ayırıştırma çerçevesi içinde çizilen veri depolarına yalnızca bu ayırıştırma içindeki işlevler tarafından erişilebilir.

Conventions for Data Flow Diagrams

Replicated Items



- TEMEL BASİT İŞ SÜRECİ
 - İşletmeyi bir tutarlılık halinden diğerine götüren veya işletmenin durumunu hiç değiştirmeyen bir iş süreci
 - Başladıktan sonra bir iş süreci daima ya sonuca kadar devam etmeli veya tamamen geri alınmalıdır.
 - Bir ara aşama geçerliyse , iş söz konusu olduğunda , bu temel bir süreç veya işlem değildir.

- Metodolojiler , DFD de temel süreçleri nasıl gösterdiklerine göre farklıdır. CASE metodunda süreçler , ayrıştırma işareti göstermedikçe temeldir. Başka yerlerde , temel süreçler DFD de açıkça tanımlanabilir, örneğin işlem kutusunun sağ alt köşesinde yanıp sönen bir yıldız işaretiyle
- CASE method da temel işlemler , temel işlevler olarak ayrıntılı olarak tanımlanır. (fonkdiyon modelleme bölümü)
- Bazı metodolojilerde , bütün temel süreçler , yapılandırılmış ingilizce de temel süreç açıklamaları(EPD) veya benzeri olarak belgelenir.
- VERİ AKIŞ DİYAGRAMLARI
 - Verileri dönüştüren veya kullanan işletme tarafından gerçekleştirilen işlev
 - Verilerin işletme içinde ve işletme ile kuşatan çevre arasındaki hareketi
 - işletme tarafından saklanan ve kullanılan veriler ör: kitap, karteks, mikrofiş, dosya dolabı , hareketsiz veriler
 - kısıtlar: zaman ve şartlar normalde bir DFD de gösterilmez.
- TEMEL BASİT İŞLEMLER
 - DFD farklı derinliklerde hiyerarşiye ayrıştırılır. Hiyerarşinin her bir bacağı temel bir duruma ulaşana kadar ayrıştırılır.
 - Şu hallerde bir işlemin temel (en alt seviye) olduğu söylenebilir
 - daha fazla ayrıştırmak pek mantıklı değildir.
 - işlem tek bir iş birimidir bütünüyle tamamlanmış veya yapılmamıştır.
 - işlem yapılandırılmış ingilizce ile bir sayfadan daha az tutacak şekilde yazılabilir
- DFD SEVİYELENDİRME / HİYERARŞİ (MERTEBE)
 - Bir DFD açık ve anlaşılması kolay olmalıdır. Karmaşık sistemlerin bu şekilde modellenmesi için DFD, DFD kademelenmesi üreten farklı seviyelere bölünmüştür.
 - Bağlam diyagramı . Bu uygulamayı tek bir işlem olarak tanımlayan ve tüm harici varlıkları ve sisteme giren ve çıkan tüm veri akışlarını gösteren en üst düzey

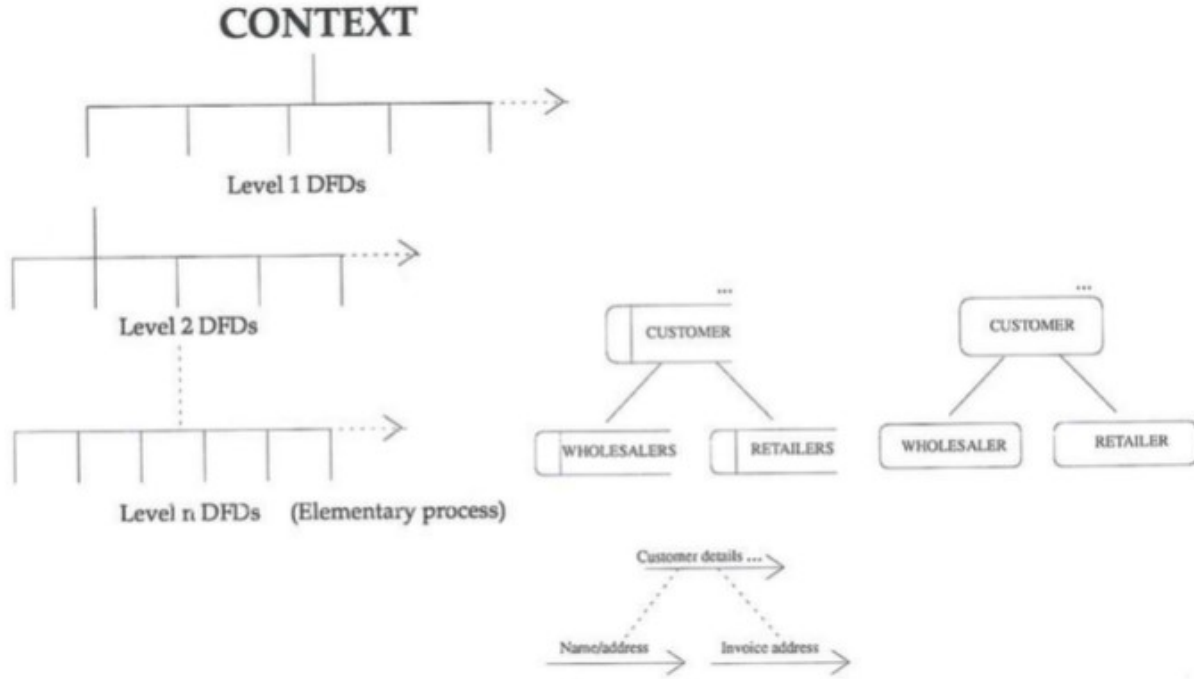
DFD dir.

- Bağlam diyagramları sistem sınırlarını ve sistemi etkileyen gerçek dünya olaylarını tanımlamak için özellikle yararlıdır.
- Bu "düzey 0" işlemi 1. düzey DFD ye parçalanır.
- DFD SEVİYELENDİRME / HİYERARŞİ(MERTEBE)
 - Seviye 1
 - Bu irtibatlı akışlar depolar vb. ana iş gruplarını içerir.
 - Bu DFD deki işlemler 2. seviye DFD ler elde etmek için parçalanabilir
 - Seviyeli set
 - DFD her işlem temel bir düzeye ulaşana kadar parçalanır . Bunlar daha sonra tamamen belgelenir.

VERİ Akışı Modelleme Kavramları

- Cüzdan
- Veri depoları, dış varlıklar ve veri akışları
- Bazı metodolojiler veri deposu , dış varlıklar ve veri akışları modelleri basitleştirmek için de seviyelendirilebilir (ayrıştırılabilir)

20



DATA FLOW DIAGRAMS (veri akış diyagramları)

- anahtar kelimelerdir
- her bir "kutunun" ve okun etiketlendiğinden emin olun
- süreç - etkin fiil: anlamlı
- veri akışları - varlıklar / öznitelikler
- veri depoları - varlıklar / öznitelikler
- (external) dış varlıklar - isimler

Veri Akışı Modelindeki Unsurları İsimlendirme ve Tanımlama

- Tüm Veri Akışı diyagramlarının başlığı, ayrıştırılan süreçle birlikte olmalıdır.
- Bir Veri Akışı diyagramındaki tüm semboller açıkça adlandırılmalıdır.
- Tüm işlem, veri deposu ve harici varlık adları, uygulamada benzersiz olmalıdır.

İşlem Adları

- Etkin fiil ile başlayın.
- Süreci kendi başına ve kullanıcılar için anlaşılır kılmak için yeterli sözcükler kullanın.
- Yalnızca veri modelinde bilinen veya bilinebilecek şeylere bakın.
- Bir düzeyde farklılaşma sağlamak ve düzeyler arasındaki hiyerarşiyi göstermek için her süreci numaralandırın.

Veri akışı Adları

- İsimler (ve uygun sıfatlar). Varlık Modelinde bulunan varlıklar ve öznitelikler veya eşanlamlılar olmalıdır. Aynı isim birkaç veri akışında görünebilir, ancak bir işleme giren ve çıkan akışların isimleri aynı olmamalıdır.
- Mantıksal DFD'lerde, akış için kullanılan medyaya herhangi bir referans olmamalıdır; örneğin; formlar, belgeler, telefon görüşmeleri vb.

Veri deposu Adları

- İsim ayrıca, genellikle varlıklar. Bir veri deposu, birden fazla varlık türü içerebilir; adı bunu yansıtmalıdır. Mantıksal DFD'lerde Veri deposu mantıksal öğelerdir ve bu nedenle isimleri herhangi bir fiziksel çağrışım içermemelidir. Fiziksel DFD'lerde isimleri fiziksel dosyalara vb. atıfta bulunabilir.

Harici Varlık Adları

- Normalde isimler ör. Müşteri, Müşteri, Banka. Bazen başka isimler tercih edilecektir, örn. Pazarlama, Döküm Süreci vb.

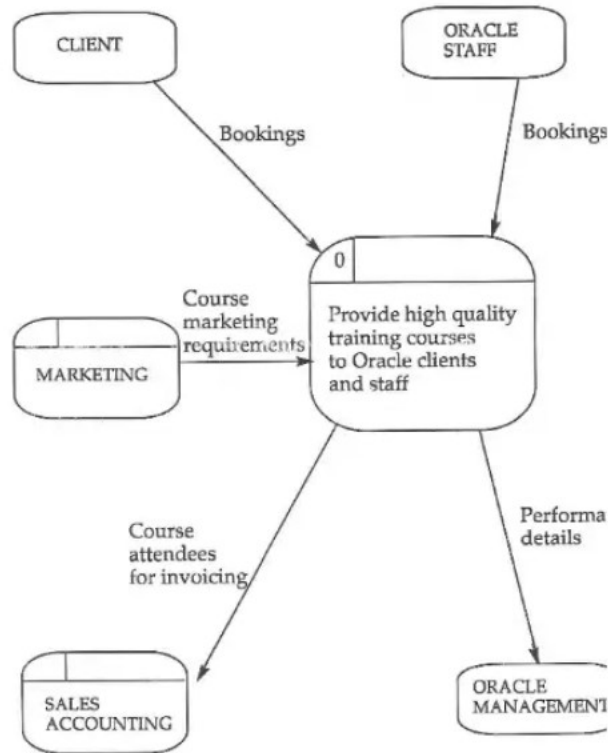
Veri Akışı Modeli Nasıl Oluşturulur

Önkoşullar

- "iş yönü" bilgisi
- Modellemeye başlamak için temel oluşturmak üzere yeterli veri toplamının tamamlanması.
- Veri Akışı Modelinin ilerlemesi, amaçlanan amaca bağlı olacaktır. DFD'ler in analiz için merkezi olduğu yöntemler, muhtemelen mevcut fiziksel sistem modelinden başlayarak çeşitli versiyonlarda eksiksiz bir DFD seti gerektirir.

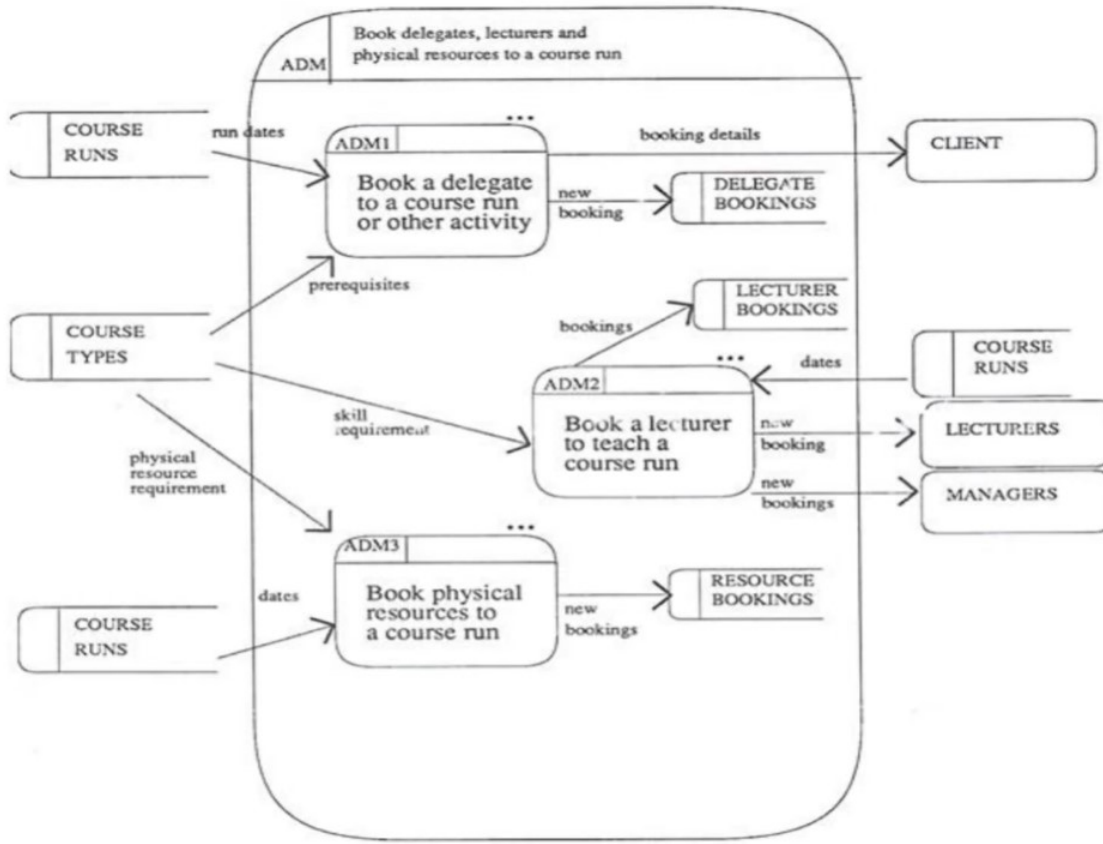
How To Build a Data Flow Model

Context Diagram



Context Diagram (Bağlam Diyagramı)

- Bu, sistem sınırını sağlar ve incelemelerin kapsamını tanımlar.
- Bir işlem kutusu ve analiz altındaki alanın adını içeren bir etiket çizin. Bir alandaki tüm işlem kutularının hiyerarşiyi göstermek için numaralandırılması gerekir. Bu ilk seviyede sol üst köşeye "0" sayısını ekleyin
- Sistemle etkileşime giren harici varlıkları tanımlayın. Sisteme bilgi vermeli veya ondan bilgi almalıdırlar. Yazılım kutuları ve ad olarak çizin.
- Düzleştirilmiş seti üretin.
- Daha fazla tanım gerektiren ana süreç alanlarını, genellikle ana organizasyonel alanları, Hesapları, Satışları vb. Tanımlayın.
- Hedef süreç için bir çerçeve çizin ve bu işlev tarafından görüldüğü gibi dış dünyayı temsil edecek marjlar bırakın.
- Çerçevenize başlık oluşturun ve ayrıştırılacak işlemin numarasını ve etiket adını girin.
- Her düzey sürecinin ayrıntılı incelenmesi sırasında, önceki düzeyde henüz mevcut olmayan yeni dış akışlar belirlenecektir. Bunlar yukarı doğru seviyelendirilmelidir.
- Alt fonksiyonlarla bir veri ara yüzüne sahip veri depolarını tanımlayın. Her birini çizin ve etiketleyin. Hedef süreçte tamamen korunan ve kullanılan veri depoları ile çerçeve içinde gösterilir. Diğerleri "dış dünyada" çizilmelidir.
- İşlemler ve diğer işlemler veya veri depoları arasındaki tüm veri akışlarını ekleyin ve adlandırın.
- Birçok kesişen veri akışının önlendiği yerlerde harici varlıkları ve veri depolarını tekrarlayın. HER ÜZERİNDE çubuk çizgileriyle göster.
- DFD'yi eksiksizlik ve mantık açısından değerlendirin.
- Tüm sorunları keşfedin, hataları düzeltin ve Kullanıcı sözleşmesi alın. Bu düzey ayrıştırmalarının her biri için bunu yapana kadar Veri Akışı diyagramının sonraki düzeyine BAŞLAMAYIN.



VERİ AKIŞ DİYAGRAMI KURALLARI

İşlem (Process)

- A. Hiçbir proses sadece çıktıya sahip olamaz. Eğer bir nesne sadece çıktıya sahipse o bir kaynak (source) olsa gerektir.
- B. Hiçbir proses sadece girdiye sahip olamaz. Eğer bir nesne sadece girdiye sahipse o bir depo (sink) (hazne) kap olsa gerektir.
- C. Bir proses fiil ifade eden bir etiket taşır; çünkü proses bir iş demektir.

Veri Deposu (Data Store)

- D. Bir veri, bir veri deposundan bir başka veri deposuna doğrudan gidemez; bir prosesten geçtikten sonra gidebilir.
- E. Veri, bir dış kaynaktan doğrudan veri deposuna gidemez; bir prosesten geçtikten sonra gidebilir.
- F. Veri, bir veri deposundan doğrudan bir dış kaynağa gidemez; bir prosesten geçtikten sonra gidebilir.
- G. Bir veri deposu bir isim ile etiketlenir.

Kaynak/kap (Source/Sink)

- H. Veri bir kaynaktan bir kaba doğrudan gidemez. Eğer veri bizim sistemle ilgili ise bir proses tarafından götürülmelidir; aksi takdirde veri akışı DFD üzerinde gösterilmez.
- I. Bir kaynak/kap isim ifadesi ile etiketlenir.

Veri Akışı (Data Flow)

- J. Bir veri akışı semboller arasında bir yönde akar. Bir proses ve veri deposu arasında bir okuma ve bir güncelleme göstermek durumunda her iki yönde de akabilir. Son bahsedilen genellikle iki ayrı ok ile gösterilir çünkü okuma ve güncelleme daha ziyade farklı zamanlarda olur.
- K. Bir veri akışında bir çatallanma aynı verinin ortak bir yerden iki veya daha fazla prosese-veri deposuna-veyakaynağa/kaba gitmesi anlamına gelir (genellikle aynı verinin farklı yerlere giden iki kopyası gibi gösterilir).
- L. Bir veri akışının birleşmesi aynı verinin iki veya daha fazla farklı prosesten veri deposundan veya kaynaktan/kaptan ortak bir yere doğru gitmesi anlamına gelir.
- M. Bir veri akışı doğrudan çıktığı prosese geri dönmez; en azından bir başka prosesin onu işleyip bir başka veri akışı yoluyla asıl veriyi başladığı prosese geri döndermesi gerekir.
- N. Bir veri deposuna bir veri akışı giriş, güncelleme veya silme demektir.
- O. Bir veri deposundan veri akışı demek veri kullanımı veya alınması demektir.
- P. Bir veri akışı isim ifadesi ile etiketlenir. Birden fazla veri bir paket içinde hareket ediyorsa tek bir okla ama birden fazla isim ifade ile etiketlenebilir.

İleri Veri Akış Diyagramı Kuralları (Advanced Rules Governing Data-Flow Diagramming)

- Q. Bir düzeydeki birleşik bir veri akışı daha alt düzeyde bileşen veri akışlarına ayrılabilir fakat üstte olmayan bir veri altta fazladan eklenemez veya birleşik akıştaki bütün veri daha alt düzeyde tek veya daha fazla alt akışta yer almalıdır(yani seviyelendirmede veri kaybı olmamalıdır).
- R. Bir prosese gelen veri çıktı üretmeye(bir veri deposuna yerleştirilen veri dahil olmak üzere) yeterli olmalıdır.
- S. En alt seviyedeki DFD'de yeni bir veri akışı ancak istisnai şartlarda mümkün olabilir; bunlar tipik olarak hata mesajlarından (Bu müşteri kayıtlı değil, yeni oluşturmak ister misin?" gibi) veya teyid mesajlarından ("Bu kaydı silmek istiyor musun?" gibi) oluşur.
- T. Veri akış çizgilerinin birbirini kesmesi istenmiyorsa aynı veri deposu, kaynağı/kabı tekrarlı yazılabilir.