YMH311-Yazılım Tasarım Ve Mimarisi
Gerçekleştirme/Kodlama

Prof. Dr. Resul DAŞ

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bu Haftaki Konular

- Yazılım Geliştirme Ortamları.....6
- Veri Modelleri.....17
- Kodlama Stili 26
- Program Karmaşıklığı.....33
- Olağan Dışı Durum Çözümleme.....37
- Kod Gözden Geçirme.....38

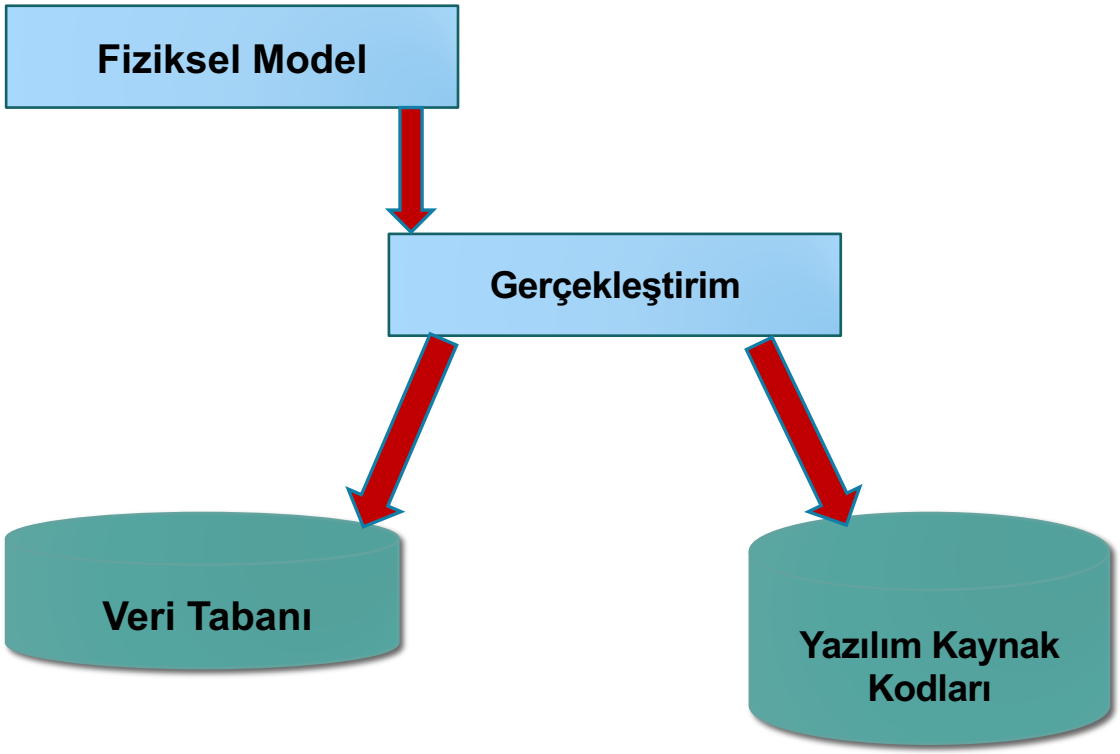
Amaçlar

- Yazılım Gerçekleştirimini Kavramak
- Yazılım Geliştirme Ortamları Hakkında Bilgilenmek
- Veri Tabanı Sistemlerini Anlamak
- Veri Modelleme Yöntemlerini Öğrenmek
- Kodlama Stilleri Hakkında Bilgi Edinmek
- Yapısal Programlama Yapılarını Öğrenmek
- Program Karmaşıklığı Hesaplamasında Kullanılan Yöntemler

Giriş

- Tasarım sonucu üretilen süreç ve veri tabanının fiziksel yapısını içeren fiziksel modelin bilgisayar ortamında çalışan yazılım biçimine dönüştürülmesi çalışmasıdır.
- Her şeyden önce bir yazılım geliştirme ortamı seçilmelidir (programlama dili, veri tabanı yönetim sistemi, yazılım geliştirme araçları (CASE)).
- Kaynak kodların belirli bir standartta üretilmesi düzeltme için faydalıdır.

Gerçekleştirim Çalışması



Yazılım Geliştirme Ortamları

- Yazılım geliştirme ortamı, tasarım sonunda üretilen fiziksel modelin, bilgisayar ortamında çalıştırılabilmesi için gerekli olan:

Programlama Dili

Veri Tabanı Yönetim Sistemi

Hazır Program Kitapçıkları

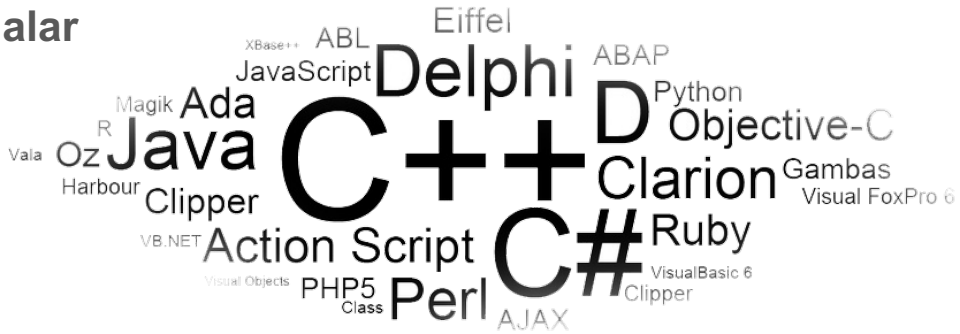
CASE Araçları

bileşenlerinden oluşur.

- Günümüzde söz konusu bileşenler oldukça farklılık ve çeşitlilik göstermekte ve teknolojinin değişimine uygun olarak gelişmektedir. Bu bileşenler aşağıda açıklanmaktadır.

Programlama Dilleri

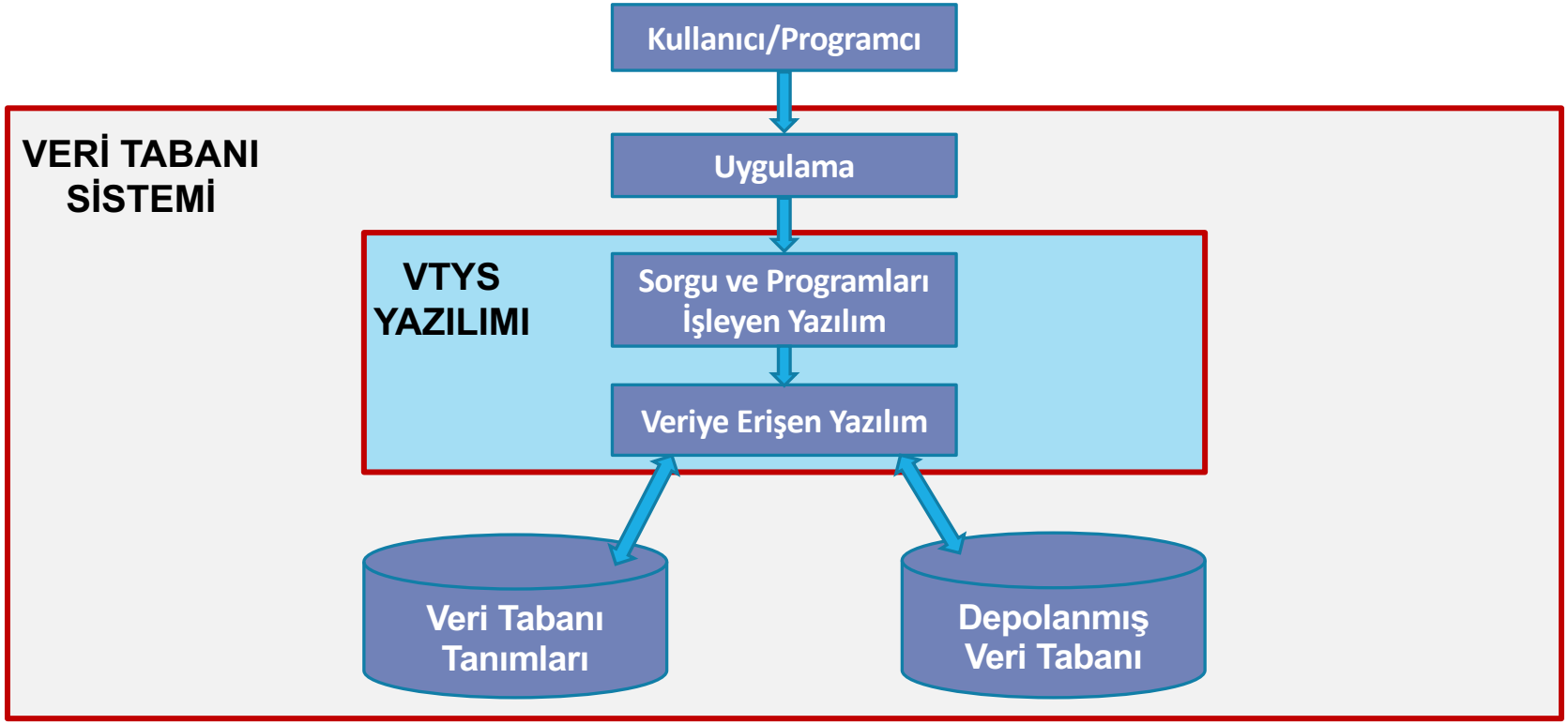
- Geliştirilecek Uygulamaya Göre Programlama Dilleri seçilmelidir.
  - **Veri İşleme Yoğunluklu Uygulamalar**
  - Cobol,
  - Görsel Programlama Dilleri ve Veri Tabanları
  - **Hesaplama Yoğun Uygulamalar**
  - Fortran
  - C
  - Paralel Fortran ve C
  - **Süreç ağırlıklı uygulamalar**
  - Assembly
  - C
  - **Sistem programlamaya yönelik uygulamalar**
  - C
  - **Yapay Zeka Uygulamaları**
  - Lisp
  - Prolog



Veri Tabanı Yönetim Sistemleri

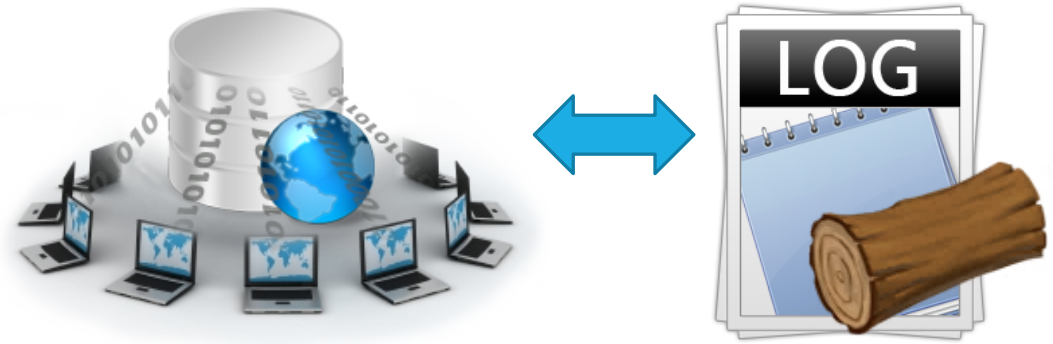
- Birbiri ile ilişkili veriler topluluğu veri tabanı olarak tanımlanmaktadır. Veri tabanı herhangi bir boyutta ya da karmaşıklıkta olabilir.
- Kişisel telefon rehberinizdeki adres bilgileri bir veri tabanı örneği oluşturduğu gibi, maliye bakanlığı bünyesinde saklanan ve vergi ödemesi gereken tüm kişilerin bilgilerinin saklandığı uygulama da bir başka veri tabanı örneğidir.
- Veri tabanını oluşturan veriler birbiriyle ilişkili verilerdir. Bir veritabanında veriler arası ilişkiler ile veri değerleri bulunur.
- Kullanıcıların veritabanındaki verileri soruşturmasını, veritabanına yeni veriler eklemesini, varolan verilerde değişiklik yapmasını sağlayan yazılım Veri Tabanı Yönetim Sistemi (VTYS) olarak tanımlanır. VTYS, genel amaçlı bir yazılımdır.
- Bütün VTYS yazılımları (Oracle, Informix, Sybase vb), kullanıcıya, veri tabanı yapısı tanımlama, veri tabanını sorgulama, değiştirme ve raporlama olanakları verirler.

Veri Tabanı Sistemleri



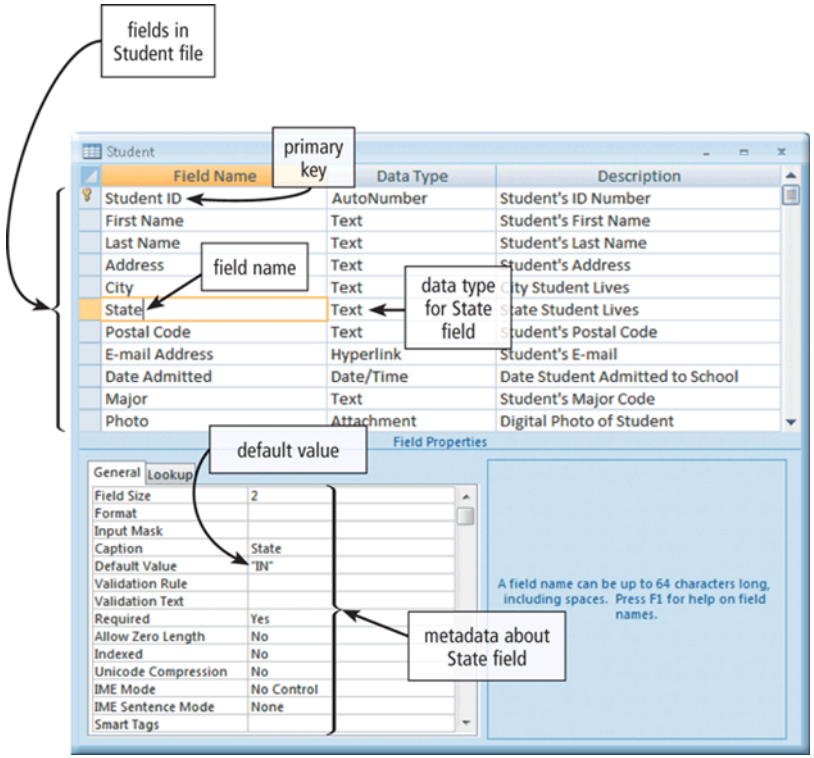
Veri tabanı Yaklaşımının Geleneksel Kütük Kavramından Ayıran Özellikler

1. Veri Sözlüğü
2. Veri Soyutlama
3. Program-Veri ve Program-İşlem Bağımsızlığı
4. Birden Çok Kullanıcı Desteği
5. Verinin Birden Fazla İşlem Arasında Paylaşımı



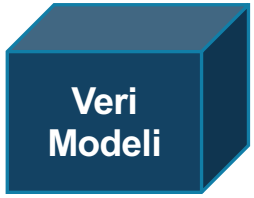
Veri Sözlüğü

- Uygulama yazılımında kullanılan tüm veri tanımlarının yapısal ve ayrık bir biçimde saklanmasını sağlayan bir katalog bilgisi veya veri sözlüğünün varlığıdır.



Veri Soyutlama

- Veri tabanı yaklaşımının bir temel karakteristiği kullanıcıdan verinin saklanması konusundaki detayları gizleyerek veri soyutlamasını sağlamasıdır. Bu soyutlamayı sağlayan ana araç veri modelidir. Veri modeli şu şekilde tanımlanabilir.



Bir veri modeli bir veri tabanının yapısını açıklamakta kullanılan kavramlar setidir.

- Bir veri tabanının yapısı ile veri tipleri, ilişkiler ve sınırlamalar kastedilmektedir.
- Pek çok veri modeli ayrıca veri tabanı üzerinde belirtilen getiri (çağrı) ve güncellemeler için temel işlemler setini içerir.
- Davranışı belirlemek üzere veri modelinin içine kavramları da eklemek mümkündür.
- Bu da, temel işlemlere ek olarak kullanıcı tarafından tanımlanmış işlemlerin de veri modeline eklenmesiyle mümkündür.
- Bir veri modelindeki genel işlemlere örnek olarak Ekle, Sil, Değiştir, Eriş verilebilir

Program-Veri ve Program-İşlem Bağımsızlığı

- Geleneksel dosya işleme yönteminde, veri dosyalarının yapısı bu dosyalara erişim programları içine gömülmüştür.
- Bundan dolayı da dosyanın yapısındaki herhangi bir değişiklik bu dosyaya erişen tüm programlarda değişiklik yapılmasını gerektirir.
- Tersine VTYS erişim programları veri dosyalarından bağımsız olarak tasarlanmışlardır. VTYS de saklanan Veri dosyalarının yapısı erişim programlarından ayrı olarak katalogda tutulduğundan program ve veri bağımsızlığı sağlanır.
- Örneğin bir veri dosyasına yeni bir alan eklenmek istendiğinde, bu veri dosyasını kullanan tüm programlara bu alanın eklenmesi gerekir. VTYS yaklaşımında ise sadece katalogdaki veritabanı tanımlarına bu alan eklenerek sorun çözülebilir.
- Nesne-kökenli veritabanları ve programlama dillerindeki son gelişmeler kullanıcının veri üzerindeki işlemleri veritabanı tanımının bir parçası olarak tanımlamasına olanak tanımaktadır.

Birden Çok Kullanıcı Desteği

- Bir veritabanının birçok kullanıcısı vardır ve bunların herbiri veritabanının farklı bir görüntüsüne gereksinim duyabilir.
- Bir görüntü veritabanının alt kümesi olabilir veya veritabanından elde edilmiş fakat kesin olarak saklanmamış sanal veri içerebilir.
- Bazı kullanıcılar, kullandıkları verilerinin veritabanından türetilmiş veriler mi? Yoksa veritabanında gerçekte saklanan veriler mi olup olmadığı ile ilgilenmezler.
- Kullanıcıları farklı uygulamalara sahip çok kullanıcılı bir VTYS çoklu görüntü tanımlama olanaklarını sağlamalıdır.

Verinin Birden Fazla İşlem Arasında Paylaşımı

- Çok kullanıcılı VTYS yazılımlarının temel rolü eş zamanlı işlemlerin karışıklık olmadan doğru bir şekilde yapılmasına olanak tanımak işlemlerin doğru olarak yapılmasını garanti etmektir.

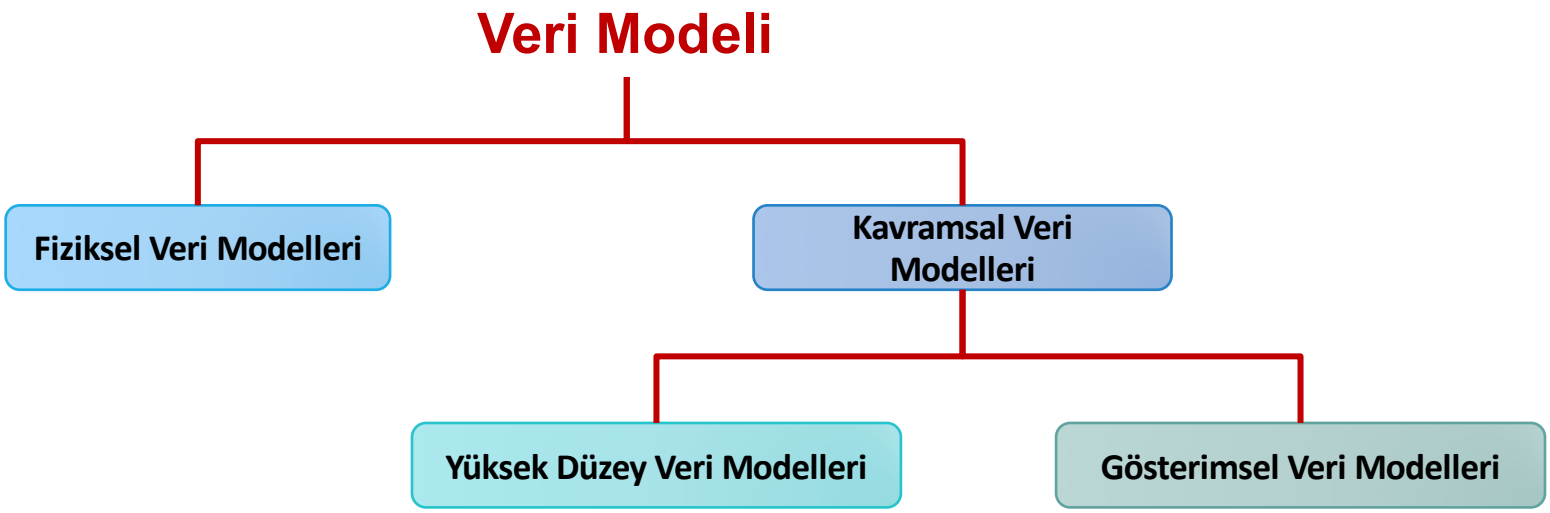
- Bu özellik veri tabanı (VT) kavramını, dosya işleme kavramından ayıran en önemli özelliktir.

- Bu kontrol aynı anda birden çok kullanıcının aynı veriyi güncellemeye çalışmasını, güncellemenin doğru olması açısından garanti eder.

VTYS Kullanımının Ek Yararları

1. Genişleme Potansiyeli,
2. Esneklik
3. Uygun geliştirme zamanının azalması,
4. Güncel bilgilerin tüm kullanıcılara aynı zamanda ulaşması,
5. Ölçümde ekonomi,
6. İşletme ortamındaki ortak verilerin tekrarının önlenmesi verilerin merkezi denetimin ve tutarlılığın sağlanması,
7. Fiziksel yapı ve erişim yönetimi karmaşıklıklarının her kullanıcıya yalnız ilgilendiği verilerin kolay anlaşılır yapılarda sunulması,
8. Uygulama yazılımı geliştirmenin kolaylaşması,
9. Fiziksel yapı ve erişim yöntemi karmaşıklıklarının her kullanıcıya yalnız ilgilendiği verilerin kolay anlaşılır yapıda sunulması.

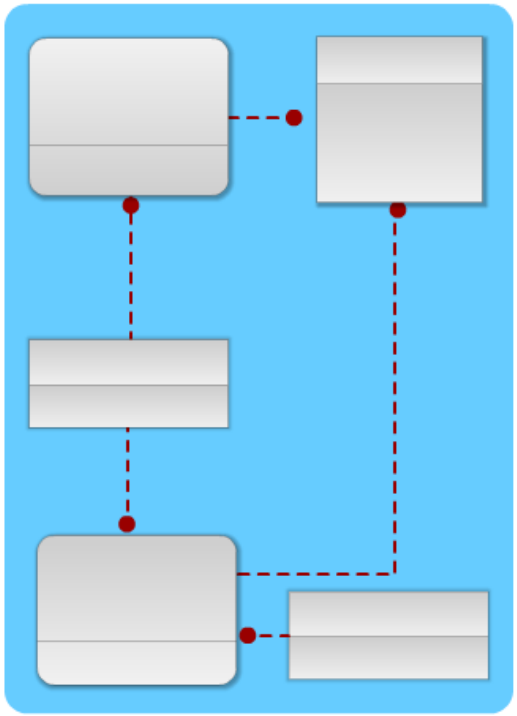
Veri Modelleri



Veri Modelleri

- Fiziksel veri modelleri verinin bilgisayarda nasıl saklandığının detayları ile ilgili kavramları sağlar.
- Kavramsal veri modelleri hem kullanıcı tarafından anlaşılan hem de verinin bilgisayar içerisindeki gösteriminden çok da fazla uzak olmayan kavramları sağlar.
  - Yüksek Düzey Veri Modelleri
    - Varlık: Veri tabanında saklanan, gerçek dünyadan bir nesne veya kavramdır (proje, işçi).
    - Özellik: Varlığı anlatan bir özelliği gösterir (işçinin adı, ücreti).
    - İlişki: Birden fazla varlık arasındaki ilişkidir (işçi ve proje arasındaki çalışma ilişkisi).
  - Gösterimsel Veri Modelleri, ticari veri tabanlarında sıklıkla kullanılır ve belli başlı veri üç modeli içerir. İlişkisel, ağ ve hiyerarşik veri modeli. Nesne yönelimli veri modelleri daha yüksek seviyeli gerçekleştirim veri modelleridir ve kavram veri modeline daha yakındır.

Şemalar



- Herhangi bir veri modelinde veri tabanının tanımlaması ile kendisini ayırmak önemlidir.
- Veri tabanının tanımlanması, veri tabanı şeması veya meta-veri olarak adlandırılır.
- Veri tabanı şeması tasarım sırasında belirlenir ve fazla değişmesi beklenmez.

VTYS Mimarisi

- Bu mimarinin amacı kullanıcı uygulamaları ile fiziksel veri tabanını birbirinden ayırmaktır. Bu mimaride şemalar 3 seviyede tanımlanabilir. Bu nedenle üç şema mimarisi olarak da anılır.

İçsel Düzey

- Veri tabanının fiziksel saklama yapısını açıklar.

Kavramsal Düzey

- Kavramsal şema içerir ve kullanıcılar için veri tabanının yapısını açıklar.

Dışsal Düzey

- Dış şemalar ve kullanıcı görüşlerini içerir.

Veritabanı Dilleri ve Arabirimleri

- Veri tabanı tasarımı tamamlandıktan sonra bir VTYS seçilir. Seçilen VTYS'de bulunan dil olanakları aşağıda verilmiştir.

Veri Tanımlama Dili (VTD)	Kavramsal şemaları tanımlamak üzere kullanılır.
Saklama Tanımlama Dili (STD)	İçsel şemayı tanımlamak üzere kullanılır.
Görüş Tanımlama Dili (GTD)	Dışsal şemayı tanımlamak için kullanılır.
Veri İşleme Dili (VİD)	Veri tabanı oluşturulduktan sonra veri eklemek, değiştirmek ve silmek için kullanılır.

Veritabanı Dilleri ve Arabirimleri

- İki tip VİD vardır; yüksek düzeyli ve düşük düzeyli.
  - Yüksek düzeyli VİD, bir bilgisayar terminalinden etkileşimli olarak kullanılabildiği gibi bir programlama dili içerisine de yerleştirilebilir.
  - Düşük düzeyli VİD‘ de ise VİD bir programlama dili içerisine gömülü olarak çalışır.
- VTYS Arabirimleri olarak Menü-Tabanlı, grafiksel, form tabanlı ve doğal dil arabirimleri kullanılmaktadır.
- Menü tabanlı arabirimlerde kullanıcıya çeşitli seçenekler sunulurken, grafiksel arabirimde kullanıcıya veritabanı şeması diyagramı halinde sunulur.
- Kullanıcı bu diyagram yardımıyla sorgu belirtebilir.
- Form tabanlı arabirimler ise kullanıcıya doldurulmak üzere bir form sunarlar. Doğal dil arabirimleri İngilizce yazılan sorguları kabul eder ve anlamaya çalışırlar.

VTYS nin Sınıflandırılması

Sınıflandırmalar kullanılan Veri Modeline göre yapılır.

- İlişkisel Model: Veri tabanı tablo yığınından oluşmuştur. Her bir tablo bir dosya olarak saklanabilir.
- Ağ Modeli: Veriyi kayıt ve küme tipleri olarak gösterir.
- Hiyerarşik Model: Veri ağaç yapısında gösterilir.
- Nesne Yönelimli Model: Veri tabanı nesneler, özellikleri ve işlemleri biçiminde gösterilir.

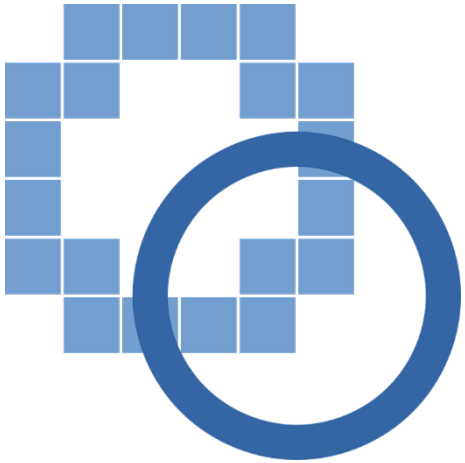
Hazır Program Kütüphaneleri

- Hemen hemen tüm programlama platformlarının kendilerine özgü hazır kütüphaneleri bulunmaktadır.
  - Pascal *.tpu
  - C *.h
  - Java *.jar

- Günümüzde bu kütüphanelerin temin edilmesi internet üzerinden oldukça kolaydır.

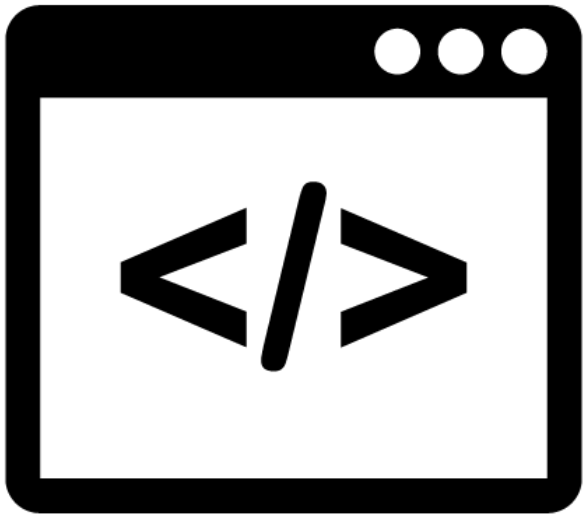
CASE Araç ve Ortamları

- Günümüzde bilgisayar destekli yazılım geliştirme ortamları oldukça gelişmiştir.
- CASE araçları yazılım geliştirme sürecinin her aşamasında üretilen bilgi ya da belgelerin bilgisayar ortamında saklanmasına ve bu yolla kolay erişilebilir ve yönetilebilir olmasına olanak sağlar.



Kodlama Stili

- Hangi platformda geliştirilirse geliştirilsin yazılımın belirli bir düzende kodlanması, yazılımın yaşam döngüsü açısından önem kazanmaktadır.
- Etkin kod yazılım stili için kullanılan yöntemler:
  - Açıklama Satırları
  - Kod Yazım Düzeni
  - Anlamlı İsimlendirme
  - Yapısal Programlama Yapıları



Açıklama Satırları

İYİ:

```
// Satış fiyatı hesaplaması 25.10.2011 İrfan Mevsim
//
// Fonksiyon birim fiyatı ve KDV orani olarak gönderilen
// bilgilerin satış fiyatını geri döndürür.
//
// Örnek Kullanım
// Dim SatisFiyati as Double = SatisHesapla(1000,18)
//
// Dönüş Değeri --> 1018.0 (double)
//
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
      return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

~~KÖTÜ:~~

```
// Satış fiyatı hesaplaması  (yetersiz açıklama)
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
      return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

~~KÖTÜ:~~

```
private double SatisHesapla(int birimFiyati, int kdvOrani)
{
// Dönüş Değeri --> 1018.0 (yanlış ve yetersiz açıklama)
      return birimFiyati + (birimFiyati * kdvOrani) / 100;
}
```

- Modülün başlangıcına
  - genel amaç,
  - işlevi
  - desteklenen platformlar,
  - eksikler,
  - düzeltilen yanlışlıklar

  gibi genel bilgilendirici açıklamaları yapılır.
- Aynı zamanda modülün kritik bölümleri vurgulanarak açıklanır.

Kod Biçimlemesi

İYİ:

```
string[] test = { "bir", "iki", "üç" };
for ( int i = 0; i < test.length; i++ )
{
      MessageBox.Show(test[i]);
}
```

~~KÖTÜ:~~

```
int i;
string[] test = { "bir", "iki", "üç" };
for(i;i<test.length;i++)
      MessageBox.Show(test[i]);
```

İYİ:

```
string test = "bir,iki,uc";
foreach ( string item in test.Split(',') )
{
      MessageBox.Show(item);
}
```

~~KÖTÜ:~~

```
string test = "bir,iki,uc";
string item;
foreach (item in test.split(','))
            MessageBox.Show(item);
```

- Programın okunabilirliğini artırmak ve anlaşılabilirliğini kolaylaştırmak amacıyla açıklama satırlarının kullanımının yanı sıra, belirli bir kod yazım düzeninin de kullanılması gerekmektedir.

Anlamlı İsimlendirme

İyi:

```
KodlamaStandartlari.csproj
namespace KodlamaStandartlari

KodlamaStandartlari.Models.dll
namespace KodlamaStandartlari.Models

KodlamaStandartlari.csproj projesinde Models dizini
namespace KodlamaStandartlari.Models
```

~~KÖTÜ:~~

```
KodlamaStandartlari.csproj
namespace Kodlama

KodlamaStandartlari.Models.dll
namespace KodlamaStandartlari.Modeller

KodlamaStandartlari.csproj projesinde Models dizini
namespace KodlamaStandartlari.Modeller
```

“İsim Alanı” İsimlendirme

- Kullanılan tanımlayıcıların (değişken adları, dosya adları, veri tabanı tablo adları, fonksiyon adları, yordam adları gibi) anlamlı olarak isimlendirilmeleri anlaşılabilirliği büyük ölçüde etkilemektedir.

Yapısal Programlama Yapıları

- Program kodlarının, okunabilirlik, anlaşılabilirlik, bakım kolaylığı gibi kalite etmenlerinin sağlanması ve program karmaşıklığının azaltılması amacıyla "yapısal programlama yapıları" kullanılarak yazılması önemlidir.

- Yapısal Programlama Yapıları, temelde, içinde "go to" deyimi bulunmayan, "tek giriş ve tek çıkışlı" öbeklerden oluşan yapılardır.

- Teorik olarak herhangi bir bilgisayar programının, yalnızca Yapısal Programlama Yapıları kullanılarak yazılabileceği kanıtlanmıştır.

- Üç temel Yapısal Programlama Yapısı bulunmaktadır:
  - Ardışıl işlem yapıları
  - Koşullu işlem yapıları
  - Döngü yapıları

Yapısal Programlama Yapıları

Ardışıl İşlem Yapıları

- Ardışıl işlemler, herhangi bir koşula bağlı olmaksızın birbiri ardına uygulanması gereken işlemler olarak tanımlanır. Hemen her tür programlama dilinde bulunan, aritmetik işlem deyimleri, okuma/yazma deyimleri bu tür yapılara örnek olarak verilebilir.

Koşullu İşlem Yapıları

- Üç tür Koşullu işlem yapısı bulunmaktadır: tek koşullu işlem yapısı (if-then), iki koşullu işlem yapısı (if-then-else) ve çok koşullu işlem yapısı (case-when). 70'li yılların ortalarından sonra gelen programlama dillerinin hemen hepsinde, bu yapılar doğrudan desteklenmektedir.

Yapısal Programlama Yapıları

Döngü Yapıları

- Döngü yapıları, belirli bir koşula bağlı olarak ya da belirli sayıda, bir ya da dah çok kez yinelenecek işlemler için kullanılan yapılardır. Temelde üç tür döngü yapısı bulunmaktadır. Bunlar:

1. Belirli sayıda yinelenecek işlemler için kullanılan yapılar (for yapısı),
2. Bir koşula bağlı olarak, sıfır yada birden çok kez yinelenecek işlemler için kullanılan yapılar (while-end yapısı),
3. Bir koşula bağlı olarak, bir ya da daha çok kez yinelenecek işlemler için kullanılan yapılar (repeat-until yapısı).

Bu yapıların her biri ‘tek girişli ve tek çıkışlı’ yapılardır.

Program Karmaşıklığı

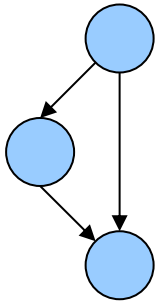
- Program karmaşıklığı konusunda çeşitli modeller geliştirilmiştir.
- Bunların en eskisi ve yol göstericisi McCabe tarafından 1976 yılında geliştirilen modeldir.
- Bunun için önce programın akış diyagramına dönüştürülmesi, sonra da karmaşıklık ölçütünün hesaplanması gerekmektedir.
- McCabe ölçütü, bir programda kullanılan "koşul" deyimlerinin program karmaşıklığını etkileyen en önemli unsur olduğu esasına dayanır ve iki aşamada uygulanır:
  1. Programın Çizge Biçimine Dönüştürülmesi
  2. Mc Cabe Karmaşıklık Ölçütünün Hesaplanması

Programın Çizge Biçimine Dönüştürülmesi-1

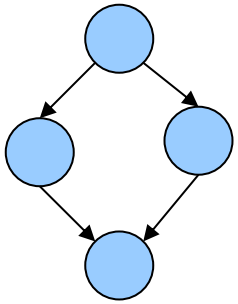
Sıradan İşlemler:

Bir ya da birden fazla ardışık işlem

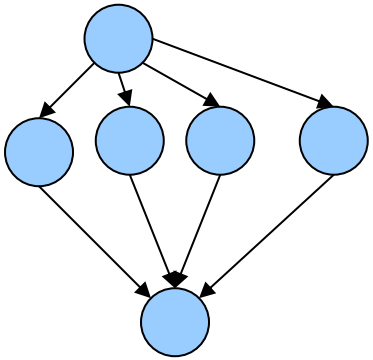
Koşullu İşlemler:



If-Then işlemi



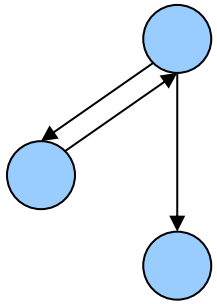
If-Then-Else işlemi



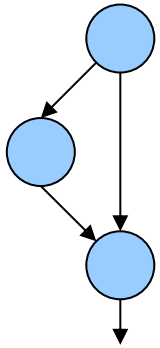
Case işlemi

Programın Çizge Biçimine Dönüştürülmesi-2

Döngü İşlemleri:



While Döngüsü



Repeat Döngüsü

McCabe Karmaşıklık Ölçütü

$$V(G)= k – d + 2p$$

K: Diyagramdaki kenar çizgi sayısı

D: Diyagramdaki düğüm sayısı

P: programdaki bileşen sayısı (ana program ve ilgili alt program sayısını göstermektedir. Alt program kullanılmadı ise p=1, 3 alt program kullanıldı ise p=4 tür)

Olağandışı Durum Çözümleme

- Olağandışı durum: Bir programın çalışmasının, geçersiz ya da yanlı veri oluşumu ya da başka nedenlerle istenmeyen bir biçimde sonlanmasına neden olan durumdur.
- Genelde kabul edilen; program işletiminin sonlandırılmasının bütünüyle program denetiminde olmasıdır.

Kod Gözden Geçirme

- Bir gazetede hiç bir yazı editörün onayı alınmadan basılamayacağı gibi, kod gözden geçirme olmadan da yazılım sistemi geliştirilemez.
- Kod gözden geçirme ile program sınama işlemleri birbirlerinden farklıdır.
- Kod gözden geçirme, programın kaynak kodu üzerinde yapılan bir işlemdir ve bu işlemlerde program hatalarının %3-5'lik bir kısımı yakalanabilmektedir.

Gözden Geçirme Sürecinin Düzenlenmesi

- Gözden geçirme sürecinin temel özellikleri;
  - Hataların bulunması, ancak düzeltilmemesi hedeflenir,
  - Olabildiğince küçük bir grup tarafından yapılmalıdır. En iyi durum deneyimli bir inceleyici kullanılmasıdır. Birden fazla kişi gerektiğinde, bu kişilerin, ileride program bakımı yapacak ekipten seçilmesinde yarar vardır.
  - Kalite çalışmalarının bir parçası olarak ele alınmalı ve sonuçlar düzenli ve belirlenen bir biçimde saklanmalıdır.

biçiminde özetlenebilir. Burada yanıtı aranan temel soru, programın yazıldığı gibi çalışıp çalışmayacağının belirlenmesidir. Gözden Geçirme çalışmasının olası çıktıları:

- ? Programı olduğu gibi kabul etmek
- ? Programı bazı değişikliklerle kabul etmek
- ? Programı, önerilen değişikliklerin yapılmasından sonra tekrar gözden geçirmek üzere geri çevirmek.

Gözden Geçirme Sırasında Kullanılacak Sorular

- Bir program incelenirken, programın her bir öbeği (yordam ya da işlev) aşağıdaki soruların yanıtları aranır.
- Bu sorulara ek sorular eklenebilir.
- Bazı soruların yanıtlarının "hayır" olması programın reddedileceği anlamına gelmemelidir.

Öbek Arayüzü

1. Her öbek tek bir işlevsel amacı yerine getiriyor mu?
2. Öbek adı, işlevini açıklayacak biçimde anlamlı olarak verilmiş mi?
3. Öbek tek giriş ve tek çıkışlı mı?
4. Öbek eğer bir işlev ise, parametrelerinin değerini değiştiriyor mu?

Giriş Açıklamaları

1. Öbek, doğru biçimde giriş açıklama satırları içeriyor mu?
2. Giriş açıklama satırları, öbeğin amacını açıklıyor mu?
3. Giriş açıklama satırları, parametreleri, küresel değişkenleri içeren girdileri ve kütükleri tanıtıyor mu?
4. Giriş açıklama satırları, çıktıları (parametre, kütük vb) ve hata iletilerini tanımlıyor mu?
5. 5. Giriş açıklama satırları, öbeğin algoritma tanımını içeriyor mu?
6. 6. Giriş açıklama satırları, öbekte yapılan değişikliklere ilişkin tanımlamaları içeriyor mu?
7. 7. Giriş açıklama satırları, öbekteki olağan dışı durumları tanımlıyor mu?
8. 8. Giriş açıklama satırları, Öbeği yazan kişi ve yazıldığı tarih ile ilgili bilgileri içeriyor mu?
9. 9. Her paragrafı açıklayan kısa açıklamalar var mı?

Veri Kullanımı

1. İşlevsel olarak ilintili bulunan veri elemanları uygun bir mantıksal veri yapısı içinde gruplanmış mı?
2. Değişken adları,işlevlerini yansıtacak biçimde anlamlı mı?
3. Değişkenlerin kullanımları arasındaki uzaklık anlamlı mı?
4. Her değişken tek bir amaçla mı kullanılıyor?
5. Dizin değişkenleri kullanıldıkları dizinin sınırları içerisinde mi tanımlanmış?
6. Tanımlanan her gösterge değişkeni için bellek ataması yapılmış mı?

Sunuş

1. Her satır, en fazla bir deyim içeriyor mu?
2. Bir deyimin birden fazla satıra taşması durumunda, bölünme anlaşılabilirliği kolaylaştıracak biçimde anlamlı mı?
3. Koşullu deyimlerde kullanılan mantıksal işlemler yalın mı?
4. 4Bütün deyimlerde, karmaşıklığı azaltacak şekilde parantezler kullanılmış mı?
5. 5. Bütün deyimler, belirlenen program stiline uygun olarak yazılmış mı?
6. 6. Öbek yapısı içerisinde akıllı "programlama hileleri" kullanılmış mı?

Sorular

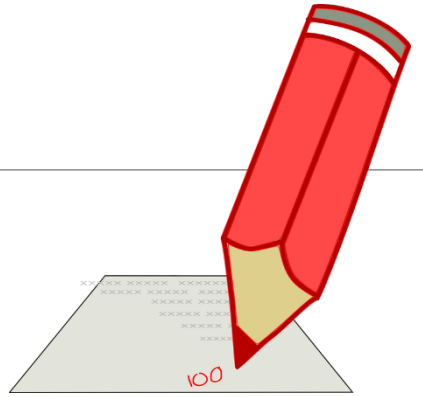
1. Kendi yazılım geliştirme ortamınızı açıklayınız.
2. Veri tabanı ile veri tabanı yönetim sistemi arasındaki farkı belirtiniz.
3. Veri tabanı Yönetim Sistemi kullanarak, uygulama geliştirme zamanının hasıl kısalacağını açıklayınız.
4. Veri tabanı Yönetim Sistemi kullanımının yararlarını ve aksak yönlerini belirtiniz?
5. Kullandığınız bir veri tabanı yönetim sistemini inceleyiniz. VTD, STD, GİD, GTD dillerinin özelliklerini araştırınız.
6. Kendi kullanımınız için kodlama stili geliştiriniz.
7. Kodlama stilleri ile programlama dilleri arasındaki ilişkiyi belirtiniz.
8. Bildiğiniz bir programlama dilinin, yapısal programlama yapılarından hangilerini doğrudan desteklediğini, hangilerini desteklemediğini araştırınız. Desteklenmeyen yapıların, bu programlama dilinde nasıl gerçekleştirilebileceğini belirtiniz.
9. Verilen bir N doğal sayısının asal olup olmadığını belirleyen bir programı dört değişik biçimde yazınız. Programlar arasındaki zaman farklılıklarını ölçünüz.
10. Program geliştirirken kullandığınız olağan dışı durum çözümleme yöntemlerini açıklayınız?
11. Geliştirdiğiniz bir program için, bölüm içerisinde verilen gözden geçirme sorularını yanıtlayınız. Programınızın niteliği hakkında ne söyleyebilirsiniz?

Kaynaklar

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- ” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- ”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsiz, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (http://www.buzluca.info/dersler.html)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

Ödev

?



Sorularınız