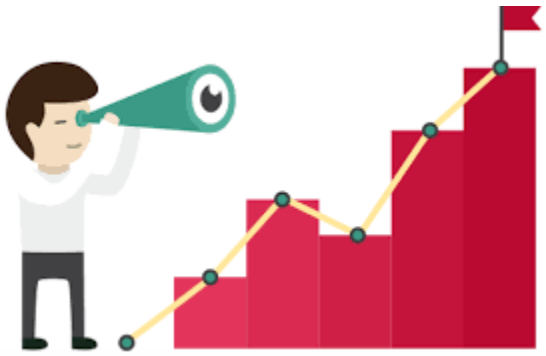
YAZILIM MÜHENDİSLİĞİNİN TEMELLERİ

7.Hafta

GERÇEKLEŞTİRME

Bölüm Hedefi

- Bu bölümde, yazılım geliştirme ortamları tanıtılmakta ve kodlama yöntemleri açıklanmaktadır. Program karmaşıklığının ölçümü üzerine yaygın olarak kullanılan yöntemlerden biri olan Mc Cabe karmaşıklık ölçütü ve kod gözden geçirme teknikleri bu bölümde tanıtılmaktadır.



Gerçekleştirme Fazı

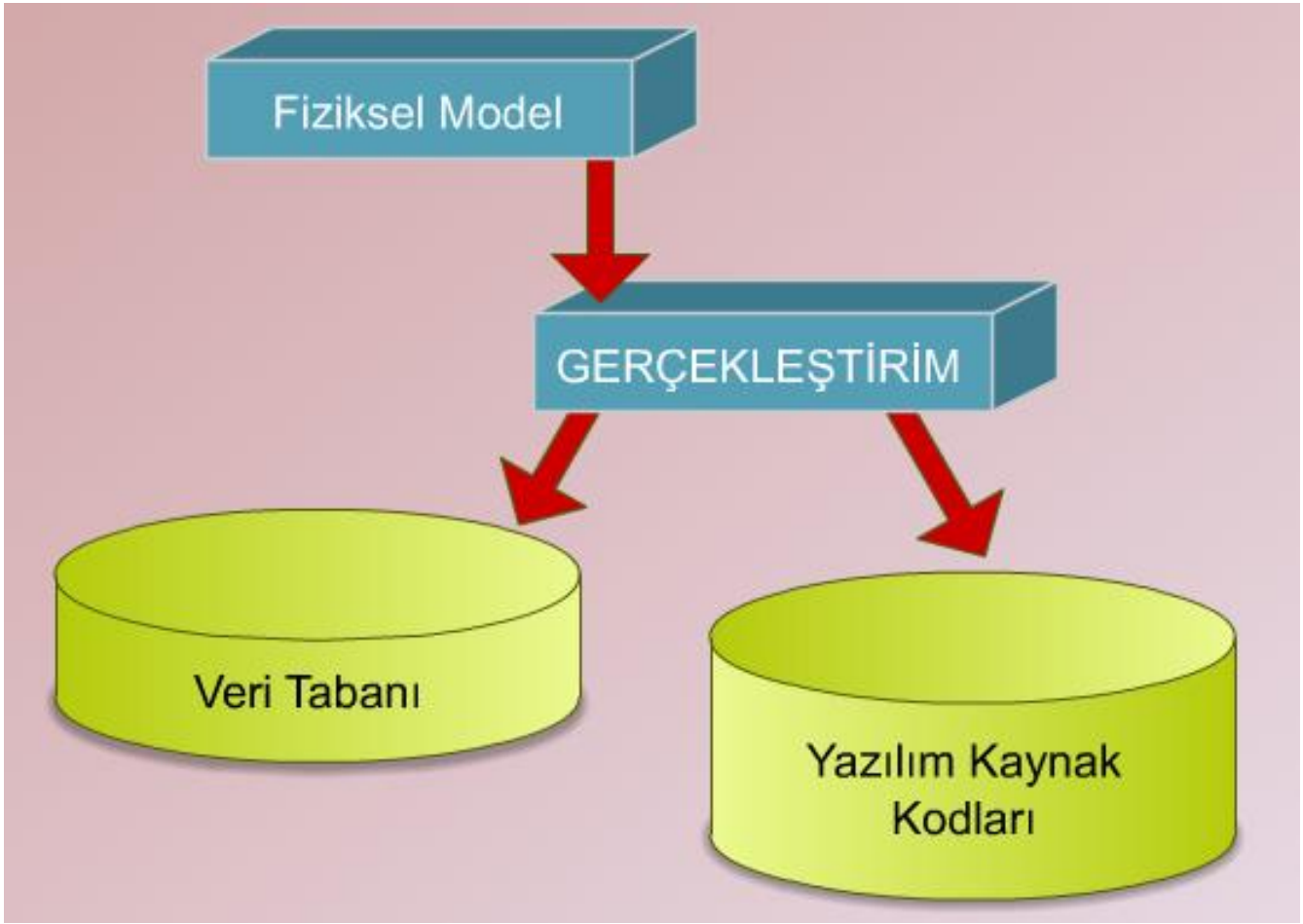
- Programlama Dilleri
- Yazılım Gerçekleştirme
- Derleyiciler
- Kodlama Etkinliği ve Kuralları
- Modüler Gerçekleştirme
- Gerçekleştirmenin Belgelendirilmesi

Giriş

- Gerçekleştirim çalışması, tasarım sonucu üretilen süreç ve veri tabanının fiziksel yapısını içeren fiziksel modelin bilgisayar ortamında çalışan yazılım biçimine dönüştürülmesi çalışmalarını içerir.

- Yazılımın geliştirilmesi için her şeyden önce belirli bir yazılım geliştirme ortamının seçilmesi gerekmektedir.

Gerçekleştirim Çalışması



Yazılım Geliştirme Ortamları

- Programlama Dili
- Veri Tabanı Yönetim Sistemi
- Hazır Program Kitapçıkları
- CASE Araçları

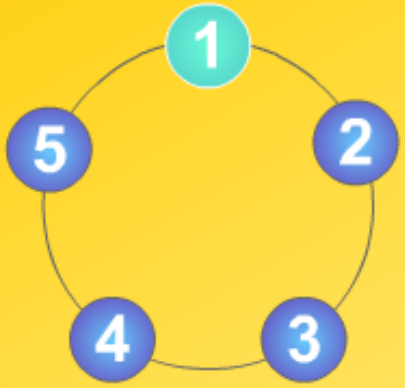
Programlama Dilleri

Geliştirilecek uygulamaya göre kullanılabilecek programlama dili de değişmektedir. Uygulama alanlarının bir sınıflaması:

- Veri işleme yoğunluklu uygulamalar
- Hesaplama yoğunluklu uygulamalar
- Süreç ağırlıklı uygulamalar
- Sistem programlamaya yönelik uygulamalar
- Yapay us (Y. zekâ- Y.Akıl) uygulamaları

biçiminde verilmektedir.

Uygulama Çeşitleri

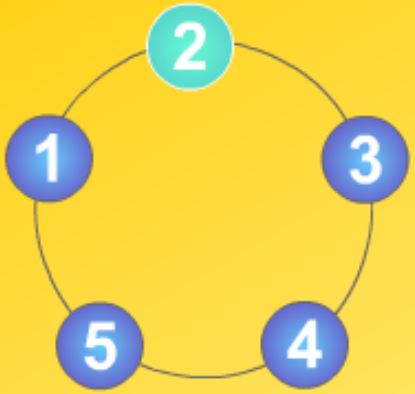


1

Veri İşleme Yoğunluklu Uygulamalar

Günümüzde, bilgi sistemi uygulamaları olarak bilinen ve yoğun veri işlemeyi gerektiren uygulamalar (Nüfus, Seçmen Kütükleri, Adli Sicil vb) için 90'lı yıllara kadar COBOL en yaygın olarak kullanılan programlama dili idi. 90'lı yılların başlarında gelişen ve veri tabanı bağlantılı çalışan Görsel Programlama Dilleri (Visual Basic, Delphi, Visual Age, Developer vb) bugünün uygulamalarında yaygın olarak kullanılmakta ve COBOL ve geçmişteki uygulamalar da bu dillere dönüştürülmektedir. Öte yandan Nesne-Kökenli Programlama dillerinin (C++, Java, Smalltalk vb) kullanımı giderek yaygınlaşmaktadır.

Uygulama Çeşitleri



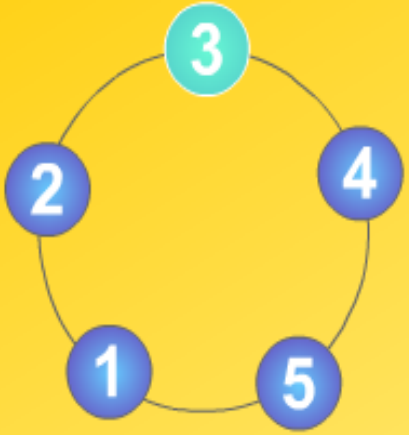
2

Yoğun Matematiksel ve Mantık İşlemi Gerektiren Uygulamalar

Bu uygulamalarda (NASA, uçak mühendisliği vb) program kütüphanelerinin zenginliği nedeniyle halen en yaygın olarak kullanılan programlama dilleri arasında, FORTRAN yer almaktadır. Yeni geliştirilen, bu tür uygulamalarda ağırlıklı olarak C programlama dili kullanılmaktadır.

Teknolojik gelişim, bu tür uygulamaları hızlandırmak amacıyla paralel işlem yapmayı olanaklı kılmış, bu bağlamda Paralel Programlama dilleri gelişmiştir. Ayrıca mevcut programlama dillerinin paralel sürümleri geliştirilmiştir (Paralel FORTRAN, Paralel C vb). Bu alanda kullanılan programlama dillerinin bir başka özelliği ise, yüksek duyarlıkta işlem yapmayı olanaklı kılmalarıdır.

Uygulama Çeşitleri

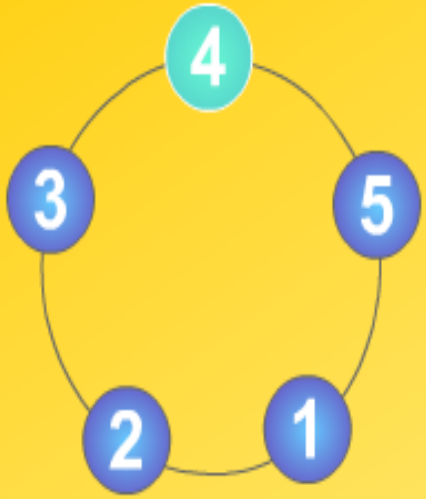


3

Süreç Ağırlıklı Uygulamalar

Bir başka deyişle "gerçek zamanlı" uygulamalarda, veri işleme ve matematiksel işlem yapma boyutlarının yanı sıra süreçler arası eşgüdüm, donanımın yazılım aracıyla sürülmesi boyutları önem kazanmaktadır. Bazı işletim sistemi yazılımları, uçak kontrol sistemleri, bankacılık sistemleri bu tür yazılımlara örnek olarak verilebilir. Bu tür uygulamalarda genellikle düşük düzey programlama dilleri (assembly dili) ve özel programlama dilleri ve henüz yaygınlaşmamış olmakla birlikte C programlama dili kullanılmaktadır.

Uygulama Çeşitleri

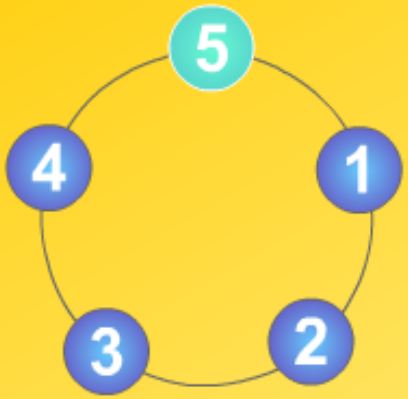


4

Sistem Programlamaya Yönelik Uygulamalar

Bilgisayarın yönetimine ilişkin uygulamalardır (İşletim sistemi vb). Bu tür yazılımlarda kullanılan programlama dili kapalı sistemler için assembly programlama dili, açık sistemler için ise, assembly dili ile birlikte C programlama dilidir.

Uygulama Çeşitleri



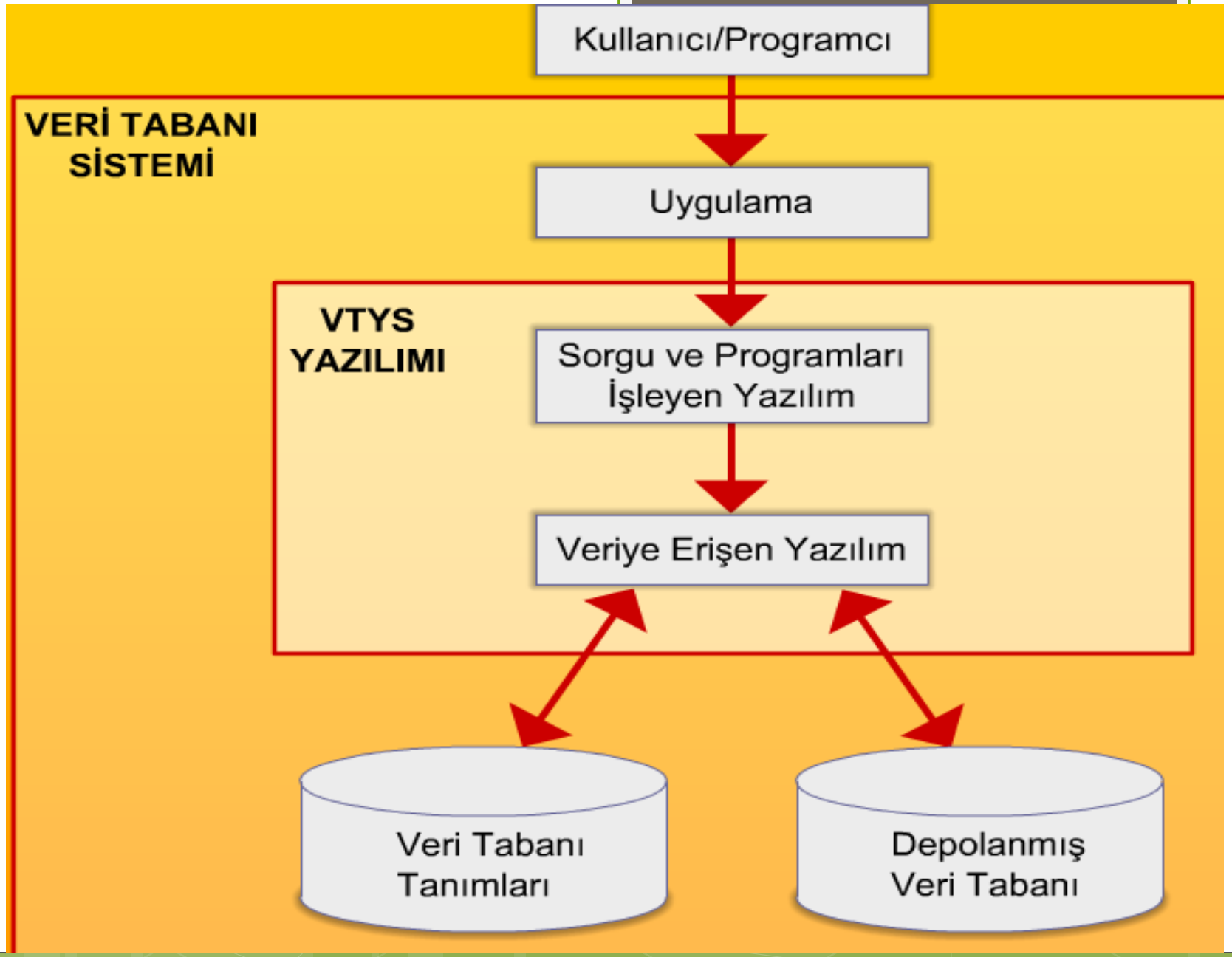
5

Yapay Us Uygulamaları

Kural-çıkarım ilkelerine dayalı uygulamalardır. Kuralların tanımlanması, bu kurallardan çıkarımlar yapılması ve çıkarımlardan yeni kurallar tanımlanması biçiminde etkileşimli olarak çalışmayı gerektiren uygulamalardır. Bu tür uygulamalarda en yaygın olarak kullanılan programlama dilleri ise Lisp ve Prolog'dur.

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.


Kullanıcı/Programcı
VERİ TABANI SİSTEMİ
Uygulama
VTYS YAZILIMI
Sorgu ve Programları İşleyen Yazılım
Veriye Erişen Yazılım
Veri Tabanı Tanımları
Depolanmış Veri Tabanı

Veritabanı 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 sözlüğü**nün varlığı.

Dosya Adı	Değişken Adı	Veri Tipi	Uzunluk	Açıklama
Users	user_id	int	4	Kullanıcı Numarası (ID)
	nick	varchar	10	Kullanıcı Adı
	fullname	varchar	20	Kullanıcı Adı Soyadı
	email	varchar	20	Kullanıcı E-Mail Adresi
	password	varchar	8	Kullanıcı Şifresi
	accesslevel	int	1	Kullanıcı Yetkilendirme Seviyesi (0..9)
Codes	code_id	int	6	Kod Numarası (ID)
	language	varchar	15	Programlama Dil İsmi
	code_category	varchar	15	Kod Kategorisi
	add_date	datetime		Kod Ekleme Tarihi
	score	int	2	Kod Puanı (0-10)
	search_word	varchar	30	Kod Arama Kelimeleri
Forum	comments	longtext		Forum Yorumları
	forum_category	varchar	15	Forum içerisindeki dil kategorileri
	forum_date	datetime		Foruma bilgi giriş tarihi
Code_DB	VT_id	int	2	Veri Tabanı Numarası (ID)
	VT_code_lang	varchar	15	VT' deki Programlama Dilleri
User_DB	user_number	int	4	(Derived) Kullanıcı ID

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 veritabanının yapısını açıklamakta kullanılan kavramlar setidir.

Veri Soyutlama

- 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.

Program-Veri ve Program-İşlem Bağımsızlığı

ÖRNEK

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 bu sorun çözülebilir.

Birden Çok Kullanıcı Desteği

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

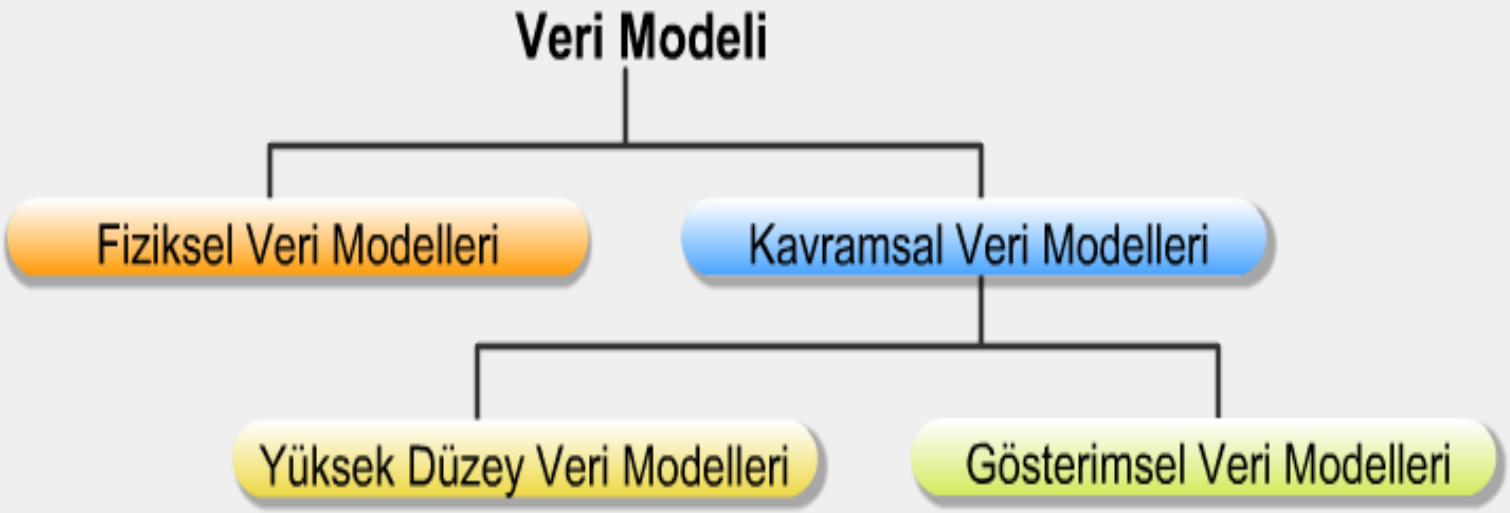
Verinin Birden Fazla İşlem Arasında Paylaşımı

- 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

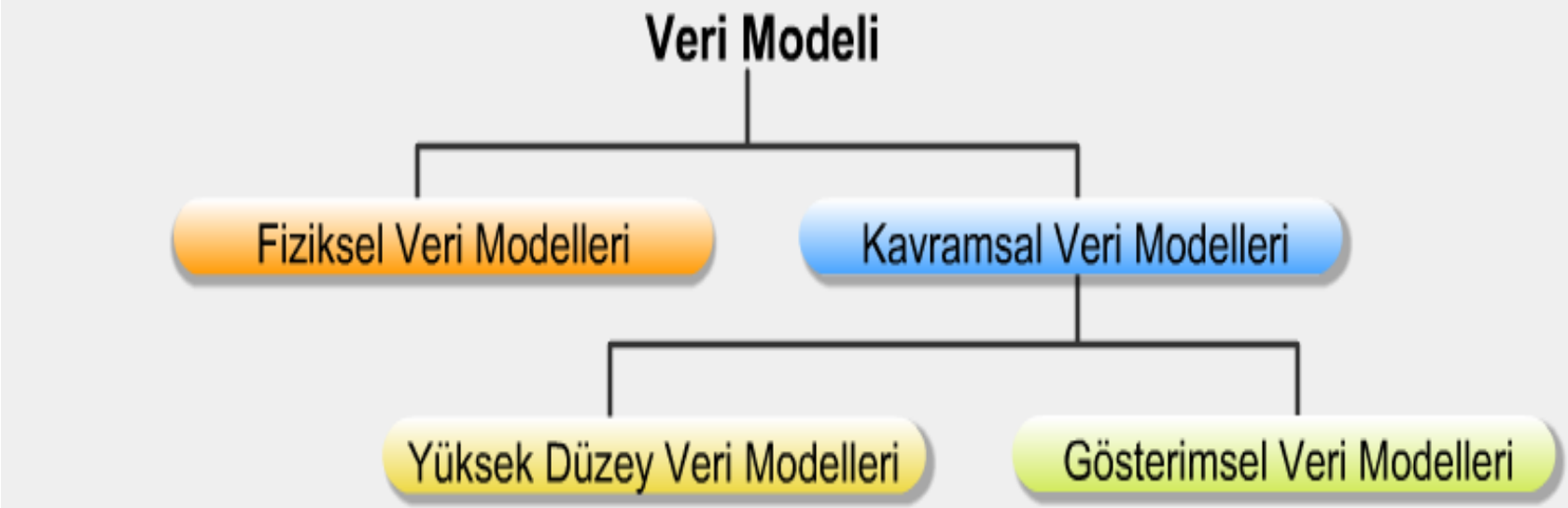
1. Genişleme potansiyeli,
2. Esneklik,
3. Uygulama geliştirme zamanının azalması,
4. Güncel bilgilerin tüm kullanıcılara aynı zamanda ulaşması,
5. Ölçümde ekonomi,
6. İşletme ortamındaki ortak verilerin tekrarının önlenmesi verilerin merkezi denetiminin ve tutarlılığının sağlanması,
7. Fiziksel yapı ve erişim yöntemi karmaşıklıklarının Her kullanıcıya yanlız ilgilendiği verilerin kolay anlaşılır yapılarda sunulması,
8. Uygulama yazılımı geliştirmenin kolaylaşması,
9. Fiziksel yapı ve erişim yöntemi karmaşıklıklarının Her kullanıcıya yanlız ilgilendiği verilerin kolay anlaşılır yapılarda sunulması,

Veri Modelleri



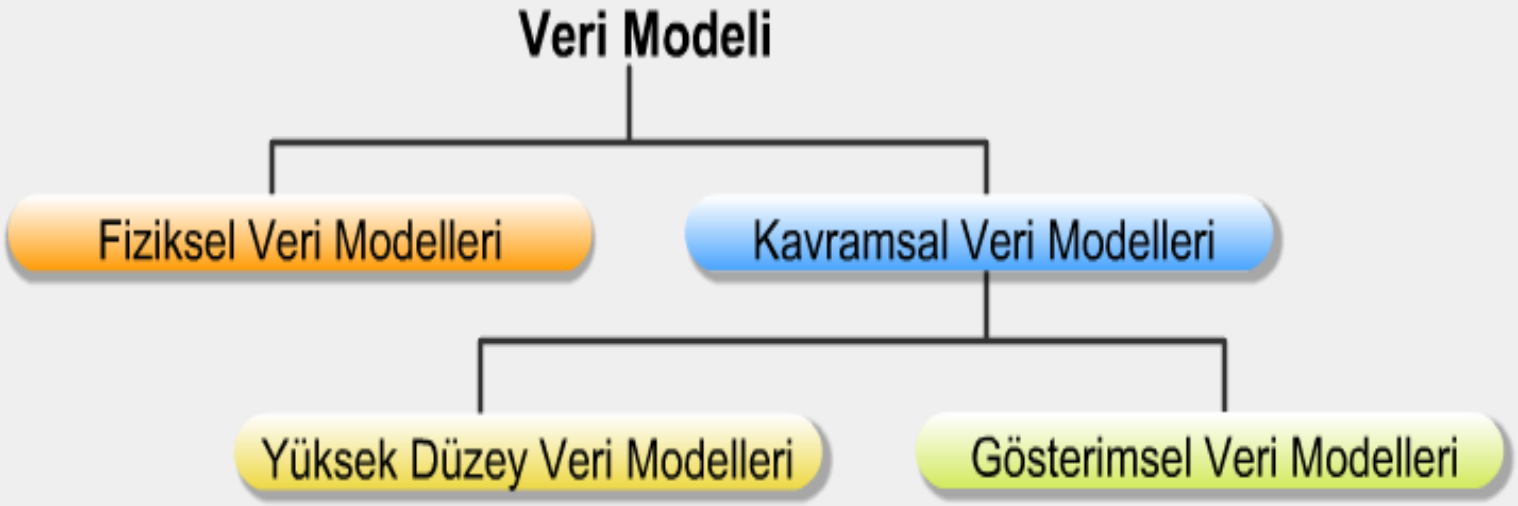
Fiziksel veri modelleri ise verinin bilgisayarda nasıl saklandığının detayları ile ilgili kavramları sağlarlar. Bu ikisinin arasında ise gösterimsel veri modelleri vardır. Bu modeller bilgiyi kayıt biçemi, sırası ve erişim yollarıyla göstererek verinin bilgisayar ortamında nasıl saklandığını açıklar. Burada erişim yolu, belirli bir veritabanı kaydını etkin bir şekilde aramayı sağlayan yapıdır.

Veri Modelleri



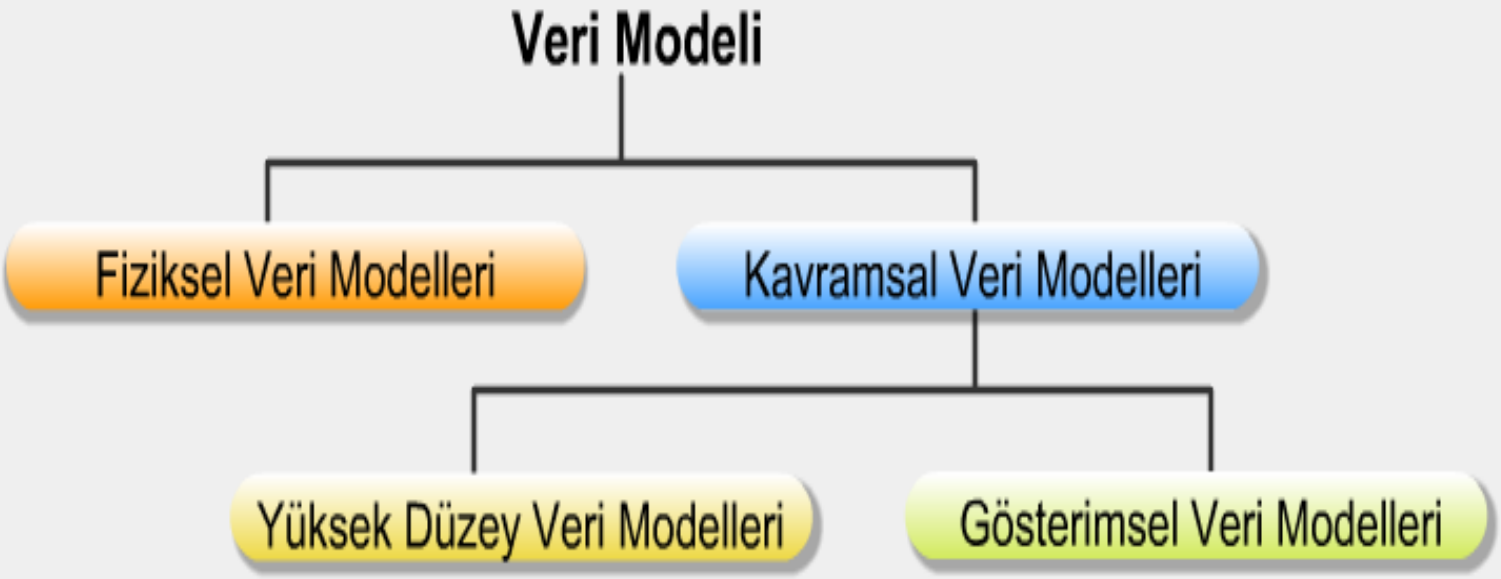
Bunlar hem kullanıcı tarafından anlaşılan hem de verinin bilgisayar içerisindeki gösteriminden çok da fazla uzak olmayan kavramları sağlarlar.

Veri Modelleri



Yüksek düzey veri modelleri varlık, özelik ve ilişki gibi kavramları kullanır. Varlık veri tabanında saklanan ve gerçek dünyadan bir nesne veya kavramdır; proje, işçi gibi. Özelik, varlığı anlatan bir özelliği gösterir işçinin adı, ücreti gibi. Bir veya daha fazla varlık arasındaki ilişki ise bir işçi ve proje arasındaki çalışma ilişkisidir.

Veri Modelleri



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 tanımlanması ile kendisini ayırmak önemlidir. Veri tabanının tanımlamaları veri tabanı şeması veya meta-veri olarak adlandırılır. Veri tabanı şeması, tasarım sırasında belirtilir ve sıkça değişmesi beklenmez.

- Pek çok veri modeli şemaları, diyagramlar halinde göstermek için belli gösterim biçimlerine sahiptir. Diyagramlar her kayıt tipinin yapısını gösterir fakat kaydın gerçek örneğini göstermez.

VTYS Mimarisi

İçsel Düzey

Veritabanının fiziksel saklanma yapısını açıklar. Fiziksel veri modeli kullanır ve veritabanına erişim yolu ile veri saklamanın tüm detaylarını açıklar.

Kavramsal Düzey

Kavramsal düzey ise kavramsal şema içerir ve kullanıcılar için veritabanının yapısını açıklar. Gerçek fiziksel yapının detaylarını kullanıcıdan gizler, veri tipleri, varlıklar ilişkiler, kullanıcı işlemleri ve sınırlamalar üzerine konsantre olmamızı sağlar. Daha yüksek seviyede veri modeli veya gerçekleştirim veri modeli bu seviyede kullanılabilir.

Dışsal Düzey

Bu düzey, dış şemalar veya kullanıcı görüşleri içerir. Her dış şema veritabanının bir bölümünü açıklar ve her gruba kendi ilgilendiği görüşü sunarken, diğer bir gruptan ilgilenmediği görüşü saklar. Daha yüksek düzeyde veri modeli veya gerçekleştirim veri modeli bu seviyede kullanılabilir.

Veri Tabanı Dilleri ve Arabirimleri

Veri Tanımlama Dili (VTD)	Kavramsal şemaları tanımlamak üzere veritabanı yönetici ve tasarımcısı tarafından kullanılır
Saklama Tanımlama Dili (STD)	İçsel şemayı tanımlamak için kullanılır.
Görüş Tanımlama Dili (GTD)	Görüş tanımlama dili kullanıcı görüşlerini tanımlamak ve kavramsal şemaya dönüştürmek amacıyla kullanılır.
Veri İşleme Dili (VİD)	Veri işleme dilidir. Veri tabanı oluşturulduktan sonra Veri tabanına veri eklemek, değiştirmek, silmek veya eklenmiş veriyi getirmek amacıyla kullanılır.

Veri Tabanı Sistem Ortamı

- VTYS, karmaşık bir yazılım sistemidir. Burada bir VTYS'yi oluşturan yazılım bileşenlerden söz edilmektedir.
- Tipik bir VTYS yazılımı bileşenleri, VTYS Kataloğu, veri tabanı derleyicisi (VTD), veri yöneticisi, VİD derleyicisi, VT işlemcisi ve yardımcı yazılımlar biçimindedir.

VTYS'nin Sınıflandırılması

- Bir VTYS'ni sınıflandırmada kullanılan temel kriter VTYS'nin dayandığı veri modelidir.

- En fazla kullanılan veri modelleri ilişkisel, ağ, hiyerarşik, nesne-yönelimli ve kavramsal modellerdir.

Hazır Program Kitaplıkları

- Hemen hemen tüm programlama platformlarının kendilerine özgü hazır program kitaplıkları bulunmaktadır.
- Söz konusu kitaplıklar, hemen hemen tüm uygulamalarda kullanılabilecek ortak program parçaları (OCX,VBX vb), sistem yönetimine ilişkin yazılımları içerir.
- Söz konusu yazılımlar program geliştirme hızını oldukça arttırmaktadır. Günümüzde bu tür kitaplıkların edinilmesi, internet sayesinde oldukça kolaylaşmıştır.

CASE Araç ve Ortamları

- Günümüzde bilgisayar destekli yazılım geliştirme ortamları (CASE) oldukça gelişmiş durumdadır.
- CASE araçları, yazılım üretiminin hemen her aşamasında (planlama-çözümleme-tasarım-gerçekleştirim-sınama) üretilen bilgi ya da belgelerin bilgisayar ortamında saklanmasını, bu yolla kolay erişilebilir ve yönetilebilir olmasını olanaklı kılar.
- Hemen tüm CASE araçları, belirli bir standarda uygun olarak geliştirme yapmayı zorlar. Bu yolla yapılan üretimin yüksek kalitede olması sağlanır. CASE araç ve ortamları Bölüm 12'de ayrıntılı olarak incelenmiştir.

Kodlama Stili

- Yazılım Kod stili konusunda herhangi bir kabul edilmiş standart bulunmamaktadır.
- Bu konuda, yazılım geliştiren ekiplerin, kodlama aşamasına başlamadan kodların düzeni konusundaki standartlarını ya da kurallarını geliştirmeleri ve bu kuralların uygulamaya geçmesini sağlamaları önerilmektedir.

Açıklama Satırları

- Bir program parçasını anlaşılabilir kılan en önemli unsurlardan biri bu program kesiminde içerilen açıklama satırlarıdır. Her bir program modülü içerisine

- Modül başlangıç açıklamaları
- Modül Kod Açıklamaları
- Boşluk satırları

eklenmelidir.

Açıklama Satırları

```
/*****************************************************************
*  ©
*****************************************************************
*  Modül Adı:
*  Kütük Adı:
*  İşlevi:
*  Desteklenen Platformlar:
*  Desteklenen Derleyiciler:
*  Taşınabilirlik Konuları:
*  Kullanılan İşlevler:
*  Hatalar:
*  Bilinen Düzeltilmemiş Yanlışlar:
*  Programcı:
*  Tarihçe Günlüğü:
*  İsim                    Tarih                    Açıklama
*
*****************************************************************
```

Kod Biçimlemesi

- Ancak Program kodlama düzeni ise gerçekleştirim aşamasında çözümlenir.
- Programların okunabilirliğini arttırmak, anlaşılabilirliğini kolaylaştırmak amacıyla açıklama satırları kullanımının yanı sıra belirli bir kod yazım düzeni kullanılması gerekmektedir

```
DO I=1 TO N-1;T=I; DO J=I TO I+1;IF A(J)< A(T) THEN DO
T=J;
END;
IF T<>I THEN DO H=A(T);A(T)=A(I); A(I)=T;END;END;
```

(a) Kötü stilde kodlanmış program parçası

```
DO I = 1 TO N-1;
   T=I;
   DO J = 1 TO I+1;
   IF A(J) < A(T) THEN DO
      T=J;
   END;
   IF  T <> I  THEN DO
      H = A(T);
      A(T) = A(I);
      A(I) = H;
   END;
END;
```

(b) İyi stilde kodlanmış program parçası

Anlamlı İsimlendirme

- Hangi değişkenlerin hangi veri tabanı tablosuna ilişkin oldukları,
- Hangi değişkenlerin klavye ve ekran aracılığı ile değer aldıkları
- Hangi değişkenlerin yazıcı çıktılarında görülmek üzere kullanıldıkları,
- Hangi değişkenlerin yalnızca ilgili kod içerisinde ara değişkenler olarak kullanıldıkları

Tekrar

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.

Yapısal Programlama Yapıları

Ardışıl İşlem Yapıları

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

Yapısal Programlama Yapıları

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 daha çok kez yinelenecek işlemler için kullanılan yapılardır. Temelde üç tür döngü yapısı bulunmaktadır:

- Belirli sayıda yinelenecek işlemler için kullanılan yapılar (for yapısı),
- Bir koşula bağlı olarak, sıfır ya da birden çok kez yinelenecek işlemler için kullanılan yapılar (while-end yapısı),
- 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. Programlar, yalnızca bu yapılar kullanılarak kodlandığında ve uygun açıklama satırları ile desteklendiğinde, program bakımı kolaylaşmakta ve geliştirme hızlanmaktadır.

PROGRAM KARMAŞIKLIĞI

- Program karmaşıklığını ölçmek için bir çok teorik model geliştirilmiştir. Bu modellerin en eskisi ve yol göstericisi McCabe karmaşıklık ölçütüdür.
- Bu bölümde bu ölçüt anlatılmaktadır. Söz konusu ölçüt 1976 yılında McCabe tarafından geliştirilmiştir. Bu konuda geliştirilen diğer ölçütlerin çoğu, bu ölçütten esinlenmiştir.

1. Programın Çizge Biçimine Dönüştürülmesi

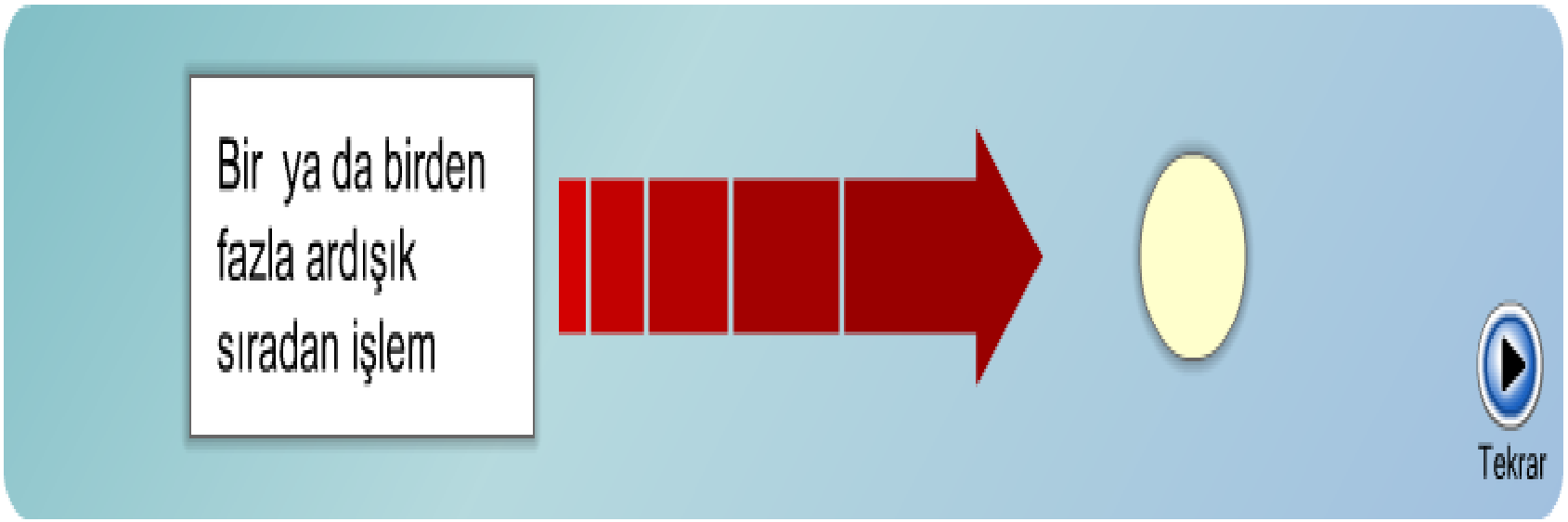
2. Mc Cabe Karmaşıklık Ölçütünün Hesaplanması

Programın Çizge Biçimine Dönüştürülmesi

- Bir program bir ana program ve onunla ilgili birden fazla alt programdan oluşur.
- Gerek ana program gerekse alt programların tümü, McCabe Karmaşıklık ölçütünün hesaplanmasından önce çizge biçimine dönüştürülmelidir.
- Bir program parçasının çizge biçimine dönüştürülmesi için gerekli kurallar aşağıda açıklanmaktadır.

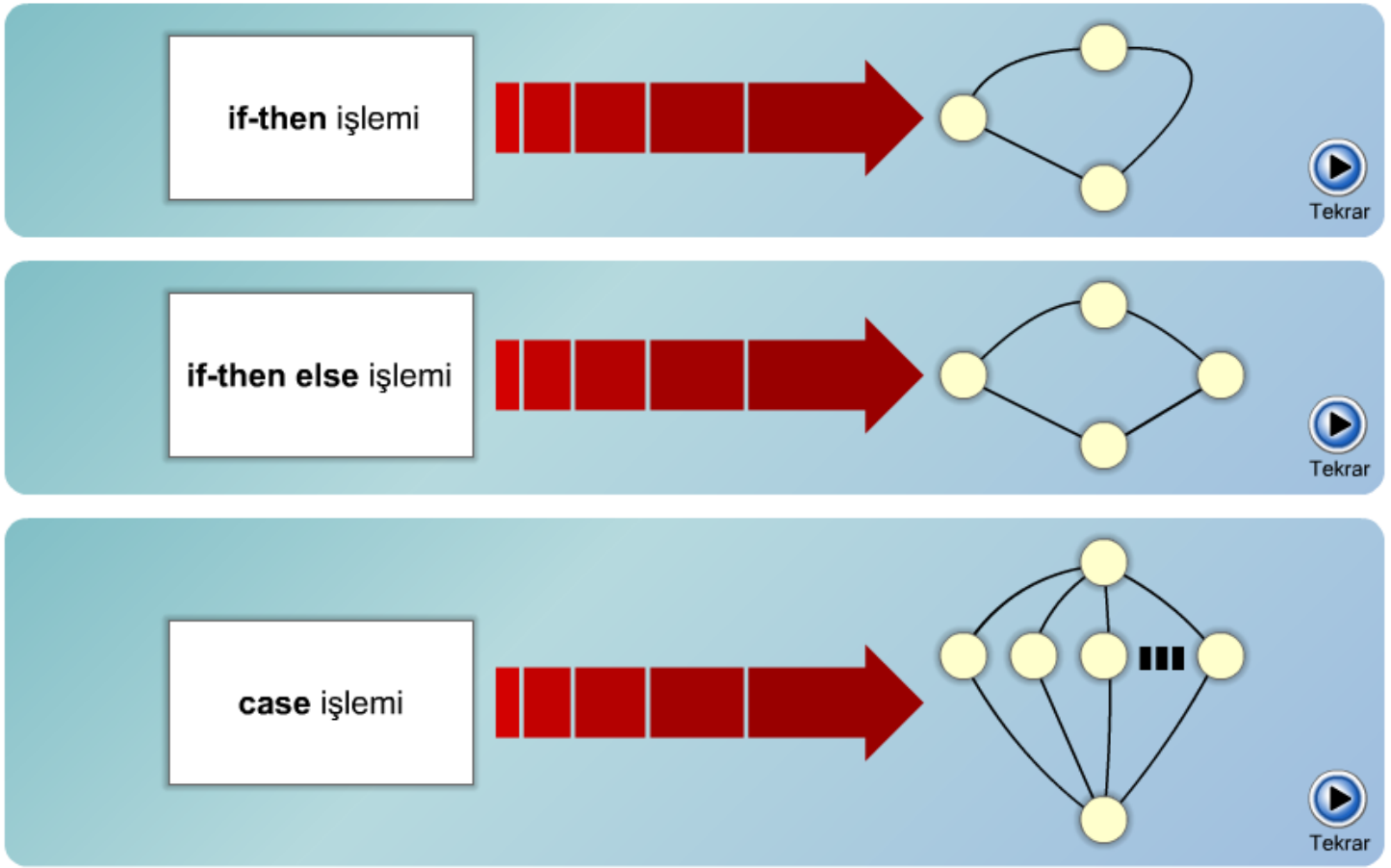
Programın Çizge Biçimine Dönüştürülmesi

1. Sıradan İşlemler: Bir sıradan işlem ya da birden fazla ardışık sıradan işlem çizgede bir düğüme dönüştürülür.



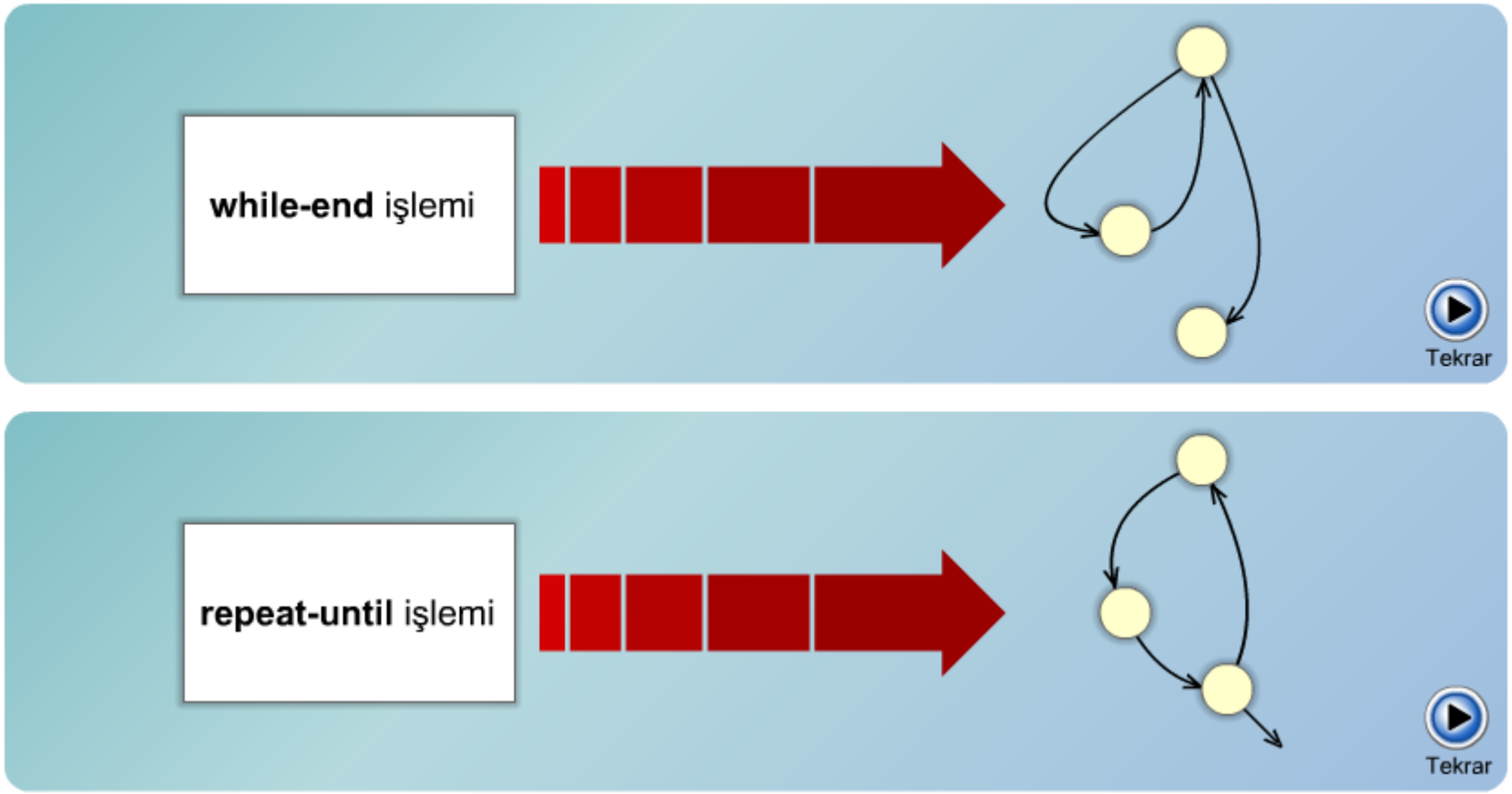
Programın Çizge Biçimine Dönüştürülmesi

2. Koşullu İşlemler:



Programın Çizge Biçimine Dönüştürülmesi

3. Döngü İşlemleri:



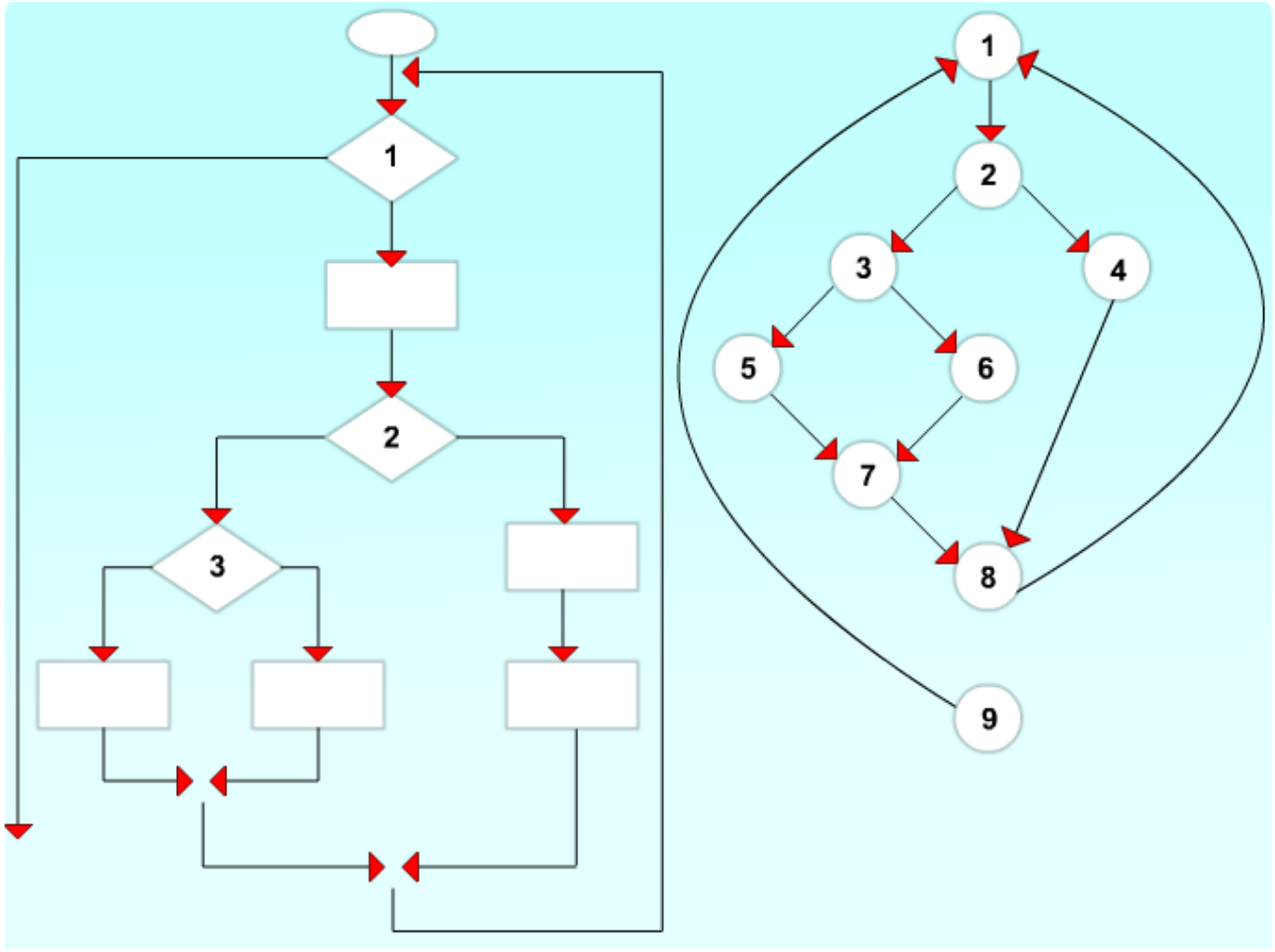
McCabe Karmaşıklık Ölçütü Hesaplama

- Ana program ve alt programlar çizge biçimine dönüştürüldükten sonra programın McCabe karmaşıklık ölçütü (V(G)) aşağıdaki biçimde hesaplanır.

$$V(G) = k - d + 2p$$

- k: kenar sayısı
- d: düğüm sayısı
- p: bileşen sayısı

McCabe Karmaşıklık Ölçütü Hesaplama



McCabe Ölçütü

- Şekil 6.5 (a) da bir programın iş akış şeması verilmekte, (b) kısmında ise önceki kesimde verilen kurallar uygulanarak programın çizge biçimine dönüştürülmüş biçimi verilmektedir. McCabe Karmaşıklığını hesaplamak için kenar ya da dallar ve düğümler sayılıp formülün uygulanması gerekir. Bu örnekte:

- **k = 11** Kenar sayısı
  d = 9 Düğüm sayısı
  p = 1 Bileşen sayısı

karmaşıklık:

$$V(G) = 11 - 9 + 2 \times 1 = 4$$

OLAĞANDIŞI DURUM ÇÖZÜMLEME

- Olağan dışı durum, bir programın çalışmasının, geçersiz ya da yanlış veri oluşumu ya da başka nedenlerle istenmeyen bir biçimde sonlanmasına neden olan durum olarak tanımlanmaktadır.
- Genelde kabul edilen kural, bir programın işletiminin sonlandırılması işleminin -elektrik kesintisi vb donanım hataları dışında- bütünüyle program denetiminde olmasıdır. Bu kural olağan dışı sonlandırma işlemini de kapsamaktadır.
- Bu nedenle program kodları oluşturulurken, olağandışı durumların da dikkate alınması ve bu durumlardaki program davranışına ilişkin yöntemler geliştirilmesi gerekmektedir.

Ada Dili Olağandışı Durum Çözümleme Yapısı

```
begin
     . . . .
     . . . .
     . . . .          Program kod satırları
     . . . .
     . . . .
     . . . .
exception
     when CONSTRAINT_ERROR   =>      SINIR_HATASI;
.    .
     when TABLO_DOLU         =>      .   .   .
end;
```

Olağandışı Durum Tanımları

- Olağan dışı durumlar, programlama dili tarafından tanımlı durumlar olduğu gibi kullanıcı tarafından da tanımlanabilmektedir.

- Yukarıdaki örnekte, CONSTRAINT_ERROR adlı olağan dışı durum ADA Programlama dili tarafından tanımlanmış, TABLO_DOLU adlı olağan dışı durum ise programcı tarafından tanımlanmıştır.

Olağandışı Durum Tanımları

Bu tür olağan dışı durumlar, olağan dışı durumu tanımlayan ve sonucunda "doğru" ya da "yanlış" değeri üreten biri mantıksal yordam ya da fonksiyon tanımından oluşmaktadır.

Şekilde ADA Programlama dili platformunda kendiliğinden tanımlı olağandışı durum örnekleri verilmektedir.

Farklı Olağandışı Durum Çözümleme Yaklaşımları

Anında Durdurma:

Hata bulunduğunda program açıklayıcı bir ileti vererek sona erer. Bu iletinin yanı sıra onaltılı sistemde bir döküm de verir. MS Office paketlerinde alınan "This Program Performed an illegal operation" iletisi ve ardından programın durması, bu tür olağandışı durum çözümleme yaklaşımına örnek olarak verilebilir. Bu yaklaşımda, verilen hata iletisinin anlamlı olması ve programcıya düzeltebilmek amacıyla yol gösterici olması önemlidir.

Hata Kodu Verme:

İlgili program modülünden başarısız şekilde çıkıldığında, programın bir hata kodu vermesi biçminde bir yaklaşımdır. Söz konusu kod "başarılı" ya da "başarısız" durumu belirten mantıksal bir kod olabileceği gibi, bazı uygulamalarda sayısal ya da metin biçiminde olabilir. Bu yaklaşımda, ilgili programında hata oluştuğu anlaşılabilmekte ancak hatanın hangi deyimde oluştuğu anlaşılamamaktadır.

Farklı Olağandışı Durum Çözümleme Yaklaşımları

Tanımlı Olağandışı Yordam Çalıştırma:

Programlama dili platformunun tanımladığı olağan dışı durum çözümleme olanaklarını kullanmak biçimindedir. COBOL Programlama dilindeki "ON READ ERROR" deyimi gibi.

Hata Yordamı Çalıştırma:

Olmayacak Değer Döndürme: Yordamın beklenmeyen bir değer geri döndürmesi gibi bir yaklaşımdır. Örneğin yaş hesaplaması yapıp, yaş bilgisini geri döndüren bir programın eksi değer döndürmesi gibi.

Tekrar

KOD GÖZDEN GEÇİRME

- Hiç kimse, önceki sürümlerini gözden geçirmeden ve incelemeden okunabilir bir program yazamaz.
- Hiçbir yazı editörün onayını almadan basılamayacağı gibi hiçbir program da incelenmeden, gözden geçirilmeden işletime alınmamalıdır.
- Kod gözden geçirme ile program sınama işlemlerini birbirinden ayırmak gerekir.

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.

Gözden Geçirme Sürecinin Düzenlenmesi

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

Gözden Geçirme Sürecinin Düzenlenmesi

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

Gözden Geçirme Sırasında Kullanılacak Sorular

6.6.2.1. 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?

Gözden Geçirme Sırasında Kullanılacak Sorular

6.6.2.2. 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. Giriş açıklama satırları, öbeğin algoritma tanımını içeriyor mu?

6. Giriş açıklama satırları, öbekte yapılan değişikliklere ilişkin tanımlamaları içeriyor mu?

7. Giriş açıklama satırları, öbekteki olağan dışı durumları tanımlıyor mu?

8. Giriş açıklama satırları, Öbeği yazan kişi ve yazıldığı tarih ile ilgili bilgileri içeriyor mu?

9. Her paragrafı açıklayan kısa açıklamalar var mı?

Gözden Geçirme Sırasında Kullanılacak Sorular

6.6.2.3. Veri Kullanımı

1. Işlevsel olarak ilintili bulunan veri elemanları uygun bir mantıksal veri yapısı içinde gruplanmış mı?

2. Değişken adları,işlevlerini yansıtacak biçimde anlamlı mı?

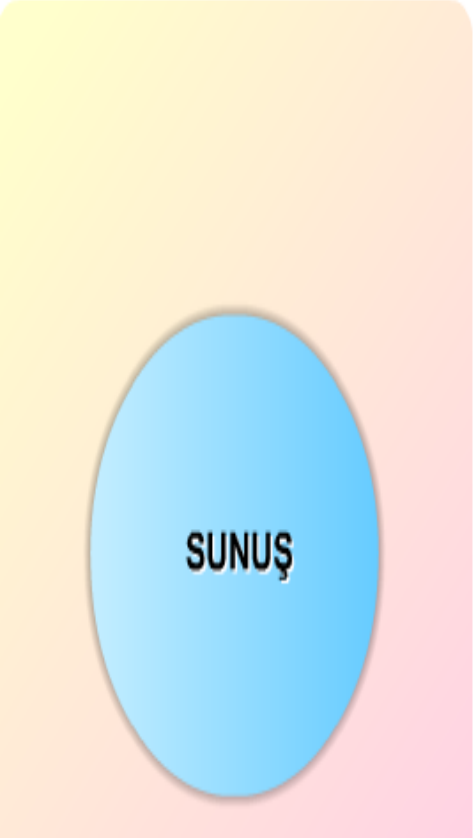
3. Değişkenlerin kullanımları arasındaki uzaklık anlamlı mı?

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?

Gözden Geçirme Sırasında Kullanılacak Sorular



6.6.2.5. 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. Bütün deyimlerde, karmaşıklığı azaltacak şekilde parantezler kullanılmış mı?

5. Bütün deyimler, belirlenen program stiline uygun olarak yazılmış mı?

6. Öbek yapısı içerisinde akıllı "programlama hileleri" kullanılmış mı?

Gerçekleştirme Öznitelikleri

- Etkinlik
- Aykırı Durumların Kotarılması
- Hata Ayıklama
- Atık/Çöp toplama
- Soyutlama
- Bilgi Gizleme
- Etiketleme
- Güvenlik
- Basitlik
- Belirgin Arayüz
- Taşınabilirlik
- Çok düzeyli koruma
- Gözden geçirme
- Belgelendirme

Alınan Dersler 1

- Bakım Programcısı kavramı uygulamada çalışmamakta, genellikle yazılımı geliştiren, bakımını da yapmakta ve usta-çırak ilişkisi içinde işler devredilmeye çalışılmaktadır.
- Bu durum, zaten kısıtlı olan bilişim insan gücü kaynağının verimsiz olarak kullanılmasına neden olmakta ve "kişiye bağlı" yazılımlar geliştirilmektedir.

Alınan Dersler 2

- Özellikle C programlama dili için programlama stili çok önemlidir. Sözdizim tanımları gereği C programları, okunmaları ve anlaşılmaları kolay olmayan programlardır.
- Bu nedenle, kodlamaya başlanılmadan önce, bu konuya ilişkin kullanacağınız standartları oluşturulmalı ve kodlama sırasında bu standartlara uyulup uyulmadığını denetlemeniz gerekmektedir.

Alınan Dersler 3

- Açıklama satırları ile ilgili kod uyumlu olmalı. Kod satırları arasında, süreç içerisinde yapılan değişiklikler, yeni eklentiler açıklama satırlarına yansıtılmamakta, bu durum, programların okunabilirliğini giderek zorlaştırmaktadır.

Alınan Dersler 4

- Süreç içinde anlamlı isimlendirmeler kaybolabilmektedir. Program bakımı, belli bir sistematik içerisinde yapılmamakta ve izlenmemektedir.
- Programların, süreç içerisinde, "yamalı bohça" ya dönüşmesi söz konusudur. Bir sistematik ve izleme olmadığında, bakım giderek zorlaşmaktadır.

Alınan Dersler 5

- Gözden Geçirme çalışmasının yapılması amacıyla yönetimi ikna etmek zor.
- Bu tür bir örgütlenmeye sıcak bakılmamakta, öte yandan, programcılar da benzer tepkiyi vermekte ve "sanki kendilerini denetliyorlarmış" mantığı ile davranmaktalar.
- Oysa, Gözden geçirme çalışması yapılmadığında, özellikle büyük yazılım projelerinin geliştirilmesi olanaksız.

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?

Genel

- Ders Kitabı: Yazılım Mühendisliği

  Erhan Sarıdoğan- papatya Yayıncılık (kitapyurdu.com)

  Diğer Kaynaklar:
  - Ders Notları.
  - Ali Arifoğlu, Yazılım Mühendisliği. SAS bilişim Yayınları
  - İnternet, UML Kaynakları
  - Roger S. Pressman, Software Engineering – Practitioner's Approach