YZM 3017
Yazılım Tasarımı ve Mimarisi

Prof. Dr. Hamdi Tolga KAHRAMAN
Arş. Gör. M. Hakan BOZKURT
Arş. Gör. Sefa ARAS

Yazılım Tasarımı ve Mimarisi

Konuyu anlamanız için bilmeniz gereken ön şartlar

i. Nesneye dayalı programlama kavram ve tekniklerini (object oriented programming, OOP) iyi bilmelisiniz (nesne, sınıf, soyutlama, kalıtım, çok biçimlilik).

ii. Tümleştirilmiş modelleme dilinin (UML) temel öğelerini ve kullanımını bilmeniz işinizi kolaylaştıracaktır.

iii. Yazılım geliştirme süreçleri hakkında temel düzeyde de olsa bilgi sahibi olmanız faydalı olacaktır.

Yazılım Tasarımı ve Mimarisi

KAYNAKLAR

1. Feza Buzluca, itü, ders slaytları, ninova
2. C# ile Tasarım Desenleri ve Mimarileri, Pusula Yayıncılık, Ocak 2015, Ali Kaya ve Engin Bulut.
3. Yazılım Mimarının El Kitabı, C++ Java ve C# ile Uml ve Dizayn Paternleri, Pusula Yayıncılık, Eylül 2014, Aykut Taşdelen
4. Head First Design Patterns, O'Reilly, Eric Freeman, Elisabeth Freeman, Kathy Sierra, Bert Bates, First Edition October 2004.
5. Design Patterns: Elements of Reusable Object Oriented Software, E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Addison –Wesley Professional, 1995.
6. Analysis Patterns: Reusable Object Models, Martin Fowler, (1996-11-27). Addison-Wesley.
7. Pattern-Oriented Software Architecture: A System of Patterns, Buschmann F., Meunier R., Rohnert H. & Sommerlad P. & Stal M. (1996), John Wiley & Sons.
8. Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design and Iterative Development, Craig Larman, 3rd Edition, Prentice Hall, 2005.
9. Joshua Kerievsky, Refactorin to Patterns, Addison-Weslay Professional, 2004.

Yazılım Tasarımı ve Mimarisi

FACEBOOK sayfası

KTÜ Yazılım Mühendisliği Yazılım Tasarımı ve Mimarisi

Yazılım Tasarımı ve Mimarisi

AMAÇ

- Bu dersin amacı, tasarımdan kaynaklanan sorunları gidererek kaliteli yazılımlar geliştirmeyi sağlayan Nesneye Dayalı Çözümleme ve Tasarım (Object-Oriented Analysis and Design – OOA/D) yöntemlerini tanıtmaktır.

Yazılım Tasarımı ve Mimarisi

AMAÇ

- Derste, bir çok işlevsel yeteneğe sahip ve çok sayıda modülden oluşan endüstriyel boyutlu yazılımların tasarımına yönelik konular ele alınacaktır.
- Yazılım projeleri çok sayıda kişinin yer aldığı takımlar halinde geliştirilmektedir. Takım çalışması sağlanacaktır.
- Değişime uyum sağlayan esnek yazılımların geliştirilmesinin yolu öğrenilecektir.

Yazılım Tasarımı ve Mimarisi

Kısa açıklamalar

- Bir programlama dilini iyi bilmek kaliteli bir yazılım geliştirmek için yeterli değildir.
- Kodlama zevkli bir konudur ancak kaliteli bir yazılım sistemi oluşturmak karmaşık ve zor bir iştir *(Philippe Kruchten)*
- İyi bir yazılım uluşturabilmek için uygun yazılım geliştirme tekniklerini de bilmek ve uygulamak gerekiyor.

Yazılım Tasarımı ve Mimarisi

Yazılım dünyasındaki sorunlardan bir kaçı

- Yazılımın zamanında tamamlanamaması
- Bütçenin aşılması ve bakım maliyetlerinin yüksek olması
- Çok sayıda giderilemeyen hatanın ortaya çıkması
- Yazılımın değişen şartlara göre uyarlanamaması
- Eski projelerde hazırlanan yazılım modüllerinin yeni projelerde kullanılamaması

Nihai hedef

- Bu dersin temel amacı, yukarıda bahsedilen sorunları giderecek kaliteli yazılımlar geliştirmeyi sağlayan Nesneye Dayalı Çözümleme ve Tasarım (Object Oriented Analysis and Design) yöntemlerini tanıtmaktır.

Yazılım Tasarımı ve Mimarisi

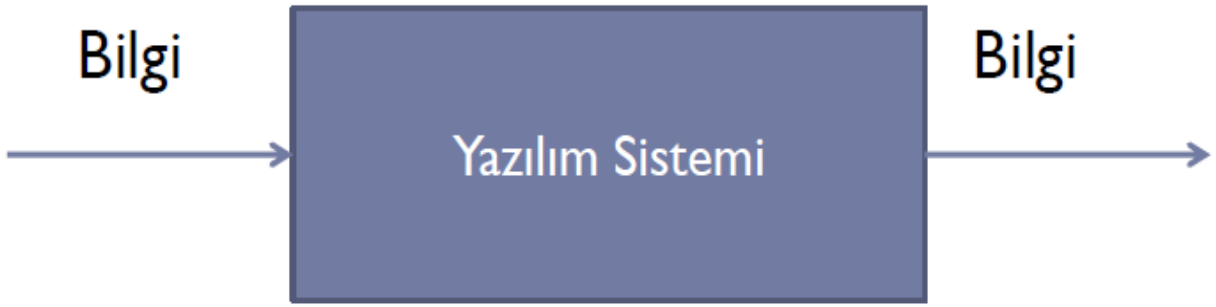
Derste işlenen konular:

1. Yazılım Tasarımı ve Mimarisine Giriş
2. Yazılım Geliştirme Süreçleri
3. Nesne Yönelimli Programlamanın Temel İlkeleri
4. Tümleştirilmiş modelleme dili (UML)
5. İsteklerin Çözümlenmesi, Kullanım Senaryoları
6. Senaryoların Gerçeklenmesi, Tasarımdan Kodlamaya Geçiş
7. Yazılım Tasarımı Kalıpları
8. Grasp Kalıpları ve yaygın biçimde kabul gören GoF kalıpları

Tasarım Yöntemleri

1. Yapısal Tasarım
   - Veri akışına Yönelik Tasarım
   - Veriye Yönelik Tasarım
2. Nesneye Yönelik Tasarım

Veri Akışına Yönelik Tasarım



- Yazılım tasarımında bilgi akışı dikkate alınarak veri akış diyagramları kullanılır.
- Veri akışı çözümlemesi, yüksek uyumlu modüller oluşturabilmek için kullanılan klasik tasarım tekniğidir.
- Ana nokta:
  - Veri akış diyagramı oluşturulduğu anda, yazılım tasarımcısı ürünle ilgili girdi ve çıktılarla ilgili açık ve net bir bilgiye sahip olur.

Veri Akışına Yönelik Tasarım

- Bu yöntem, yazılım belirtiminden şunları tanımlayan yazılım yapısına ulaşmayı sağlar:
  - yazılımın oluşturulduğu modüller
  - bu modüller arasındaki ilişkiler
- Programlama dilleri genellikle modüller için yapılar sağlarlar:
  - Cobol: subprogram
  - Fortran: subroutine
  - Pascal: procedure, function
  - C: function
  - C++: class
  - Java: class

Veri Akışına Yönelik Tasarım

- Her türlü yazılım veri akış diyagramı ile gösterilebilir.
- Özellikle sıradüzensel veri yapılarının bulunmadığı ve bilgilerin ardışık olarak işlendiği yazılımlar için daha kullanışlıdır.
- Kullanıcı etkileşimi çok azsa veya yoksa, sistemin çalışması sürekli veri akışına bağlıysa kullanılır.
  i. Karmaşık sayısal çözümleme yazılımları
  ii. Kontrol sistemleri yazılımları
- Veritabanı sistemleri, nesneye yönelik arayüzlerin bulunduğu sistemlerde diğer yöntemlerin kullanılması daha uygun olur.

Yazılım Tasarımı ve Mimarisi

- **COBOL** (Common Business Oriented Language), bir programlama dili. Ticaret alanı ve özellikle iş yerlerinin yönetimiyle ilgili konularda, tüm dünyada kullanılmak üzere hazırlanmıştır.
- ISAM (Indexed Sequential Access Method) yapısına izin veren sınırlı sayıdaki dilden biridir.
- ISAM (Indexed Sequential Access Method) IBM firmasinin gelistirdigi sirali (sequential) ve rastgele (random) index ile erisim saglayan data erisim sistemidir. IBM mainframe sistemlerinde kullanilmis. Cobol programlama dilinin varsayilan data file (dosya veritabani) seklinde kullanilirdi eskiden.
  - *ISAM sisteminde ya data ayri, indexler ayri tutulur (örnek: musteri.data, musteri.idx gibi) yada bir dosyada hem datalar ve indexler tutulurdu. (Tipik MS Cobol5, Micro Focus Cobol ve RM Cobol örneklerinde görülebilir.)*
  - *ISAM dosyalar cok cabuk bozulabilir(ozellikle data write modunda acikken elektirik kesintisi veya pc kapanmasinda), yeniden indexlemede veri kaybi yasanabilir. Read only acildiginda datalar cok hizli ve guvenli okunabilir.*
  - *Su anki iliskisel veritabani yonetim sistemleri (RDBMS) ile kiyaslanamicayak kadar ilkel bir kullanim sekline sahipti.*

Yazılım Tasarımı ve Mimarisi

- **COBOL** (Common Business Oriented Language),
- Sayı tipi sınırsızdır.
- COBOL 2002 'den beri Nesne Yönelimli Programlama'yı desteklemektedir .
- COBOL 1959'da Üniversiteler, Hükümetler ve Ticari Kuruluşlar tarafından oluşturulan bir komite tarafından yaratılmıştır.
- "COBOL" ismi 18 Eylül 1959'da toplanan komitenin kararıdır.

Yazılım Tasarımı ve Mimarisi

- **COBOL** toplam olarak dört bölümden oluşur.
- **Tanımlama bölümü**: Yazılan programın sürümü, yazarı gibi bilgiler tanımlanabilir.
- **Ortam bölümü**: Program geliştirilirken kullanılacak olan değişkenler, program içerisinde kullanılacak kütüphane tanımlamaları burada yapılabilir.
- **Prosedür bölümü**: Program içerisinde çalışma zamanı anında çalışacak asıl kodlar tutulur. Bu bölümde kullanılan değişkenler ve tip tanımlamaları daha önce de bahsedildiği gibi ortam bölümden çağrılır.
- **Kesim bölümü**: Ayrıca tanımlanılan bölümler içerisinde değişik yordam tanımlamalarının yapıldığı "kesim" adı verilen komut tanımlama alanları vardır.

Yazılım Tasarımı ve Mimarisi

- **COBOL**, daha önceden ilk geliştirilme amacı olarak ticaret ile uğraşan kurum veya kuruluşlarda kurumları temsil eden kişiler ile müşterileri arasındaki her türlü ilişkiyi bilgisayar ortamında geliştirilmiş programlamlarla gerçekleştirilmesini sağlayan bir yazılım olarak ortaya çıkmıştır. Kişiler arasında ilişkileri yukarıda anlatılan bölümler arasında tanımlanan mantıksal yordamlarla gerçekleştirmeyi amaçlayan bir yazılım dilidir.

Yazılım Tasarımı ve Mimarisi

- Günümüzde COBOL'un bulunduğu konum çok önemlidir. Bunun nedenlerinden biri Microsoft tarafından 2001 yılında piyasaya sürülen .NET Framework 1.1 versiyonu ile desteklenmeye başlamış olmasıdır.
- Gelişen ticaret dünyasına ve bunun yanında gelişen yapay zeka teknolojisine destek vermek amacı ile Microsoft tarafından destek verilmiştir.
- .NET Framework bileşenlerini kullanarak basit bir metin düzenleyici aracılığıyla yazılan COBOL kodları ".cb" uzantısı ile kaydededilir ise, .NET derleyicisi tarafından derlenebilir.
- Prolog gibi sınırlı sayıda koda sahiptir. Programcı mantıksal olarak tanımlamak istenen durumları belirler ve dilin kendisine sunduğu yapıları kullanarak bunların kombinasyonlarını sağlayarak sonuç üretmeye çalışır.
- Eski bir yazılım olmasına rağmen COBOL günümüzde bankalarda hala kullanılmaktadır.

Yazılım Tasarımı ve Mimarisi

- **Yapısal programlama**, yordamsal programlama dillerinin pek çoğu ile yapılabilmektedir. 1970'lerin başlarında popülerleşmeye başlayan yapısal programlama ile pek çok yeni yordamsal programlama dili yapısal programlamayı destekleyecek özellikleri barındırmaya başladılar. Bu dillere örnek olarak Pascal ve Ada verilebilir.

- Küçük kod parçacıkları seviyesinde yapısal programlama hiyerarşik program akışı yapılarını tavsiye eder. Bu yapılar pek çok modern dilde kolayca elde edilebilen, "while", "repeat", "for" gibi yapılardır. Yapısal programlama bu yapılar için tek giriş ve tek çıkış noktalarını tavsiye eder. Bu tavsiyeyi zorunlu kılan dillere rastlanmaktadır.

Yazılım Tasarımı ve Mimarisi

Yapısal programlamaya sahip bir dilde kontrol işlemleri (şartlar) aşağıdaki şekilde üçe ayrılırlar:

1. Akış (sequence) bir alt programdan diğerine geçiş işlemi. (fonksiyon veya prosedür, prosedürel programlama)
2. iki alt programdan birisini bir bool mantık işlemine göre çalıştırmak. (if , eğer)
3. bir şart sağlanana kadar bir alt programın çalıştırılması (döngüler, loop , iteration, for, while)

yukarıdaki şartları destekleyen bir dil yapısal programlama mantığına sahiptir denilebilir

Yazılım Tasarımı ve Mimarisi

- **Yapısal programlama,** bir programlama yaklaşımı olup, güncel gelişmelerle birlikte kullanılmaya devam etmektedir.
- Örneğin nesne yönelimli programlama yaklaşımlarını kullanan dillerin neredeyse tamamı yapısal programlamayı da bünyelerinde barındırmaktadır.

Yazılım Tasarımı ve Mimarisi

Sorun Ne?

- Dinamik yapıdaki problemler yapısal programlama teknikleri ile modellendiğinde değişen şartlara ve ihtiyaçlara uyum sağlayamamaktadır.

- Problemlerin karmaşıklığı arttıkça mevcut tekniklerle yazılım geliştirmek imkansız hale gelmektedir.

Yazılım Tasarımı ve Mimarisi

Sorun Ne?

- Tekrar eden kodlar yazmak,
- değişen şartlar karşısında yazılımda ciddi değişiklikler yapmak ve
- en küçük bir değişimde kodlara müdahalede bulunmak

geliştirilen yazılımların ömrünü kısaltmaktadır.

Yazılım Tasarımı ve Mimarisi

Çözüm için neler yapıldı?

- 80 li yıllar itibariyle yapısal programlamanın devamında nesneye dayalı programlama (NDP) teknikleri ortaya çıkmıştır.

- Modüler ve yeniden kullanılabilir NDP teknikleri esnek ve gürbüz yazılımların geliştirilmesini sağlamıştır.

Yazılım Tasarımı ve Mimarisi

Çözüm için neler yapıldı?

- 90'lı yıllar boyunca NDP, karmaşıklaşan problemlerin çözümünde önemli kolaylıklar sağlanmıştır.

- 2000 li yıllardan itibaren yazılım sektörünün çok daha karmaşık problemlere çözüm üretmesi ve büyük çaplı projelere uygulanması gerekmiştir.

Yazılım Tasarımı ve Mimarisi

Büyük çaplı karmaşık projelerde yazılımların

- okunabilirliği,
- genişletilebilirliği,
- değişen şartlara uyum sağlama yeteneği ve
- isteklere cevap verememesi/kod hatası

gibi sorunları tekrar gündeme gelmiştir.

Yazılım Tasarımı ve Mimarisi

Büyük çaplı karmaşık projelerin yazılımları gerçekleştirilirken,

- NDP teknikleri uygulandığı halde dahi iyi bir tasarım izlenmezse ciddi problemlerle karşılaşıldığı görülmüştür.
- Dolayısıyla günümüz yazılım dünyasında NDP tekniklerinin yanı sıra tasarım süreci de hayati bir öneme sahiptir.

Yazılım Tasarımı ve Mimarisi

Büyük çaplı karmaşık projelerin yazılımları gerçekleştirilirken,

- Problemi gerçeğe en yakın haliyle modelleyen iyi bir tasarımla oluşturulmuş nesnelerle başarılı çözümler üretilirken, kötü tasarlanan nesnelerde ise NDP teknikleri uygulansa dahi başarısızlık söz konusu olabilmektedir.

Yazılım Tasarımı ve Mimarisi

Çözüm Tasarımda!

Tasarım Nitelikleri Ne Olmalı?

i. İsterler ile izlenebilirliği olmalıdır
ii. Geliştirilen birimin kodu ve testleri ile izlenebilirliği olmalıdır.
iii. Programlama dilinden olabildiğince bağımsız olmalıdır
iv. Öğrenmesi ve kullanımı kolay bir ürünü hedeflemelidir
v. Tekrar kullanılabilir olmalıdır
vi. Kolay anlaşılmalıdır
vii. Gerektiğinde kolaylıkla değiştirilebilmelidir.

Yazılım Tasarımı ve Mimarisi

Yapışıklık (Kohezyon) nedir?

- Modül içerisindeki etkileşim derecesi
- Az mı çok mu?
  i. Çok olmalı
  ii. Eğer çok olursa, modüller diğer modüllerin karmaşıklığıyla uğraşmaya gerek duymaksızın tasarlanabilir, kodlanabilir ve test edilebilirler.
  iii. Modül içerisinde bir hata olursa diğer modüllere yayılması önlenmiş olur

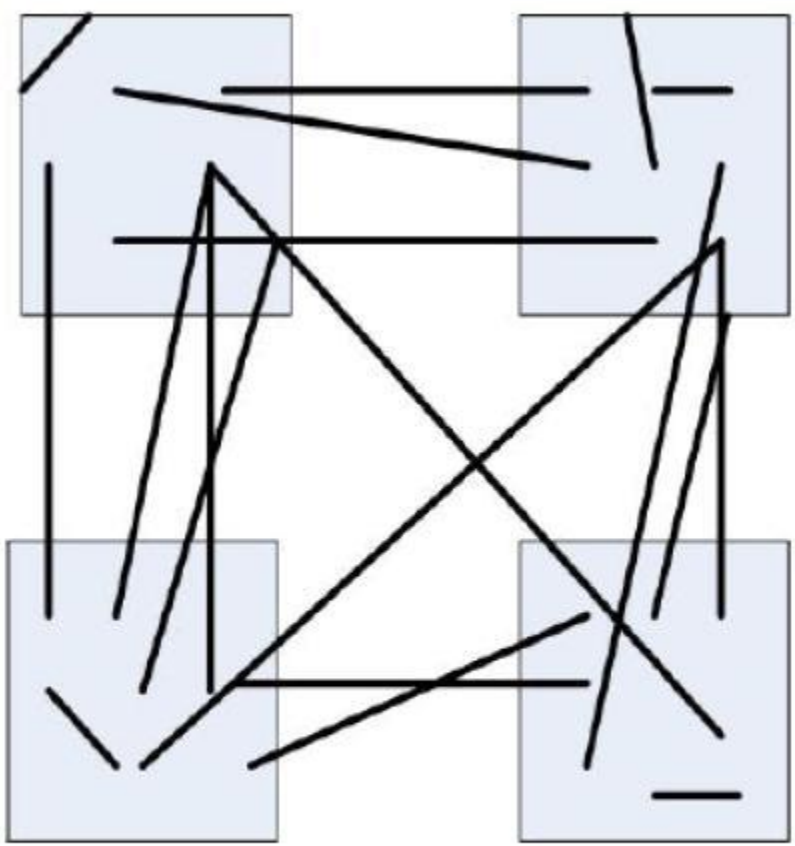
Yazılım Tasarımı ve Mimarisi

Bağlaşım (coupling) nedir?

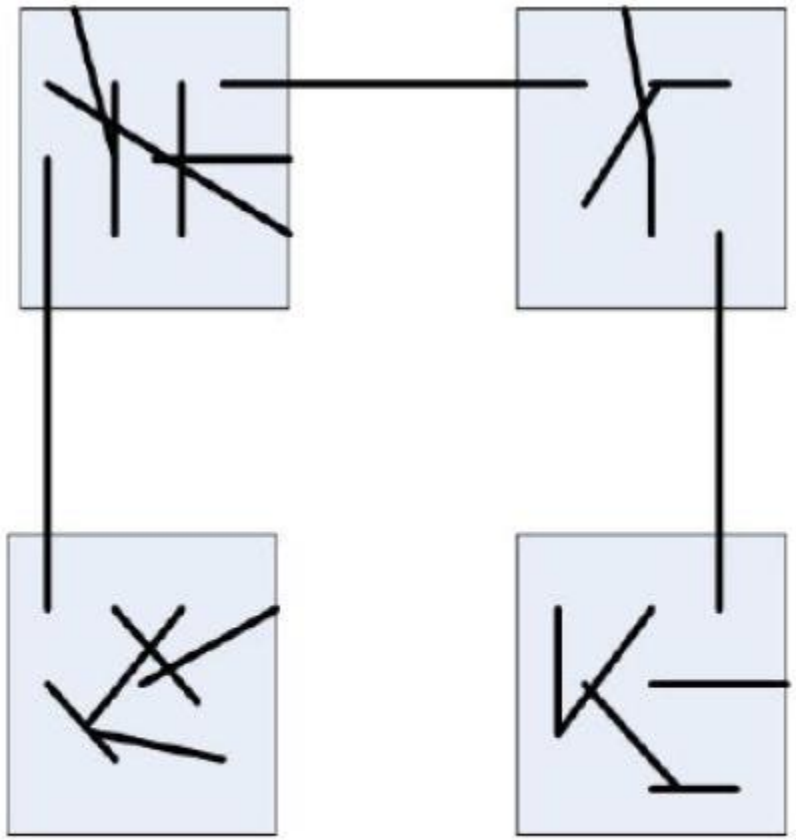
- Modüller arası etkileşim derecesi
- Az mı çok mu?
  i. Az olmalı
  ii. Tavsiye edilen en fazla ikiden dörde kadar parametrenin kullanılması
  iii. Karmaşıklığı azaltır

Yazılım Tasarımı ve Mimarisi

Yapışıklık ve bağlaşım



Az kohezyon, çok bağ



Çok kohezyon, az bağ

Yazılım Tasarımı ve Mimarisi

Anlaşılabilirlik

- Tasarımla ilgilenecek herkes onu kolaylıkla anlayabilmelidir.
  - **Yapışıklık ve bağlaşım:** Bileşen başka bileşenlerden bahsetmeden de anlaşılabilir mi?
  - **İsimlendirme:** Bileşenler için kullanılan isimlendirmeler anlamlı mı?
  - **Belgelendirme:** Bileşenler gerçek dünya ve bileşenler arasında eşleştirme yapabilmeyi sağlayacak şekilde belgelendirilmişler mi?
  - **Karmaşıklık:** Bileşeni gerçekleştirmek için uygulanacak algoritmalar ne kadar karmaşık?

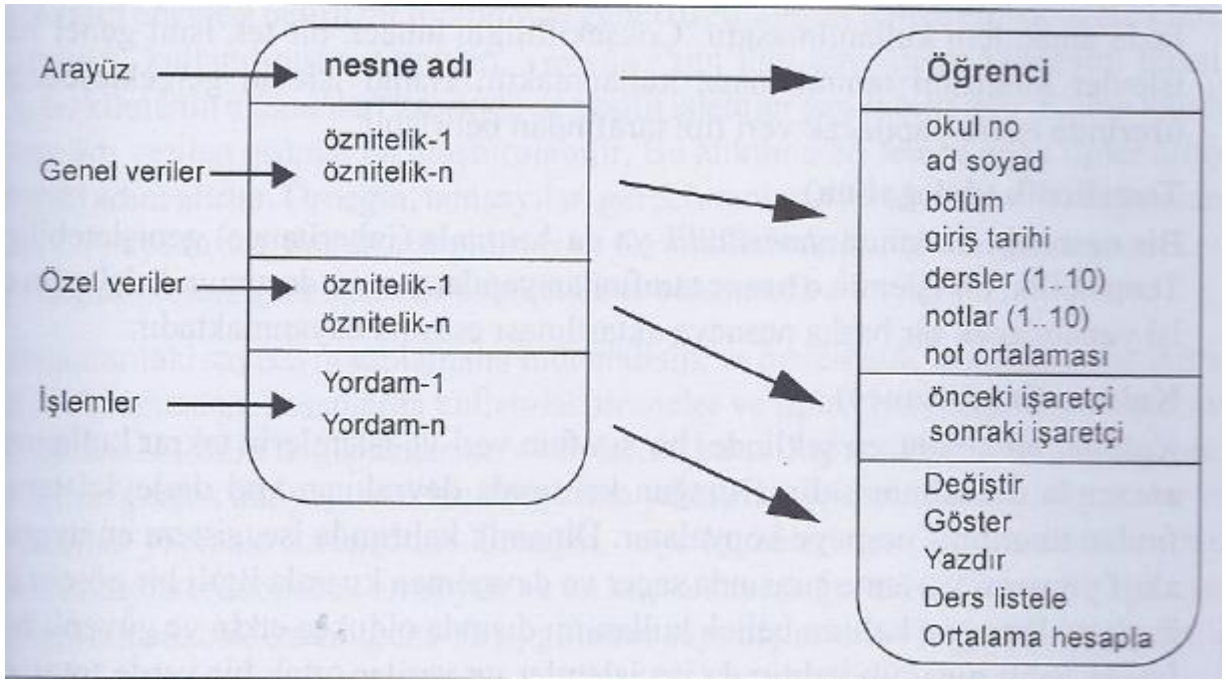
Adapte olabilirlik

- Tasarımın ne kadar kolay değiştirilebileceğidir.
  - **Bağlaşım:** Bileşenler düşük bağlaşımlı olmalı
  - **Anlaşılabilirlik:** Belgelendirme anlaşılabilir hazırlanmış olmalı
- Adapte olabilir sistem, farklı düzeydeki tasarım modelleri arasında yüksek oranda takip edilebilirliğin olduğu sistemlerdir.

Yazılım Tasarımı ve Mimarisi

Nesneye Yönelik Tasarım

- Nesneye yönelik yaklaşım
  - nesnelerin özellikleri ve ilgili oldukları süreçler ön planda



Yazılım Tasarımı ve Mimarisi

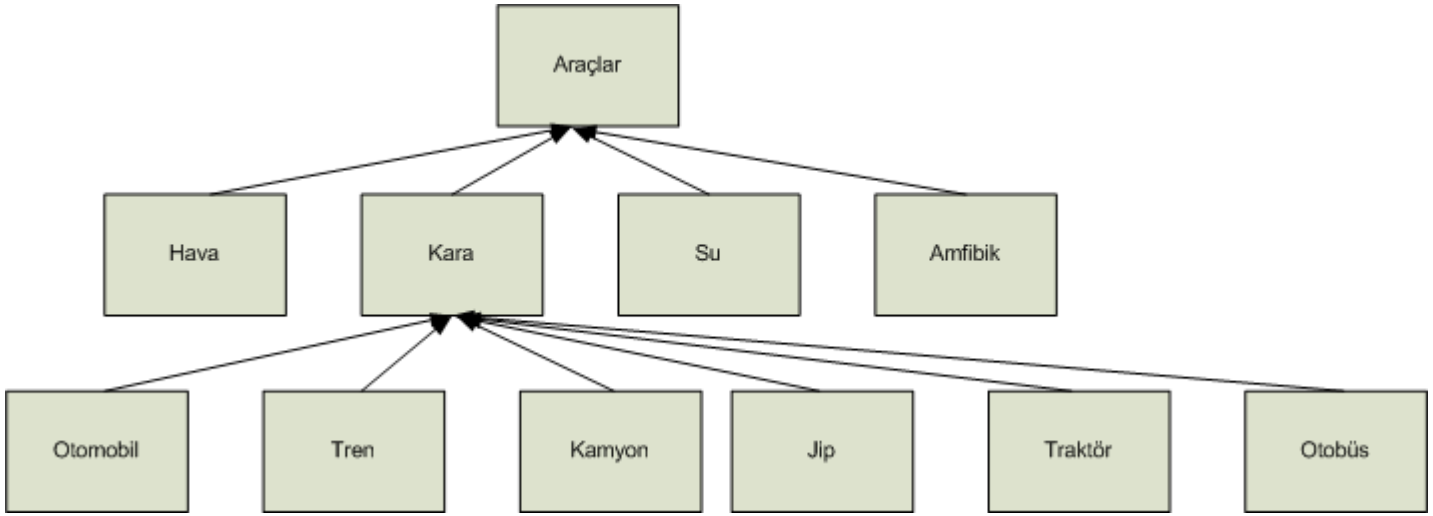
Nesneye Yönelik Tasarım: Nesneye yönelik temel kavramlar

- **Kapsülleme(Encapsulation):** Veri ve süreç paketlerinin bir arada paketlenmesi
- Gereksiz detayların gizlenmesi de bu özellik tarafından sağlanmaktadır.

Yazılım Tasarımı ve Mimarisi

Nesneye Yönelik Tasarım: Nesneye yönelik temel kavramlar

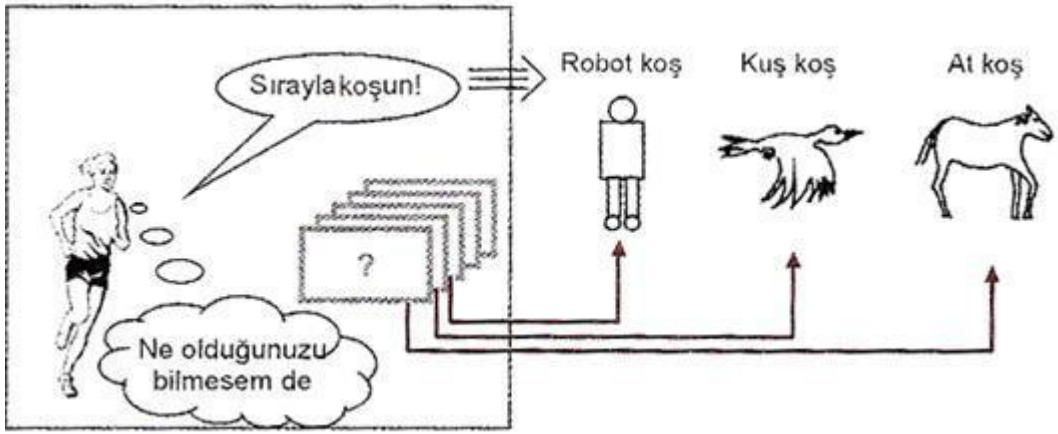
- **Kalıtım(Inheritance):** Daha genel nesne sınıflarından daha özel nesne sınıflarının özellik ve süreç edinmeleri
- Sınıftan sınıf üretimi



Yazılım Tasarımı ve Mimarisi

Nesneye Yönelik Tasarım: Nesneye yönelik temel kavramlar

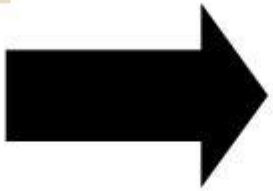
- **Çok şekillilik (Polymorphism):** Aynı isimle tanımlanan bir sürecin, hangi nesne tarafından harekete geçirildiği ile ilgili olarak kendiliğinden değişik davranışlara girebilmesi.
- Örn: Şekildeki nesnelerin türüne bakılmaksızın koşturulması



Yazılım Tasarımı ve Mimarisi

Nesneye Yönelik Tasarım: Nesneye yönelik temel kavramlar

Kavramsal sınıflar



Yazılım sınıfları

- Sınıflar arası etkileşimler
- Sorumlulukların sınıflara atanması = nesnelere metodların eklenmesi

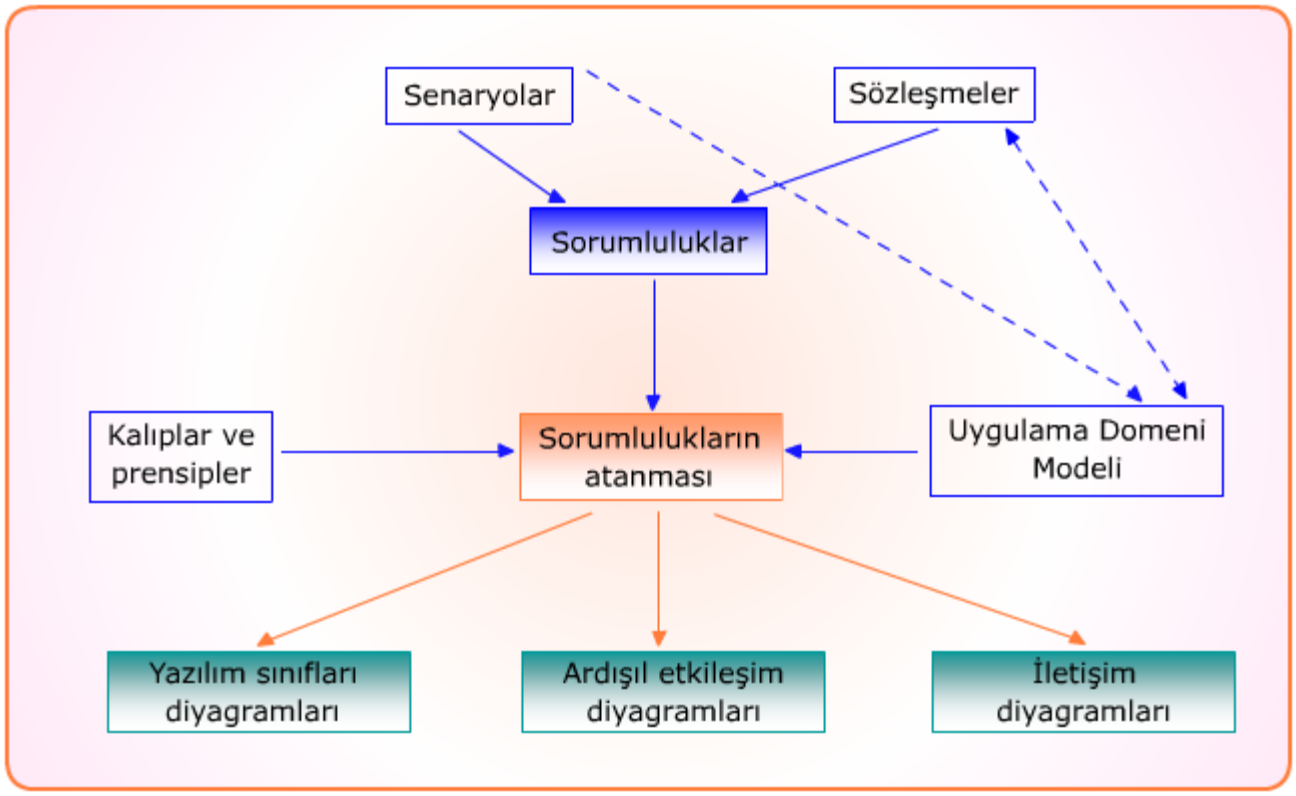
Yazılım Tasarımı ve Mimarisi

Nesneye Yönelik Tasarım: Nesne Yönelimli Yazılım Tasarımının Genel Şeması

- Tasarım aşamasında öncelikle senaryo ve sözleşmelerden sorumluluklar belirlenir.
- Ardından tasarım kalıpları (desenleri) ve prensipleri kullanılarak bu sorumluklar uygun sınıflara atanır.
- Tamamlanan tasarım **UML** diyagramları ile ifade edilir.
  - Use-case
  - Sequence

Yazılım Tasarımı ve Mimarisi

Nesneye Yönelik Tasarım: Nesne Yönelimli Yazılım Tasarımının Genel Şeması



Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri *(Design Patterns)*

Desen:Örüntü:Kalıp

- Profesyonel yazılım geliştiricilerin, deneyimleri neticesinde kazandıkları, değişen şartlara uyum sağlayabilen kod yazma alışkanlıkları günümüzde standart çözümler haline gelmiş ve *tasarım desenleri* adını almıştır.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri (Design Patterns) nedir? Nasıl ortaya çıkmıştır?

- Tasarım desenleri, yazılımın geliştirilmesi sırasında sıkça karşılaşılan ve birbirine benzeyen problemleri, tasarım aşamasında çözmek için geliştirilmiş ve işlerliği kanıtlanmış genel çözüm önerileridir.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri (Design Patterns) nedir?

- Genel olarak tasarım desenleri, programlama dillerinden bağımsız olarak tanımlansalar da, Nesne Yönelimli Programlama dillerine uygun tasarım desenleri daha çok bilinir.

Yazılım Tasarımı ve Mimarisi

*Tasarım Desenleri **nedir**? **Nasıl ortaya çıkmıştır**?*

- Tasarım desenleri, nesnelerin (sınıf örnekleri - class instances) nasıl yaratılacağı hakkında öneriler sunar.
- Ana fikir, iyi bir yazılımın, içinde barındırdığı nesnelerin nasıl yaratıldığından bağımsız olarak tasarlanması gerekliliğidir.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri nedir? Nasıl ortaya çıkmıştır?

- Diğer bir deyişle, nesnelerin nereden ve nasıl yaratıldığı, ait oldukları yazılımın işleyişini etkilememeli; yeni özellikler eklenmesine ve değişikliklere karşı sorun oluşturmamalıdır.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri nedir? Nasıl ortaya çıkmıştır?

- Yazılım sistemleri geliştikçe, nesnel bileşimler, sınıf kalıtımına göre daha fazla önem kazanır.

- yani, nesnelere davranışların bileşim olarak eklenmesi, daha sonra bu davranışların yazılımın gelişimine göre değiştirilmesine olanak sağlar.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri nedir? Nasıl ortaya çıkmıştır?

- geliştirilen yazılım için gereken temel davranış şekillerine dayalı bir tasarım, nesne arayüzleri (interface) değiştirmeden farklı ya da daha karmaşık davranışların kullanılabilmesini olanaklı kılar.

- Bunun nedeni, yazılım sistemleri için basit temel davranış (behavior) şekillerinin tanımlanması üzerine kurulu tasarımların, sabit davranışlara dayalı tasarımlara göre daha esnek olmasındandır.

Yazılım Tasarımı ve Mimarisi

*Tasarım Desenleri **nasıl uygulanır?***

- Tasarım aşamasında, yazılım sınıfları ve aralarındaki işbirliği (etkileşim) belirlenir.

- Burada amaç, senaryolarda belirlenmiş olan sorumlulukların (sistemden beklenenler) uygun sınıflara atanması (assignment of responsibilities) ve bu sınıfların tasarlanmasıdır.

- Nesnel tasarım (object design) gerçekleştirilirken sınıfların nasıl oluşturulacağı ve sorumlulukların nasıl atanacağı konusunda tasarım kalıplarından (design patterns) yararlanılır.

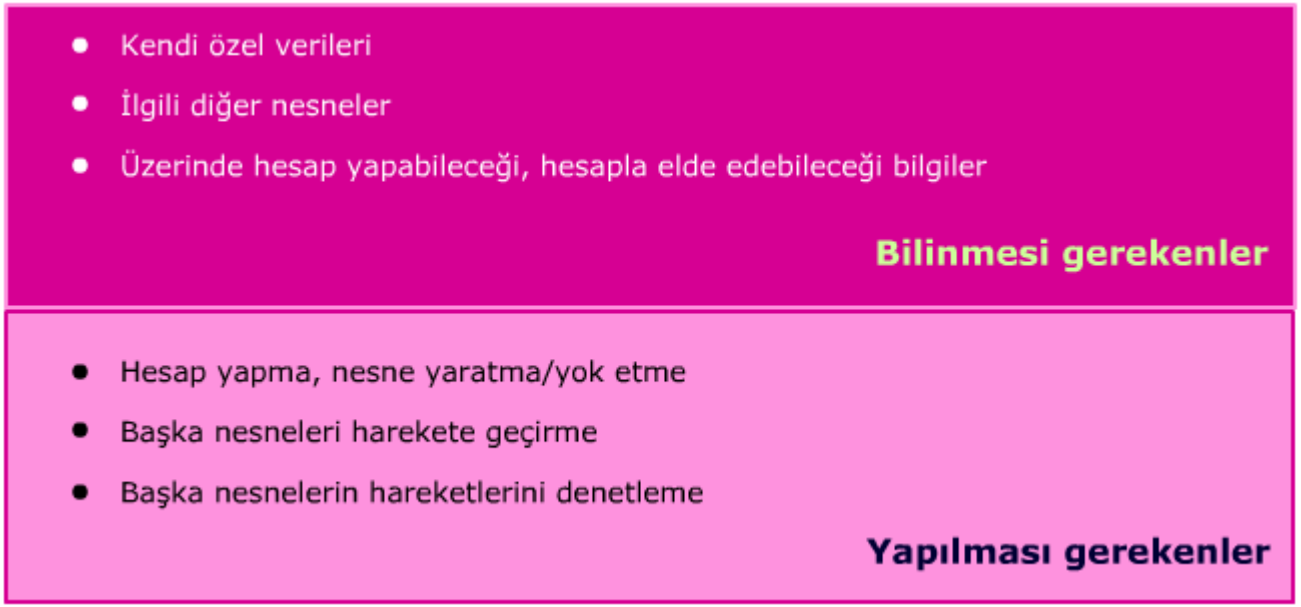
Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri: Sınıfların Sorumlulukları nelerdir?

- Nesnel Tasarımın (Object Design)genel ifadesi
  - İsteklerin belirlenmesi ve uygulama alanı, ortamı modelinin oluşturulmasından sonra;
    - Yazılım sınıflarına metotların eklenmesi (sorumlulukların atanması)
    - İstekleri yerine getirmek üzere nesneler arası mesajların belirlenmesi
- Nesnel tasarımın temeli nesnelere sorumlulukların atanmasıdır.
- Nesnelerin sorumlulukları 2 gruba ayrılır:
  - Bilinmesi gerekenler
  - Yapılması gerekenler

Yazılım Tasarımı ve Mimarisi

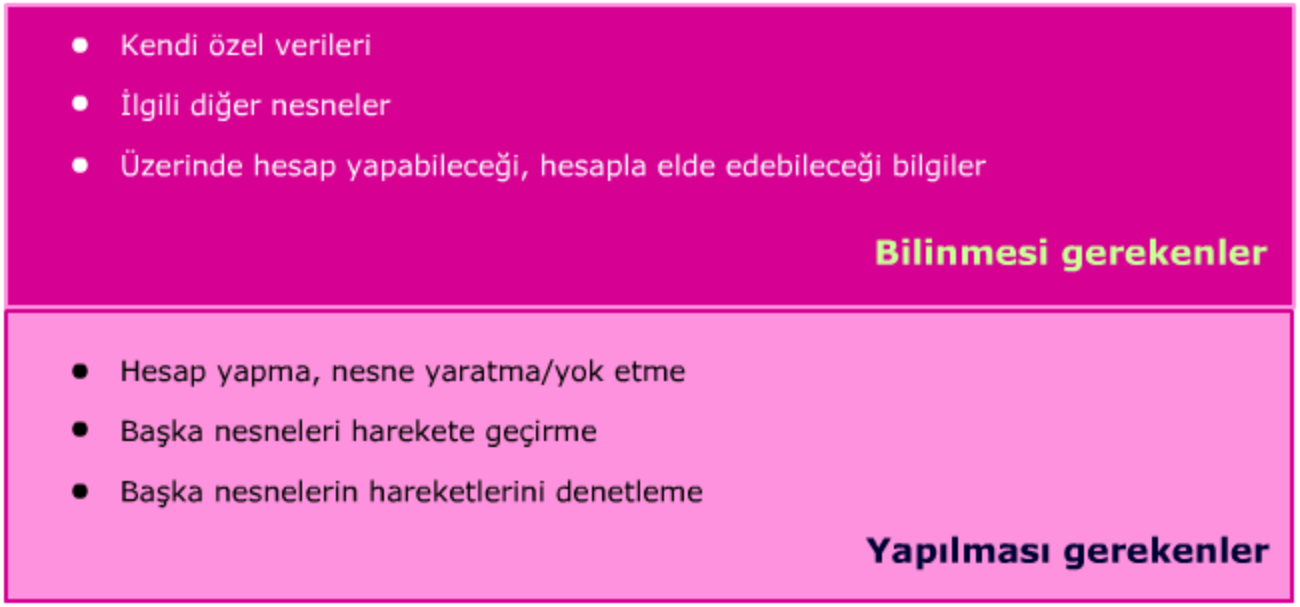
Tasarım Desenleri: Sınıfların Sorumlulukları nelerdir?



- Sorumlulukları yerine getirmek için metotlar oluşturulur.
- Bir sorumluluğu yerine getirmek için bir metot başka nesnelerdeki metotlarla işbirliği yapabilir.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri: Sınıfların Sorumlulukları nelerdir?

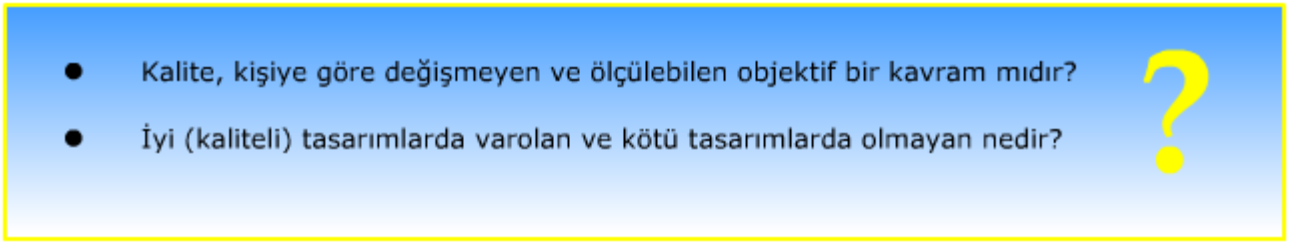


- Bir sistemdeki sorumluluklar o sistem için yazılmış olan senaryolardan (use-case) ve sözleşmelerden (contract) elde edilir.
  - Daha sonra bu sorumluluklar tasarım prensipleri ve kalıpları kullanılarak uygun sınıflara atanır.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri: Sınıfların Sorumlulukları nelerdir?

- Hangi yazılım sınıfına hangi sorumlulukların atanacağını ve hangi sınıflarla işbirliği yapılacağını belirlemek için tasarım prensiplerinden ve kalıplardan yararlanmak gerekir.
- Tasarım kalıplarının varlığı ilk olarak bir mimar olan **Christopher Alexander** tarafından ortaya konulmuştur. Bu süreçte sorulan sorular şunlardır:



Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri:

- **Christopher Alexander** yaptığı araştırmalar sonunda benzer problemleri çözmek için oluşturulan ve beğenilen *(kaliteli)* mimari yapılarda ortak özellikler *(benzerlikler)* olduğunu belirlemiştir.

- Bu benzerliklere kalıplar/desenler/örüntüler *(patterns)* adını vermiştir.

- Her kalıp gerçek dünyada defalarca karşılaşılan bir problemi ve o problemin çözümünde izlenmesi gereken temel yolu tarif etmektedir.

- Türkçesi: ***"Aklın yolu birdir."***

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri:

- Bir problemle karşılaşan tasarımcı eğer daha önce benzer problemle karşılaşan tasarımcının uyguladığı başarılı çözümü biliyorsa *(kalıp)* herşeyi yeniden keşfetmek yerine aynı çözümü tekrar uygulayabilir.

- Mimarlıkta olduğu gibi yazılım geliştirmede de benzer problemlerle defalarca karşılaşılmaktadır.

  – Yazılımcılar deneyimleri sonucunda birçok problemin çözümünde uygulanabilecek **prensipler ve deneyimler** *(kalıplar)* oluşturmuşlardır.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri

Bu konudaki ilk önemli yayın dört yazar tarafından hazırlanan bir kitap olmuştur:

- Yazılım tasarım örüntüleri 1994 tarihinde *"Tasarım Örüntüleri: Tekrar kullanılabilir Nesneye Yönelik Yazılımın Temelleri (Design Patterns: Elements of Reusable Object-Oriented Software)"* adıyla yayınlanan kitap ile yaygınlaşmaya başlamış.

- Kitabın yazarları Erich Gamma, Richard Helm, Ralph Johnson ve John Vlissides bilgisayar bilimleri çevresinde **Dörtlü Çete** olarak anılır olmuştur.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri

- Gof Design Patterns, **"Dörtlü Çetenin Kitabı"** (Book of GoF) olarak anılır ve gerçek hayattaki nesnelere en yakın olan, ortak ve sürekli karşılaşılan problemlere getirilen 23 adet çözümden oluşur.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri

- Gang of Four'a göre tasarım desenleri üç (3) sınıfa ayrılır. Fakat bu sınıfları birbirinden ayıran keskin kriterler yoktur;

1. Creational patterns (Yaratışsal Tasarım Desenleri)

   Abstract Factory, Builder, Factory Method, Prototype, Singleton

2. Structural patterns (Yapısal Tasarım Desenleri)

   Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy

3. Behavioral patterns (Davranışsal Tasarım Desenleri)

   Chain of responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy , Template method, Visitor

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri

Günümüzde kurumsal büyük ölçekli karmaşık projeler temel olarak;

- Veri tabanına bağlanma,
- Servis temelli geliştirme,
- Katmanlı mimari,
- katmanlar arası haberleşme gibi

tasarımsal özelliklere ve yazılım süreçlerine sahiptir.

Bu tür büyük çaplı projelerde karşılaşılan problemler neticesinde Martin Fowler isimli yazar 51 adet tasarım kalıbı/deseni geliştirmiştir.

Yazılım Tasarımı ve Mimarisi

Tasarım Desenleri

- Martin Fowler'ın kurumsal yazılımlar için mimari tasarım kalıpları her bir katman için doğru tasarım desenleri standartlar olarak önerilmekte ve sunum katmanında ise Model View tasarım kalıpları ile arayüz ile modelin birbirinden ayrılması sağlanmaktadır.

- Tasarım kalıplarının/desenlerinin temelinde nesneler arası bağımlılığın olabildiğince sıfıra yaklaşması fikri yatmaktadır. Nesneler arası bağımlılık ne kadar az ise tasarım o kadar esnek olmaktadır.

EKLER

Ek Materyaller

Veri Modelleri

- Verileri mantıksal düzeyde düzenlemek için kullanılan yapılar, kavramlar ve işlemler topluluğuna veri modeli (data model) denir.
- Her VTYS belirli bir veri modelini kullanır.
- Bir VTYS'yi kullanarak oluşturulacak her veri tabanında yer alacak veriler ve veriler arası ilişkiler, mantıksal düzeyde ilgili veri modeline göre düzenlenir; bu veri modeli kullanılarak veri tabanının kavramsal ve dış şemaları oluşturulur.

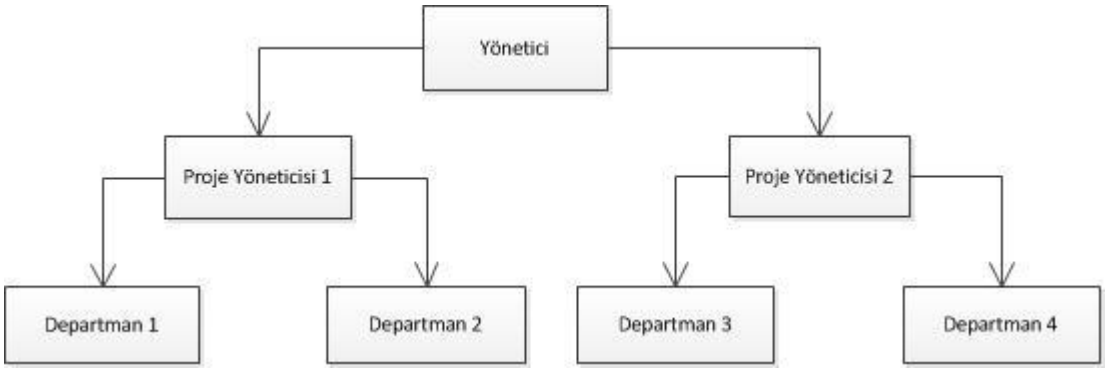
Veri Modelleri

- Bugüne kadar geliştirilmiş olan çok sayıda veri modeli vardır.
- Ancak geçmişte ve günümüzde yaygın kullanılan veri modellerini 4 grupta toplamak mümkündür:
  - Sıradüzensel Veri Modeli (Hierarchical Data Model)
  - Ağ Veri Modeli (Network Data Model)
  - İlişkisel Veri Modeli (Relational Data Model)
  - Nesneye-Yönelik Veri Modeli (Object-oriented Data Model)
- Yukarıdaki sıralama aynı zamanda kronolojik bir sıralamadır.

Veri Modelleri

1. Basit Veri Modelleri

- **Sıradüzensel Veri Modeli (Hierarchical Data Model)**
  - Tarihine değinecek olursak; en eski veri modeli ve en çok 60'lı 70'li yıllarda kullanılmıştır. Ağaç veri yapısına benzer. Her kaydın 1 ebeveyn kaydı ve birden çok çocuk kaydı vardır. Farklı şekilde ifade edecek olursak, bu modelde bir düğüm alt seviyedeki n düğüme, üst seviyedeki 1 düğüme bağlanabilir.
  - Örnek: IBM IMS (*Information Management System*)



Veri Modelleri

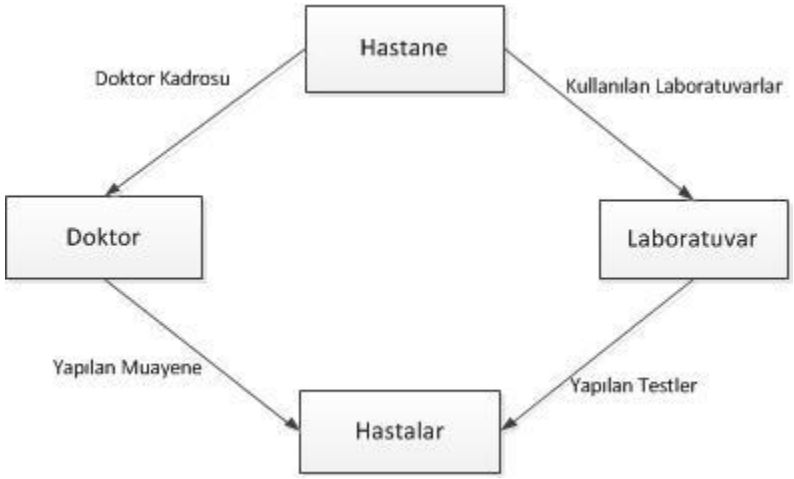
- **Ağ (şebeke) Veri Modeli (Network Data Model)**
  - 1969 yılında ortaya çıkmış ve 1970 ile 1980 yıllarının ilk yarısında kullanılmıştır. Tablo ve grafik temellidir. Grafikteki düğümler varlık tiplerini gösterir ve tablolara karşılık gelir.
  - İki veri yapılandırma aracından oluşur: Kayıt tipi ve bağlantı.

Veri Modelleri

- **Ağ (şebeke) Veri Modeli (Network Data Model)**

  – Hiyerarşik modelde bir düğüm alt seviyedeki n düğüme, üst seviyedeki 1 düğüme bağlanıyordu. Ağ veri modelinde ise bu sınırlama yoktur, bir eleman herhangi bir elemana bağlanabilir. Bağlantılar tarafından belirlenmiş ilişkiler dışında kayıt tipleri arasında ilişki belirlenemez.

  – Ağ modelinde düğümler arası çoklu ilişki kurulamadığından kısıtlı yapıdır. Ancak hiyerarşik model daha kısıtlıdır.



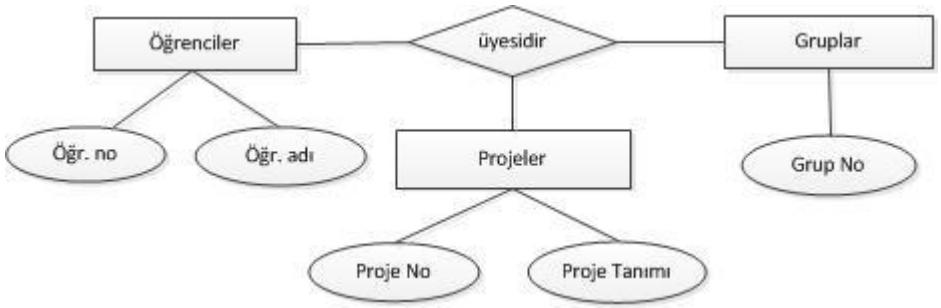
Veri Modelleri

2. Geliştirilmiş Veri Modelleri

- **Varlık-İlişki Veri Modeli (Entity-Relationship Data Model)**

  – Bu modelleme tekniği ilk olarak 1976 da Peter Chen tarafından ortaya çıkarılmış. 1977 de farklı kişiler tarafından geliştirilmiş ve 1980'li yıllarda son şeklini kazanmıştır.

  – Ağ ve hiyerarşik veri modellerinde sadece ikili fonksiyonel bağlantılar vardı. Burada ise varlıklar arasında n-adet ilişki olabilir. Tekrar eden bağlantılar da kullanılabilir.



Veri Modelleri

- **İlişkisel Veri Modeli(Relational Data Model)**
  - 1980'lerde ortaya çıkarılmış veri modeli tipidir. Veriler ve ilişkiler "tablolar" üzerinde tanımlanır ve tüm bilgiler görülebilecek şekilde tasarlanmış veri modelidir.

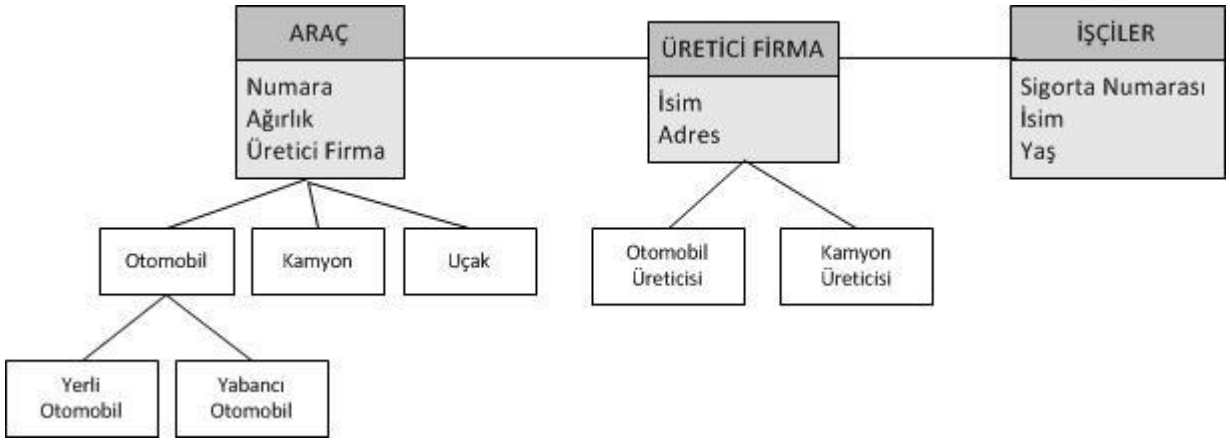
Kitap_ID	K_Adı	Yayınevi_ID	Yazar_ID
1	Kadın Sultanlar	2	1
2	Hasret	1	2
3	Ruhi Mücerret	3	3
4	Aşk	1	4

Yayınevi_ID	Yayınevi_Adı
1	Doğan Kitap
2	Timaş
3	April

Yazar_ID	Yazar_Adı
1	Sibel Eraslan
2	Canan Tan
3	Murat Menteş
4	Elif Şafak

Veri Modelleri

- **Nesneye-Yönelik Veri Modeli (Object-oriented Data Model)**
  - 1990 yıllarında ortaya çıkarılmıştır. İlişkisel modelin ilişki kavramı burada sınıf kavramına denk geliyor. Veriler nesne olarak modellenir ve yaratılır. Nesneye Yönelik Programlama(OOP) da olan sınıf ve miras kavramları burada da geçerlidir.
  - Karmaşık veriler üzerinde işlem yaparken yüksek performans sunan bir yaklaşımdır.



Veri Modelleri

- **Nesneye Yönelik Veri Modeli'nin İlişkisel Veri Modeline Göre Avantajları**
  - Veri tipleri esnektir
  - Nesne tanımlarında soyutlama yapılabilir
  - Veri bütünlüğü daha kolay sağlanır
  - Veri yapısında daha fazla genişleme ve yeniden düzenleme imkanı vardır.

Ana Sunuya Geri Dön