Programlamaya Giriş

Ders Tanıtımı & Yazılım Geliştirme Yaşam Döngüsü

Konular

- ✓ Ders Tanıtımı
- ✓ Temel Kavramlar
- ✓ Büyük Resim – Algoritmik Problem
- ✓ Yazılım Geliştirme Yaşam Döngüsü
- ✓ Bir Yazılımın Kalitesi
- ✓ Programlama Dilleri
- ✓ Programlama Teknikleri
- ✓ Kaynaklar

Ders Tanıtımı

Ders ile ilgili tanıtım sayfasının adresi:
http://ebs.sabis.sakarya.edu.tr/DersTumDetay/tr/2016/255/21/2/68615/0

Ders İçeriği

Bilgisayar yapısı, yazılım, programlama, (nesne yönelimli, fonksiyonel), makine kodu, yazılım geliştirme, yaşam döngüsü, tasarım (UML), kodlama
Yazılım geliştirme ortamı (Program yazma, editör, Eclipse, V.Studio, ilk program)
Gereksinim analizi, tasarım/modelleme, sözde kod, UML
Yazılımı gerçekleme/Kodlama, değişkenler, karar yapıları, tekrarlı yapılar, diziler
Fonksiyon tanımlama, özyineli fonksiyonlar
Sınıf-nesne, static, fonksiyon aşırı yükleme
Kalıtım, operatör overloading, fonksiyonu geçersiz kılma
İşaretçiler (new, delete)
String sınıfı, Dosyalama işlemleri (nesne yönelimli)
Proje Değerlendirme

Temel Kavramlar

Bilgisayar

Genel olarak, girilen veriler üzerinde aritmetiksel, mantıksal ve ilişkisel işlemler yapabilen ve verileri depolama yeteneğine sahip olan elektromekanik cihazlara bilgisayar denir. **Donanım, Yazılım**

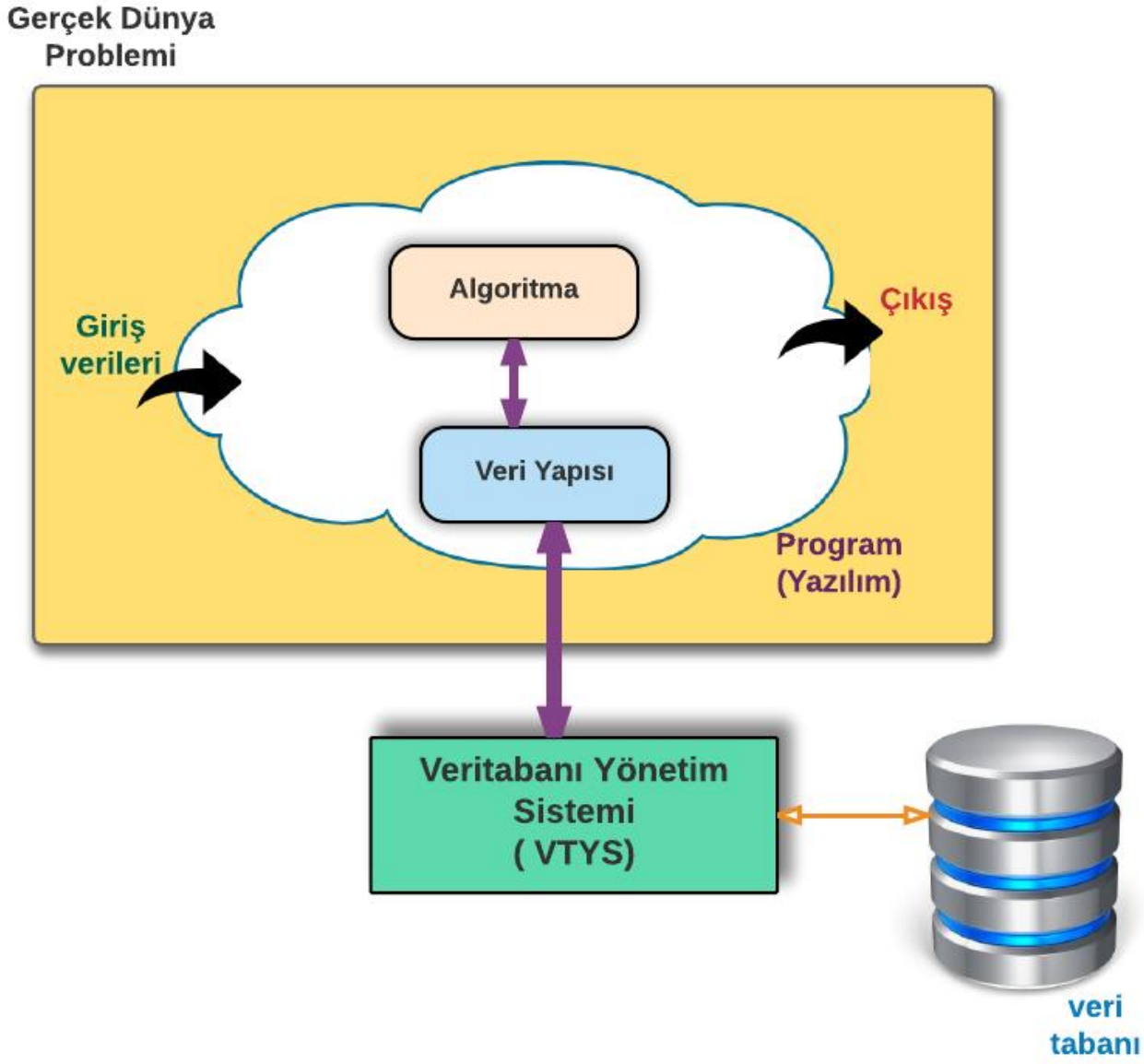
Donanım

Bilgisayarı oluşturan fiziksel parçalar.

- Mikroişlemci
- Bellek
- Giriş/Çıkış Birimi

Büyük Resim - Algoritmik Problem

- ✓ Gerçek dünya problemlerini bilgisayar yardımıyla çözmek için yazılım (program) geliştirilir.



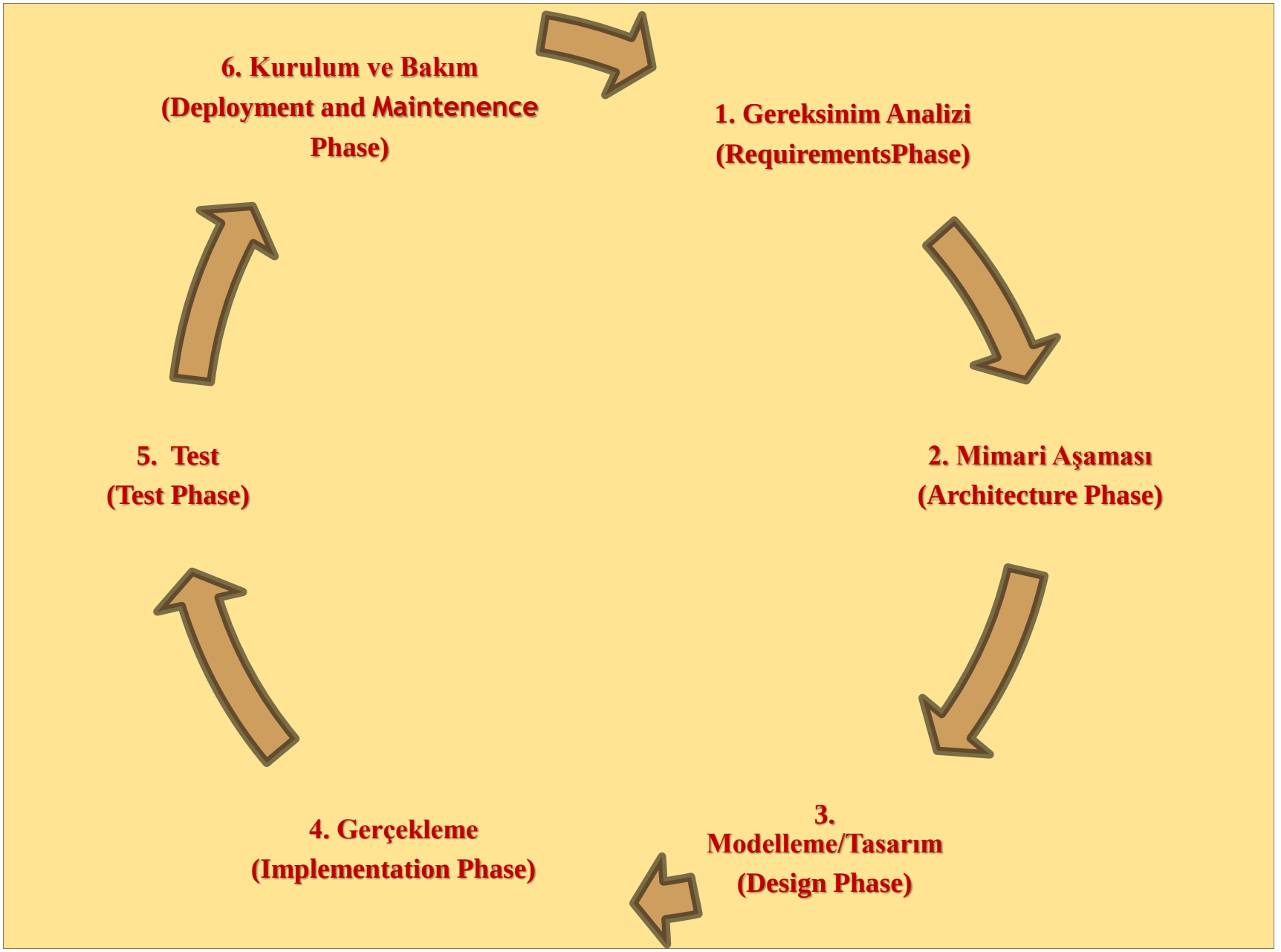
Temel Kavramlar

- ✓ **Algoritma:** Bir problemin çözümüne yönelik işlem basamaklarının sıralı olarak ifade edilmesi

- ✓ **Program:** Probleme ilişkin çözümlerin, herhangi bir programlama dili kullanarak, bilgisayarın anlayacağı komutlar dizisi şeklinde yazılmasına **program**, bilgisayarda kullanılan programların genel adına da **yazılım** denir.

- ✓ Ders boyunca; herhangi bir gerçek dünya problemini bilgisayar yardımıyla çözebilecek **algoritmaların nesne yönelimli olarak geliştirilmesi** anlatılacaktır.

Yazılım Geliştirme Yaşam Döngüsü



Yazılım Geliştirme Yaşam Döngüsü

Çözümleme (Analiz)/ Gereksinim Analizi: Belirsizlik kalmayacak şekilde çözülmesi istenen problem anlaşılır ve yazılımdan yapması beklenenler ayrıntılı olarak listelenir. İhtiyaç listesi oluşturulur.

Tasarım: Bu aşamada amaç, gerçek dünyadaki problemin bilgisayarda temsil edilebilecek soyut bir modelinin oluşturulmasıdır. Çözümün hangi unsurlardan oluşacağı ve bu parçaların nasıl modelleneceği tamamıyla programlama yöntemine bağlıdır.
Tasarım sonrası ortaya çıkacak olan yazılımın kalitesini doğruda etkilediğinden bu aşamada iyi ve doğru bir çözümün oluşturulması çok önemlidir. Bu nedenle kodlamaya geçilmeden önce kurulan modelin sağlamasının (verification) yapılması gerekir.

UML (Unified Modelling Language)

Gerçekleme: Tasarım aşamasında oluşturulan model, bir programlama dili ile bilgisayara bu aşamada aktarılır.

Test: programın olası giriş değerleri için nasıl davrandığı incelenir. Özellikle kritik değerler için programın çalışması test edilmelidir.

Hata Ayıklama: (Yazım Hatası, Mantıksal Hata, Çalışma Zamanı Hatası)

Kurulum ve Bakım (Deployment and Maintenence)

Dokümantasyon: Yazılım projesinin her aşamasında yapılan işleri açıklayan dokümanlar hazırlamak gereklidir.

Bir Yazılımın Kalitesi

- Etkinlik
  - Hız ve kaynak kullanımı
- Bakım Kolaylığı
- Sağlamlık
  - Hatalardan etkilenme
- Anaşılabilirlik
- Taşınabilirlik
- Geliştirilebilirlik
- Güvenlik

Nesneye dayalı programlama yukarıdaki maddelerde belirtilen kalite kriterlerini sağlamak üzere ortaya konmuş bir yöntemdir.

Programlama Dilleri

1. **Makine Dili (Düşük Seviye)**

   Anlaşılması zor, ikili (ya da hex) sayılardan oluşur, makine bağımlıdır

```
+1300042774
+1400593419
+1200274027
```

2. **Assmbly Dili (Orta Seviye)**

   Anlaşılması biraz daha kolay, ingilizce kısaltmalardan oluşur, makine bağımlıdır

   Assembler derleyicisi ile makine koduna dönüştürülür.

```
LOAD sayi1
ADD  sayi2
STORE toplam
```

3. **Yüksek Seviyeli Diller (High Level Language)**

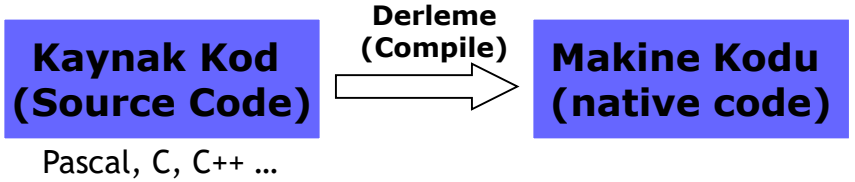
   Anlaşılması çok daha kolay, komutlar ingilizce ve matematiksel ifadelerden oluşur,

   Derleyici ile makine koduna dönüştürülür.

```
toplam = sayi1 + sayi 2
```

Programların Çalıştırılması

- ✓ Derleme (Compiled Language)



Pascal, C, C++ …

- ✓ Yorumlama (Interpreted Language)

  Komutlar teker teker çalıştırılır
  Komut okuma ve çevirme işlemi çalışma zamanında yapılır (düşük hız)
  Hata düzeltme daha kolaydır.
  Derleyiciden kaynaklanan sınırlamalar kalktığı için daha esnek bir çalışma ortamı.

  Perl, tcl, basic, matlab…

- ✓ Karma (Hybrid)

  Kodlar derlenerek byte code oluşturulur.
  Byte code yorumlanarak çalıştırılır.

  Java, phyton

Programlama Teknikleri

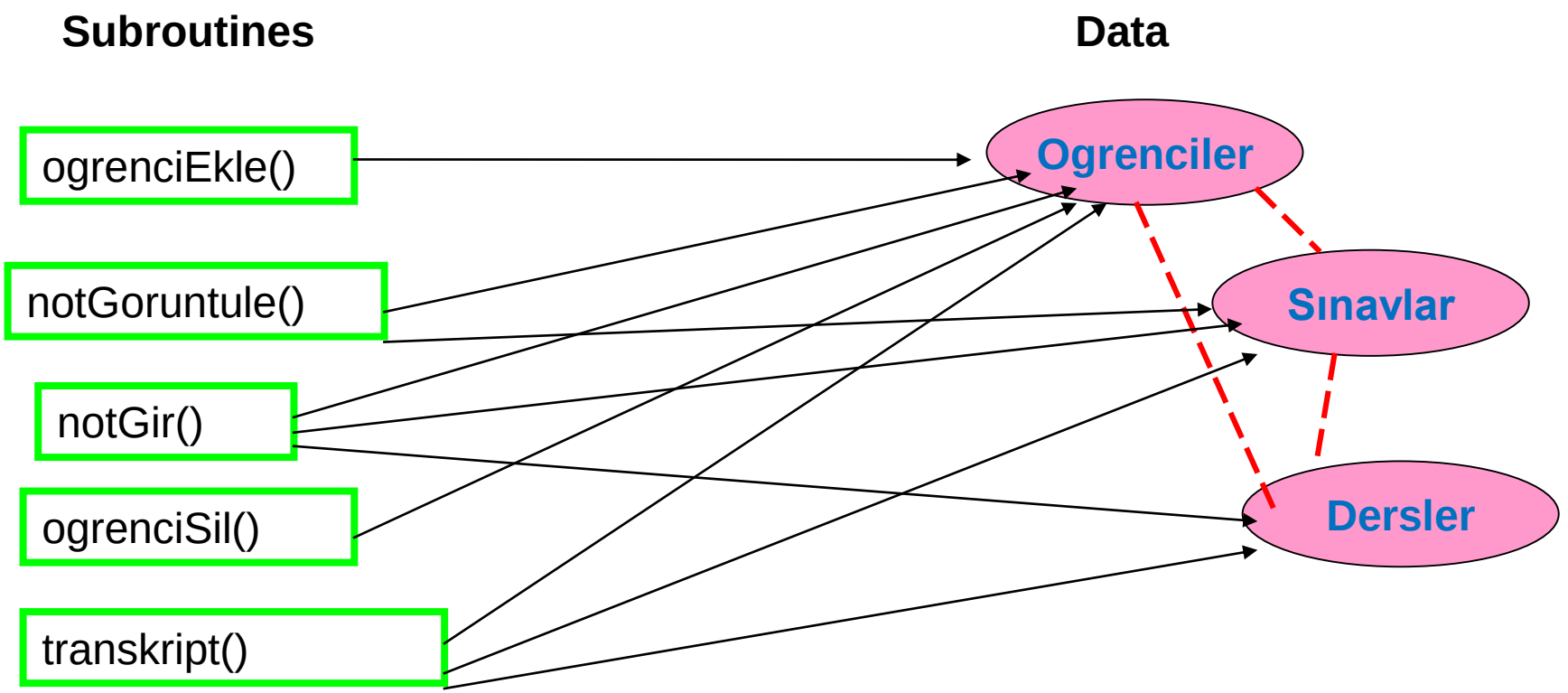
- ✓ Yapısal (modüler) programlama tekniği (Structured Programming)
- ✓ Nesne Yönelimli programlama tekniği (Object Oriented Programming)

Yapısal Programlama

•Yapısal teknikte programcı doğrudan probleme odaklanır ve problemin çözümüne ilişkin fonksiyonları/yöntemleri geliştirir. Ana fonksiyon/yöntem (main) programı başlatır ve her biri özel görevleri yerine getiren yöntemler çağrılarak yapılması gereken işlem gerçekleştirilir.

•Yöntemlerin ihtiyaç duyduğu veriler genellikle veritabanlarında ya da (her taraftan erişilebilen) değişkenlerde saklanır.

•Öğrenci, açılan ders, not girişleri v.b. İşlemler gerçekleştiren bir bilgi sistemi düşünüldüğünde;

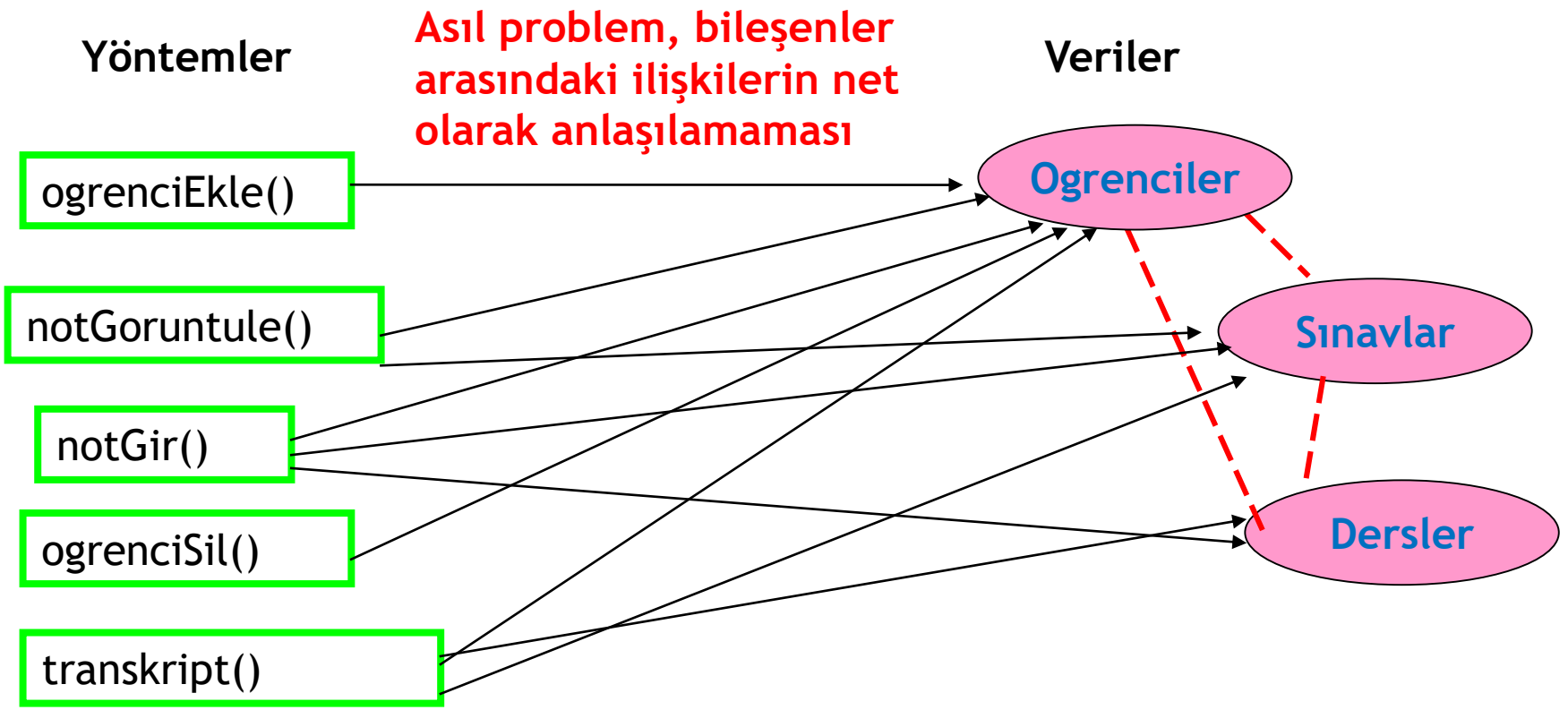


Sistem büyüdükçe ilişkiler ve bağımlılık daha da karmaşıklaşır.

Hata bulma zorlaşır

Program içerisinde değiştirme, ekleme, çıkarma vs. yapmak zorlaşır ve beklenmeyen etkilere neden olabilir…

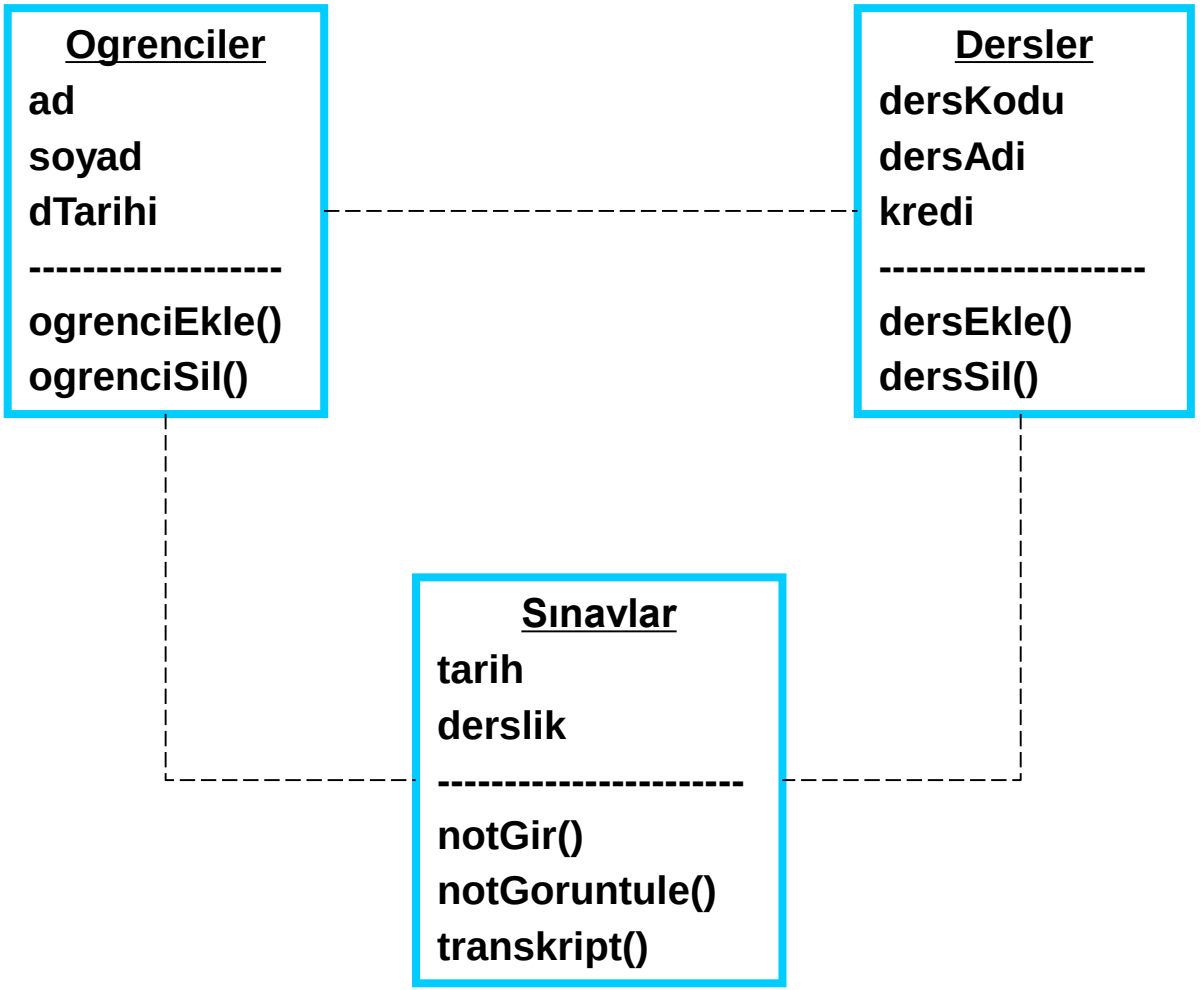
Örneğin;
Ogrenciler tablosunda ogrencinin dogum tarihi iki haneli
Bu alanı 4 haneli yapmak istiyoruz…
Ogrenciler tablosu Sınavlar ve dersler tablolarıyla ilişkili olduğundan beklenmeyen bir sorun çıkabilir…
Tüm yöntemler Ogrenciler tablosunu bir şekilde kullanıyor. Buyöntemlerde de hata olma ihtimali var..
ogrenciEkle() yontemi kesinlikle sorun cikaracaktır…
milenyum problemi…



Nesne Yönelimli Programlama

Yapısal teknikte programcı doğrudan probleme odaklanır ve problemin çözümüne ilişkin yöntemleri geliştirir. Nesenye yönelimli programlama tekniğinde ise temel bileşen nesnedir ve programlar nesnelerin birlikte çalışmasından meydana gelir. Nesne hem veriyi hem de bu veriyi işleyen fonksiyonları içerir. Programcılar dikkatlerini nesneleri oluşturan sınıfları geliştirmeye yoğunlaştırır.

Yapısal teknikte bir fonksiyon herhangi bir görevi yerine getirmek için veriye ihtiyaç duyarsa, gerekli veri parametre olarak gönderilir. NYP de ise yerine getirilmesi gereken görev nesne tarafından icra edilir ve fonksiyonlar verilere parametre gönderimi yapılmaksızın erişebilirler.



Nesne Yönelimli Programlamanın Özellikleri/Avantajları

Encapsulation, data abstraction, inheritance, and polymorphism.

- ✓ Problemler daha kolay tanımlanıp çözülebilir. Gerçek dünya düşünülerek geliştirilmiştir. Geliştirme süreci daha kolay olur.

- ✓ Bilgi Gizleme (Information Hiding, Data abstraction)
  - ✓ Nesne içerisindeki işlemler (nasıl) diğer nesnelerden soyutlanır-sadece ne yapacağını bilirler. Nesne içerisindeki değişiklik diğer nesneleri etkilemez-regression fault. Dolayısıyla bakım aşaması daha kolay olur.
  - ✓ Gereksiz ayrıntılarla uğraşılmaz, probleme odaklanılır (arabanın gitmesi icin gaza basmak yeterli)- Daha hızlı geliştirme süreci.

- ✓ Modüler Programlama (Moduler Programming)
  - ✓ Nesneler birbirinden tamamen bağımsız (veri + fonksiyon) (encapsulation, responsibility driven design)
  - ✓ Büyük ve karmaşık bir problem küçük parçalara ayrılarak daha kolay çözülebilir. Geliştirme ve bakım daha kolay olur.
  - ✓ Programların geliştirilmesi daha hızlı, geliştirilen bir nesne diğer programlarda da rahatlıkla kullanılabilir. (. Hata bulma-bakım daha kolay) Bisikleti başkasına verdiğimiz zaman da çalışır. string nesneleri her programda kullanılır.
  - ✓ Geliştirme sürecinde grup çalışmalarına olanak sağlanır.

Nesne Yönelimli Programlamanın Özellikleri/Avantajları

✓Kodların Tekrar Kullanımı (Code Reuse)
- ✓ Nesneler başka programlara kolaylıkla aktarılabilir. Bakım ve geliştirme zamanı/maliyeti düşer
- ✓ Kalıtım, Çok şekillilik

✓Hata Bulma – Bakım/Onarım (Maintainence)
- ✓ Bileşenler arasındaki ilişkiler açık olduğundan (veri+fonksiyon aynı yapı içerisinde) güncelleme, hata bulma ve bakım daha kolay

✓Design Patterns

✓Günümüzdeki en iyi yaklaşım. Gelecekte ?

Kaynaklar