- 1. Bilgisayar Mühendisliğine Giriş
  - 1.1. Temel Kavramlar
  - 1.2. Algoritma ve Akış Şemaları
    - 1.2.1. Algoritmada Olması Gereken Özellikler
    - 1.2.2. Akış Şeması Simgeleri
  - 1.3. Programlama Dilleri
    - 1.3.1. Yazılım Geliştirme Süreci
    - 1.3.2. Programlama Dilinin Önemi
    - 1.3.3. Nesneye Yönelik Programlama
    - 1.3.4. Programlama Ortamı
  - 1.4. İşletim Sistemleri
    - 1.4.1. Bilgisayar Sistemi Bileşenleri
    - 1.4.2. İşletim Sistemlerinin Temel İşlevleri
    - 1.4.3. Sistem Performansı
    - 1.4.4. Çekirdek Sistem
    - 1.4.5. Proses Yönetimi
    - 1.4.6. Bellek Yönetimi
- 2. Bilgisayar Mimarisi ve Tasarım
- 3. Bilgisayar Ağları
  - 3.1. Ağ Yapısına Giriş
    - 3.1.1. Veri İletimi
    - 3.1.2. Veri İletim Tipleri
    - 3.1.3. Veri İletim Yöntemleri
    - 3.1.4. Veri İletim Ortamları
    - 3.1.5. Haberleşme Yöntemleri
    - 3.1.6. Ağ Mimarileri
    - 3.1.7. Sunucu Türleri
    - 3.1.8. Coğrafi Açıdan Ağ Türleri
    - 3.1.9. Topolojiler
  - 3.2. Ağ Bağlantı Cihazları
  - 3.3. İnternet ve TCP/IP
    - 3.3.1. Katmanlar
    - 3.3.2. TCP
    - 3.3.3. UDP
    - 3.3.4. IP
  - 3.4. Ağ Güvenliği
    - 3.4.1. Bilgi Güvenliği
    - 3.4.2. IPsec
    - 3.4.3. Saldırı Tespit Sistemleri (IDS-Intrusion Detection Systems)
    - 3.4.4. Şifreleme
- 4. Veri Yapıları ve Algoritmalar
  - 4.1. Bilgisayar Yazılım Dünyası
  - 4.2. Veri Yapıları ve Veri Modelleri
    - 4.2.1. Veri Modeli Türleri
  - 4.3. Sıralama Algoritmaları
  - 4.4. Arama Algoritmaları

- 5. Java ile Programlama
  - 5.1. Veri Tipleri ve Değişkenler
- 6. İşletim Sistemleri
  - 6.1. Bilgisayarlar ve İşletim Sistemleri
  - 6.2. Bilgisayar Mimarileri
    - 6.2.1. CISC (complex instruction set computer) Mimarisi
    - 6.2.2. RISC (reduced instruction set computer) Mimarisi
    - 6.2.3. İşletim Sistemleri Ağ Özellikleri
- 7. Veri Tabanı Yönetim Sistemleri
  - 7.1. Veritabanı Yönetim Sistemleri
    - 7.1.1. Kavramlar
    - 7.1.2. Uygulamalar
    - 7.1.3. Yaklaşımlar
  - 7.2. Veri Modelleri
  - 7.3. Sorgulama
  - 7.4. Fonksiyonlar
    - 7.4.1. Tek Satır
      - 7.4.1.1. Karakter
      - 7.4.1.2. Genel
- 8. Yazılım Mühendisliği
  - 8.1. Yazılım Mühendisliğine Giriş
    - 8.1.1. Giriş
    - 8.1.2. Yazılım Süreçleri
    - 8.1.3. Çevik Yazılım Geliştirme
    - 8.1.4. Gereksinim Mühendisliği
    - 8.1.5. Sistem Mühendisliği
    - 8.1.6. Mimari Tasarım
    - 8.1.7. Tasarım ve Geliştirme
    - 8.1.8. Yazılım Testi
    - 8.1.9. Yazılım Evrimi
  - 8.2. Güvenilebilirlik ve Güvenlik
    - 8.2.1. Güvenilebilir Sistemler
    - 8.2.2. Güvenilirlik Mühendisliği
    - 8.2.3. Emniyet Mühendisliği
    - 8.2.4. Güvenlik Mühendisliği
    - 8.2.5. Hayatta Kalma
  - 8.3. Yazılım Yönetimi
    - 8.3.1. Proje Yönetimi
    - 8.3.2. Proje Planlaması
    - 8.3.3. Kalite Yönetimi
    - 8.3.4. Konfigirasyon Yönetimi
- 9. Web Programlama
  - 9.1. HTML5
    - 9.1.1. Neden Önemli
    - 9.1.2. Formlar
      - 9.1.2.1. Özellikleri

        - 9.1.3. Elemanlar
            - 9.1.3.1. Input Types
    - 9.2. CSS3
        - 9.2.1. CSS Pseudo Classes & Elements
        - 9.2.2. EcmaScript 6
- 10. Yazılım Proje Yönetimi
    - 10.1. Yazılım Geliştirme Süreci
        - 10.1.1. Yazılım Geliştirme Temel İlkeleri
        - 10.1.2. Yazılım Geliştirme Süreci
        - 10.1.3. Yazılım Süreç (Geliştirme) Modelleri
    - 10.2. Planlama
    - 10.3. Projenin Yürütülmesi
    - 10.4. Kalite Güvence Yöntemi

# 1. Bilgisayar Mühendisliğine Giriş

## 1.1. Temel Kavramlar

> **Birleşik mantık devreleri** VE, VEYA gibi kapılardan oluşur ve devrenin çıktısı sadece o anki girdilere bağlıdır. **Sıralı/Ardışıl mantık devreleri (flip-flop)** hem o andaki girdilere hem de bir önceki çıktıya bağlıdır. Bu devreler saat darbeleri ile çalışır.

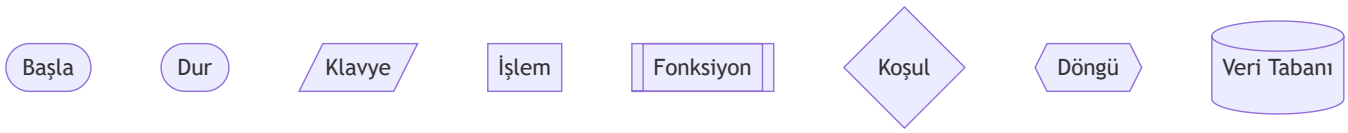
> Assembly diliyle yazılmış programların çalışması için makina koduna dönüştürülmesi gerekir. Her işlemci ailesinin kendine has makina kodu ve assembly dili vardır.

## 1.2. Algoritma ve Akış Şemaları

### 1.2.1. Algoritmada Olması Gereken Özellikler

1. Etkin ve Genel olma
2. Sonlu olma
3. Yanılmazlık
4. Giriş/Çıkış tanımlı olma
5. Başarım (Bellek gereksinimi ve çalışma süresi arasındaki denge)

### 1.2.2. Akış Şeması Simgeleri




## 1.3. Programlama Dilleri

### 1.3.1. Yazılım Geliştirme Süreci

1. Gereksinim analizi
2. Yazılım Tasarımı
3. Kodlama
4. Sertifikasyon
5. Bakım

### 1.3.2. Programlama Dilinin Önemi

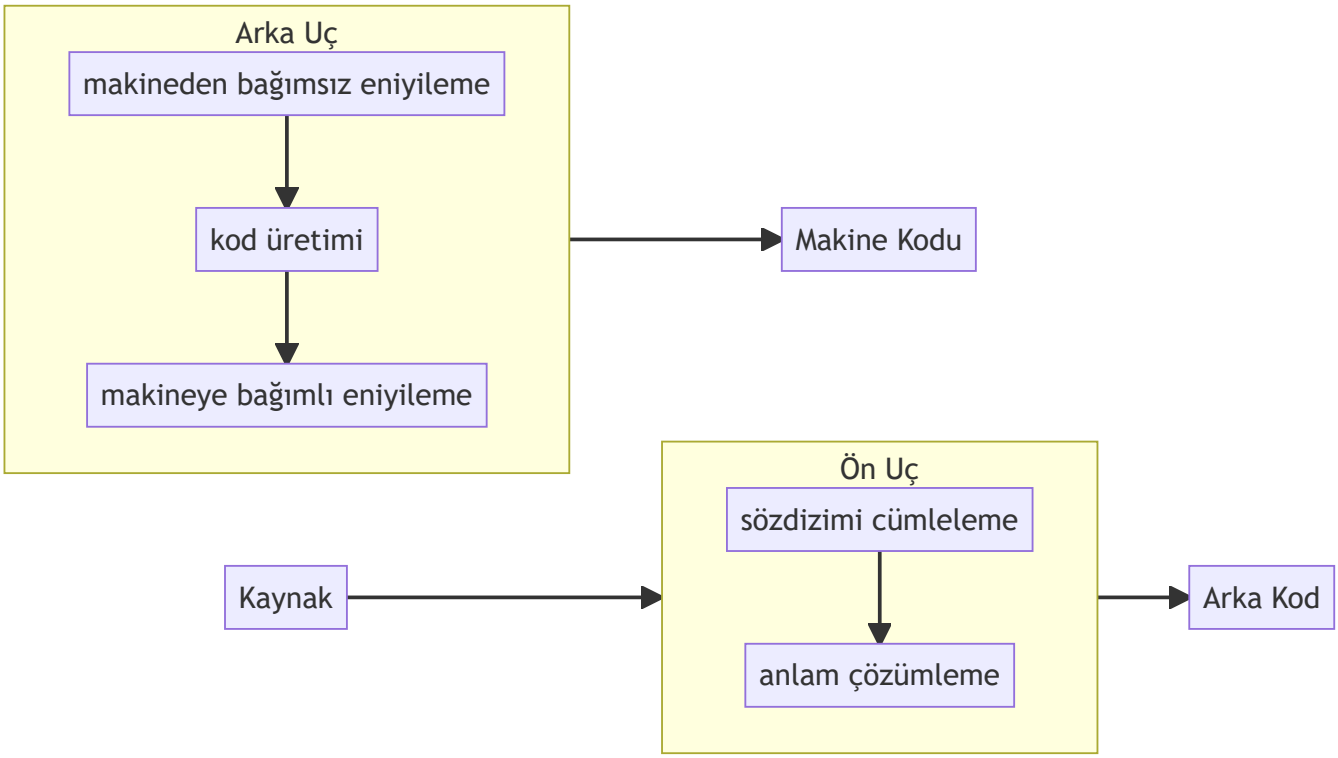
1. Güvenilir olmalı
    1. Yazılabilirlik
    2. Okunabilirlik
    3. Sıra dışı durumları karşılayabilme
2. Bakıma elverişli olmalı
3. Verimli çalışmalı

### 1.3.3. Nesneye Yönelik Programlama

1. **Veri soyutlama:** Yeni veri türlerini modelleyen sınıflar yaratması
2. **Kalıtım:** Sınıfları genişleterek ve özelleştirerek yeni sınıflar türetilmesi
3. **Çok biçimlilik:** Aynı isimdeki işlemlerin değişik nesne sınıfları için farklı algılanması

### 1.3.4. Programlama Ortamı

1. Editör
2. Derleyici (Compiler)
3. Kütüphane (Librarian): Nesne dosyaları
4. Bağlayıcı (Linker): Tüm nesne dosyalarını birleştirerek tek bir yürütülebilir dosya haline getirir
5. Yükleyici (Loader): Yürütülebilir dosyayı diskten belleğe kopyalar
6. Hata ayıklayıcı
7. Yorumlayıcı (Interpreter): Bir programın kaynak kodunu doğrudan satır satır yürüten program




## 1.4. İşletim Sistemleri

### 1.4.1. Bilgisayar Sistemi Bileşenleri

1. Donanım
2. İşletim sistemi
3. Sistem ve uygulama programları
4. Kullanıcılar

### 1.4.2. İşletim Sistemlerinin Temel İşlevleri

1. Yazılım donanım bütünlüğünün sağlanması

2. Kaynakların yönetimi
    1. Görev (task) > İş (job)
3. Kullanıcı ile sistem arasındaki ilişki, uyum ve düzen
    1. Adanmış işlem (dedicated)
    2. Toplu işlem (batch)
    3. Etkileşimli işlem (interactive, time sharing)

### 1.4.3. Sistem Performansı

1. **Verimlilik:** Sistem kaynaklarının düşük kullanımı
2. **Güvenilirlik:** Donanım ve yazılım yüzünden doğacak sorunları sezebilmeli ve önlemeli
3. **Koruyuculuk:** Bir kullanıcının yaptığı hatadan diğer kullanıcı etkilenmemeli
4. **Sezdiricilik**
5. **Elverişlilik:** Kullanıcı sistemin kaynaklarını kendi isteğine göre değil sistemin izin verdiği ölçüde kullanmalı

### 1.4.4. Çekirdek Sistem

1. İşlemciye prosesleri atamak
2. Kesmeleri yönetmek
3. Prosesler arasında iletişimi sağlamak

### 1.4.5. Proses Yönetimi

> **Proses:** kendi veri şablonu olan ve kendi başına bütünlüğü olan kod parçası

1. Çalışıyor (running)
2. Askıda (blocked)
3. Hazır (ready)

> **Zaman Çizelgeleyici (schedular):** İşlemciye atanacak olan prosesi ve prosesin hangi koşullar altında işlemciyi kullanacağını belirler.

1. Proses kuyruğundan yürütülecek prosesi seçer
2. Prosesin kullanım zaman dilimini ayarlar

> **Deadlack:** proseslerin hiç bir zaman ele geçiremeyecekleri bir birim veya kaynağa ihtiyaç duymaları sebebiyle sürekli askıda kalması

> **Semafor:** Bir çeşit değişken. Signal ve Wait'tir.

### 1.4.6. Bellek Yönetimi

| Bellek | Statik/Dinamik | Gerçek/Sanal |
|---|---|---|
| Tekli kesintisiz | S | G |
| Bölümlenmiş | S | G |
| Yer değiştirilebilir bölümlenmiş | D | G |
| Sayfalı | D | G |

| Bellek | Statik/Dinamik | Gerçek/Sanal |
|---|---|---|
| İsteğe bağlı sayfalı | D | S |
| Dilimli | D | S |
| Dilimli ve isteğe bağlı sayfalı | D | S |

# 2. Bilgisayar Mimarisi ve Tasarım

# 3. Bilgisayar Ağları

## 3.1. Ağ Yapısına Giriş

### 3.1.1. Veri İletimi

1. Paralel
2. Seri
3. Asenkron: *START, STOP ve Eşik Biti vardır.*
4. Senkron: *Saat bilgisi*

> Bilgisayar ağları üzerindeki iletişim **seri** iletişimdir.

### 3.1.2. Veri İletim Tipleri

1. BaseBand
2. BroadBand

### 3.1.3. Veri İletim Yöntemleri

1. Simplex (Tek Yön)
2. Half Duplex (Yarı Çift Yön)
3. Full Duplex

### 3.1.4. Veri İletim Ortamları

1. Koaksiyel Kablo
    1. **10Base5-Thicknet:** 10Mbps, BaseBand, 500m, 10mm
        1. Yellow ethernet
        2. 2.5m'de bir siyah bant
        3. Transceiver, AUI bağlantı
    2. **10Base2-Thinnet:** 10Mbps, BaseBand, 185m, 4mm
        1. Black ethernet
        2. 0.5m'de bir ve en fazla 30 cihaz
        3. BNC bağlantı, sonlandırıcı
2. Çift Burgulu (Twisted Pair) Kablo
    1. Kablo tipi
        1. UTP (Unshielded) *100m, 4-6 burgu**
        2. STP (Shielded) *Topraklanmalı*
        3. FTP (Foiled)
        4. S/FTP
    2. Konnektör
        1. RJ-11: *4 pin*
        2. RJ-45: *8 pin; T-568A, T-568B*
    3. Bağlama şekli
        1. Düz (Straight-through)
            1. Hub/Switch ve PC

            2. Switch ve Router
            3. Hub ve Hub *Uplink*
            4. Switch ve Switch
            5. Yazıcı ve Hub/Switch
            6. PC ve Kablo/DSL Modem
        2. Çapraz (Cross-through)
            1. Hub ve Hub
            2. Pc ve Pc
            3. Switch ve Switch
            4. Switch ve Router
            5. Yazıcı ve Hub/Switch
3. Fiber
    1. Fiber Tipine göre
        1. Tek Mod (Single Mode Fiber - SMF): *3km*
        2. Çok Mod (Multi Mode Fiber - MMF): *Dereceli, Kademeli, daha kısa mesafeler*
    2. Kablo Tipine göre
        1. Loose Tube
        2. Tight Buffer
    3. Konnektör
        1. Düşük Insartion Loss: *Konnektörün bağlantısından dolayı sinyal gücü kaybı*
        2. Yüksek Return Loss: *Konnektörden kaynaklı geri yansıma*

### 3.1.5. Haberleşme Yöntemleri

1. Yayın (Broadcast)
    1. Unicast
    2. Multicast
    3. Broadcast
2. Anahtarlamalı (Switching)
    1. Devre Anahtarlama
    2. Paket anahtarlama (Routing)
        1. Statik yönlendirme
            1. En kısa yol
                1. Dijkstra
                2. Bellman-Ford
            2. Taşma: *gelen paketin kopyası geldiği düğüm hariç her yere*
            3. Rastgele: *ilgili yol veri hızı/tüm yol hızı*
            4. Akış durumu
        2. Dinamik yönlendirme
            1. Uzaklık vektörü: *Komşulara gönderir*
            2. Hat durumu: *Herkese gönderir*

### 3.1.6. Ağ Mimarileri

1. Peer-to-Peer
2. Client-Server
    1. 2-tier

    2. 3-tier

### 3.1.7. Sunucu Türleri

1. File server
2. Database Server
3. Transaction Server
4. Web Server
5. ISA (Internet Security and Acceleration Server) / TMG (Threat Management Gateway)
6. Proxy Server

### 3.1.8. Coğrafi Açıdan Ağ Türleri

1. PAN (Personal)
2. LAN
3. MAN (Metropolitan)
4. WAN
5. VPN
6. CAN (Controller)
7. SAN (Storage)

### 3.1.9. Topolojiler

1. Fiziksel
    1. Ortak yol (Bus)
    2. Halka (Ring): *MAU, token*
    3. Yıldız
    4. Örgü (Mesh)
    5. Ağaç (Hierarchical)
2. Mantıksal
    1. Yayın (Broadcast)
    2. Jetonlu Geçiş (Token Passing)

## 3.2. Ağ Bağlantı Cihazları

1. Ağ Arayüz Kartı (NIC-Network Interface Card): *MAC 48 bit*
2. HUB: *Gelen veriyi iletir*
    1. Aktif
    2. Pasif
3. Switch: *Gelen veriyi ilgilisine iletir*
    1. Cut Throught
    2. Store and Forward
4. Repeater
5. Bridge: *Ethernet-Token Ring*
6. Router: *Akıllı switch*
7. Gateway: *Farklı protokolleri bağlar*
8. Transceiver
9. Firewall

10. Access Point: *Kablosuza dönüşüm*
11. Modem

## 3.3. İnternet ve TCP/IP

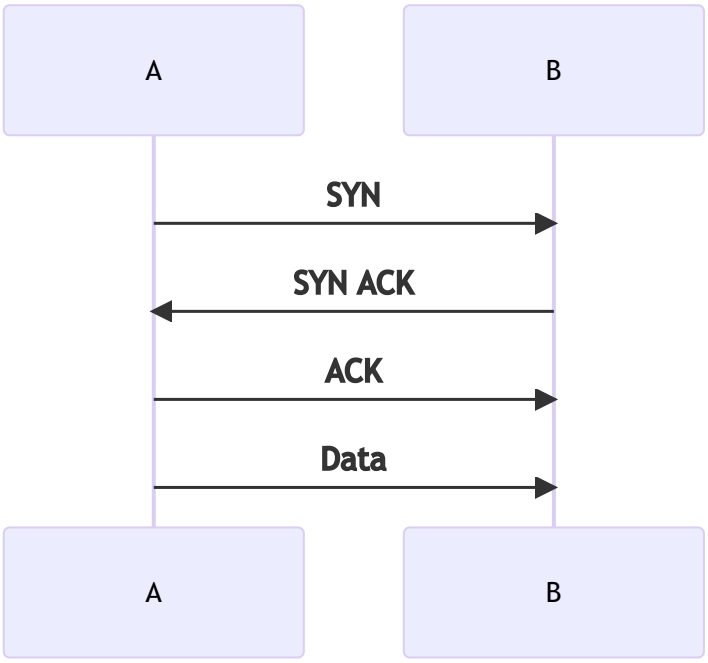
### 3.3.1. Katmanlar

1. Uygulama Katmanı
    1. DHCP (Dynamic Host Configuration)
    2. DNS
    3. HTTP/HTTPS
    4. FTP/TFTP (Trivial)
    5. SMTP (Simple Mail Transfer)
    6. POP (Post Office)
    7. IMAP (Internet Message Access)
    8. IRC (Internet Relay Chat)
    9. SOCKS (SOCKet Secure)
    10. TLS/SSL (Transport Layer Security/Secure Socket Layer)
    11. SSH (Secure Shell)
    12. TELNET
    13. SNMP (Simple Network Management)
    14. NNTP (Network News Transfer)
    15. RMON (Remote Monitoring)
    16. RPC (Remote Procedure Call)
    17. RTP (Remote Transport)
    18. LDAP
    19. NTP (Network Time)
2. Taşıma Katmanı
    1. TCP
    2. UDP
3. İnternet Katmanı
    1. IP
    2. ICMP (Internet Control Message): *ping, traceroute*
    3. IPsec
    4. RIP (Routing Information)
4. Uygulama Katmanı
    1. ARP/InARP (Address Resulation)
    2. RARP (Reverse)
    3. NDP (Neighbour Discovery): *IPv6*
    4. PPP (point-to-point)
    5. SLIP (Serial Line Interface)

### 3.3.2. TCP

| **4** | **6** | **6** | **16** |
|---|---|---|---|
| Source Port | | | Destination Port |

| 4 | 6 | 6 | 16 |
|---|---|---|---|
| Sequance Number | | | |
| Ack Number | | | |
| Header Length | Reserved | TCP Flag | Window Size |
| Checksum | | | Urgent Pointer |
| Options | | | |
| Data | | | |




### 3.3.3. UDP

| 16 | 16 |
|---|---|
| Source Port | Destination Port |
| Length | Checksum |
| Data | |

| | TCP | UDP |
|---|---|---|
| Uygulama Katmanı | Stream | Message |
| Taşıma Katmanı | Segment | Packet |
| İnternet Katmanı | Datagram | Datagram |
| Fiziksel Katman | Frame | Frame |

### 3.3.4. IP

| **4** | **4** | **6** | **2** | **3** | **13** |
|---|---|---|---|---|---|
| Version | Internet Header Length | DSCP | ECN | Total Length | |
| Identification | | | | Flags | FO |
| Time to Live | | Protocol | | Checksum | |
| Source IP Adress | | | | | |
| Destination IP Adress | | | | | |
| Options (if header length > 5) | | | | | |
| Data | | | | | |

**Diffrentiated Services Code Point:** Type of service

**Explicit Congestion Notification:** Ağ tıkanıklığı

**Identification:** veri parçalandığında

**Flags:** DF(Don't fragment): 1,0 ; MF(More Fragments): Arkadan başka veri geliyor mu

**Fragment Offset:** parçanın bütündeki yeri

**Time to Live:** Jump count

**Protocol:** TCP, UDP

# 3.4. Ağ Güvenliği

## 3.4.1. Bilgi Güvenliği

1. Gizlilik
2. Bütünlük
3. Doğruluk
4. Ulaşılabilirlik
5. Kanıt

## 3.4.2. IPsec

1. Protokol
    1. Authentication Header
    2. Encapsulation Security Payload
    3. AH + ESP
2. Bütünlük
    1. MD5: 128 bit anahtar, 512bitlik şifreler
    2. SHA1: 160 bit anahtar, 512bitlik şifreler
3. Kimlik Doğrulama
    1. Preshared Key
    2. Kerberos (Windows AD)
    3. Sertifika Yetkilisi

4. Gizlilik
    1. DES (Data Encryption Standart): 64 bit anahtar, 64bitlik şifreler
    2. 3DES
    3. AES (Advanced Encryption Standart): 128, 192 ve 256 bit anahtar
    4. Seal

### 3.4.3. Saldırı Tespit Sistemleri (IDS-Intrusion Detection Systems)

1. NIDS (Network)
2. HIDS (Host)
3. SIDS (Stack)

### 3.4.4. Şifreleme

1. Simetrik
    1. DES
    2. 3DES
    3. AES
2. Asimetrik
    1. Diffie-Hellman
    2. RSA
3. Anahtarsız
    1. MD5
    2. SHA

# 4. Veri Yapıları ve Algoritmalar

## 4.1. Bilgisayar Yazılım Dünyası

Genel olarak çalışma hızı ile bellek ihtiyacı ters orantılıdır. Hızlı algoritma yavaş olana göre daha fazla bellek ihtiyacı duyar. Algoritma karmaşıklığı iki açıdan ele alınır birisi **zaman karmaşıklığı** diğeri **bellek karmaşıklığı**dır.

## 4.2. Veri Yapıları ve Veri Modelleri

### 4.2.1. Veri Modeli Türleri

1. Bağlantı liste veri modeli
    - Ardışlı erişim
    - Dinamik yaklaşım
    - Arama maliyeti O(n)
    - Ekleme maliyeti O(1)
    - Silme maliyeti O(1) veya O(n)
    - Esnek bellek kullanımı
2. Ağaç veri modeli
    - Hızlı arama
    - Algoritma karmaşıklığı logaritmik
    - Farklı problemlere model olması
    - Dinamik bellek kullanımı
    - Tasarım esnekliği
    - Kod ağacı
3. Graf veri modeli
    - Modelleme esnekliği
    - En kısa yol
    - Greedy yöntemi
    - Kruskal algoritması
    - Graf renklendirme
    - Yol ağacı
    - DFS ve BFS yaklaşımları
    - Sosyal ilişki grafiği
4. Durum makinası veri modeli
    - Davranış modelleme
    - Ardışıl yaklaşım
    - Desen uyuşması
    - Sözce arama
    - Gramer çözümü
    - Turing makinası
5. Veri tabanında ilişkisel veri modeli
    - Bilgilerin düzenli saklanması
    - Hızlı arama/sorgulama
    - Veriler arasında ilişki oluşturulması

    - İnternet tarayıcı ortamında bilgi sorgulama
    - Koruma
    - Verinin arşivlenmesi
6. Ağ veri modeli
    - Ağ üzerinden karşılıklı çalışma
    - IP Paketi
    - İnternet
    - Protokol kümeleri
    - Ağ üzerinden veri aktarımı
    - TCP/IP protokol kümesi

## 4.3. Sıralama Algoritmaları

1. **Araya sokma:** Sıralama süreci boyunca dizi iki parça gibi düşünülür. Sıralı olan ön taraf ve henüz sıralanmamış arka taraf
2. **Seçmeli sıralama:** Önce dizinin ilk elemanı alınır ve daha sonra küme içerisindeki kalan elemanların en küçüğü arınır. Bulunduğu zaman ilk eleman ile karşılıklı yer değiştirilir. En olumlu yanı eğer herhangi bir eleman gerçek yerinde ise yer değiştirme işlemi yapılmamasıdır.
3. **Kabarcık sıralaması:** Sıralanacak elemanlar üzerinde bir yönden diğer yöne doğru ilerlerken komşu iki elemanın yer değiştirmesi işlemine dayanır. Tasarımı basit, ancak etkin bir algoritma değildir.
4. **Birleşmeli sıralama:** Böl ve yönet yaklaşımına dayanır. Rekürsif tasarlanması doğasına uygundur.
5. **Kümeleme sıralaması:** İkili Kümeleme ağacı kurulmasına dayanır. İki yolu vardır birisi bir başka dizi gerektirmeksizin sırasız elemanların bulunduğu dizi üzerinde çalışır diğeri sıralı elemanların olacağı yeni bir dizi gerektirir.
6. **Hızlı sıralama:** Böl ve yönet yaklaşımına dayanır.

## 4.4. Arama Algoritmaları

1. **Ardışıl veya doğrusal arama:** Aranan bulunana kadar komisyon çalışır.
2. **İkili arama:** Köşeye sıkıştırma yöntemi.
3. **Çırpı fonksiyonu:** Çırpı tablosu, çatışmaların çözümlenmesi
4. **Metin içerisinde sözcük arama**

# 5. Java ile Programlama

## 5.1. Veri Tipleri ve Değişkenler

> Her sınıf bir veri tipi, her veri tipi bir sınıftır

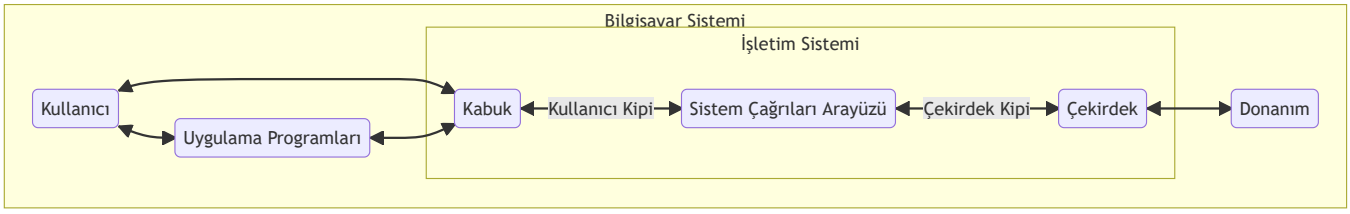
> Ram'e direk kaydedebildiğim veriler ilkeldir. Küçük harf ile başlar.

```
void topla(){
   System.out.println("toplama işlemi yapılıtor...");
}

int topla(){
   return 5;
}

Car car = new Car();
System.out.println(car.door)
/* Null */

System.out.println(-4 / 0)
/* -Infinity */

System.out.println(0 / 0)
/* NaN */

public class Demo {
   static final double AVOGADRO; /* Yanlış */
   static final double PI = 3.14 /* 3,14 değil */
   static int a; /* 0 */
   int b = 1;

   public static void main(String[] args) {
      /* Java kutsal kitabının emri */
      static int c = 3; /* Yanlış */
      b = 4; /* Yanlış */
   }
}

/*Recursive*/
public static int sum(int k) {
   if (k > 0) {
      return k + sum(k - 1);
   } else {
      return 0;
   }
} /* 10 + sum(9) ... 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0 */

/* Construction, Inheritance, Polymorphism, Interface*/
interface SporPaket{
```

```
    void otomatikVites();
    void farSensoru();
    void yagmurSensoru();
    void startStop();
}

abstract class Renault implements SporPaket{
    public void otomatikVites(){
        System.out.println("P, N, D, R");
    }
    abstract void havaYastıgı();
}

class Megane extends Renault{
    public void farSensoru(){
        System.out.println("Aydınlık, Karanlık");
    }
    public void yagmurSensoru(){
        System.out.println("Çise, Normal, Sağanak");
    }
    public void startStop(){
        System.out.println("Çalıştır, Durdur");
    }
    void havaYastıgı(){
        System.out.println("Kaza");
    }
}

class TestDrive{
    public static void main(String args[]){
        Renault r = new Megane();
        r.otomatikVites();
        r.farSensoru();
        r.yagmurSensoru();
        r.startStop();
        r.havaYastıgı();
    }
}
```

# 6. İşletim Sistemleri

## 6.1. Bilgisayarlar ve İşletim Sistemleri




## 6.2. Bilgisayar Mimarileri

| | **Harvard Mimarisi** | **von Neumann Mimarisi** |
|---|---|---|
| Düzenleme | ROM komutlar, RAM veri için. Belleğe erişim yolları bağımsız. | ROM bilgisayarın açılışı, RAM veri ve kodlar için. Belleğe erişim yolları ortak. |
| Donanım Gereksinimi | Karmaşık | Basit |
| Alan Gereksinimi | Çok | Az |
| Komut Çalıştırma Hızı | Yüksek | Düşük |
| Bellek Alan Kullanımı | Çok | Az ve efektif |
| Kontrol Karmaşıklığı | Yüksek | Az |

### 6.2.1. CISC (complex instruction set computer) Mimarisi

CISC mimarisi yüksek seviyeli programlama dillerindeki ifadelerin makine dilinde benzer şekilde doğrudan ifade edilmesine imkan sağlayacak şekilde tasarlanmıştır. Bunun gereği olarak CISC mimarisinde komut yapısı operantlardan en az bir tanesinin birlikte olmasını öngörür ve komutlar birden fazla alt operasyondan oluşurlar.

### 6.2.2. RISC (reduced instruction set computer) Mimarisi

CISC mimarisi kullanıcı odaklı iken RISC mimarisi donanım odaklı bir tasarıma sahiptir. RISC mimarisinde alt operasyonların böl ve yönet mantığına göre bellekten okuma, aritmetik lojik işlem yapma ve belleğe yazma olmak üzere birbirinden bağımsız üç ayrı komik şekilde ifade edilmesi ile ortadan kaldırmıştır. Bu yaklaşımın

sonucu RISC mimarisindeki bütün komutların uzunluğu birbirine eşittir ve aritmetik lojik işlemler sadece saklıcılar içinde gerçekleştirilir.

```
/* CISC */
MULT Addr1, Addr2

/* RISC */
LOAD A, Addr1
LOAD B, Addr2
PROD A, B
STORE Addr1, A
```

| CISC | RISC |
|---|---|
| donanıma yük biner | yazılıma yük biner |
| çok saat döngüsü ile çalışan karmaşık komutlar | tek saat döngüsünde tamamlanabilen basit komutlar |
| az birleştirici kodu | çok birleştirici kodu |
| transistörler karmaşık kod tasarım için kullanılır | transistörler saplayıcılar için kullanılır |

### 6.2.3. İşletim Sistemleri Ağ Özellikleri

**Linux:** Kullanıcı uygulamaları tüm isteklerini soket arabirimi üzerinden gerçekleştirir. Linux ağ katmanı BSD soket katmanı alt yapısı üzerine inşa edilmiştir. Herhangi bir ağ verisi bu katmana bir uygulamanın soketinden veya bir ağ aygıtı sürücüsünden geldiğinde verilerin hangi ağ protokolünü içerdiklerini belirten bir tamamlayıcıyla etiketlenmiş olması beklenir.

**Windows:** Ağ arttırım protokollerini sürücüler üzerinden uygulamaktadır.

1. Sunucu ileti blogu (SMB) protokolü: Sistem üzerinden giriş çıkış isteklerinin gönderilmesi, oturumunda denetimi, sunucudan kaynana yönlendirme bağlantısını başlatan ve sonlandıran, yeniden yönlendirme süreçlerini yöneten, dosyalara erişim kontrollerini yapan, ağ üzerindeki yazılara veri gönderim süreçleri yöneten protokol. Yeni adıı Ortak İnternet Dosya Sistemi (CIFS)
2. İletim Kontrol Protokolü: Basit ağ Yönetim Protokolü (SNMP), dinamik ana bilgisayar yapılandırma protokolü (DHCP) ve Windows İnternet Ad (WINS) gibi hizmetleri içerir.
3. Noktadan noktaya tünel protokolü (PPTP)
4. HTTP
5. Uzaktan Yordan Çağrıları (RPC)/(VPN)
6. Bileşen nesne modeli (COM)/(DCOM)

# 7. Veri Tabanı Yönetim Sistemleri

## 7.1. Veritabanı Yönetim Sistemleri

### 7.1.1. Kavramlar

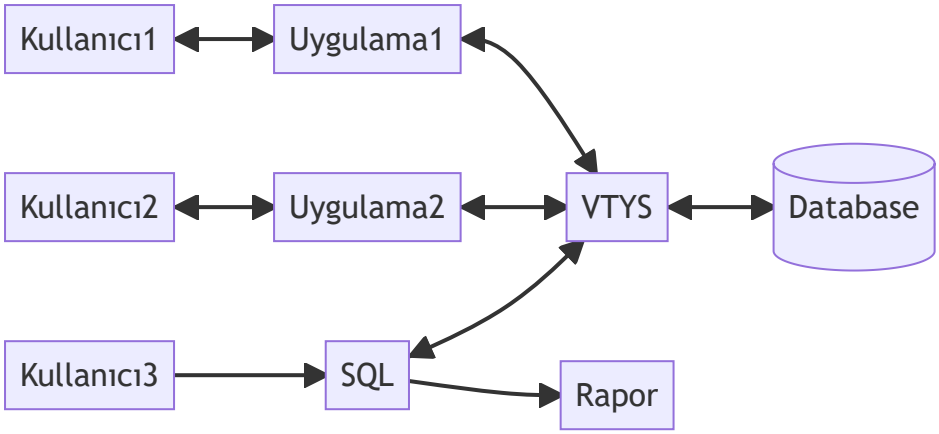
1. Ana veri tabloları (Master)
2. İşlem tabloları (Transaction)
3. Alan (Field) *col*
    1. Anahtar alan (Key)
    2. İkincil anahtar alan(Foreign key)
4. Kayıt (Record) *row*
5. Dataset
6. Index: Tabloların belirtilen alanlara göre sıralanması ile oluşan data setleri ifade eder.

### 7.1.2. Uygulamalar

| **Uygulama** | **Ortam** | **Boyut** | **Yenilik, Özellik** |
|---|---|---|---|
| MS Access | Windows (Lokal) | 25 MB | |
| MySQL | Unix/Linux (Web) | 2 GB | Trigger, Stored Procedure |
| IBM DB2 | IBM | | +Transaction Locking, Yüksek performans, yüksek maliyet |
| Informix | Postgres | | |
| MS SQL | Windows | 4 TB | Maliyetli, Güvenli |
| Postgres | | 64 TB | Veri güvenliği ön planda, açık kaynak |
| Oracle | | Sınırsız | Yüksek güvenlik, yüksek maliyet |
| Sybase | | | Orta ve büyük ölçekli banka ve kamusal alan |
| BerkeleyDB | Gömülü | 256 TB | |
| SQLite | Native | | |
| NoSQL | | | İlişkisel olmayan veriler için (Not Only SQL) Çok yüksek performans |

### 7.1.3. Yaklaşımlar

1. Geleneksel yaklaşım *(dosyalama)*
2. Veritabanı yaklaşımı




## 7.2. Veri Modelleri

1. Hiyerarşik *bire-çok*
2. Ağ *çoka-çok*
3. Nesneye yönelik: Karmaşık verileri kolay saklar, Büyük verinin işlenmesi yavaş. *Java*
4. İlişkisel
5. Varlık-İlişki modeli

## 7.3. Sorgulama

```
SELECT DISTINCT *, alan_1 AS "Alan Adı"
FROM tablo
WHERE NOT koşul AND koşul OR koşul --önce AND
ORDER BY alan DESC --varsayılan ASC
GROUP BY alan
HAVING grup_koşulu

--bir veri kümesi içinde arama
WHERE koşul NOT IN (1, 2, 3)

--arasında
WHERE alan BETWEEN 5 AND 10

--arama
WHERE alan LIKE '_a%'
```

## 7.4. Fonksiyonlar

> ** veya ^ üs alma işlemi

> **Tarih + Sayı** gün çıkarır. Sonuç tarihsel değildir.
> **Tarih1 + Tarih2** iki tarih arasındaki gün sayısı. Sonuç sayısal

### 7.4.1. Tek Satır

#### 7.4.1.1. Karakter

1. Lower
2. Upper
3. Inıtcap
4. Concat
5. Substring('Metin', kaçKarakter, kaçıncıKarakter)
6. Len
7. CharIndex('Metin', 'ArananMetin', kaçıncıKarakterdenİtibaren)
8. Left('Metin', kaçKarakter)
9. Right('Metin', kaçKarakter)
10. LTrim
11. RTrim
12. Replace('Metin', 'Bul', 'Değiştir')
13. Reverse

#### 7.4.1.2. Genel

1. NVL/ISNULL(Ucret, 0)
2. DECODE (Cinsiyet, 'E', 'ERKEK', 'K', 'KADIN', 'GAY') AS Cinsiyet
3. CASE Cinsiyet, WHEN 'E', THEN 'ERKEK', WHEN 'K', THEN 'KADIN', ELSE 'GAY'

# 8. Yazılım Mühendisliği

## 8.1. Yazılım Mühendisliğine Giriş

### 8.1.1. Giriş

1. Yazılım mühendisliği yazılım üretiminin tüm yönleriyle ilgili bir mühendislik disiplinidir.
2. Yazılım sadece bir program veya programlar değildir aynı zamanda sistemin kullanıcılarının kalite güvence personelinin ve geliştiricilerin ihtiyacı olan tüm elektronik dokümantasyonu içerir yazılımlarda bakım kolaylığı güvenilirlik ve güvenlik etkinlik ve kabul edilebilirlik bulunması şart olan özelliklerdendir
3. Yazılım süreci yazılım geliştirme de yer alan tüm etkinlikleri içerir spesifikasyon geliştirme doğrulama ve evrimin bütün etkinlikleri tüm yazılım süreçlerinin bir parçasıdır
4. Bir çok farklı sistem türü vardır ve her biri geliştirilmesi için uygun yazılım mühendisliği araçları ve tekniklerine gerek duyar eğer varsa çok az sayıda özel tasarım ve gerçekleştirim teknikleri tüm sistem çeşitlerine uygulanabilirdir
5. Yazılım mühendisliğinin temel fikirleri bütün yazılım sistem türlerine uygulanabilirdir bu temeller yönetilen yazılım süreçlerini yazılım güvenilebilirliği ve güvenliğini gereksinim mühendisliğinin ve yazılım yeniden kullanımını içerir
6. Yazılım mühendislerinin mühendislik mesleğine ve topluma karşı sorumlulukları vardır sadece teknik konularla ilgilenmemeli kendi işlerini etkileyen etik konuların farkında olmalıdırlar
7. Mesleki topluluklar etik ve mesleki standartları içeren davranış kuralları yayınlarlar bunlar kendi üyelerinden beklenen davranış standartlarını belirlerler

### 8.1.2. Yazılım Süreçleri

1. Yazılım süreçleri bir yazılım sisteminin üretilmesinde yer alan etkinliklerdir yazılım süreç modelleri bu süreçlerin soyut temsilleridir
2. Genel süreç modelleri yazılım süreçlerinin nasıl düzenlendiğini tanımlar bu genel modellere örnek olarak Çağlayan modeli artımlı geliştirme ve yeniden kullanılabilir bileşen konfigürasyonu ve bütünleştirme verilebilir
3. Gereksinim Mühendisliğe bir yazılım spesifikasyonunu geliştirme sürecidir spesifikasyonlar ile müşterinin sistem gereksinimlerinin geliştiricilere bildirilmesi amaçlanır
4. Tasarım ve gerçekleştirme süreçleri gereksinim spesifikasyonunun çalıştırılabilir bir yazılım sistemine dönüştürülmesi ile ilgilidir
5. Yazılım geçerleme sistemin spesifikasyonuna uygun olduğunun ve sistem kullanıcılarının gerçek ihtiyaçlarını karşıladığını kontrol edilmesi sürecidir
6. Yazılım evrimi yeni gereksinimlerin karşılanması amacıyla mevcut yazılım sistemlerini değiştirilmesidir değişim sürekli olduğundan yazılımın faydalı olmayı sürdürebilmesi için evrim geçirmesi gerekir
7. Süreçler değişimle baş etmeye yönelik etkinlikler içerir gereksinim ve tasarımla ilgili yetersiz kararların önlenmesine yardımcı olacak bir prototipleme evresi süreçlere dahil edilebilir yinelemeli geliştirme ve teslimat için yapılandırılan süreçlerde değişiklikler sistemin tümünü aksatmadan gerçekleştirebilir
8. Süreç iyileştirme yazılım kalitesinin artırılması ve geliştirme maliyetleri ve sürenin azaltılması amacıyla mevcut yazılım süreçlerinin iyileştirilmesi sürecidir süreç ölçümü analizi ve değişimini içeren döngüsel bir süreçtir

### 8.1.3. Çevik Yazılım Geliştirme

1. Çevik yöntemler süreç ve belgeleme yükünü azaltmaya ve artırımlı yazılım teslimine odaklanan yinelemeli geliştirmeye yönelik yöntemlerdir bu yöntemlerde müşteri temsilcileri geliştirme sürecinin içinde yer alır
2. Geliştirme için Çevik mi yoksa plan güdümlü yaklaşımlar mı kullanılması gerektiği kararı geliştirilecek projenin türüne geliştirme takımının yeteneklerine ve sistemin geliştirdiği şirketin kültürüne göre verilir pratikte Çevik ve plan güdümlü yaklaşımlardan oluşan bir karma sistem kullanılabilir.
3. Çevik geliştirme pratikleri kullanıcı öyküleri şeklinde ifade edilmiş gereksinimleri eş programlamayı yeniden üretimi sürekli bütünleştirmeyi ve test önce geliştirmeyi içerir.
4. Scrum Çevik projeleri organize etmeye olanak sağlayan bir Çevik yöntemdir. sistem artımlarının geliştirildiği sabit zaman periyotlarının bir dizi Sprint etrafında bir araya gelmesinden oluşur planlama iş listesinin önceliklendirilmesine ve en öncelikli işlerin bir Sprint için seçilmesine dayanır.
5. Çevik yöntemleri ölçeklendirmek için bazı plan güdümlü pratiklerin Çevik pratiklerle bütünleştirilmesi gerekir. Bunlar ön gereksinimleri çok sayıda müşteri temsilcisini daha fazla belgelemeyi proje takımları arasında ortak araçların kullanımını ve yayımların zamanının takımlar arasında eşitlenmesini içerir.

### 8.1.4. Gereksinim Mühendisliği

1. Bir yazılım sistemi için gereksinimler sistemin ne yapması gerektiğini ve işlemleri ve gerçekleştirimi üstündeki kısıtlamaları belirtir
2. Fonksiyonel gereksinimler sistemin sağlaması gerekli servislerin spesifikasyonu ya da bazı hesaplamaların nasıl yapılacağının açıklamasıdır.
3. Fonksiyonel olmayan gereksinimler genellikle geliştirilecek sistemi ve kullanılacak geliştirme sürecini kısıtlar Bunlar ürün gereksinimleri organizasyonel gereksinimler ya da dış gereksinimler olabilir genellikle sistemin ortaya çıkan özellikleri ile ilgilidirler ve bu nedenle sisteme bir bütün olarak uygulanırlar
4. Gereksinim mühendisliği süreci gereksinimlerin açığa çıkarılmasını gereksinimlerin spesifikasyonunu gereksinimlerin doğrulanmasını ve gereksinim yönetimini içerir
5. Gereksinimlerin açığa çıkarılması gereksinimlerin bulunması gereksinimlerin sınıflandırılması ve düzenlenmesi gereksinimlerin görüşülmesi ve gereksinimlerin belgelendirilmesi gibi eylemler Sarmalı ile gösterilebilen yinelemeli bir süreçtir
6. Gereksinim spesifikasyon süreci kullanıcı ve sistem gereksinimlerinin resmen belgelendirilmesi ve yazılım gereksinimleri dökümanının yaratılması sürecidir
7. Yazılım gereksinimleri dökümanı sistem gereksinimlerinin üzerinde anlaşılmış anlatımıdır sistem müşterilerinin ve yazılım geliştiricilerinin kullanabileceği gibi düzenlenmelidir
8. Gereksinim doğrulama gereksinimlerin geçerliklerinin tutarlılıklarının bütünlüklerinin gerçekliklerinin ve doğrulanabilir olduklarının kontrol edilmesi sürecidir
9. İş organizasyonel ve teknik değişiklikler kaçınılmaz olarak yazılım sistemi gereksinimlerinin değişmesine yol açar gereksinim yönetimi bu değişiklikleri yönetme ve kontrol etme sürecidir.

### 8.1.5. Sistem Mühendisliği

1. Bir model bir sistemin bazı detaylarını kasıtlı biçimde görmezlikten gelen soyut bir görünümüdür bir sistemin bağlamını etkileşimlerini yapısını ve davranışını göstermek için birbirini tamamlayan modeller geliştirilebilir
2. Bağlam modelleri içinde başka sistemler ve süreçler olan bir ortamda modellenen sistemin nasıl konumlandırıldığını gösterirler geliştirilmesi istenen sistemin sınırlarının çizilmesine yardım ederler

3. Kullanım durumu diyagramları ve sıra diyagramları kullanıcılar ile tasarlanan sistem arasındaki etkileşimleri Betimlemek için kullanılırlar kullanım durumları bir sistem ile dışsal aktörler arasındaki etkileşimleri göstermek için kullanılırlar sıra diyagramları ise nesneler arasındaki etkileşimleri göstererek buna katkıda bulunurlar
4. Yapısal modeller bir sistemin mimarisini ve organizasyonunu gösterirler sınıf diyagramları bir sistemdeki sınıfların ve bağlantılarının statik yapılarını tanımlamak için kullanılır
5. Davranışsal modeller çalışan bir sistemin dinamik davranışını Betimlemek amacıyla kullanılırlar bu davranış sistem tarafından işlenen verinin perspektifinden veya sistemin tepkilerini tetikleyen olayların perspektifinden modellenir
6. Etkinlik diyagramları verilerin işlenmesini modellemek de kullanılabilir ki bu durumda her bir etkinlik bir işlem adımına karşılık gelir
7. Durum diyagramları iç ve dış olaylara tepki olarak sistemin davranışını modellemekte kullanılır
8. Model güdümlü Mühendislik herhangi bir sistemin çalıştırılabilir Koda otomatik olarak dönüştürülebilecek bir modeller kümesi olarak temsil edildiği bir yaklaşımdır

### 8.1.6. Mimari Tasarım

1. Bir yazılım mimarisi bir yazılım sisteminin nasıl yapılanması gerektiğinin bir anlatımıdır bir sistemin performans güvenlik ve erişilebilirlik gibi özellikleri kullanılan mimariden etkilenir
2. Mimari tasarım kararlarından uygulamanın türü sistemin dağıtıklığı kullanılabilecek mimari stiller mimarinin belgelendirme ve değerlendirilme yöntemleri dahildir
3. Mimariler çeşitli perspektifler veya görünümlerden belgelenebilir mümkün olan görünümler içinde kavramsal görünüm mantıksal görünüm süreç görünümü geliştirme görünümü ve fiziksel görünüm vardır
4. Mimari Desenler jenerik sistem mimarileri alanındaki bilgileri kullanmanın bir aracıdır mimariyi betimler ne zaman kullanılabileceğini açıklar avantaj ve dezavantajlarına işaret eder
5. Yaygın kullanılan mimari Desenler arasında model-görünüm-kumanda, katmanlı mimari, ambar, istemci-sunucu ve boru ve süzgeç vardır
6. Uygulama sistemlerinin jenerik modelleri uygulamaların çalışma biçimini anlamamıza aynı tipten uygulamaları karşılaştırmamıza uygulama sistem tasarımlarını geçerlememize ve büyük çaplı bileşenleri yeniden kullanım açısından değerlendirmemize yardımcı olurlar
7. Hareket işleme sistemleri bir veri tabanındaki bilgiye birçok kullanıcı tarafından uzaktan erişilmesini ve bu bilgiler üzerinde değişiklik yapılabilmesini sağlayan etkileşimli sistemlerdir bilgi sistemleri ve kaynak yönetimi sistemleri hareket işleme sistemlerinin örnekleridir
8. Dil işleme sistemleri bir dilden diğerine çeviri yapmak ve bir girdi dilde ifade edilen Komutları yerine getirme amacıyla kullanılırlar bir çevirmen ve üretilen Dildeki Komutları yerine getiren bir soyut makine içerirler

### 8.1.7. Tasarım ve Geliştirme

1. Yazılım tasarımı ve gerçekleştirimi dönüşümlü etkinliklerdir tasarımdaki Detay düzeyi geliştirilmekte olan sistemin türüne ve plan güdümlü mü yoksa çevik bir yaklaşım mı kullanıldığına bağlıdır
2. Nesne yönelimli tasarım süreci sistem mimarisinin tasarlamak sistemdeki nesneleri belirlemek farklı nesne modelleri kullanarak tasarımı tanımlamak ve bileşen ara yüzlerini belgelemek için gerekli etkinlikleri içermektedir
3. Bir nesne yönelimli tasarım süreci sırasında bir dizi farklı model üretilebilir bu modeller statik modelleri (sınıf modelleri genelleştirme modelleri ilişki modelleri) ve dinamik modeller (sıra modelleri durum

makinesi modelleri) içermektedir
4. Diğer nesnelerin onları kullanabilmesi için bileşen ara yüzleri açık olarak tanımlanmalıdır bir UML ara yüz stereotipi ara yüzleri tanımlamak için kullanılabilir
5. Yazılım geliştirirken var olan yazılımları bileşen servis veya bütün sistem olarak kullanma olasılığını her zaman Düşünmeniz gerekmektedir
6. Konfigürasyon yönetimi Gelişmekte olan bir yazılım sistemindeki değişimleri yönetmenin sürecidir bir takımın yazılım geliştirmek için işbirliği yaptığı durumlarda kullanılması gereklidir
7. Çoğu yazılım geliştirme Konakçı Hedef geliştirmedir Konakçı bilgisayarda bir EGO kullanılarak geliştirilen yazılım işletmek için hedeflenen bilgisayara transfer edilmektedir
8. Açık Kaynak geliştirme bir sistemin kaynak kodunu herkese açık hale getirmeden oluşmaktadır bu birçok kişinin yazılıma değişiklikler ve geliştirmeler önerebilmesi anlamına gelmektedir

### 8.1.8. Yazılım Testi

1. Test etme sadece programdaki hataların varlığını gösterebilir programda Hiç hata kalmadığını gösteremez
2. Geliştirme testi yazılım geliştirme takımının sorumluluğudur sistemin müşterilere sunulmasından önce test edilmesi ayrı bir takımın sorumluluğu olmalıdır kullanıcı testleri sürecinde müşteriler veya sistem kullanıcıları test verileri sağlarlar ve testlerin başarılı olup olmadığını kontrol ederler
3. Geliştirme testi bireysel nesne ve metotların test edildiği birim testini birbiriyle ilişkili nesne gruplarının test edildiği bileşen testini ve sistemin kısmen veya tamamen test edildiği sistem testini kapsar
4. Sistemi test ederken amacınız yazılımı çöktürmektir
5. Mümkün olan her aşamada otomatikleştirilmiş testler yazmalısınız bu şekilde testler sistemde bir değişiklik yapıldığı tüm zamanlarda çalıştırılabilecek bir programın içine yerleştirilmiş olacaktır
6. Önce test geliştirme metodu test edilecek kodlardan önce testlerin yazıldığı bir geliştirme yaklaşımıdır tüm testler başarılı bir şekilde sonuçlanana kadar küçük kod değişiklikleri ve kodun yapısında küçük değişiklikler yapılır
7. Senaryo testi sistemin pratik kullanımının bir kopyasını yarattığından kullanışlıdır senaryo testi sistemin normal kullanım durumunun yaratılmasını ve bu senaryo kullanılarak test takımlarının elde edilmesini içerir
8. Kabul testi yazılımın kullanım için yeterince iyi olup olmadığına ve planlanan işlem ortamında kullanılıp kullanılmayacağına karar vermeyi amaçlayan bir kullanıcı test sürecidir

### 8.1.9. Yazılım Evrimi

1. Yazılım geliştirme ve evrim spiral bir model ile gösterilebilecek bütünleşik ve yinelemeli bir süreç olarak düşünülebilir
2. Müşteriye özel geliştirilen sistemler için genellikle yazılım bakım maliyetleri yazılım geliştirme maliyetlerinden fazladır
3. Yazılım Evrim süreci değişiklik istekleri tarafından yönetilir ve değişiklik etki analizi sürüm planlama ve değişiklik gerçekleştirimini içerir
4. Kalıt sistemler artık kullanılmayan yazılım ve donanım teknolojileri ile geliştirilmiş olan ve iş hayatı için hala yararlı olan daha eski yazılım sistemleridir
5. Çoğunlukla bir kalıt sistemin bakımını yapmak modern teknoloji ile onun yerine geçebilecek bir sistem geliştirmekten daha ucuz ve daha az risklidir
6. Bir sistemin bakımının mı yapılmasına ya da dönüştürülmesine ya da değiştirilmesine karar vermek için kalıt sistemin iş hayatındaki değeri ve uygulama yazılımının ve içinde bulunduğu ortamın kalitesi

değerlendirilmelidir
7. Üç tür yazılım bakımı vardır hata düzeltme yazılımın yeni bir ortamda çalışması için değiştirilmesi ve yeni veya değişen gereksinimlerin gerçekleştirimi
8. Yeniden yazılım mühendisliği yazılımın anlaşılmasını ve değiştirilmesini daha kolaylaştırmak için yeniden yapılandırılması ve yeniden belgelendirilmesi ile ilgilidir
9. Yeniden üretim fonksiyonelliği koruyan küçük program değişiklikleri yapmak önleyici bakım olarak düşünülebilir

# 8.2. Güvenilebilirlik ve Güvenlik

## 8.2.1. Güvenilebilir Sistemler

1. Sistem güvenilebilirliği önemlidir Çünkü kritik bilgisayar sistemlerinin arızalanması büyük ekonomik kayıplara ciddi bilgi kaybına fiziksel hasara veya insan hayatına olabilecek tehditlere yol açabilir
2. Bilgisayar sisteminin güvenilebilirliği sistemdeki kullanıcının Güven düzeyini yansıtan bir sistem özelliğidir güvenilebilirliğin en önemli boyutları erişilebilirlik güvenilirlik emniyet güvenlik ve dayanıklılıktır
3. Sosyoteknik sistemler bilgisayar donanımı yazılım ve insanları içerirler ve bir organizasyon içinde yer alırlar kurumsal veya iş dünyasına yönelik hedef ve amaçları desteklemek için tasarlanmışlardır
4. Bir sistemdeki hataların en aza indirilebilmesi için güvenilebilir tekrarlanabilir bir süreç kullanımı çok önemlidir süreç gereksinim tanımlamalarından sistem uygulamasına kadar her aşamada doğrulama ve geçerleme faaliyetlerini içermelidir
5. Donanımda yazılım süreçlerinde ve yazılım sistemlerinde fazlalık ve çeşitlilik kullanımı güvenilebilir sistemlerin gelişimi için çok önemlidir
6. Bir sistemin biçimsel bir modelinin geliştirmede temel olarak kullanılan biçimsel yöntemler sistem içerisindeki betimleme ve uygulama hata sayısını azaltmaya yardımcı olur ancak biçimsel yöntemler maliyet etkinliği ile ilgili endişeler nedeniyle endüstride sınırlı bir ilgi görmüştür

## 8.2.2. Güvenilirlik Mühendisliği

1. Yazılım güvenilirliği kusurların sisteme eklenmesinden kaçınarak sistem dağıtılmadan önce kusurların bulunup giderilmesi ile ve Kusura toleranslı yöntemler ekleyerek kusur arızaya sebep olduktan sonra sistemin çalışmaya devam etmesi ile elde edilebilir
2. Güvenilirlik gereksinimleri sistem gereksinim tanımlamalarında sayısal olarak tanımlanabilir güvenilirlik ölçümleri talep üzerine arıza olasılığı (TÜAO) arızanın ortaya çıkma oranı (AAO) ve erişilebilirliği (ERİS) içermektedir
3. Fonksiyonel güvenilirlik gereksinimleri fonksiyonel-olmayan güvenilirlik ihtiyaçlarını karşılamaya yardımcı olan kontrol ve yedeklilik gereksinimleri gibi sistem işlevleridir
4. Güvenilebilir sistem mimarileri Kusura toleranslı olarak tasarlanan sistem mimarileridir koruma sistemleri kendini izleyen mimariler ve N-sürümlü programlama dahil olmak üzere bir dizi mimari Stil Kusura toleransı destekleme imkanı sağlar
5. Her sürümün birbirinden tamamen bağımsız olmasını sağlamak pratik olarak neredeyse imkansız olduğu için yazılım çeşitliliğini sağlamak zordur
6. Güvenilir programlama girdilerin ve program sabitlerinin doğruluğu kontrol edilirken programda yedekliliğin sağlanmasına dayanır
7. Yazılım güvenilirliğini tahmin etmek için istatistiksel test kullanılır bu test operasyonel kullanımdaki profile uygun yazılıma kullanım sırasında gelen girdilerin dağılımını yansıtan test verilerini kullanmaya

dayanır

### 8.2.3. Emniyet Mühendisliği

1. Emniyet kritik sistemler hata yapmaları durumunda insan hayatına zarar verebilecek ve hatta ölüme sebep olabilecek sistemlerdir
2. Emniyet kritik sistemlere ait emniyet gereksinimlerini anlamak için risk temelli yaklaşım kullanılabilir olası riskler ve tehlikeler belirlenip hata ağacı analizi gibi yöntemler kullanılarak bunların ayrıştırılması sonucunda esas nedenler keşfedilebilir sonra bu problemlerden kaçınmak veya problem oluştuğunda toparlanmak için gerekli gereksinimler belirlenir
3. Emniyet kritik sistem geliştiriminde eksiksiz ve net tanımlanmış sertifikalandırılmış bir sürecin izlenmesi önemlidir
4. Statik analiz bir onaylama ve doğrulama yaklaşımı olup hatalar ve anomalileri bulmak için bir sistem kaynak kodunu inceler
5. Model denetimi biçimsel bir yöntem olup kapsamlı bir yaklaşım ile olası tüm durumları potansiyel hataları bulmak üzere kontrol eder
6. Emniyet ve güvenilirlik gerekçeleri sistemin emniyetli ve güvenilir olduğunu gösteren kanıtları bünyesinde Barındırır dış düzenleyiciler sistemleri sertifikalandırma durumunda kaldıklarında bu emniyet gerekçelerine ihtiyaç duymaktadır

### 8.2.4. Güvenlik Mühendisliği

1. Güvenlik mühendisliği bilgisayar Temelli sisteme veya onun verisine zarar vermeye çalışan kötü niyetli saldırılara karşı koyabilecek yazılım sistemlerinin geliştirilmesine ve bakımına odaklanır
2. Güvenlik tehditleri sisteme veya onun verisine ait gizlilik bütünlük ve erişilebilirlik özelliklerine yönelik tehditlerdir
3. Güvenlik risk yönetimi sisteme yönelik saldırılardan oluşabilecek kayıpların değerlendirilmesini ve bu kayıpları ortadan kaldıracak veya azaltacak güvenlik gereksinimlerinin belirlenmesini içerir
4. Güvenlik gereksinimlerini oluşturmak için korunacak varlıkları belirlemek ve güvenlik teknik ve teknolojilerinin bu Varlıkları nasıl koruyacağını tanımlamak gereklidir
5. Güvenli sistem mimarisi tasarlanırken dikkat edilmesi gereken konular temel varlıkları koruyacak sistem yapısının organize edilmesi ve başarılı bir saldırı karşısında kayıpları azaltmak için sistem varlıklarının dağıtılmasını içerir
6. Güvenlik tasarım ilkeleri sistem tasarımcılarının normal koşullarda düşünemedikleri güvenlik konularına dikkat çekmek için hazırlanmıştır bu ilkeler güvenlik denetim listeleri içinde de bir temel teşkil eder
7. Güvenlik doğrulaması zordur Çünkü güvenlik gereksinimleri neyin gerçekleşmesinden ziyade neyin gerçekleşmemesi gerektiğini tanımlar Buna ek olarak saldırganların akıllı olduğunu ve sistem zayıflıklarını aramak için güvenlik testi için ayrılan zamandan daha fazla zamanları olduğunu Unutmamak gerekir

### 8.2.5. Hayatta Kalma

1. Bir sistemin hayatta kalma etkinliği o sistemin arızalar veya Siber saldırılar gibi bozucu olaylar karşısında kritik servislerini ne kadar iyi sürdürebildiğine ilişkin bir yargıdır
2. Hayatta kalma etkinliği fark etme karşı koyma kurtarıp geri döndürme ve yeniden başlatma modeline dayanır
3. Hayatta kalma planlaması bilgisayar ağı ile bağlı sistemlerin içeriden ve dışarıdan Siber saldırılara maruz kalacağı ve bu saldırıların bazılarının başarılı olacağı varsayımına dayanmalıdır

4. Sistemler farklı türlerde bir dizi savunma katmanları şeklinde tasarlanmalıdır bu katmanlar Etkin çalışır ise insan hataları ve teknik arızalar ile Siber saldırılar fark edilebilir ve karşı konulabilir
5. Sistem yöneticileri ve operatörlerinin problemler ile başa çıkmasını sağlayabilmek için süreçler esnek ve uyarlanabilir olmalıdır süreç Otomasyonu kişilerin problemlerle başa çıkmasını zorlaştırabilir
6. Hayatta kalan sistemlerin tasarımında işe ilişkin hayatta kalma gereksinimleri başlangıç noktasıdır sistemin hayatta kalmasını sağlamak için problemleri fark etmeye ve onlardan kurtulup Geri dönmeye kritik servis ve varlıkların geri döndürülmesine ve sistemin yeniden başlatılmasına odaklanmalısınız
7. Hayatta kalma etkinliği için Tasarımın önemli bir parçası kritik servislerin belirlenmesidir kritik servisler sistemin varlık nedenini gerçekleştirmek için gerek olan servislerdir sistemler bu servisler korunacak ve bir arıza durumunda bu servisler hızlıca geri döndürecek şekilde tasarlanmalıdır

# 8.3. Yazılım Yönetimi

## 8.3.1. Proje Yönetimi

1. Yazılım mühendisliği projelerinin zamanında ve bütçe dahilinde gerçekleştirilmesi için iyi yazılım proje yönetimi esastır
2. Yazılım yönetimi diğer Mühendislik alanlarındaki yönetimden farklıdır yazılım soyuttur projeler yeni ya da yenilikçi olabilir Böylece onların yönetimini yönlendirmek için bir deneyim birikimi yoktur yazılım süreçleri geleneksel Mühendislik süreçleri gibi olgun değildir
3. Risk yönetimi risklerin ortaya çıkma olasılıklarını ve o risk ortaya çıkarsa proje için sonuçlarını belirleyecek şekilde ana proje risklerini tanımlamayı ve değerlendirmeyi içerir olası riskler ortaya çıkarsa ya da ortaya çıktığı zaman onlardan kaçınmak onları yönetmek ve onlarla başa çıkmak için planlar yapmalısınız
4. İnsan yönetimi bir projede çalışacak doğru insanları seçmeyi ve mümkün olduğunca verimli olacakları şekilde takımı ve çalışma ortamını düzenlemeyi içerir
5. İnsanlar diğer insanlarla etkileşimle yönetim ve Arkadaşları tarafından kabul görmeleri ile ve kişisel gelişim için verilen fırsatlarla motive olmaktadır
6. Yazılım geliştirme grupları oldukça küçük ve uyumlu olmalıdır bir grubun etkinliğini etkileyen önemli faktörler o gruptaki insanlar bu grubun düzenlenme biçimi ve grup üyeleri arasındaki iletişimdir
7. Bir grup içindeki iletişim bu grup üyelerinin durumu grubun büyüklüğü grubun cinsiyet dağılımı kişilikler ve mevcut iletişim kanalları gibi faktörlerden etkilenmektedir

## 8.3.2. Proje Planlaması

1. Bir sistem için ödenecek ücret sadece tahmini geliştirme maliyetlerine ve geliştirme şirketi tarafından gereksinim duyulan kâra bağlı değildir kurumsal faktörler fiyatın artan riski telafi etmek için artırılması veya rekabet avantajı kazanmak için azaltılması anlamına gelebilir
2. Yazılım genellikle bir ihale kazanmak için fiyatlandırılır ve sonra sistemin fonksiyonelliği tahmini fiyatı karşılamak için ayarlanır
3. Plan güdümlü geliştirme proje etkinliklerini planlanan çabayı etkinlik zaman planını ve her etkinlik için kimin sorumlu olduğunu tanımlayan tam bir proje planı etrafında düzenlenmiştir
4. Proje zaman planlaması proje planının bir parçası olarak çeşitli grafiksel gösterimlerin oluşturulmasını içerir etkinlik süresini ve istihdam zaman planlarını gösteren Çubuk grafikler en sık kullanılan Zaman planı temsilleridir
5. Bir proje kilometre taşı öngörülebilir bir etkinlik sonucu ya da etkinlikler kümesidir her kilometre taşında biçimsel bir ilerleme raporu yönetime sunulmalıdır bir çıktı proje müşterisine teslim edilen bir çalışma

ürünüdür
6. Çevik planlama oyunu tüm takımın proje planlamasına dahil edilmesini gerektirir plan artımı olarak geliştirilir ve problemler ortaya çıkarsa bir artımın teslimatının geciktirilmesi yerine yazılımın fonksiyonelliği azaltılır
7. Yazılım için tahmin teknikleri yöneticilerin gereken çabayı değerlendirdiği yerde tecrübeye dayalı ya da gereken çabanın diğer tahmini proje parametrelerinden hesaplandığı yerde algoritmik olabilir
8. COCOMO II maliyet modeli bir maliyet tahminini formüle ederken projeyi ürünü donanımı ve personel özelliklerini dikkate alan olgun bir algoritmik maliyet modelidir

### 8.3.3. Kalite Yönetimi

1. Yazılım kalite yönetimi yazılımın düşük sayıda kusur içermesinin ve bakım yapılabilirlik güvenilirlik taşınabilirlik ve benzeri gerekli standartlara ulaşmasını Garanti etmekle ilgilidir bu süreç ve ürün standartlarının tanımlanmasını ve bu standartların takip edilip edilmediğini kontrol etmek için süreçlerin oluşturulmasını içerir
2. Yazılım standartları en iyi uygulama tanımını gösterdiği için kalite güvencesi için önemlidir yazılım geliştirilirken standartlar iyi kalitede yazılım oluşturmak için sağlam bir temel sağlar
3. Yazılım süreci çıktıları ile ilgili incelemeler kalite standartlarının takip edildiğini kontrol eden bir ekip gerektirir yorumlar Kaliteyi değerlendirmek için en çok kullanılan tekniktir
4. Bir program değerlemesinde veya ikili inceleme sırasında küçük bir ekip sistematik olarak kodu denetler kodu detaylı bir şekilde okurlar ve muhtemel hata ve kusurları ararlar tespit edilen sorunlar daha sonra kod denetlemesi toplantısında ele alınır
5. Çevik kalite yönetimi genellikle ayrı bir kalite yönetim ekibine dayanmaz Bunun yerine yazılım kalitesini artırmak için geliştirme ekibinin birlikte çalıştığı bir kalite kültürünün oluşturulmasına dayanır
6. Yazılım ölçme yazılım ve yazılım süreci hakkında nicel veri toplamak için kullanılabilir ürün ve süreç kalitesi hakkında çıkarımlar yapmak için toplanan yazılım ölçüt değerleri kullanılabilir
7. Ürün kalite ölçütleri özellikle kalite sorunları olabilecek anormal bileşenleri ortaya çıkarmak için yararlıdır bu bileşenler daha sonra detaylı olarak analiz edilmelidir
8. Yazılım analitiği proje yöneticileri ve geliştiricileri için görüş sağlayabilen ilişkileri keşfetmek için geniş hacimli yazılım ürün ve süreç verilerin otomatik analizidir

### 8.3.4. Konfigirasyon Yönetimi

1. Konfigürasyon yönetimi Evrim geçiren bir yazılım sisteminin yönetilmesidir bir sistemin bakımını yaparken bir konfigürasyon yönetimi takımı konuşlandırılarak değişikliklerin sisteme kontrollü bir biçimde dahil edildiğinden ve gerçekleştirilmiş olan değişikliklerin detaylarını içeren kayıtların tuttuğundan emin olunmalıdır
2. Asıl konfigürasyon yönetimi süreçleri sürüm kontrolü sistem inşası değişim yönetimi ve dağıtım yönetimi ile ilgilenir Tüm bu süreçleri desteklemek için yazılım araçları mevcuttur
3. Sürüm kontrolü yazılım bileşenleri üzerinde değişiklik yapılırken yaratılmakta olan farklı sürümlerin çetelesini tutmayı gerektirir
4. Sistem inşası sistem bileşenlerini kaynaştırarak belirli bir Hedef Bilgisayar sistemi üzerinde çalışmak üzere bir çalıştırılabilir program oluşturulmasıdır
5. Yazılım sıklıkla yeniden inşa edilmeli ve yeni sürümün inşasının hemen ardından sınanmalıdır Bu yöntem hataların ve en son inşa dan bu yana ortaya çıkmış olan problemlerin saptanmasını daha kolay hale getirir

6. Değişim yönetimi sistem müşterilerinden ve diğer paydaşlardan gelen değişiklik taleplerini değerlendirmeyi ve sistemin yeni dağıtımında bu değişiklikleri gerçekleştirmenin uygun maliyetli olup olmadığına karar vermeyi gerektirir
7. Sistem dağıtımları çalıştırılabilir kodu veri dosyalarını konfigürasyon dosyalarını ve belgeleri kapsar dağıtım yönetimi sistem dağıtım tarihleri üzerine kararlar almayı ve her bir sistem dağıtımının teslimi ve belgelenmesi için gereken tüm bilgileri hazırlamayı gerektirir

# 9. Web Programlama

## 9.1. HTML5

### 9.1.1. Neden Önemli

1. Yeni standartlar
2. Hızlı kolay kodlama
3. Modern
4. Mobil cihaz desteği
5. Tarayıcı desteği
6. Temiz kodlama
7. Semantik ve genişletilebilir
8. Daha iyi veri girişi
9. Video ve ses desteği
10. CSS3 ile daha zengin içerikler
11. Çevrimdışı uygulama yeteneği
12. API desteği
13. Yerel veri depolama
14. Oyun programlama
15. Mobil uygulama geliştirme

### 9.1.2. Formlar

```
<form>
  <label for="fname">First name:</label><br />
  <input type="text" id="fname" name="fname" />
</form>
```

#### 9.1.2.1. Özellikleri

| Attributes | Açıklama |
|---|---|
| accept-charset | Specifies the character encodings used for form submission |
| action | Specifies where to send the form-data when a form is submitted |
| autocomplete | Specifies whether a form should have autocomplete on or off |
| enctype | Specifies how the form-data should be encoded when submitting it to the server (only for method="post") |
| method | Specifies the HTTP method to use when sending form-data |
| name | Specifies the name of the form |

| Attributes | Açıklama |
|---|---|
| novalidate | Specifies that the form should not be validated when submitted |
| rel | Specifies the relationship between a linked resource and the current document |
| target | Specifies where to display the response that is received after submitting the form |

### 9.1.3. Elemanlar

```
<!--açılır liste-->
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo" selected>Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

```
<!--büyük metin girişi-->
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

```
<form action="/action_page.php">
  <fieldset>
    <legend>Personalia:</legend>
    <label for="fname">First name:</label><br />
    <input type="text" id="fname" name="fname" value="John" /><br />
    <label for="lname">Last name:</label><br />
    <input type="text" id="lname" name="lname" value="Doe" /><br /><br />
    <input type="submit" value="Submit" />
  </fieldset>
</form>
```

```
<form action="/action_page.php">
  <input list="browsers" />
  <datalist id="browsers">
    <option value="Internet Explorer"></option>
    <option value="Firefox"></option>
    <option value="Chrome"></option>
    <option value="Opera"></option>
```

```
    <option value="Safari"></option>
  </datalist>
</form>
```

#### 9.1.3.1. Input Types

```
<input type="button" />
<input type="checkbox" />
<!--checkbox name attribute aynı olmalı-->
<input type="color" />
<input type="date" />
<input type="datetime-local" />
<input type="email" />
<input type="file" />
<input type="hidden" />
<input type="image" />
<input type="month" />
<input type="number" />
<input type="password" />
<input type="radio" />
<!--radio name attribute aynı olmalı-->
<input type="range" />
<input type="reset" />
<input type="search" />
<input type="submit" />
<input type="tel" />
<input type="text" />
<input type="time" />
<input type="url" />
<input type="week" />
```

## 9.2. CSS3

```
* {
  color: red;
  /*herşey*/
}

p {
  color: red;
}

.class {
  color: red;
}

#id {
  color: red;
```

```
}

p.class {
  color: red;
  /*sınıfı 'class' olan paragraflar*/
}

.class1,
.class2 {
  color: red;
  /*hem birinci hem ikinci sınıf ortak*/
}

.class1 > .class2 {
  color: red;
  /*birinci sınıfın hemen altındaki ikinci sınıf, çocuğu*/
}

.class1 .class2 {
  color: red;
  /*birinci sınıfın oğlu yada torunu olan ikinci sınıf*/
}

@font-face {
  font-family: Sansation;
  src: url(sansation_light.woff);
}

@keyframes myexample {
  from {
    top: 0px;
  }
  50% {
    top: 100px;
  }
  to {
    top: 200px;
  }
}
```

### 9.2.1. CSS Pseudo Classes & Elements

www.w3schools.com/css/css_pseudo_classes.asp

### 9.2.2. EcmaScript 6

1. Scope
2. Arrow fonksiyon
3. For of döngüsü
    1. String
    2. Array
    3. Map

    4. Set
4. Parametreler
    1. Default
    2. Rest (Dizi)
    3. Template
5. Map
6. Set
7. Built-in metotlar
    1. Objeye obje ekleme `let depo = Object.assign({}, obje1, obje2);`
    2. Array eleman arama
    3. String tekrarlama
    4. String arama
    5. Number güvenlik kontrolü
    6. Number işaret saptaması
8. Modüller
9. Class
10. Promise

# 10. Yazılım Proje Yönetimi

## 10.1. Yazılım Geliştirme Süreci

### 10.1.1. Yazılım Geliştirme Temel İlkeleri

1. Basitlik
2. Tekrar kullanılabilirlik
3. Süreklilik *(çökmemeli)*
4. İzlenebilmeli
5. Güvenlik

### 10.1.2. Yazılım Geliştirme Süreci

1. İhtiyaç
    1. Planlı istek
    2. Belirsiz başlangıçlar
        1. Mevcut projede yapılan istek
        2. Proje kapsamının değişmesi
2. Planlama
3. İhtiyaç analizi
    1. Gerçekleştirebilme
    2. Doğrulanabilme
    3. Mananın tam anlaşılması
    4. Çözüm değil istek ifade etme
    5. Tutarlılık
    6. Seviyesi ve yeri doğru olmalı
4. Prototip geliştirme
5. Tasarım
6. Gerçekleştirilme
7. Test
8. Gözden geçirme
9. Devreye alma

### 10.1.3. Yazılım Süreç (Geliştirme) Modelleri

1. Kod eksenli yazılım geliştirme modeli: code and fix (Allah belasını versin)
2. Doğrusal modeller
    1. Şelale
    2. V Modeli
3. Yinelemeli modeller
    1. Artımlı geliştirme
    2. Evrimsel geliştirme
    3. Sarmal model
4. Çevik geliştirme
5. Modüler geliştirme

6. Servis tabanlı

## 10.2. Planlama

## 10.3. Projenin Yürütülmesi

## 10.4. Kalite Güvence Yöntemi