



# YMT 312-Yazılım Tasarım Ve Mimarisi Doğrulama ve Geçerleme

Fırat Üniversitesi Yazılım Mühendisliği Bölümü



# Bu Haftaki Konular

Doğrulama Ve Geçerleme.....	5
Sınamalar Kavramları.....	21
Doğrulama ve Geçerleme Yaşam Döngüsü.....	27
Sınamalar Yöntemleri.....	29
Sınamalar ve Bütünleştirme Stratejileri.....	34
Sınamalar Planlaması.....	47
Sınamalar Belirtimleri.....	48
Yaşam Döngüsü Boyunca Sınamalar Etkinlikleri.....	50

# Amaçlar

---

- Doğrulama ve Geçerleme Kavramlarını Öğrenmek
- Doğrulama Tekniklerini Öğrenmek
- Sınamalar Hakkında Bilgi Edinmek
- Doğrulama ve Geçerleme Yaşam Döngüsünü Kavramak
- Bütünleştirme Sınamasında "koçan" Kullanımı
- Yazılım Yaşam Döngüsü Boyunca Sınamalar Etkinliklerini Öğrenmek



# Giriş

- Geliştirilecek bilgi sistemi yazılımının doğrulanması ve geçerlenmesi, üretim süreci boyunca süren etkinliklerden oluşur. Söz konusu etkinlikler:
  - Yazılım belirtimlerinin ve proje yaşam sürecindeki her bir etkinlik sonunda alınan çıktıların, tamam, doğru, açık ve önceki belirtimleri tutarlı olarak betimler durumda olduğunun doğrulanması.
  - Proje süresince her bir etkinlik ürününün teknik yeterliliğinin değerlendirilmesi ve uygun çözüm elde edilene kadar aktivitenin tekrarına sebep olması.
  - Projenin bir aşaması süresince geliştirilen anahtar belirtimlerin önceki belirtimlerle karşılaştırılması.
  - Yazılım ürünlerinin tüm uygulanabilir gerekleri sağladığının gerçekleşmesi için sınamaların hazırlanıp yürütülmesi.

biriminde özetlenebilir.

# Doğrulama ve Geçerleme

## **Doğrulama**

Doğru ürünü mü üretiyoruz ?

Ürünü kullanacak kişilerin isteklerinin karşılanıp karşılanmadığına dair etkinliklerden oluşur.

## **Geçerleme**

Ürünü doğru olarak mı üretiyoruz?

Ürünün içsel niteliğine ilişkin izleme ve denetim etkinliklerinden oluşur.

Doğrulama ve geçerleme işlemleri temel olarak çeşitli düzeylerde sınıma, gözden geçirme, denetim ve hata giderme süreçlerinden oluşur.

# Bağımsız Doğrulama ve Geçerleme

---

Tüm D&G işlemleri bir başka şirket tarafından yapılır. Kurum içi başka bir bölüm/grup tarafından da yapılabilir.

## Avantajları

- Konusunun uzmanları tarafından yapılır
- Kurum içi siyasetten etkilenmez

## Dezavantajları

- Daha pahalıdır
- Daha sıkı koordinasyon gerektirir

# Doğrulama Teknikleri

## ➤ Gözden Geçirme

- Yönetim
- Teknik
- Arkadaş
- Masaüstü

## ➤ Üstünden Geçme

## ➤ Denetleme

## ➤ İnceleme

# Gözden Geçirme - Yönetim

---

## Amaç: [IEEE1028-97]

- Durum izlemek
- Planların ve takvimin durumlarını belirlemek
- Gereksinimleri ve sistem kaynaklarını onaylamak
- Yönetim şeklinin hedefe uygunluğunu değerlendirmek

## Yöntem:

- Proje Yöneticisi veya Lideri tarafından yöneticilere sunum

## Katılımcılar:

- Karar verme yetkisi olan yöneticiler

## Neler:

Raporlar (Denetleme, Durum, Sonuçlar, ...)

Planlar (Risk, Proje, Konfigürasyon, ...)

## Beklenen Sonuçlar:

Yapılması planlanan değişikliklere ve iyileştirme kararlarına destek ve onay almak

Plan, takvim, ve gereksinimlerin yeterliliğini belirlemek

Projenin yolunda gidip gitmediğini belirlemek

# Gözden Geçirme - Teknik

---

## Amaç: [IEEE1028-97]

- Ürünün kullanıma uygunluğunu değerlendirmek
- Ürünün onaylanmış gereksinimlere uymayan yanlarını belirlemek

## Yöntem:

- Teknik Lider tarafından sunum

## Katılımcılar:

- Karar verme yetkisi olan yöneticiler,
- Gözden geçirme sorumlusu,
- Kaydedici,
- Teknik Uzmanlar

## Neler:

Ürüne ait:

- Amaç ve hedefler
- Proje yönetim planı
- Problem listesi

## Beklenen Sonuçlar:

- Ürünün beklenilere ve standartlara uygun olup olmadığını yöneticilere göstermek
- Değişiklikleri kontrol etmek,
- Devam edip etmemeye kararını almak

# Gözden Geçirme - Arkadaş

## Amaç:

- Ürünün kullanıma uygunluğunu değerlendirmek
- Ürünün onaylanmış gereksinimlere uymayan yanlarını belirlemek.

## Yöntem:

- Ürün konuya hakim bir takım arkadaşına verilir ve bulunan hatalar/düzeltilmeler ürün sahibine açıklanır

## Katılımcılar:

- Konusunda uzman takım elemanı
- Ürün sahibi

## Neler:

- Gereksinimler, tasarım dokümantasyonu, kaynak kodu
- Planlar (Proje, Yazılım Geliştirme, Test, ...)

## Beklenen Sonuçlar:

- Bulunan hatalar/düzeltilmeler/tavsiyeler belgenin üzerine yazılır ve yazara açıklanır

# Gözden Geçirme – Masa Üstü

## **Amaç:**

- Ürünün kullanıma uygunluğunu değerlendirmek
- Ürünün onaylanmış gereksinimlere uymayan yanlarını belirlemek.

## **Yöntem:**

- Ürün konuya hakim uzmanlara dağıtilır ve bulunan hatalar/düzeltilmeler ürünün bir kopyası üzerine kaydedilir

## **Katılımcılar:**

- Uzmanlar
- Belgenin müşterileri
- Belgenin sahibi

## **Neler:**

- Gereksinimler, tasarım dokümantasyonu, kaynak kodu
- Planlar (Proje, Yazılım Geliştirme, Test, ...)

## **Beklenen Sonuçlar:**

- Bulunan hatalar/düzeltilmeler/tavsiyeler belgenin üzerine yazılır ve yazara açıklanır

# Üstünden Geçme

## Amaç: [IEEE1028-97]

- Ara veya son ürünü değerlendirmek,
- Katılımcıları eğitmek

## Yöntem:

- Konu ile ilgili proje elemanlarına sunum

## Katılımcılar:

- Yazar (Sunucu)
- Konusunda uzmanlar
- Ürünün müşterileri

## Neler:

Sistem Gereksinim Belirtimleri (SRS)  
Tasarım  
Proje Planı

## Beklenen Sonuçlar:

Bulgu listesi  
İyileştirme ve alternatif tavsiyeleri

# Denetleme

---

## Amaç: [IEEE1028-97]

- Ürünün ve süreçlerin bağımsız uzmanlar tarafından değerlendirilip regülasyonlara, standartlara, kılavuzlara, planlara ve protokollere uygunluğunun belirlenmesi.

## Yöntem:

- Bağımsız bir uzman takımı tarafından değerlendirme

## Katılımcılar:

- Bağımsız uzmanlar

## Neler:

- Gereksinimler, Sistem Mimarisi, Tasarımlar
- Proje Planı

## Beklenen Sonuçlar:

- Bulgu listesi
- İyileştirme ve alternatif tavsiyeleri

# İnceleme

---

## Amaç: [IEEE1028-97]

- Üründeki hata ve anomalilikleri belirlemek.

## Yöntem:

- İnceleme toplantısı düzenlenir
- Katılımcılar toplantıdan önce ürünü detaylı olarak değerlendirir
- Toplantı sırasında ürün satır satır okunarak bulgular tartışıılır

## Katılımcılar : (Yöneticiler katılmaz)

- İnceleme sorumlusu (Uzlaştırıcı)
- Yazar,
- Okuyucu,
- Kayıtçı,
- İnceleyiciler
  - Tekrar İncelenmesi gerekir

## Neler:

Gereksinimler, tasarım ve kaynak kod

## Beklenen Sonuçlar:

Hata listesi

Kabul Durumu:

Olduğu gibi veya önemi düşük  
düzeltmelerle kabul

Önemli değişikliklerle kabul

# İncelemenin Önemi

---

- İnceleme maliyeti artırır. Faydası nedir?
- Sonraki aşamalarda, özellikle Test ve Müşteri tarafından bulunan hataların onarılması maliyeti daha fazla etkiler
- Bu hataları giderirken, başka hataların sisteme girme olasılığı artar
- Test ve Müşteri tarafından hataların düzeltilmesi proje süresini artırır
- Müşterinin ürün ve firma hakkındaki fikirleri zedelenir
- Proje elemanlarının işlerinden tatmini azalır

# İnceleme Yapmayan Bir Proje

Proje büyülüğu aşağıdaki rakamlarla verilmiştir:

- Gereksinimler : 300 sayfa
- Tasarım : 150 sayfa
- Kaynak Kodu : 10 000 satır

Varsayımlar:

- Gereksinim, Tasarım ve Kodlama aşamaları 100'er hata yapıyor
- Tüm hatalar Test ve Müşteri tarafından bulunuyor
- Testin Hata Bulma Etkinliği (HBE) %75 ve geriye kalan hataların hepsi müşteri tarafından bulunuyor
- Test tarafından bulunan hataların onarılması Gereksinim, Tasarım ve Kodlama hataları için sırasıyla 3, 2.5, ve 2 gündür
- Müşteri tarafından bulunan hataların onarılması Gereksinim, Tasarım ve Kodlama hataları için sırasıyla 4, 3.5, ve 3 gündür

# İnceleme Yapmayan Bir Proje (Devam)

---

Aşama	Hata	Tst	Mlyt	Mşt	Mlyt	Top Mlyt
Gerek	100	75	225	25	100	325
Tasar	100	75	188	25	88	275
Kod	100	75	150	25	75	225
Toplam	300	225	563	75	263	825

Tüm Hataların Onarılma Maliyeti

**825 adam-gün**

# Aynı Proje – Gereksinimler İncelenirse

---

İnceleme için varsayımlar:

- Gereksinim İncelemesi hızı 15 sayfa/saat
- İnceleme Takımı 5 kişi
- İncelemenin HBE %75
- İncelemede bulunan hataların onarılma süresi  $\frac{1}{2}$  adam-gün
- İş Günü 8 saat

İncelemenin maliyeti

$$(300/15) \times 5 = 100 \text{ saat} = 12.5 \text{ gün}$$

İncelemede bulunan hataları onarma süresi

$$75 \times 0.5 = 37.5 \text{ gün}$$

Toplam

$$12.5 + 37.5 = \underline{\underline{50 \text{ gün}}}$$

# Aynı Proje – Gereksinimler İncelenirse (Devam)

---

Test ve Müşteri Tarafından Bulunan Hataların Onarılma Maliyeti

Aşama	Hata	Tst	Mlyt	Mşt	Mlyt	Top Mlyt
Gerek	25	19	57	6	24	81
Tasar	100	75	188	25	88	275
Kod	100	75	150	25	75	225
Toplam	225	169	395	56	187	581

Tüm Hataların Toplam Maliyeti

$$581 + 50 = 631 \text{ adam-gün}$$

# İncelemenin faydaları

---

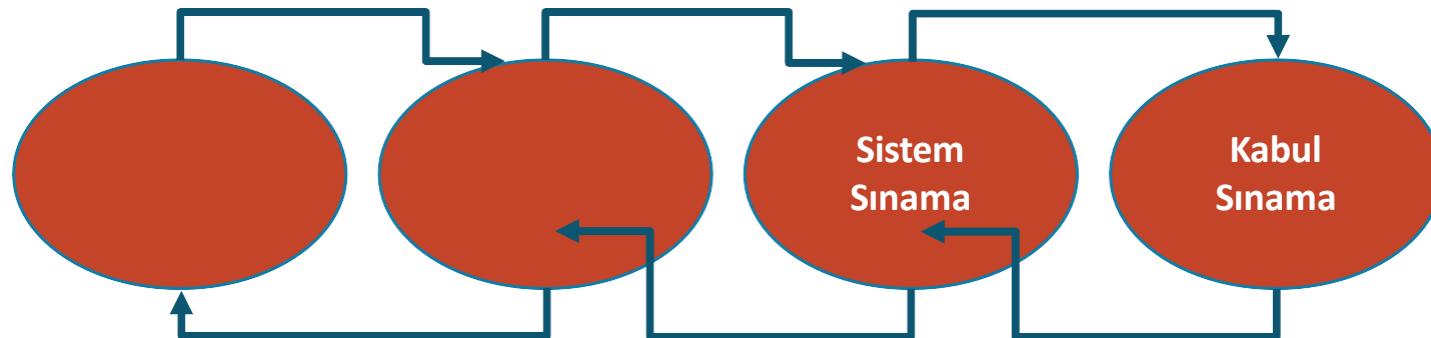
- Hataların Onarılması:
  - İncelemesiz : 825 adam-gün
  - İnceleme ile : 631 adam-gün
  - Kazanılan zaman : 194 adam-gün
    - 39 adam-hafta
    - 9 adam-ay
- Maliyet azaldı
- Takvim kısaldı
- Kalite arttı
- Müşteri memnun (Neden?)
- Proje elemanları memnun (Neden?)
- Yöneticiler memnun (Neden?)

# Sınamalar

- Sınamalar ve bütünlendirme işlemlerinin bir strateji içinde gerçekleştirilmesi, planlanması ve tekniklerin seçimi gerekmektedir. Büyüklendirme işleminde, en küçük birimlerden başlanarak sistem düzeyine çıkılmaktadır.
- Bu değişik düzeylere hitap edecek sınamalar yöntemleri olmalıdır.
- Sınamalar, hatalardan kurtulmanın bir güvencesi değildir.
- Hatalardan bütünüyle arınıldığı gibi bir kanı edinilmemelidir.
- Yalnızca, sınamalar uzadıkça hata bulma sıklığı azalır, daha zor bulunacak hatalar gizli kalmağa devam eder.
- Ne kadar hata sıklığına erişildiğinde sınamalarının durdurulacağı, maliyet ve kalite arasında bir en iyileştirme yapma gereğini öne çıkarır.
- Aynı zamanda vakit de önemli bir unsurdur.
- Daha uzunca süreler vakitler harcanarak daha az hatalar bulunmaya devam eder.
- Yazılımın kritiklik düzeyine göre, sınamaya ayrılan süre ve çaba artar.

# Sınamalar Kavramları

---



# Birim Sınaması

- Bağlı oldukları diğer sistem unsurlarından bütünüyle soyutlanmış olarak birimlerin doğru çalışmalarının belirlenmesi amacıyla yapılır.
- Birimler, ilişkili yapıtaşlarının bütünleştirilmesinden oluşurlar.

# Alt Sistem Sınaması

- Alt sistemler, modüllerin bütünlüğü ile ortaya çıkar. Yine bağımsız olarak sınamaları yapılmalıdır.
- Bu düzeyde en çok hata arayüzlerde bulunmaktadır, arayüz hatalarına yönelik sınamalara yoğunlaşılmalıdır.

# Sistem Sınaması

- Üst düzey bileşenlerin sistem ile olan etkileşimlerinde çıkacak hatalar aranmaktadır.
- Ayrıca belirtilen ihtiyaçların doğru yorumlandıkları da sınanmalıdır.

# Kabul Sınama

---

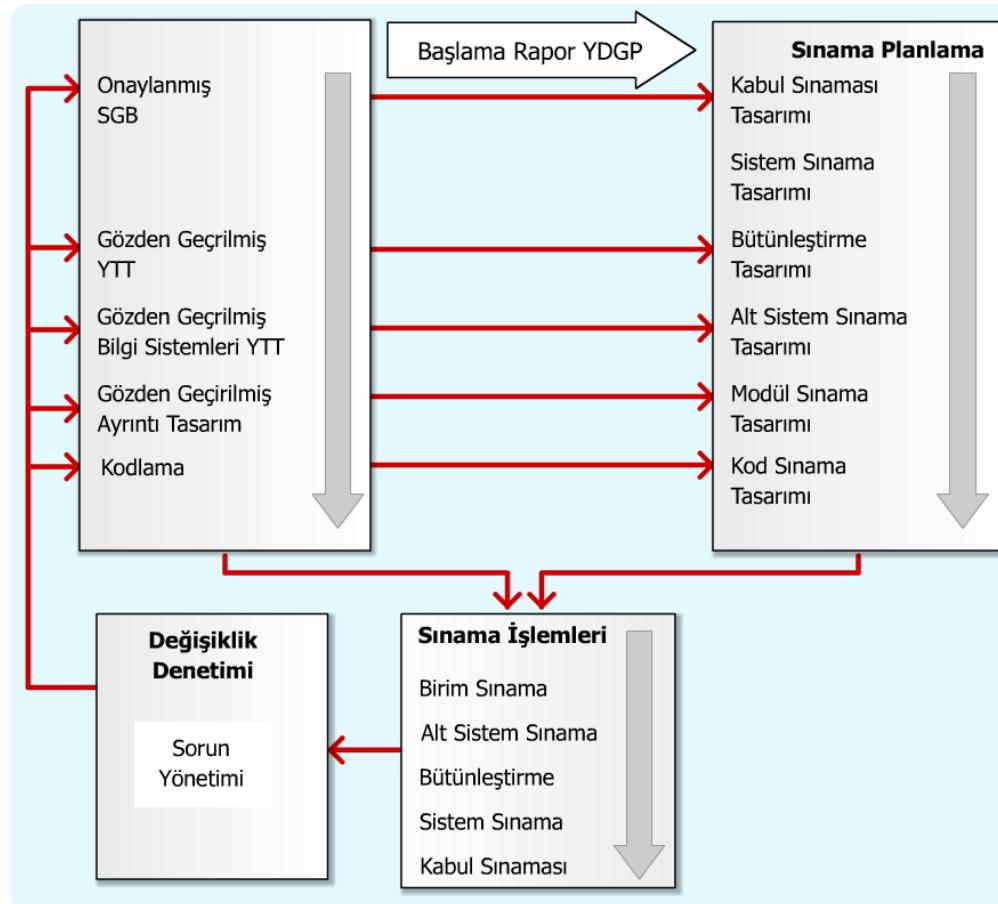
- Çalıştırılmadan önce sistemin son sınamasıdır. Artık yapay veri yerine gerçek veriler kullanılır.
- Bu sınama türü alfa sınaması ve beta sınaması olarak da bilinir.
- Alfa sınamada, tanımında, sınamanın geliştirici organizasyonun yerleşkesinde, kullanıcıların da gelerek katkıda bulunması içerilir.
- Daha sonra ürünün pazarlama işlemi sırasında beta sınama denilen, sınama kullanıcının kendi yerleşkesinde geliştirici gözetiminde yapılır.

# Doğrulama ve Geçerleme Yaşam Döngüsü

---

- Doğrulama ve geçerleme işlemleri yazılım üretim yaşam döngüsünün tüm süreçlerinde ve bu süreçlere koşut olarak sürer.
- Gerçekleştirim aşamasına kadar olan süreçlerde doğrulama ve geçerleme işlemlerinin planlaması yapılır.
- Planlama, genel olarak, birim, alt sistem, bütünlendirme, sistem ve kabul sınavlarının tasarımlarını içerir. Gerçekleştirim aşamasının sonunda ise söz konusu planlar uygulanır.
- Uygulama sonucu elde edilen bulgular, Yazılım Doğrulama ve Geçerleme raporları biçiminde sürekli olarak raporlanır.
- Bu bilgiler, değişiklik denetim sistemi ve sorun yönetim sistemlerinde girdi olarak kullanılır.
- Değişiklik Denetim sistemi, sınavma süresince elde edilen bulguların izlenmesi amacıyla oluşturulan bir sistemdir.
- Bu sistemde, sınavma sonucu elde edilen bulgular ve bunlara karşı sorun yönetimi tarafından alınan önlemler izlenir.

# Doğrulama ve Geçerleme Yaşam Döngüsü



# Beyaz Kutu Sınaması

Beyaz kutu sınaması tasarlanırken, birimin süreç belirtiminden yararlanılır.

Yapılabilecek denetimler arasında:

- Bütün bağımsız yolların en azından bir kere sınanması,
- Bütün mantıksal karar noktalarında iki değişik karar için sınamaların yapılması,
- Bütün döngülerin sınır değerlerinde sınanması,
- İç veri yapılarının denenmesi

bulunur.

Bunun için hataların bazı özelliklerinin bilinmesinde yarar vardır:

- Bir program kesiminin uygulamada çalıştırılma olasılığı az ise o kesimde hata olması ve bu hatanın önemli olması olasılığı fazladır.
- Çoğu zaman, kullanılma olasılığı çok az olarak kestirilen program yolları, aslında çok sıkça çalıştırılıyor olacaktır.
- Yazım hataları rasgele olarak dağılır. Bunlardan bazılarını derleyiciler bulur, bazıları da bulunmadan kalır.

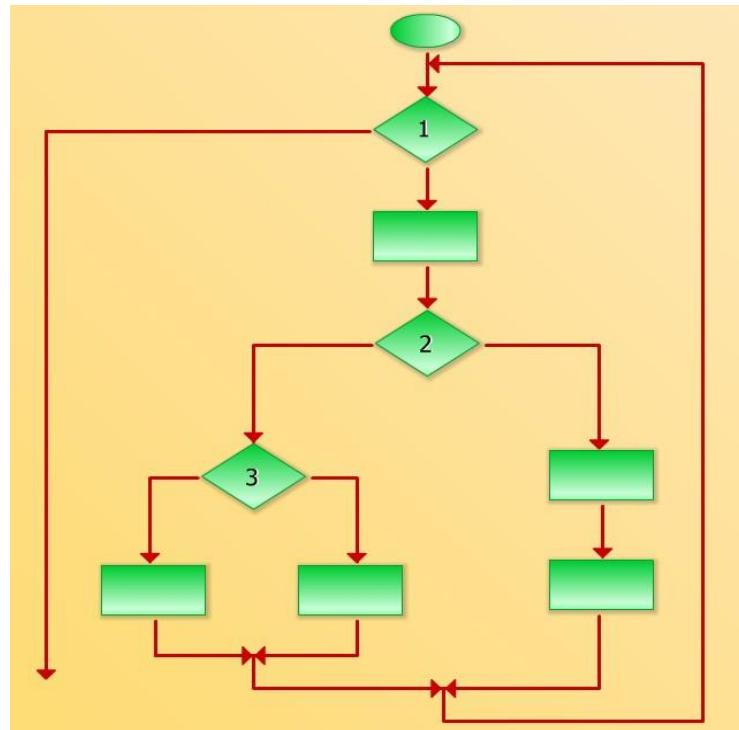
# Temel Yollar Sınaması

---

- Daha önce çevrimsellik karmaşıklığı konusunda gördüğümüz hesap yöntemi ile bir programdaki bağımsız yollar bulunduktan sonra, bu kadar sayıda sınama yaparak programın her birimini bir şekilde sınamalara dahil etmiş oluruz.
- Bağımsız yolların saptanması için önce, program çizgesel bir biçimde çevrilir.
- Bunu yapmak için ise, program iş akış şemaları diyagramları iyi bir başlangıç noktasıdır.

# Temel Yollar Sınaması

---



Bir Programın İş Akış Şeması

# Temel Yollar Sınaması

- Program akış diyagramları matematiksel titizlik ile tanımlanmayan daha serbestçe program yapılarını alt düzeyde modelleyen çizimlerdir. Akış diyagramları ise Çizge Teorisi dalında kabul görecek şekilde bir "Çizge"dir.
- Her çizgede olduğu gibi burada da düğümler ve dallar (veya kırışlar) bulunur.
- Program akış diyagramından akış diyagramına geçmek için süreç kutuları ortadan kaldırılır, koşul baklavaları yerine düğümler çizilir ve her koşul düğümüne karşı düşecek birleştirme düğümleri eklenir.
- Birleştirme düğümleri, koşul kollarının kapandığı noktaya konur.
- Artık bu çizge üzerinden temel yol sayısını da verecek olan çevrimsellik karmaşıklığı sayısını hesaplayabiliriz:

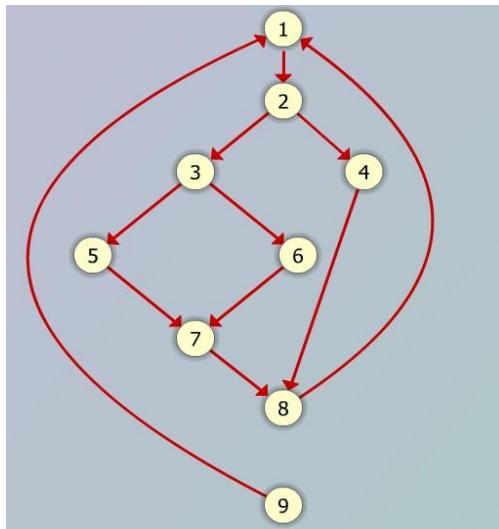
$$E - N + 2$$

E: toplam dal sayısı  
N: toplam düğüm sayısı

- Bağımsız temel yol sayısı kadar temel yolları çizge üzerinde (sonunda programa yansıtılmak üzere) veya program üzerinde işaretlenir. Sonra bu yolların hepsinin koşturulacağı sınama programları tasarlanır.

# Temel Yollar Sınaması

---



Programın Grafik Diyagramı

Formüle göre bağımsız yol sayısı:

$$11 - 9 + 2 = 4$$

Bunun anlamı da şöyle açıklanabilir: Çizgedeki her dal sınaması işleminde kapsanmalıdır. Bu, sınaması sırasında her işlemin çalıştırılması demektir. Her dalın en az bir kere kapsanacağı ve en az sayıda yollar bulunmalıdır. Programa ortadan giremeyeceğimize göre de bu yollar başlangıç noktasından bitiş noktasına kadar uzanmalıdır. Son olarak da minimum sayıda yol ile bu şartları sağlamalıyız. Bu sayının 4 olduğu, daha önce yukarıdaki formülde hesaplanmıştır.

# Sınamalar ve Bütünleştirme Stratejileri

---

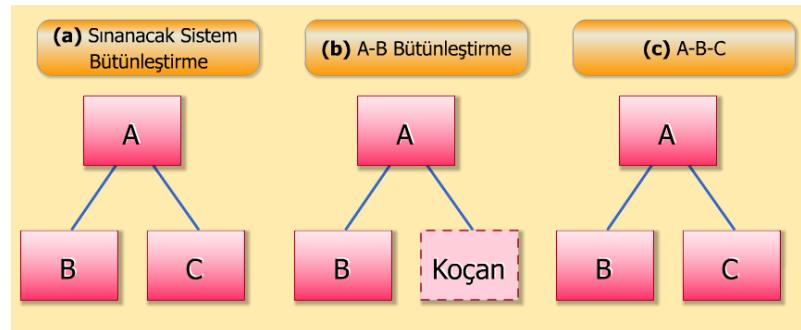
- Genellikle sınavma stratejisi, bütünlendirme stratejisi ile birlikte değerlendirilir. Ancak bazı sınavma stratejileri bütünlendirme dışındaki tasaları hedefleyebilir.
  
- Örneğin, yukarıdan aşağı ve aşağıdan yukarı stratejileri bütünlendirme yöntemine bağımlıdır.
  
- Ancak işlem yolu ve gerilim sınavmaları, sistemin olaylar sırasında değişik işlem sıralandırmaları sonucunda ulaşacağı sonuçların doğruluğunu ve normal şartların üstünde zorlandığında dayanıklılık sınırını ortaya çıkarır.

# Yukarıdan Aşağı Sınama ve Bütünleştirme

---

- Yukarıdan aşağı bütünləştirmədə, önce sistemin en üst düzeylerinin sınanması ve sonra aşağıya doğru olan düzeyleri, ilgili modüllerin takılarak sınanmaları söz konusudur.
- En üst noktadaki bileşen, bir birim/modül/alt sistem olarak sınandıktan sonra alt düzeye geçilmelidir.
- Ancak bu en üstteki bileşenin tam olarak sınanması için alttaki bileşenlerle olan bağlantılarının da çalışması gereklidir.
- Alt bileşenler ise bu stratejiye göre henüz hazırlanmış olamazlar.
- Bunların yerine üst bileşenin sınaması için kullanılmak üzere 'koçan' programları yazılır.
- Koçanlar, bir alt bileşenin, üst bileşen ile ara yüzünü temin eden, fakat işlevsel olarak hiç bir şey yapmayan, boş çerçeve programlarıdır.
- Üst bileşenin sınanması bittikten sonra bu koçanlar, içleri doldurularak kendi kodlama ve birim sınama işlemlerini tamamladıktan sonra üst bileşen ile yeniden sınanırlar.

# Bütünleştirme Sınamasında "koçan" Kullanımı



A, B, C birimlerinden oluşan ve birim şeması (a) şıklıkta belirtilen bir sistemin bu tür koçan kullanılarak sınaması (b) şıklık ve şekil (c) şıklıkta belirtilmektedir.

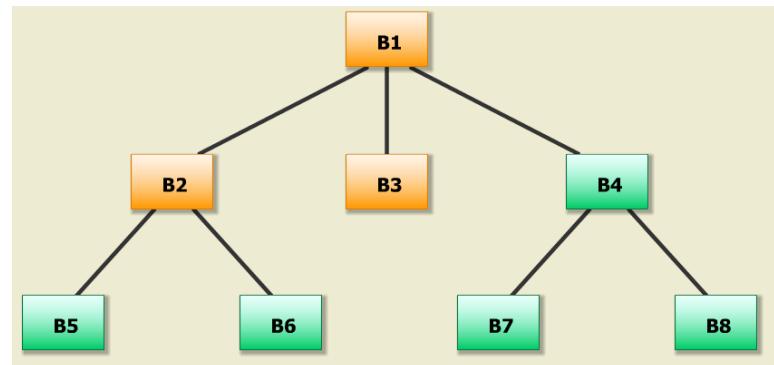
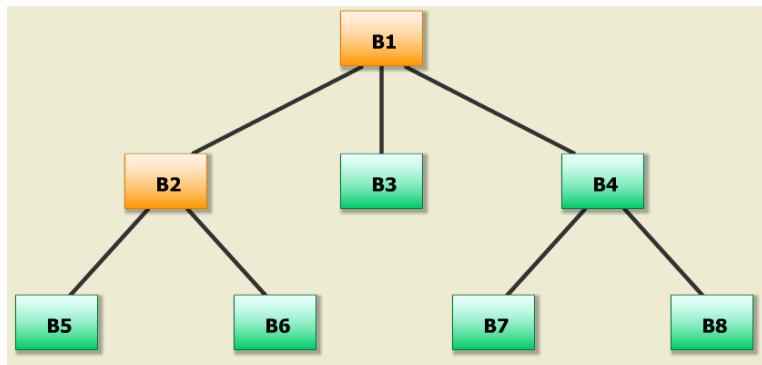
İlk adımda A ve B birimleri bütünleştirilir; C için bir "koçan" yazılır. İkinci adımda ise "koçan" kaldırılır ve C ile yer değiştirilerek A-B-C bütünleştirilir.

Yukarıdan aşağıya doğru bütünleştirme işleminde iki yaklaşım izlenebilir:

**1. Yaklaşım:** Düzey Öncelikli Bütünleştirme (En üst düzeyden başlanır, öncelikle aynı düzeylerdeki birimler bütünleştirilir.)

**2. Yaklaşım:** Derinlik Öncelikli Bütünleştirme (En üst düzeyden başlanır. Birim şemasında bulunan her dal soldan sağa olma üzere ele alınır. Bir dala ilişkin bütünleştirme bitirildiğinde diğer dalın bütünlestirmesi başlar.)

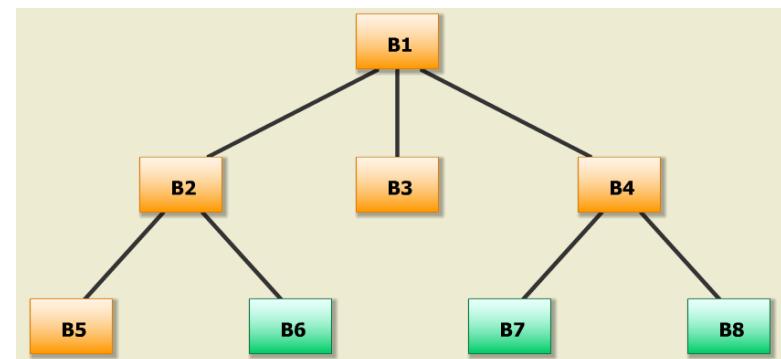
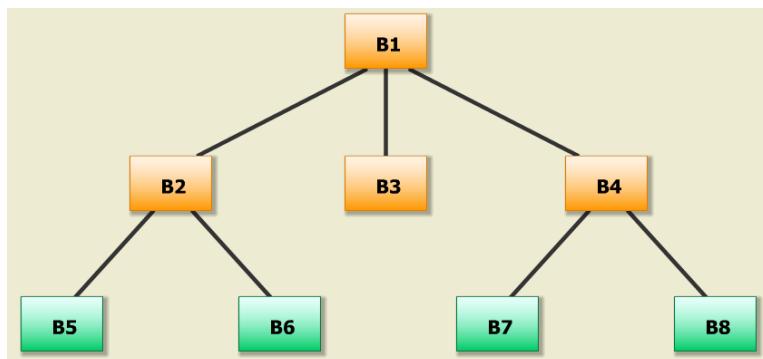
# 1. Yaklaşım: Düzey Öncelikli Bütünleştirme



**1. Adım:** B1-B2 ( KoçanB3 - KoçanB4- KoçanB5 - KoçanB6)

**2. Adım:** B1-B2-B3 (KoçanB4- KoçanB5 - KoçanB6)

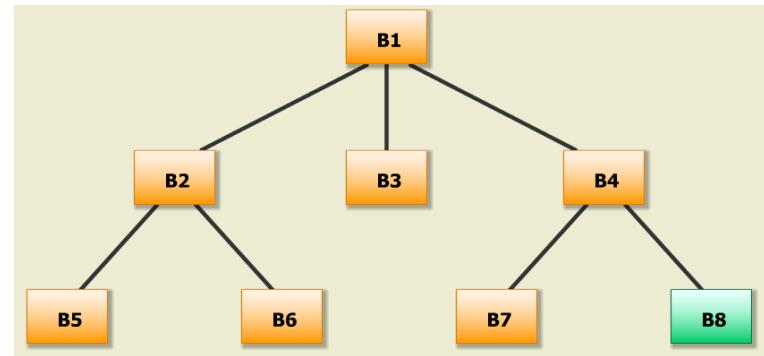
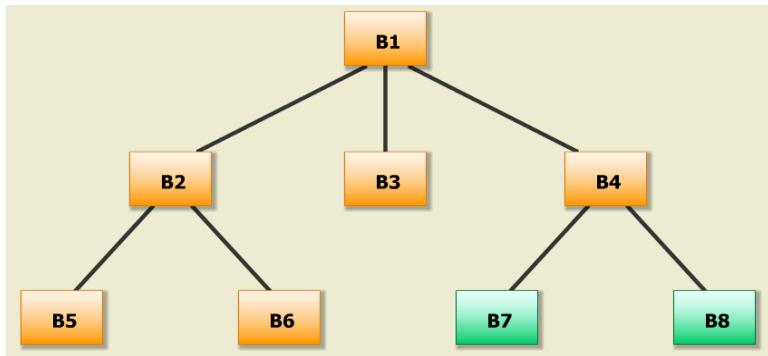
# 1. Yaklaşım: Düzey Öncelikli Bütünleştirme



**3. Adım:** B1-B2-B3-B4 ( KoçanB7 - KoçanB8- KoçanB5 - KoçanB6)

**4. Adım:** B1-B2-B3-B4-B5 ( KoçanB7 - KoçanB8- KoçanB6 )

# 1. Yaklaşım: Düzey Öncelikli Bütünleştirme

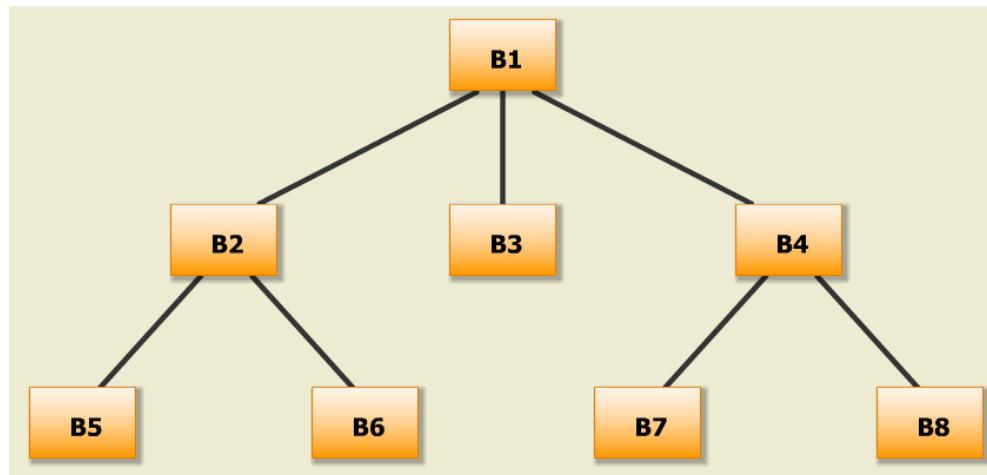


**5. Adım:** B1-B2-B3-B4-B5-B6 ( KoçanB7 - KoçanB8 )

**6. Adım:** B1-B2-B3-B4-B5-B6-B7 ( KoçanB8 )

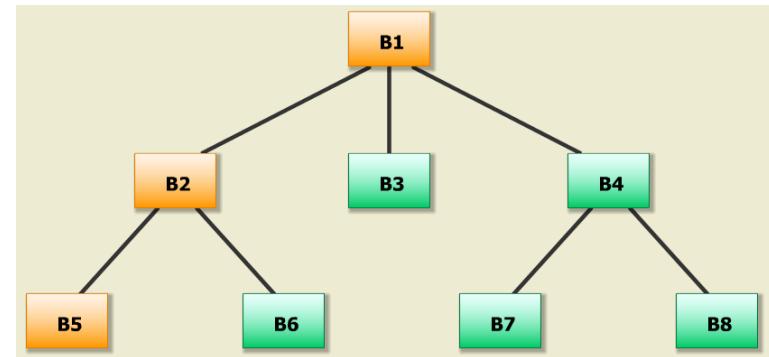
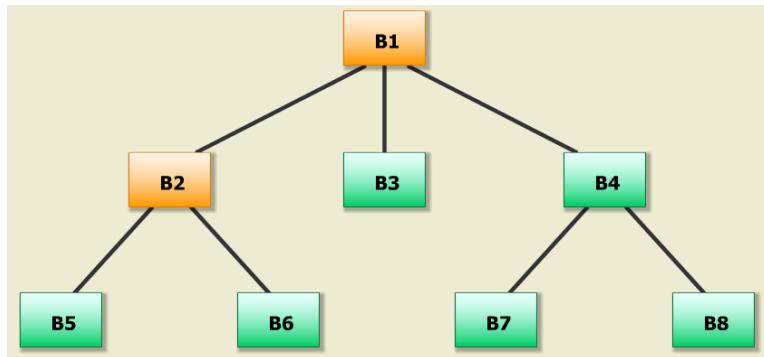
# 1. Yaklaşım: Düzey Öncelikli Bütünleştirme

---



**7. Adım:** B1-B2-B3-B4-B5-B6-B7-B8

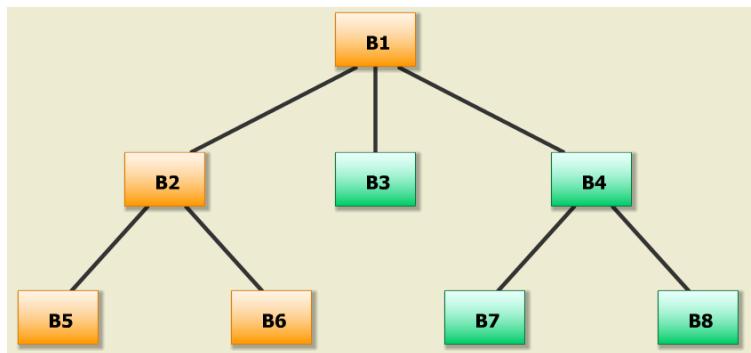
## 2. Yaklaşım: Derinlik Öncelikli Bütünleştirme



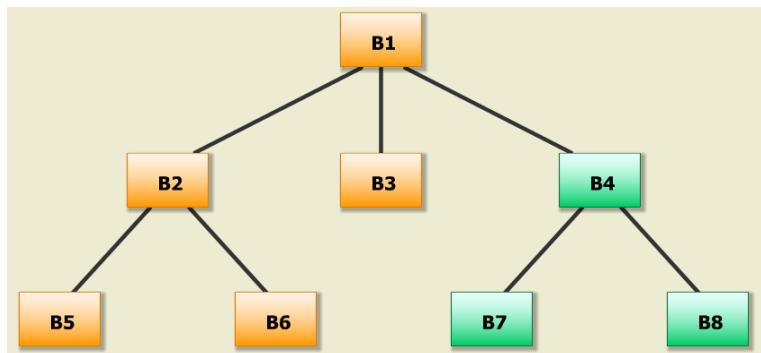
**1. Adım:** B1-B2 ( KoçanB3 - KoçanB4- KoçanB5 - KoçanB6)

**2. Adım:** B1-B2 - B5 (KoçanB3- KoçanB4 - KoçanB6)

## 2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

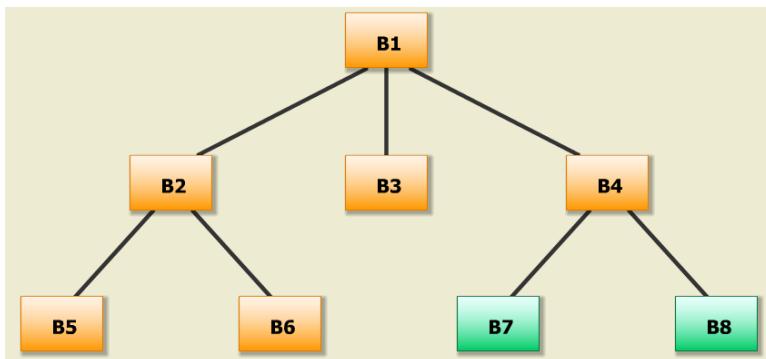


**3. Adım:** B1-B2-B5-B6 (KoçanB3 - KoçanB4)

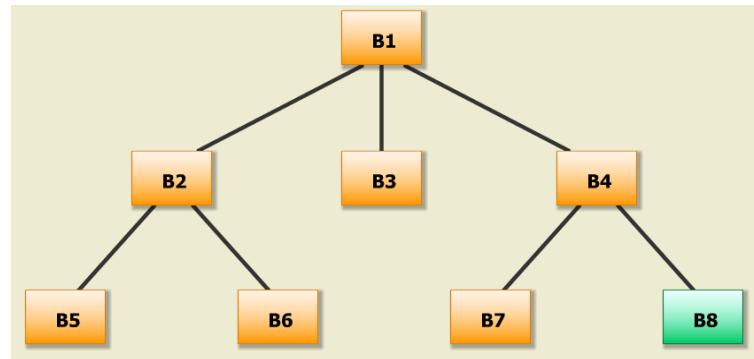


**4. Adım:** B1-B2-B5-B6-B3 (KoçanB4 )

## 2. Yaklaşım: Derinlik Öncelikli Bütünleştirme



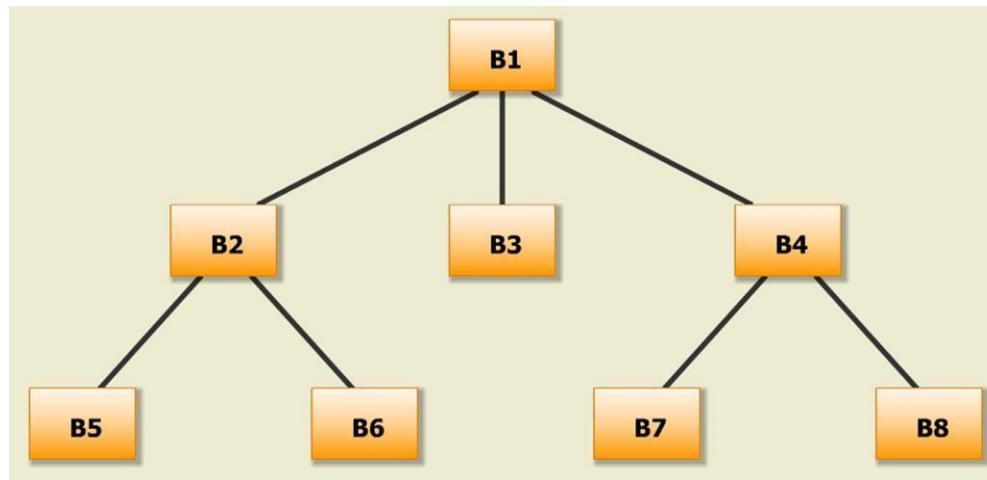
**5. Adım:** B1-B2-B5-B6-B3-B4 ( KoçanB7 - KoçanB8 )



**6. Adım:** B1-B2-B5-B6-B3-B4-B7 (KoçanB8)

## 2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

---



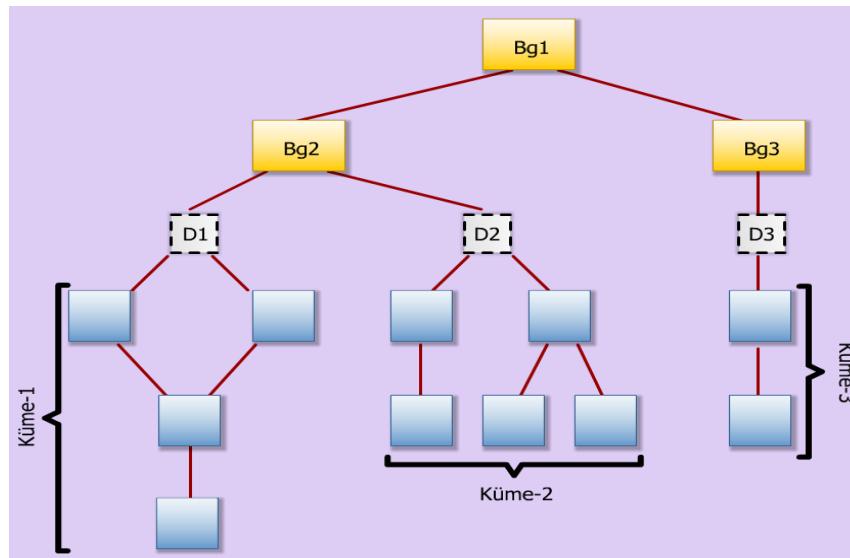
**7. Adım:** B1-B2-B5-B6-B3-B4-B7-B8

## Aşağıdan Yukarıya Sınama ve Bütünleştirme

---

- Aşağıdan yukarı bütünlştirmede ise, önceki yöntemin tersine uygulama yapılır.
- Önce en alt düzeydeki işçi birimleri sınanır ve bir üstteki birimle sınama edilmesi gerekiğinde bu üst bileşen, bir 'sürücü' ile temsil edilir.
- Yine amaç, çalışmasa bile arayüz oluşturacak ve alt bileşenin sınanmasını sağlayacak bir birim edinmektir.
- Bu kez kodlama, bütünlştirme ve sınama aşağı düzeylerden yukarı düzeylere doğru gelişir ve yukarı düzeylerde önce sürücü olarak yazılan birimler sonra gerçekleriyle yer değiştirerek o düzeyin birimleri/alt sistemleri olurlar.
- Bütünlştirme yukarı doğru yapıldıkça daha az sürücü gereği duyulur.
- Uygulamada, hem aşağıdan yukarıya, hem de yukarıdan aşağıya sınama stratejilerinin iki stratejinin birleştirildiği de olur. 'Sandviç' adı verilen bu karma yaklaşımda hem üstten hem alttan sınama etkinliği sürer.

# Aşağıdan Yukarıya Sınama ve Bütünleştirme



Şekilde;

1. Belirli bir yazılım alt işlevini gören alt düzey birimler **kümeler** biçiminde oluşturulurlar,
2. Denetim amaçlı bir **sürücü** programı sınama işlemi için girdi ve çıktı oluşturmak amacıyla yazılar,
3. Sürcüler aşağıdan yukarı kaldırılır ve gerçek birim ya da birim kümeleriyle değiştirilerek sınama işlemi sürdürülür.

# Sınaması Planlaması

---

- Sınaması işlemi çok kapsamlıdır.
- Bir plan güdümünde gerçekleştirilmelidir.
- Böyle bir planın temel bileşenleri önceki sayfalarda belirtilmiştir.
- Yazılım yaşam döngüsünün süreçlerine koşut olarak, farklı ayrıntı düzeylerinde birden fazla sınaması planı hazırlanır. Sınaması planları;
  - Birim (Modül) Sınaması Planı,
  - Alt Sistem Sınaması Planları,
  - Bütünleştirme Sınaması Planları,
  - Kabul Sınaması Planları,
  - Sistem Sınaması Planları biçimindedir.

Her sınaması planı, sınaması etkinliklerinin sınırlarını, yaklaşımını, kaynaklarını ve zamanlamasını tanımlar. Plan neyin sınanacağını, neyin sınanmayacağını, sorumlu kişileri ve riskleri göstermektedir. Sınaması planları, sınaması belirtimlerini içerir.

# Sınamalar Belirtimleri

➤ Sınamalar belirtimleri, bir sınamanın nasıl yapılacağına ilişkin ayrıntıları içerir. Bu ayrıntılar temel olarak:

- sınamanın program modülü ya da modüllerinin adları,
- sınamanın türü, stratejisi (beyaz kutu, temel yollar vb.),
- sınamanın verileri,
- sınamanın senaryoları

türündeki bilgileri içerir.

➤ Sınamanın verilerinin elle hazırlanması çoğu zaman kolay olmayabilir ve zaman alıcı olabilir. Bu durumda, otomatik sınamanın verisi üreten programlardan yararlanılabilir.

➤ Sınamanın senaryoları, yeni sınamanın senaryosunu üretebilmeye yardımcı olacak biçimde hazırlanmalıdır.

➤ Zira sınamaların belirtimlerinin hazırlanmasındaki temel maç, etkin sınamanın yapılması için bir rehber oluşturmasıdır. Sınamanın işlemi sonrasında bu belirtimlere,

- sınamayı yapanı,
- sınamanın tarihini,
- bulunan hataları ve açıklamalarını

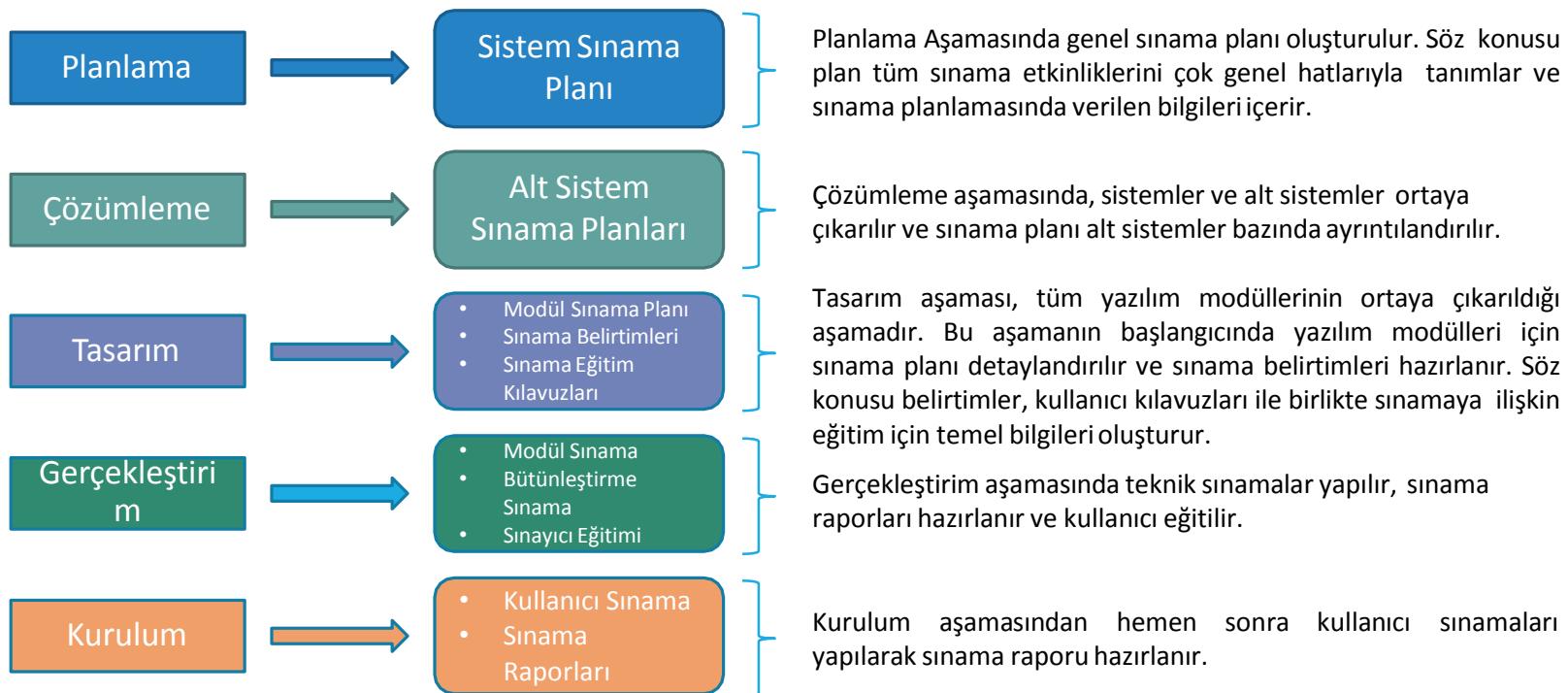
türündeki bilgiler eklenerek sınamanın raporları oluşturulur.

# Sınamalar Belirtimleri

---

- Sınamalar raporları, sınavma bitiminde imzalanır ve yüklenici ile iş sahibi arasında resmi belge niteliği oluşturur.
- Sınamalar planları; Birim (Modül) Sınamalar Planı, Alt Sistem Sınamalar Planları, Bütünleştirme Sınamalar Planları, Kabul Sınamalar Planları, Sistem Sınamalar Planları biçimindedir.
- Her sınavma planı, sınavma etkinliklerinin sınırlarını, yaklaşımını, kaynaklarını ve zamanlamasını tanımlar.
- Plan neyin sınanacağını, neyin sınanmayacağını, sorumlu kişileri ve riskleri göstermektedir. Sınamalar planları, sınavma belirtimlerini içerir.

# Yazılım Yaşam Döngüsü Boyunca Sınamalar Etkinlikleri



## Yazılım Yaşam Döngüsü Boyunca Sınamalar Etkinlikleri

---

- Hazırlanan sınavma raporları, doğrulama ve geçerleme yaşam döngüsü işlemleri gereği "sorun yönetimi "ne iletilir. Bu bölümde hatalar kaydedilir ve bulunan hatalara karşı yapılacak işlemler planlanır.
- Sınamalar sırasında bulunan her bulgu ya da hata olarak belirtilen her durum其实 hata olmayabilir.
- Farklı sınavıcılardan biri, bir durumu hata olarak nitelerken diğerinin aynı durumu doğru olarak değerlendirebilir.
- Bu nedenle sınavma raporlarında hata olarak bildirilen her durum hemen düzeltilmek üzere ele alınmaz.
- Önce çözümlenir, kullanıcı çelişkileri giderilir ve gerçekten hata olduğuna karar verilirse düzelttilir. Söz konusu karar kullanıcı temsilcileri ile birlikte alınır.
- Sınamalar sırasında bulunan her hata için, değişiklik kontrol sistemine (DKS), "Yazılım Değişiklik İsteği" türünde bir kayıt girilir.

## Yazılım Yaşam Döngüsü Boyunca Sınamalar Etkinlikleri

---

- Hatalar, DKS kayıtlarında aşağıdaki gibi gruptara ayrılabılır:
  - **Onulmaz Hatalar:** BT projesinin gidişini bir ya da birden fazla aşama gerileten ya da düzeltmesi mümkün olmayan hatalardır.
  - **Büyük Hatalar:** Projenin kritik yolunu etkileyen ve önemli düzeltme gerektiren hatalardır.
  - **Küçük Hatalar:** Projeyi engellemeyen, ve giderilmesi az çaba gerektiren hatalardır.
  - **Şekilsel Hatalar:** Heceleme hatası gibi önemsiz hatalardır.

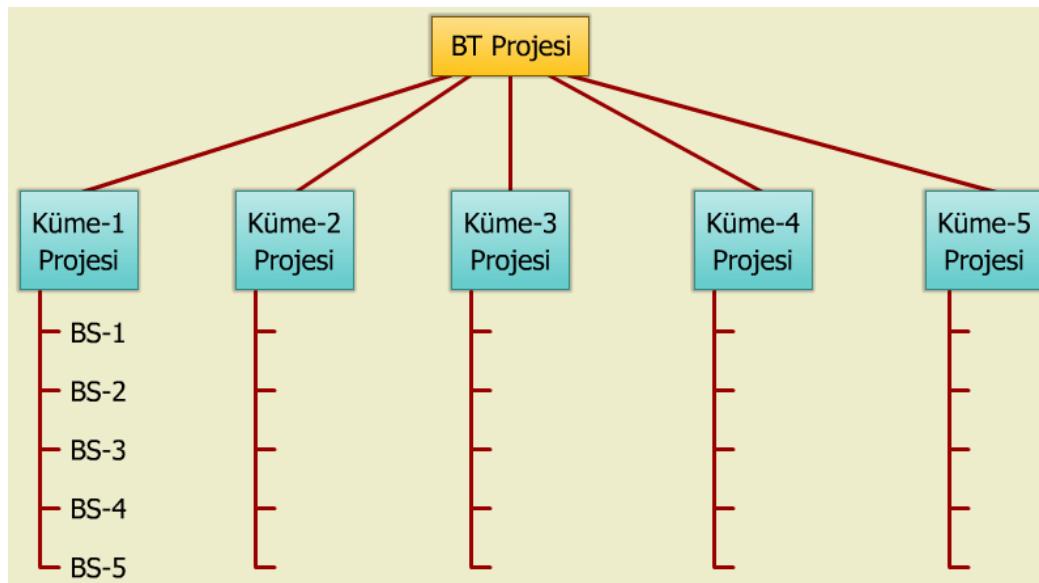
# Uygulama : Görsel Yazılım Geliştirme Ortamında Sınama

---

- Bu kısımda, gerçek yaşam ortamında, **Oracle Designer CASE** aracı ve **Developer** görsel yazılım geliştirme platformu kullanılarak geliştirilen yazılım modüllerinin sınanması işleminin nasıl yapılacağı ve buraya kadar açıklanan sınama yöntemlerinin nasıl uygulandıkları bir örnek üzerinde anlatılmaktadır.
- Oracle Developer kullanılarak geliştirilen her yazılım formlardan oluşur. Bir form, bir ekran ve bu ekranada yapılan işlemlere karşılık gelen PL/SQL kodları biçiminde tanımlanır.
- Bu örnekte elimizde, sınama işlemine koşulacak ve uygulamanın çeşitli işlevlerine ilişkin bir dizi form olduğunu düşünebiliriz.
- Bu örnekte söz edilen uygulama, 2000'den fazla kullanıcısı olan, ülkenin çeşitli yörelerine dağılmış birimlerde çalışacak biçimde tasarlanmış ve 1000'den fazla Developer formundan oluşmaktadır.
- Uygulamanın sınama aşamasına gelmesi, 2 yıllık bir süre ve yaklaşık 100 kişi-yıl'lık bir iş gücü gerektirmiştir. Uygulama beş ana kümeye bölünmüştür ve her kümeye belirli sayıda bilgi sistemini içermektedir.
- Toplam olarak 30 bilgi sistemi bulunmaktadır.
- Uygulama sıra düzeni bir sonraki slaytta gösterilmektedir.

# Uygulama : Görsel Yazılım Geliştirme Ortamında Sınama

---



- Şimdi böyle kapsamlı bir uygulamanın sınama aşamasında kullanılan yöntemleri ve metodolojiyi bir sonraki slaytta inceleyelim.

# Sınamı Ortamı Oluşturulması

- Üretimin etkilenmemesi amacıyla, yalnızca sınayıcıların kullanacakları ve ayrı bilgisayarlardan oluşan bir sınav ortamı oluşturuldu.
- Oluşturulan sınav ortamı ile üretim ortamının bir bir aynı olması sağlandı.
- Üretimi biten yazılım parçaları, bir kayıt düzeni içerisinde sınav ortamına alındı.

# Sınamaya Yöntemlerine Karar Verilmesi

---

- Uygulamada kullanılacak sınavma yöntemleri aşağıdaki gibidir:
  - Teknik Sınaması
  - Biçimsel Sınaması
  - İşletimsel Sınaması
  - Senaryo Sınaması
  - Kullanıcı Sınaması

# Teknik Sınamalar

---

- Üretim ortamında yapılacak sınavma olarak karar verildi. Bu sınavma, modül sınavması ve bütünlendirme sınavması olarak üretim ekipleri tarafından gerçekleştirildi. Modül sınavma yöntemi olarak ‘Beyaz kutu’ sınavma yöntemi uygulandı. Tüm program deyimleri en az bir kez, tüm döngüler en az 10 kez yinelenerek biçimde sınavma yapıldı.
- Bütünlendirme sınavma yöntemi olarak ‘yukarıdan aşağıya sınavma yöntemi’ ve ‘derinlik öncelikli bütünlendirme’ stratejisi uygulandı.

# BİÇİMSEL SİNAMA

---

- Üretim ekiplerinden bağımsız olarak, sınama ekipleri tarafından yapılan sınamadır. Bu sınama, Developer formları üzerinde görsel olarak yapıldı. Amaç, formların, önceden kararlaştırılan standartlara uygunluğunun saptanmasıydı.
- Örneğin,
  - form alanları, kararlaştırılan uzunlukta mı?
  - Başlıklar istenilen gibi koyu mu?
  - Yardım düğmesi hep aynı yerde mi vb.
- Sınama, formlar işletilmeden yapılır. Tüm formlar tek tek incelenir ve standartlara uygun olmayanlar belirlenip, düzeltilmek üzere üretim ekibine geri iletılır.
- Biçimsel sınavlarının yapılması amacıyla denetim listeleri hazırlanır ve sınama sırasında bu listeler kullanılır. Listelere kaydedilen her sonuç DKS'ye aktarılır.
- Bu yolla üretim ekiplerinin performansı izlenebilir.

# İşletimsel Sınama

---

- Üretim ekiplerinden bağımsız olarak, sınavma ekipleri tarafından yapılan sınavmadır.
- Biçimsel sınavma işlemi bittikten sonra yapılır.
- Bu sınavada her form ayrı ayrı çalıştırılarak işlem yapılır.
- Amaç, formun çalışıp çalışmadığının belirlenmesidir.
- Form alanlarının sınır değerlerle çalışıp çalışmadığı, aykırı değer verildiğinde uygun hata iletisi alınıp alınmadığı vb. belirlenmeye çalışılır.

# Senaryo Sınama

---

- Sınama ekipleri tarafından yapılan sınamadır.
- Ancak, senaryoların hazırlanması sırasında üretim ekipleri ile birlikte çalışılır.
- Amaç, birden fazla formun bir arada sıyanmasıdır. Bu amaçla, ‘senaryo’lar hazırlanır.
- Her senaryo, çözümleme aşamasında belirlenen bir iş fonksiyonuna karşılık gelecek biçimde hazırlanır.

# Kullanıcı Sınaması

- Kullanıcılar tarafından yapılması öngörülen sınavadır.
- Senaryo sınavasının kullanıcı tarafından yapılan biçimini olarak düşünülebilir.

# Kullanıcı Sınaması Eğitimi

- Sınamaya yapılacak kullanıcı sınayıcılarına, sınamaların nasıl yapılacağına ilişkin eğitim verilmesi gerekmektedir.
- Eğitim kitaplıklarının hazırlanması amacıyla, senaryo sınavlarında kullanılan "senaryo"lar ve kullanıcı kitaplıkları kullanılır.

# Sınamaların Yapılması

- Sınamalar sırasında yapılan her işlem, DKS'de izlenir. Özellikle kullanıcı sınamalarının izlenmesi ve ortaya çıkabilecek tartışmaların önlenmesi bu yolla sağlandı.
- Yaklaşık 100 kullanıcı sınayıcısının, birbirinden farklı yerlerde yaptıkları sınamalar için "haftalık sınavma sonuçları" formları toplandı.
- Yerinde destek elemanları, sürekli olarak kullanıcı sınayıcılarını ziyaret ederek sınavma sonuçlarının düzenli olarak toplanmasını sağladı.
- Bu formlar DKS'ye aktarılıarak, daha sonra Yazılım Doğrulama ve Geçerleme Raporları için önemli girdiler oluşturdu.

# Özet

---

Doğrulama: Doğru ürünü mü üretiyoruz ?

- Ürünü kullanacak kişilerin isteklerinin karşılanıp karşılanmadığına dair etkinliklerden oluşur.

Geçerleme: Ürünu doğru olarak mı üretiyoruz?

- Ürünün içsel niteliğine ilişkin izleme ve denetim etkinliklerinden oluşur.

Doğrulama ve geçerleme işlemleri temel olarak çeşitli düzeylerde sınama, gözden geçirme, denetim ve hata giderme süreçlerinden oluşur.

Doğrulama Teknikleri: Gözden Geçirme, Üstünden Geçme, Denetleme ve İncelemedir.

Sınama planları;

- Birim (Modül) Sınama Planı,
- Alt Sistem Sınama Planları,
- Bütünleştirme Sınama Planları,
- Kabul Sınama Planları,
- Sistem Sınama Planları biçimindedir.

# Özet

---

Hatalar, DKS kayıtlarında aşağıdaki gibi gruptara ayrılabılır:

- **Onulmaz Hatalar:** BT projesinin gidişini bir ya da birden fazla aşama gerileten ya da düzeltilmesi mümkün olmayan hatalardır.
- **Büyük Hatalar:** Projenin kritik yolunu etkileyen ve önemli düzeltme gerektiren hatalardır.
- **Küçük Hatalar:** Projeyi engellemeyen, ve giderilmesi az çaba gerektiren hatalardır.
- **Şekilsel Hatalar:** Heceleme hatası gibi önemsiz hatalardır.

Uygulamada kullanılacak sınıma yöntemleri aşağıdaki gibidir:

- Teknik Sınıma
- Biçimsel Sınıma
- İşletimsel Sınıma
- Senaryo Sınıma
- Kullanıcı Sınıma

# Sorular

---

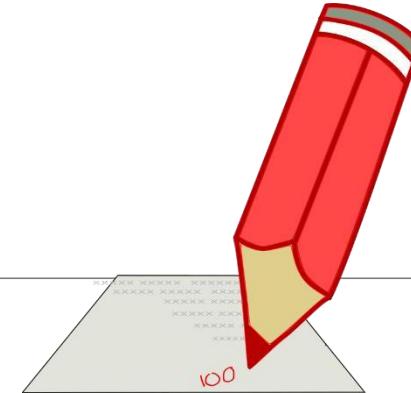
1. Doğrulama ile Geçerleme arasındaki farklılıklarını belirtiniz. Birer örnekle açıklayınız.
2. Sınamaya Yöntemlerini açıklayınız.
3. "Beyaz Kutu" sınaması ile "Temel Yollar Sınaması" yöntemleri arasındaki farklılıkları belirtiniz.
4. Sınamaya Yöntemleri ile sınamaya belirtimleri arasındaki farkı belirtiniz.
5. Yukarıdan aşağıya doğru bütünlendirme ve aşağıdan yukarıya bütünlendirme yöntemlerinin zorluklarını ve kolaylıklarını belirtiniz.
6. Sınamaya belirtimlerinin önemi nedir.
7. Kullanıcı sınaması sırasında yaşanabilecek sorunları belirtiniz.

# Kaynaklar

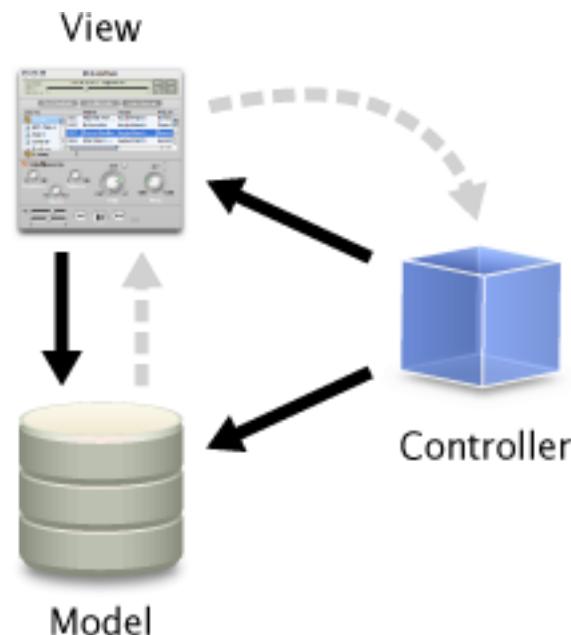
---

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

# Ödev



- Yazılım Mimarileri Hakkında Araştırma Yapınız.
- İki Katmanlı ve Üç Katmanlı Mimarilerle yapılmış örnek sistemleri araştırınız.





# YMT312 Yazılım Tasarım ve Mimarisi

## Yazılım Yaşam Döngüsü ve Süreç Modelleri

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bölüm-2

# Bu Haftaki Konular

---

Yazılım Yaşam Döngüsü.....	4
Süreç Modelleri.....	16
Metodojiler.....	71

# Amaçlar

---

- Yazılım Yaşam Döngüsü ’nın Projelerde ki Önemini Kavramak?
- Yazılım Yaşam Döngüsünde Önemli Kavramlar
- Yazılım Geliştirmede Süreç Modellerinin Önemi?
- Süreç Modellerinin Gelişimi
- Süreç Modeli Çeşitleri ve Uygulanması
- Süreç Modellerinin Avantajları ve Dezavantajları
- Metodoloji Yaklaşımı
- Örnek bir Metodoloji



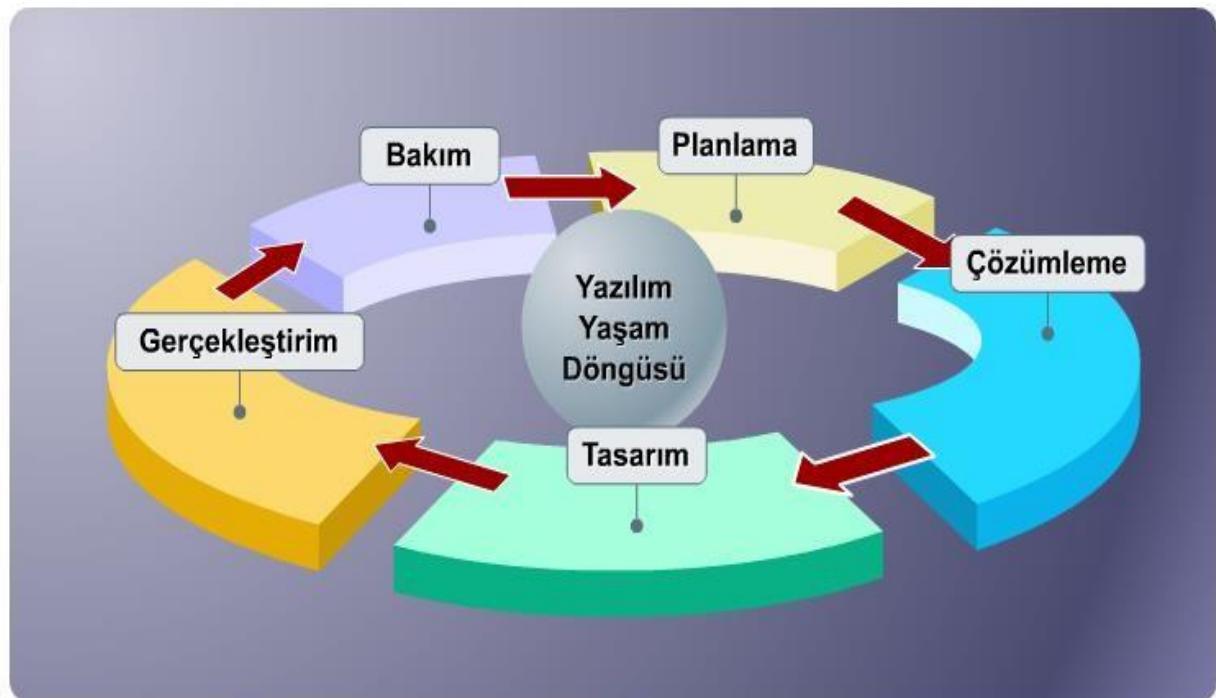
# Yazılım Yaşam Döngüsü Nedir?

- Yazılım yaşam döngüsü, herhangi bir yazılımin, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar biçiminde tanımlanır.
- Yazılım işlevleri ile ilgili gereksinimler sürekli olarak değiştiği ve genişlediği için, söz konusu aşamalar bir döngü biçiminde ele alınır.
- Döngü içerisinde herhangi bir aşama da geriye dönmek ve tekrar ilerlemek söz konusudur.
- Yazılım yaşam döngüsü tek yönlü ve doğrusal olduğu düşünülmemelidir.

# Yazılım Yaşam Döngüsü Temel Adımları

---

- Planlama
- Çözümleme
- Tasarım
- Gerçekleştirim
- Bakım



# Planlama

---

Üretilenek yazılım ile ilgili olarak, personel ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşamadır.



# Çözümleme

---

Yazılım işlevleri ile gereksinimlerin ayrıntılı olarak çıkarıldığı aşamadır.

Bu aşamada temel olarak mevcut sistemde var olan işler incelenir, temel sorunlar ortaya çıkarılarak, yazılımın çözümleyebilecekleri vurgulanır.

Temel amaç, bir yazılım mühendisi gözüyle mevcut yapıdaki işlerin ortaya çıkarılması ve doğru olarak algılanıp algılanmadığının belirlenmesidir.

Bu aşamada temel UML diyagramlarının çizimine başlanır (Use Case, Activity, Class diagram... vs.)

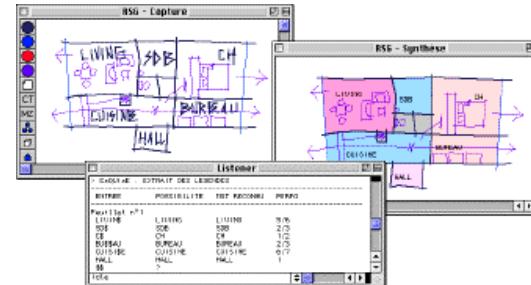


# Tasarım

Çözümleme aşamasından sonra belirlenen gereksinimlere karşılık verecek yazılım ya da bilgi sisteminin temel yapısının oluşturulması çalışmalarıdır.

Bu çalışmalar, mantıksal tasarım ve fiziksel tasarım olarak iki gruba ayrılır.

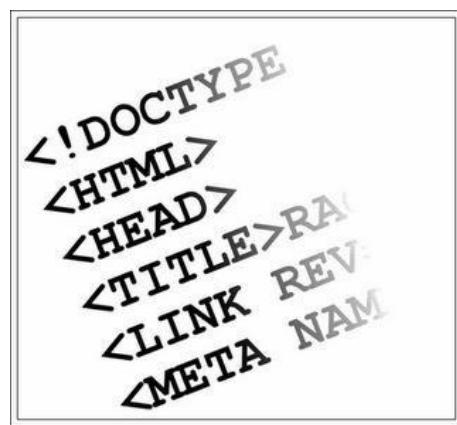
- **Mantıksal Tasarım:** Mevcut sistem değil önerilen sistemin yapısı anlatılır. Olası örgütsel değişiklikler önerilir.
  - **Fiziksel Tasarım:** Yazılımı içeren bileşenler ve bunların ayrıntıları içerilir.



# Gerçekleştirim

- Kodlama
- Test etme
- Kurulum

çalışmalarının yapıldığı aşamadır.



# Bakım

---

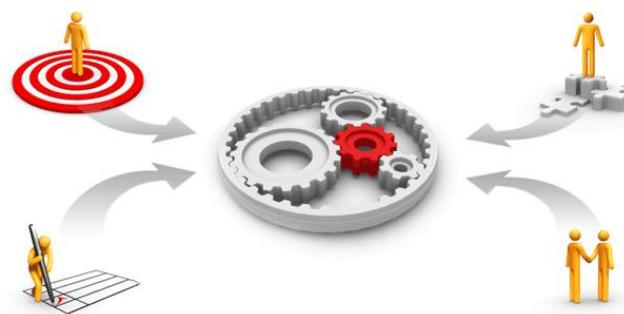
- İşletime alınan yazılım ile ilgili olarak, hata giderme ve yeni eklentiler yapma aşamasıdır.
- Bu aşama yazılımın tüm yaşamı boyunca sürer.



# Yazılım Yaşam Döngüsü Temel Adımları

---

- Yazılım yaşam döngüsünün temel adımları çekirdek süreçler (core processes) olarak da adlandırılır.
- Bu süreçlerin gerçekleştirilmesi amacıyla;
  - **Belirtim Yöntemleri (Software Specification Methods)** - bir çekirdek süreç ilişkin fonksiyonları yerine getirmek amacıyla kullanılan yöntemler.
  - **Süreç Modelleri (Software Process Models)** - yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlayan modeller kullanılır.

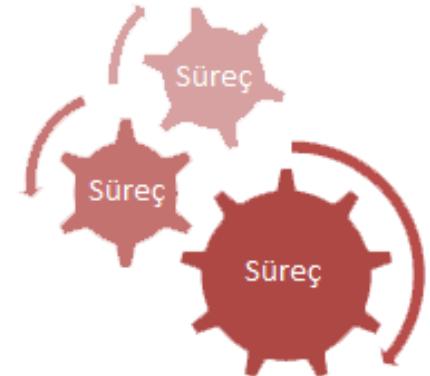


# Belirtim Yöntemleri

---

## ➤ Süreç Akışı İçin Kullanılan Belirtim Yöntemleri

- Süreçler arası ilişkilerin ve iletişimini gösterdiği yöntemler  
(Veri Akış Şemaları, Yapısal Şemalar, Nesne/Sınıf Şemaları).



## ➤ Süreç Tanımlama Yöntemleri

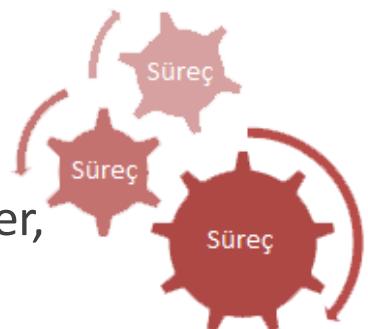
- Süreçlerin iç işleyişini göstermek için kullanılan yöntemler  
(Düz Metin, Algoritma, Karar Tabloları, Karar Ağaçları, Anlatım Dili).

## ➤ Veri Tanımlama Yöntemleri

- Süreçler tarafından kullanılan verilerin tanımlanması için kullanılan yöntemler  
(Nesne İlişki Modeli, Veri Tabanı Tabloları, Veri Sözlüğü).

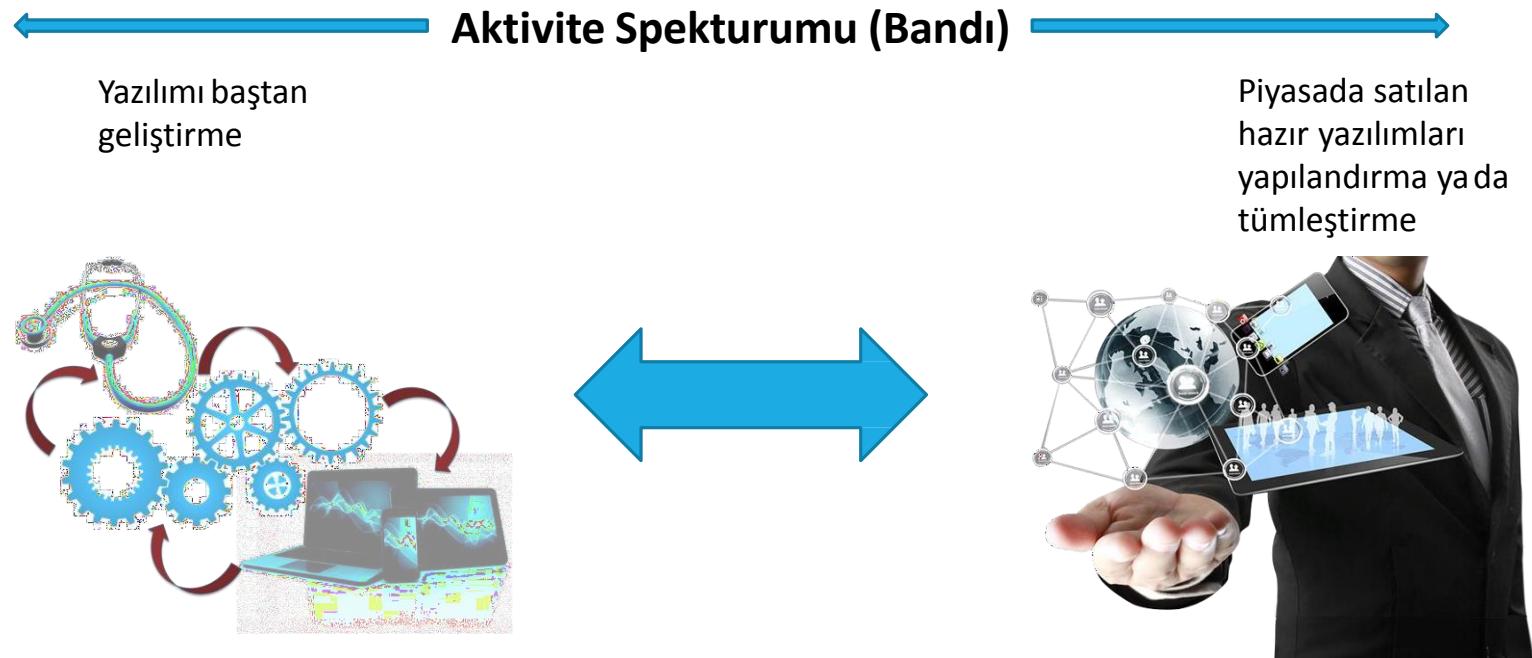
# Süreç (process) nedir?

- Süreç olguların ya da olayların, belli bir taslağa uygun ve belli bir sonuca varacak biçimde düzlenmesi, ***sıralanması***.
- Bir şeyin yapılışını, üretiliş biçimini oluşturan sürekli işlemler, eylemler dizisi (**Kaynak: <http://tr.wikipedia.org>**)
- Aralarında birlik olan veya belli bir düzen veya zaman içinde tekrarlanan, ilerleyen, gelişen olay ve hareketler dizisi, proses.



# Yazılım süreci nedir?

Bir *yazılım ürünü* üretmeyi sağlayan birbiriyle **tutarlı aktivite grubudur.**



# Yazılım süreci nedir?

- Ne yapılmak istendiğini tüm uygulama detaylarına girmeden tanımlar.
- Yazılım süreci bizim yazılım üretme yolumuzdur. \*



\*Kaynak: Object-Oriented and Classical SWE, 7th Edition, Stephen R. Schach, p71.

# Süreç Modelleri

---

**Süreç Modelleri**, Yazılım Yaşam Döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlar.

Yazılım geliştirmenin bahsedilen zorluklarıyla bahsedebilmek için, geliştirmeyi sistematik hale getirmeyi hedefleyen çeşitli süreç modelleri ortaya çıkmıştır.

- **Bu modellerin temel hedefi;** proje başarısı için, yazılım geliştirme yaşam döngüsü (“software development life cycle”) boyunca izlenmesi önerilen mühendislik süreçlerini tanımlamaktır.

Modellerin ortaya çıkmasında, ilgili dönemin donanım ve yazılım teknolojileri ile sektör ihtiyaçları önemli rol oynamıştır.

Süreçlerin içsel ayrıntıları ya da süreçler arası ilişkilerle ilgilenmez.

Özetle yazılım üretim işinin genel yapılmış çalışma düzenine ilişkin rehberler olarak kullanılabilir.

**Örnek:**

- Geleneksel modeller (Çaglayan (“waterfall”), evrimsel, döngüsel ...vb.)
- Çevik (“Agile”) Modeller (Uçdeger (“extreme”) programlama modeli – XP )

# Süreç Modelleri Neden Önemlidir?

---

- Endüstri kaliteye önem vermektedir.
  - (örn. performans, üretkenlik)
- Deneyimler göstermektedir ki süreçlerin ürünlerin kalitesine kayda değer etkisi vardır.
  - Ürünlerin istenen kalitede olmasını süreçleri kontrol ederek daha iyi sağlayabiliriz.
- Yönetici ve geliştiricilerin, ***yazılım geliştirme sürecinin karışıklığı ile baş etmelerini*** sağlarlar .

# Süreç Modelleri

## Düzenleyici Süreç Modelleri

- Kodla ve Düzelt (Code and Fix)
- Gelişigüzel Model
- Barok Modeli
- Çağlayan/Şelale Modeli (Waterfall Model)
- V Modeli (V-shaped Model)
- Prototipleme
- Helezonik Model (Spiral Model)
- Evrimsel Geliştirme Modeli (Evolutionary Development)
- Artırımsal Geliştirme Modeli (Incremental Development)
- Araştırma Tabanlı Model (Resource Based Model)
- Formal Sistem Geliştirme (Formal System Development)
- Bileşen Tabanlı Geliştirme (Component Based Development)

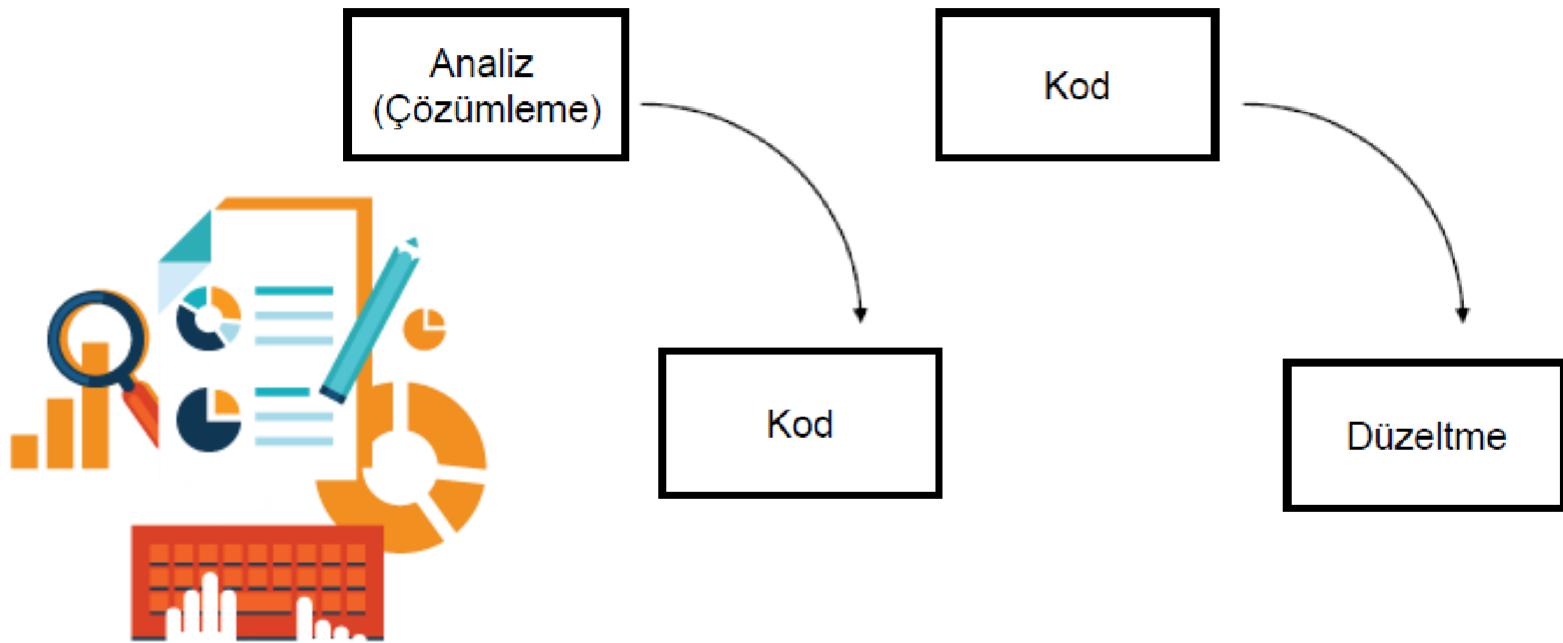
## Birleşik Süreç

- Birleşik Sürecin Aşamaları

## Çevik Yazılım Süreci

- Uçdeger Programlama (“Extreme Programming – XP”)
- Scrum
- Özellik Güdümlü Geliştirme (“Feature-Driven Development – FDD”)
- Çevik Tümlesk Süreç (“Agile Unified Process – AUP”)

# Kodla ve Düzelt ( Code and Fix)



# Kodla ve Düzelt - Avantajları

- Tüm gereken yeterli olacak kadar gayrettir.
- Tüm adımlardaki gayret direk olarak ürüne katkı sağladığından çoğu müşteri ödeme yapmaktan mutlu olur.
- Eğer ürün onu yapanlar tarafından kullanılacaksa avantajlıdır.



# Kodla ve Düzelt - Dezavantajları

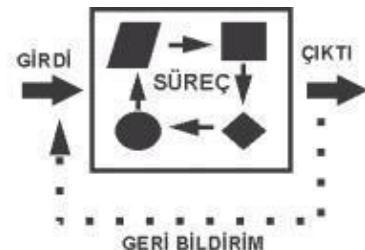
---

- Kodlamaya başlamadan önce değişiklik tahmin edilmediğinden, birbirini izleyen değişikliklerden sonra kod karmaşık bir hale gelir ve daha sonraki düzeltmeleri yapmak daha da zorlaşır.
- Geliştirilen sistemin boyutunun artması, yapısal olmayan bir şekilde karmaşıklığının yönetilmesini zorlaştırır.
- Müşterinin sürece dahil edilmemesi kullanıcı ihtiyaçlarına uygun olmamasına yol açar.
- Bireysel geliştiriciler için uygundur, takımlar için değil.

# Gelişigüzel Model

---

- Geliştirme ortamında herhangi bir model ya da yöntem kullanılmaz.
- Geliştiren kişiye bağımlı (belli bir süre sonra o kişi bile sistemi anlayamaz ve geliştirme güçlüğü yaşar).
- İzlenebilirliği ve bakımı oldukça zor.
- 60'lı yıllarda, daha çok tek kişilik üretim ortamlarında kullanılan yöntemlerdir.
- Yani basit programlama yöntemidir..



# Barok Modeli

---

**Gerçekleştirim aşamasına daha fazla ağırlık veren bir model olup, günümüzde kullanımı önerilmemektedir.**

İnceleme

Çözümleme

Tasarım

Kodlama

Modül Testleri

Altsistem Testleri

Sistem testi

Belgeleme

Kurulum

- Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği model.
- Barok modeli 70'li yılların ortalarından başlanarak kullanılmaya başlanmıştır.
- Belgelemeyi ayrı bir süreç olarak ele alır, ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür.
- Halbuki, günümüzde belgeleme yapılan işin doğal bir ürünü olarak görülmektedir.
- Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlı değil.

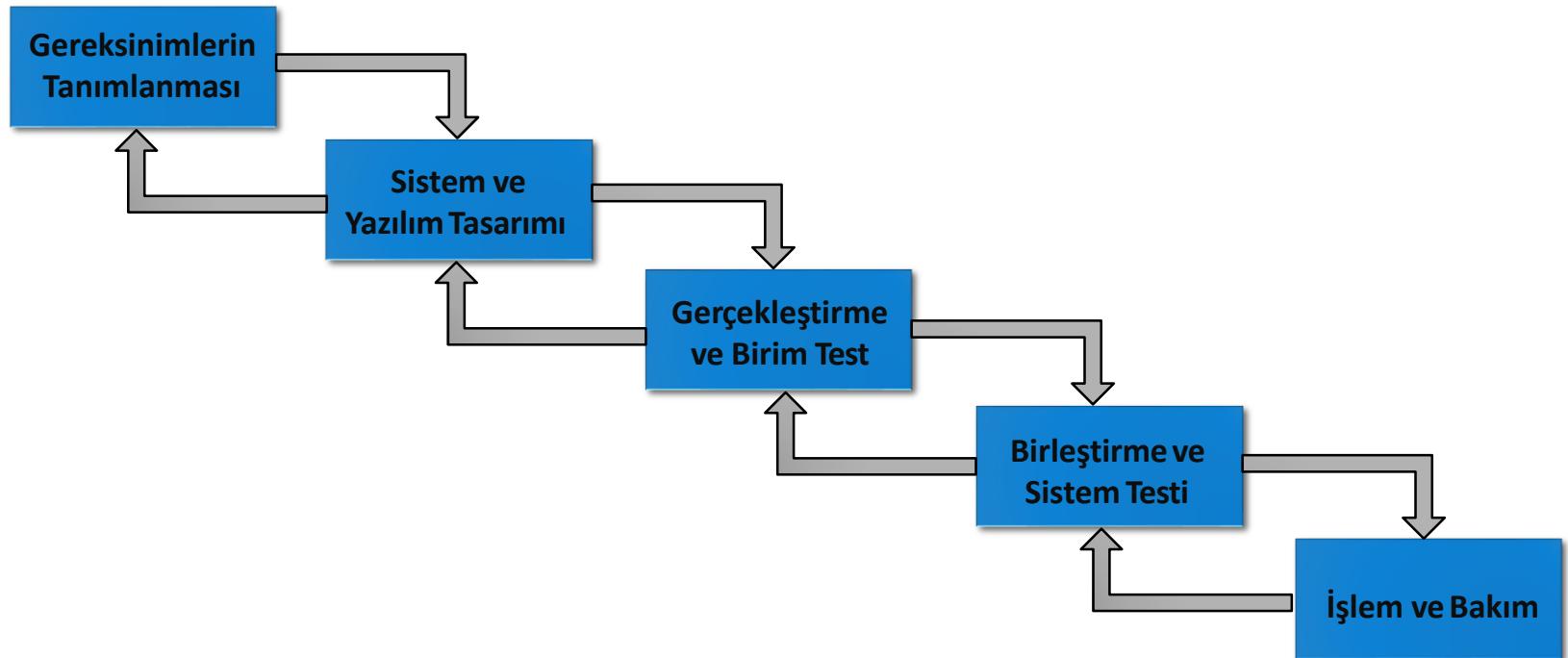
# Çağlayan Modeli

---

- Yaşam döngüsü temel adımları baştan sona en az bir kez izleyerek gerçekleştirilir.
- **İyi tanımlı** projeler ve **üretimi az zaman gerektiren** yazılım projeleri için uygun bir modeldir.
- **Geleneksel model** olarak da bilinen bu modelin kullanımı günümüzde giderek azalmaktadır.
- Barok modelin aksine **belgeleme** işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür.
- Barok modele göre geri dönüşler iyi tanımlanmıştır.
- Yazılım tanımlamada belirsizlik yok (ya da az) ise ve yazılım üretimi çok zaman almayacak ise uygun bir süreç modelidir.
- Bir sonraki aşama, önceki aşama tamamlanmadan başlayamaz.
- Her aşamanın sonucu bir ya da birden fazla onaylanan (imzalanan) belgedir.
- Gerektiğinde geliştirme aktivitelerinde iterasyonlar (tekrarlamalar) olabilir.

# Çağlayan/Şelale Modeli (Waterfall Model)

---



# Çağlayan Modeli - Aşamaları

---

**Gereksinim Tanımlama:** Gerçekleştirilecek sistemin gereksinimlerinin belirlenmesi isidir.

- Müşteri ne istiyor? Ürün ne yapacak, ne işlevsellik gösterecek?

**Sistem ve Yazılım Tasarımı:** Gereksinimleri belirlenmiş bir sistemin yapısal ve detay tasarımını oluşturma isidir.

- Ürün, müşterinin beklediği işlevselligi nasıl sağlayacak?

**Gerçekleştirme ve Birim Test:** Tasarımı yapılmış bir yazılım sisteminin kodlanarak gerçekleştirilmesi isidir.

- Yazılım ürünü, tasarımı gerçekleştirecek şekilde kodlandı mı?

**Birleştirme ve Sistem Testi:** Gerçekleştirilmiş sistemin beklenen işlevselligi gösterip göstermediğini sınama işlemidir.

- Ürün, müşterinin beklediği işlevselligi sağlıyor mu?

**İşlem ve Bakım:** Müşteriye teslim edilmiş ürünü, değişen ihtiyaçlara ve ek müşteri taleplerine göre güncelleme isidir.

- Ürün müşteri tarafından memnuniyetle kullanılabilir mi?

# Çağlayan Modeli - Avantajları

---

- Müşteriler ve son kullanıcılar tarafından da iyi bilenen anlaşılabilen adımlardan oluşur.
- İterasyonlar (tekrarlamalar) bir sonraki ve bir önceki adımlarla gerçekleşir, daha uzak adımlarla olması nadirdir.
- Değişiklik süreci yönetilebilir birimlere bölünmüştür.
- Gereksinim adımı tamamlandıktan sonra sağlam bir temel oluşur.
- Erken işin miktarını arttırır.

# Çağlayan Modeli - Avantajları

- Proje yöneticileri için işin dağılımını yapma açısından kolaydır.
- Aşamaları iyi anlaşılabilir.
- Gereksinimleri iyi anlaşılabilen projelerde iyi çalışır.
- Kalite gereksinimlerinin bütçe ve zaman kısıtlamasında göre çok daha önemli olduğu projelerde iyi çalışır.

# Çağlayan Modeli Problemleri

## Problem - 1

Test aşaması geliştirme sürecinin en sonunda yapılır. Hatalar önemli yeniden tasarım gerekliliğini oluşturur.

## Çözüm

1. Çözümleme aşamasının önüne bir ön-tasarım aşaması eklenir böylece programlama kısıtlamaları önceden anlaşılabilir.

2. Her aşamanın sonunda genişletilmiş belgelendirme yapılır

## Neden?

Erken aşamalarda tasarım= belgelendirme

Etkili yeniden tasarıma izin verir

Proje ile ortak anlayış



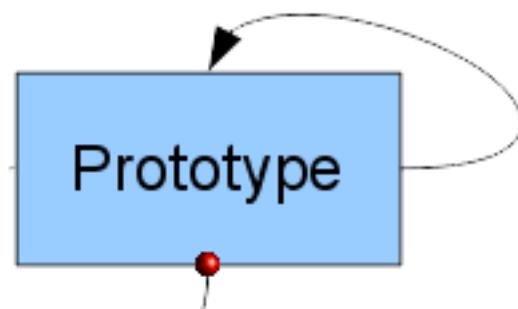
# Çağlayan Modeli Problemleri

## Problem - 2

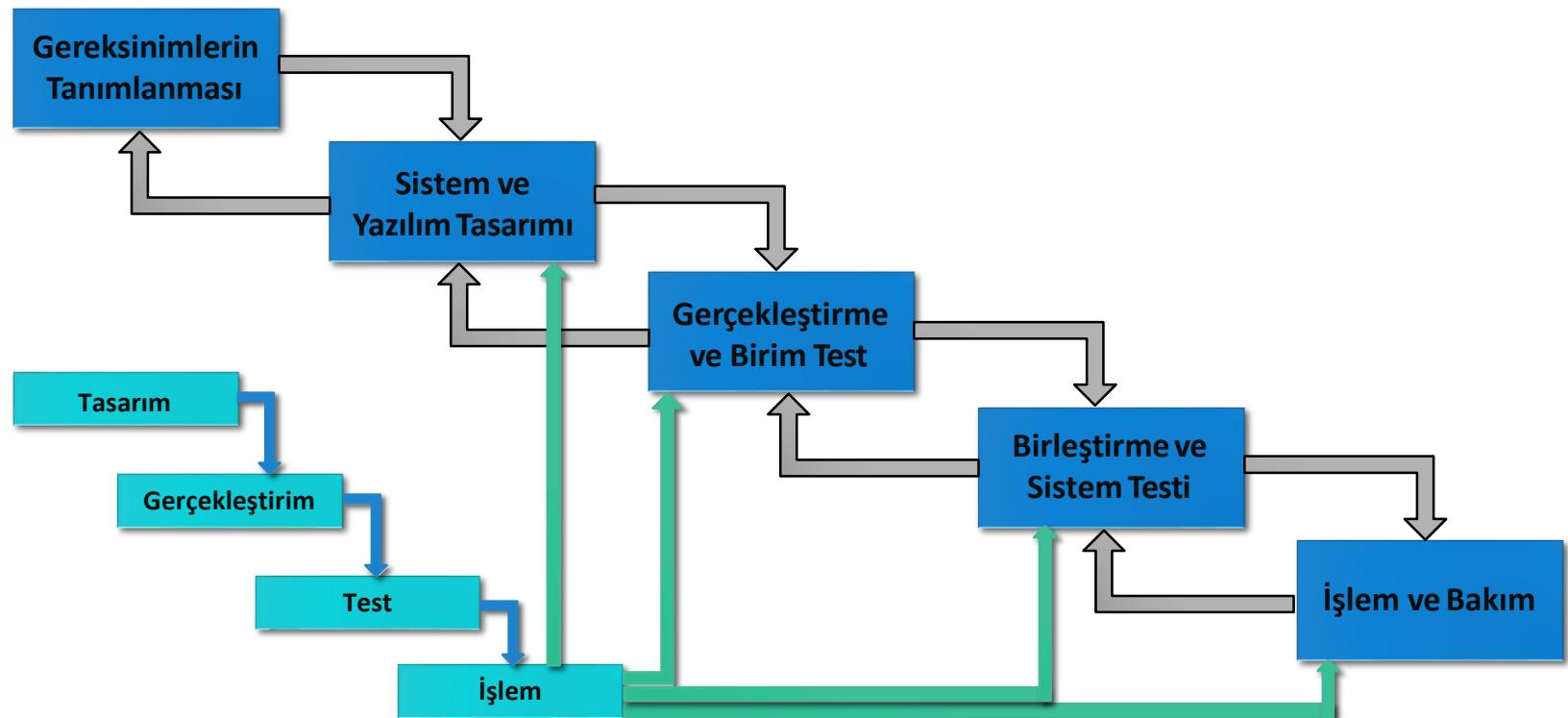
Eğer ürün tamamıyla orijinal ise, sistemi yapmadan önce biraz deneysel testlerin yapılması gereklidir.

## Çözüm

Bazı anahtar hipotezleri sınamak için bir prototip yap.



# Prototip yaptıktan sonra



# Çağlayan Modeli Problemleri

## Problem - 3

Önceden anlaşma sağlanırsa bile yazılımın ne yapacağı konusu yorumu açiktır.  
Kullanıcılar kaliteyi en sondan önce anlayamazlar

## Çözüm

Teslim etmeden önce sürece müşteriyi de dahil et.  
- Gözden geçirmeler



# Çağlayan Modeli - Diğer dezavantajları

---

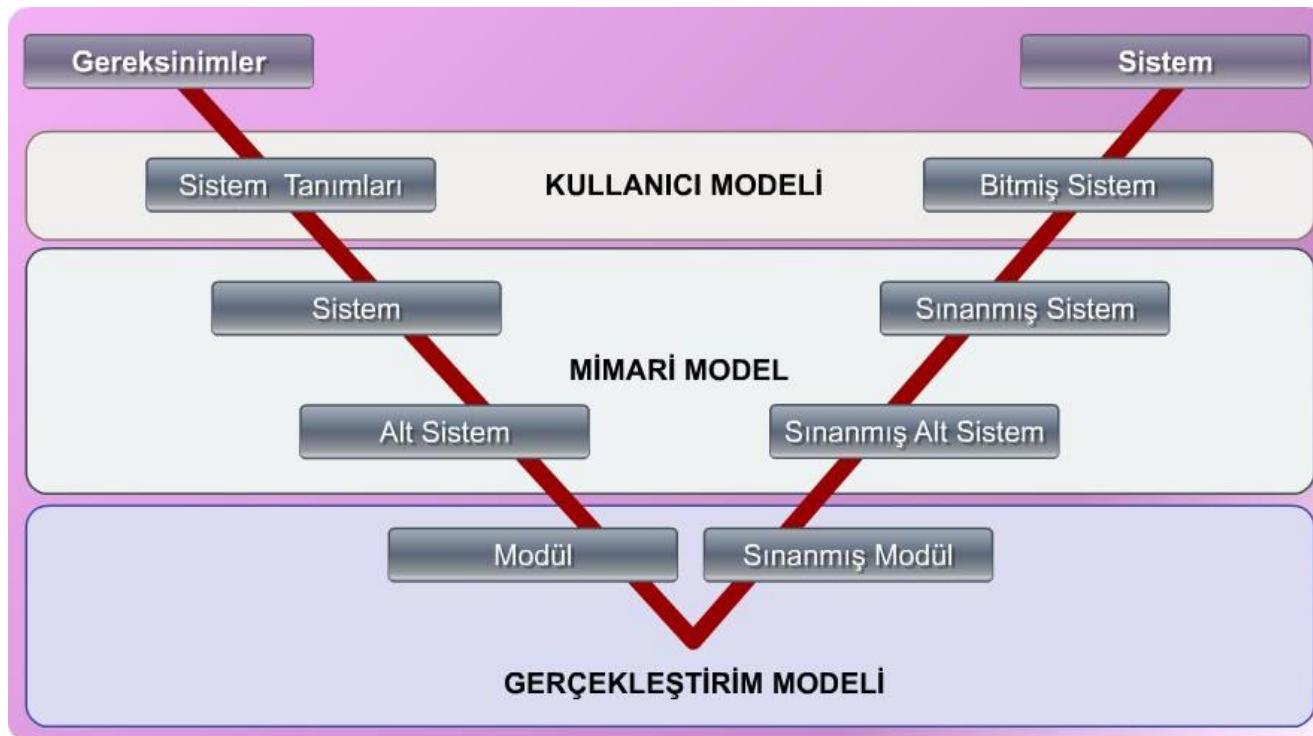
- Bitirme kriteri olarak belgelendirmeye önem verilmektedir.
  - Bazı alanlar için mümkünken (derleyiciler, işletim sistemleri, vb.) etkileşimli son kullanıcı uygulamaları gibi alanlar için zordur.
- Sistem geliştirilmesi süresince de gereksinimler sıklıkla değişir.
  - Çağlayan modeli gereksinimlerin çok iyi anlaşılabildiği durumlarda kullanılmalıdır.
- İki ya da daha önceki fazlara gitmek çok maliyetlidir.
  - bu durumda da gerektiğinde tüm fazı yeniden gerçekleştirmek çok büyük bir iştir.
- Bir faz tamamlanmadan diğerine geçilememesi riski arttıran.

# V Modeli (V-shaped Model)

- Proje ve gereksinim planlaması
- Ürün gereksinimleri ve belirtim çözümlemesi
  - Mimari ve yüksek seviye tasarım
  - Detaylı tasarım
  - Kodlama
- Birim testi
  - Tümleştirme ve test
- Sistem ve kabul edilme testleri
- Üretim, işletim ve sürdürülebilirlik

# V Modeli

---



# V Modeli

---

Sol taraf üretim, sağ taraf sınama işlemleridir.

V süreç modelinin temel çıktıları;

## Kullanıcı Modeli

Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır.

## Mimari Model

Sistem tasarımı ve oluşacak altsistem ile tüm sistemin sınama işlemlerine ilişkin işlevler.

## Gerçekleştirim Modeli

Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar.

# V Modeli

- Belirsizliklerin az, **iş tanımlarının belirgin** olduğu BT projeleri için uygun bir modeldir.
- Model, **kullanıcının projeye katkısını** artırmaktadır.
- BT projesinin **iki aşamalı olarak ihale** edilmesi için oldukça uygundur:
  - İlk ihalede kullanıcı modeli hedeflenerek, iş analizi ve kabul sınamalarının tanımları yapılmakta,
  - İkinci ihalede ise ilkinde elde edilmiş olan kullanıcı modelitasarlanıp, gerçekleştirmektedir.

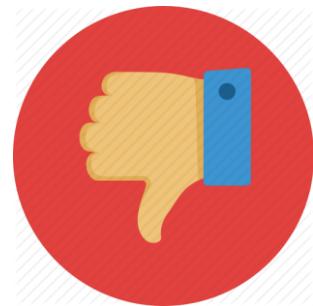
# V Modeli - Avantajları

- Verification ve validation planları erken aşamalarda vurgulanır.
- Verification ve validation sadece son üründe değil tüm teslim edilebilir ürünlerde uygulanır.
- Proje yönetimi tarafından takibi kolaydır
- Kullanımı kolaydır.



# V Modeli - Dezavanatjları

- Aynı zamanda gerçekleştirilebilecek olaylara kolay imkan tanımaz.
- Aşamalar arasında tekrarlamaları kullanmaz.
- Risk çözümleme ile ilgili aktiviteleri içermez.

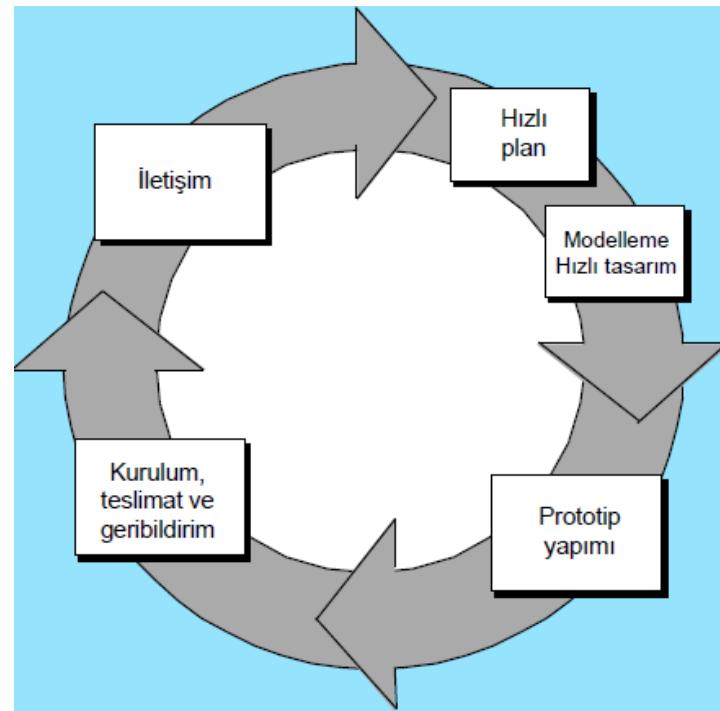


# Prototipleme

---

- Gereksinim tanımılama fazında hızlıca yapılan kısmi gerçekleştirmeye.
- Gereksinimler netleşikçe prototipi düzelt.
- Müşteri memnun olana kadar düzeltmelere devam et.

# Prototipleme



# Prototipleme - Avantajları

Kullanıcı sistem gereksinimlerini görebilir.

Karmaşa ve yanlış anlaşılmaları engeller.

Yeni ve beklenmeyen gereksinimler netleştirilebilir.

Risk kontrolü sağlanır.



# Prototipleme - Dezavantajları

Belgelendirmesi olmayan hızlı ve kirli (quick and dirty) prototipler.

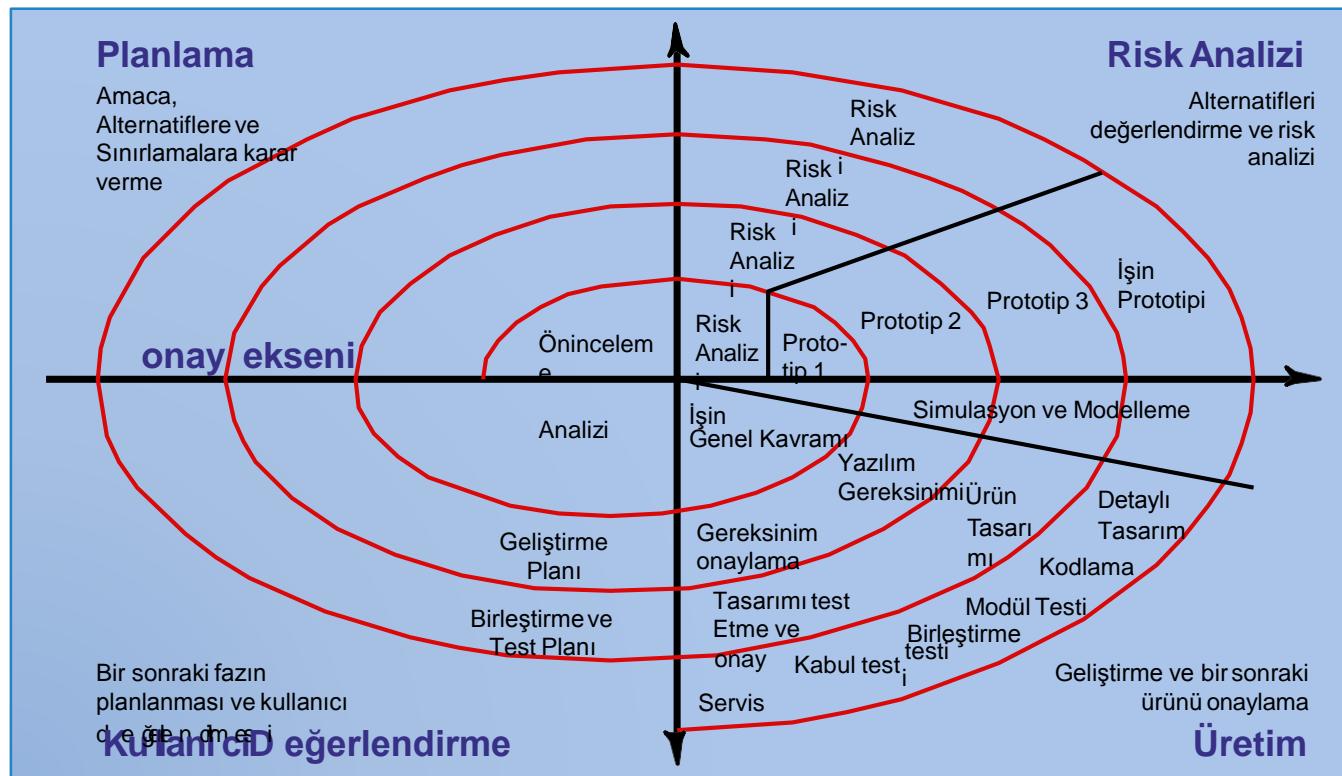
Prototip hedefleri net değilse kod hackleme ya da jenga başlar.

Düzelte aşaması atlanırsa, düşük performansa yol açar.



Müşteri prototipten de son ürün gibi görünüm ve etki bekler.

# Helezonik Model (Spiral Model)



# Helezonik Model - Aşamaları

## **Planlama**

- Üretilen ara ürün ile bütünlendirme

## **Risk Analizi**

- Risk seçeneklerinin araştırılması ve risklerin belirlenmesi

## **Üretim**

- Ara ürünün üretilmesi

## **Kullanıcı Değerlendirmesi**

- Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sınamalar ve değerlendirmeler

# Helezonik Model

- Risk Analizi Olgusu ön plana çıkmıştır.
- Hedefler, alternatifler ve kısıtlamalar belirlenir.
- Alternatifler değerlendirilir, riskler belirlenip çözülür.
- Aşamanın ürünü geliştirilir.
- Sonraki aşama planlanır.
- Her döngü bir aşamayı ifade eder. Doğrudan tanımlama, tasarım,... vs. gibi bir aşama yoktur.
- Yinelemeli artımsal bir yaklaşım vardır.
- Prototip yaklaşımı vardır.

# Helezonik Geliştirme

- Süreç arka arkaya devam eden sıralı aktiviteler şeklinde gösterilmek yerine spiral şekilde gösterilir.
- Spiral üzerindeki her bir halka bir fazı gösterir.
- Belirtim, tasarım gibi kesin fazlar yoktur – spiral deki halkalar neye ihtiyaç varsa onu gerçekleştirmek için seçilir.
- Süreç boyunca risklerin değerlendirilmesi ve çözümü açık olarak yapılır.

# Helezonik Model - Avantajları

- Kullanıcılar sistemi erken görebilirler.
- Geliştirmeyi küçük parçalara böler . En riskli kısımlar önce gerçekleştirilir.
- Pek çok yazılım modelini içinde bulundurur.
- Riske duyarlı yaklaşımı potansiyel zorlukları engeller.
- Seçeneklere erken dikkate odaklanır.
- Hataları erken gidermeye odaklanır.
- Yazılım-donanım sistemi geliştirme için bir çerçeve sağlar.

# Helezonik Model - Problemeler

- Küçük ve düşük riskli projeler için pahalı bir yöntemdir.
- Komplekstir (karmaşık).
- Spiral sonsuza gidebilir.
- Ara adımların fazlalığı nedeniyle çok fazla dokümantasyon gerektirir.
- Büyük ölçekte projeler
- Kontrat tabanlı yazılıma uymaz.
  - Yazılımın içten geliştirileceğini varsayar.
  - Kontrat tabanlı yazılımlar adım adım anlaşma esnekliğini sağlamaz.
- Öznel risk değerlendirme deneyimine dayanır.
  - Yüksek riskli öğelere yoğunlaşmak, yüksek riskli öğelerin doğru belirlenmesini gerektirir.

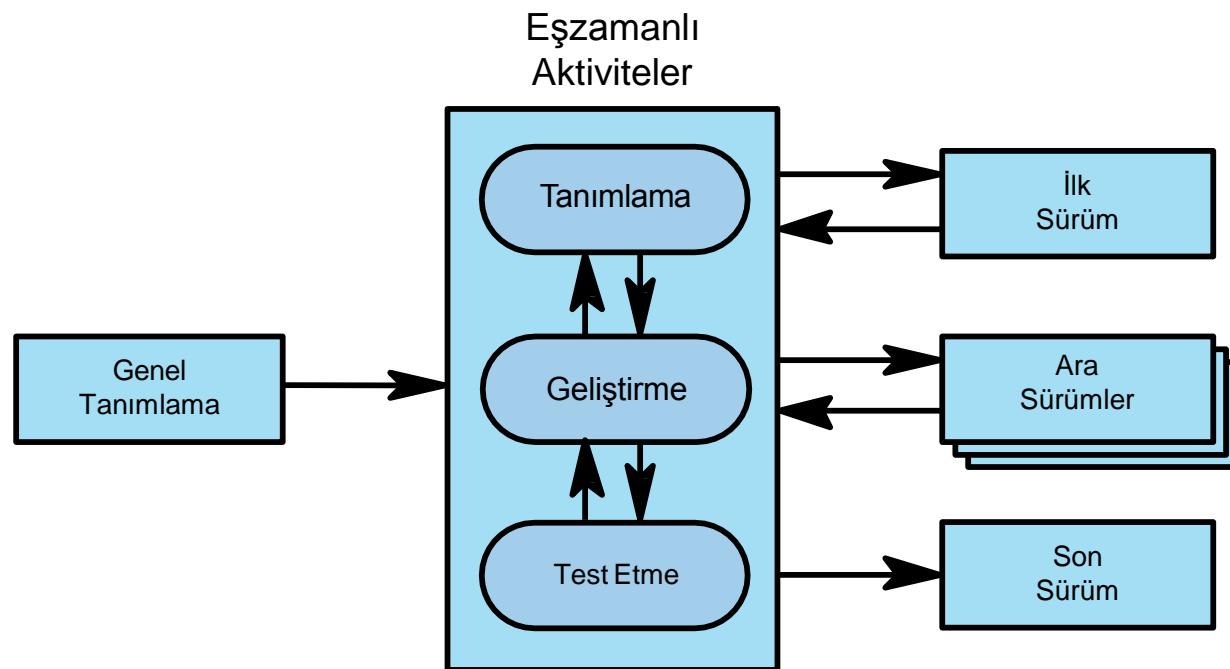
# Evimsel Geliştirme Modeli

(Evolutionary Development Model)

---

- İlk tam ölçekli modeldir.
- Anahtar gereksinimleri ile başlangıç sistemi geliştirilir.
- Müşteri geribildirimleri ile sistem pek çok versiyonla yavaş yavaş geliştirilir.
- Belirtim (specification), geliştirme ve geçerleme (validation) aktivitleri koşut zamanlı yürütülür.
- **Coğrafik olarak geniş alana yayılmış, çok birimli organizasyonlar** için önerilmektedir (banka uygulamaları).
- Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselligi içermektedirler.
- **Pilot uygulama** kullan, test et, güncelle diğer birimlere taşı.
- Modelin başarısı **ilk evrimin başarısına** bağlıdır.

# Evrimsel Geliştirme Modeli



# Evrimsel Geliştirme Modeli

**İki çeşit evrimsel geliştirme vardır:**

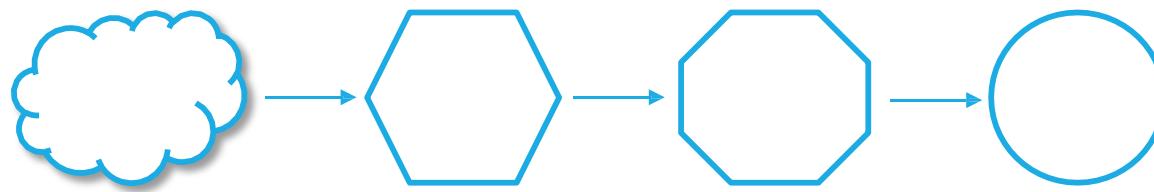
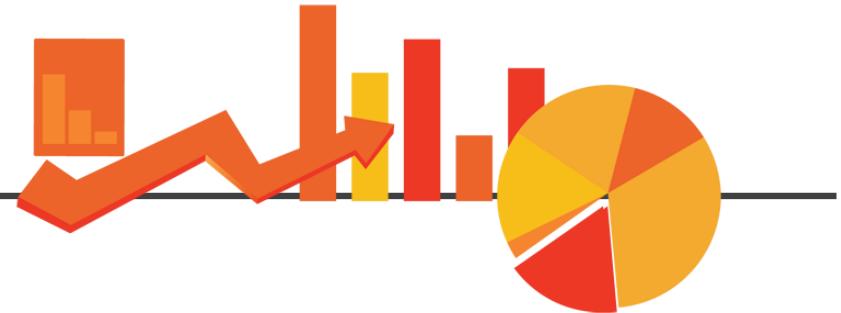
- Keşifçi geliştirme (exploratory development)
  - Hedef: Müşterinin gereksinimlerini incelemek için müşteri ile çalışıp son sistemi teslim etmek
  - İyi anlaşılan gereksinimlerle başlanmalıdır.

**“ Ne istediğimi sana söyleyemem ama onu gördüğümde bilirim”**

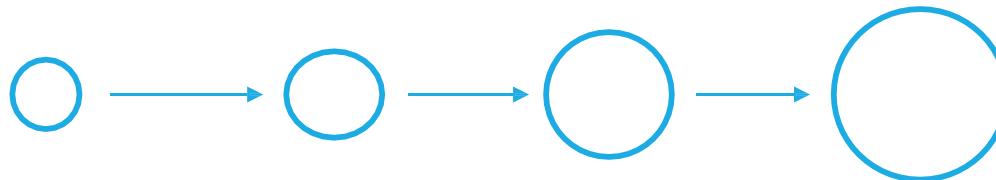
- Atılacak prototipleme (throw-away prototyping)
  - Hedef: Sistem gereksinimlerini anlamak
  - Tam anlaşılmamış gereksinimlerle başlar



# Karşılaştırma



Çağlayan Modeli



Evrimsel Geliştirme

Zaman



# Evrimsel Geliştirme Modeli - Avantajlar

---

- Kullanıcıların kendi gereksinimlerini daha iyi anlamalarını sağlar.
- Sürekli değerlendirme erken aşamalardaki geliştirme risklerini azaltır.
- Hatalar azalır.



# Evrimsel Geliştirme Modeli - Problemler

---

- Sürecin görünürlüğü azdır (düzenli teslim edilebilir ürün yoktur).
- Sistemler sıkılıkla iyi yapılandırılmaz (sürekli değişiklik yazılımın yapısına zarar verir).
- Bakımı zordur.
- Yazılım gereksinimini yenilemek gerekebilir.



# Evrimsel Geliştirme - Uygulanabilirliği

---

- Küçük ve orta boyutlu etkileşimli sistemler (500.000 LOC dan daha az olan).
- Büyük bir sistemin parçaları (ör. Kullanıcı ara yüzleri).
- Kısa süreli kullanılacak sistemler.

# Örnek

---

- Çok birimli banka uygulamaları.
- Önce sistem geliştirilir ve Şube-1'e yüklenir.
- Daha sonra aksaklılıklar giderilerek geliştirilen sistem Şube-2'ye yüklenir.
- Daha sonra geliştirilen sistem Şube-3'e,... yüklenir.
- Belirli aralıklarla eski şubelerdeki güncellemeler yapılır.

# Yazılım Süreç Modellerinde Süreç Tekrarı (“Process Iteration”)

---

- Yazılım süreç modelleri tek bir defada uygulanmak yerine, birkaç tekrarda uygulanabilir.
  - Örneğin, geniş kapsamlı 5 alt sistemden oluşan bir sistemin; ilk alt sistemi için çağrıyan modeli uygulandıktan sonra, geri kalanı için çağrıyan modeli tekrar uygulanabilir.
  - Bu şekilde geliştirme riskleri en aza indirilerek ilk tekrarda kazanılan deneyimden, sistemin geri kalanı geliştirilirken faydalанılabilir.
- Hangi süreç modelinin, sistemin hangi bölümleri için ve kaç tekrarda uygulanacağına proje basında karar verilir.
- Süreç tekrarıyla yakından ilişkili iki geleneksel model vardır:
  - Artırımsal (“incremental”) model
  - Döngüsel (“spiral”) model

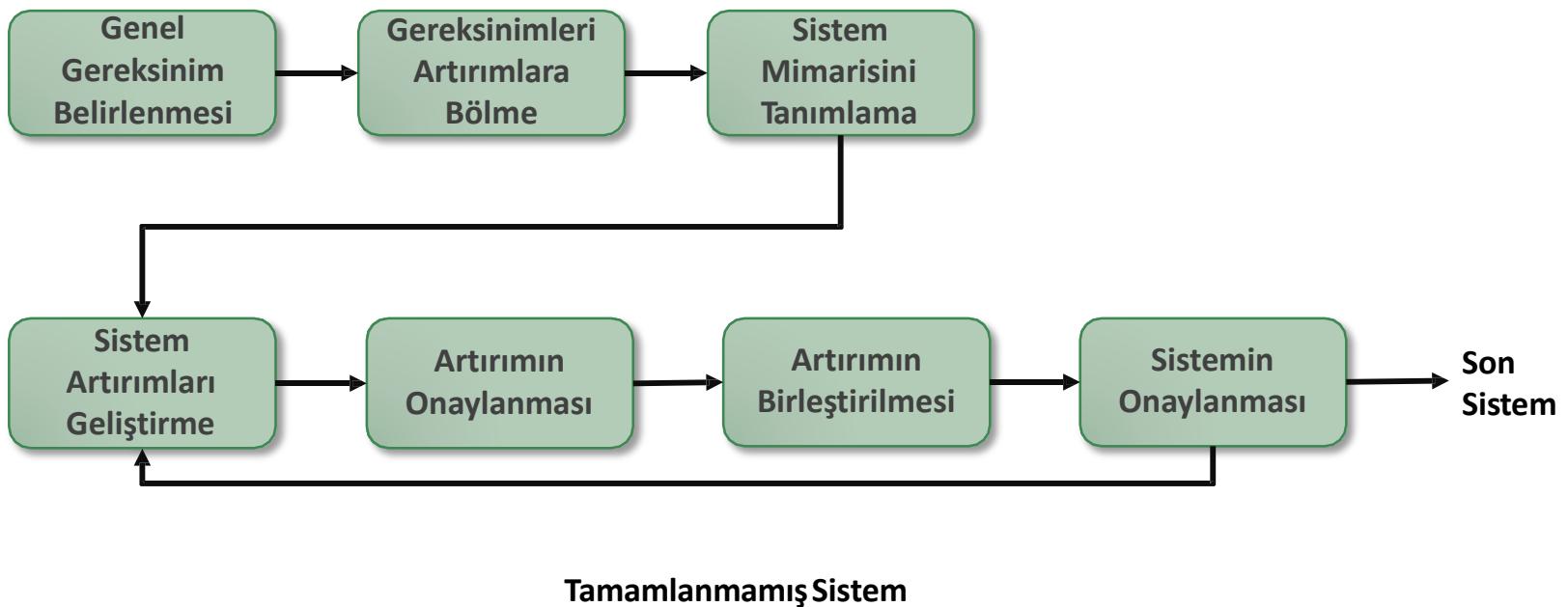
# Artırımsal Geliştirme Modeli

(Incremental Development Model)

---

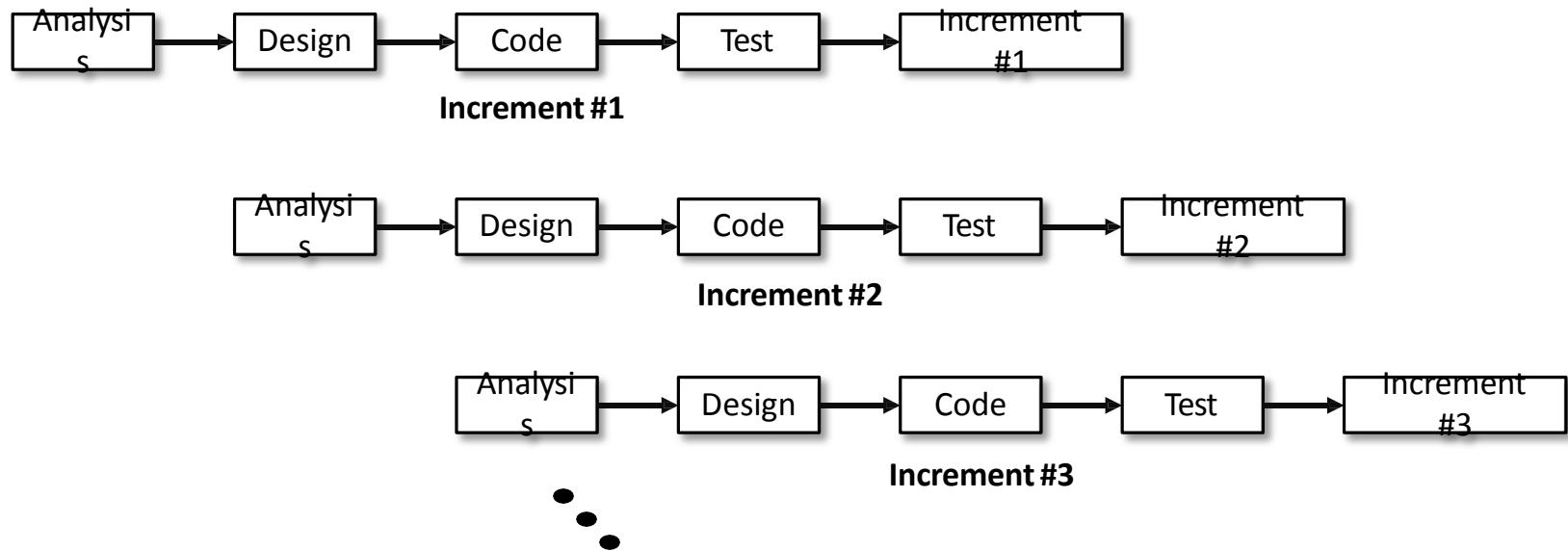
- Üretilen **her yazılım sürümü birbirini kapsayacak** ve giderek artan sayıda işlev içerecek şekilde geliştirilir.
  
- Öğrencilerin bir dönem boyunca geliştirmeleri gereken bir programlama ödevinin 2 haftada bir gelişiminin izlenmesi (bitirme tezleri).
  
- Uzun zaman alabilecek ve sistemin eksik işlevlikle çalışabileceği türdeki projeler bu modele uygun olabilir.
  
- Bir taraftan kullanım, diğer taraftan üretim yapılır.

# Artırımsal Geliştirme Modeli



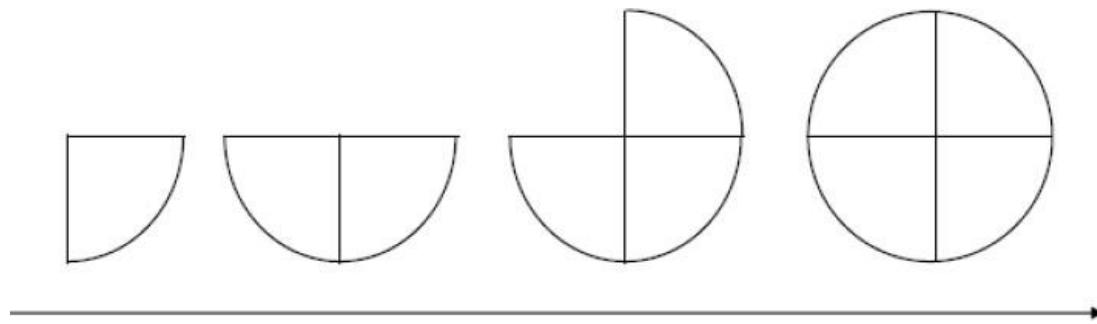
# Arttırımsal Geliştirme Modeli

Aslında Çağlayan modelinin örtüsen şekilde uygulanmasıdır.



# Artırımsal Geliştirme Modeli

Çağlayan modeli ve Evrimsel geliştirme arası bir model:



# Artırımsal Geliştirme Modeli - Avantajları

---

- Sistem için gerekli olan gereksinimler müşterilerle belirlenir
- Gereksinimlerin önemine göre teslim edilecek artımlar belirlenir
- Öncelikle en önemli gereksinimleri karşılayan çekirdek bir sistem geliştirilir.
- Erken artımlar prototip gibi davranışarak, gereksinimlerin daha iyi anlaşılmasını sağlar
- Tüm projenin başarısız olma riskini azaltır
- En önemli sistem özellikleri daha fazla sınanma (test edilme) imkanı bulmuş olur.
- Divide and Conquer (Böl ve Yönet) yaklaşımıdır



## Artırımsal Geliştirme Modeli - Dezavantajları

---

- Artımları tanımlamak için tüm sistemin tanımlanmasına ihtiyaç vardır.
- Gereksinimleri doğru boyuttaki artımlara atamak bazen zor olabilir.
- Deneyimli personel gerektirir.
- Artımların kendi içlerinde tekrarlamalara izin vermez.



# Araştırma Tabanlı Model

---

- **Yap-at** prototipi olarak da bilinir.
- Araştırma ortamları **bütünüyle belirsizlik** üzerine çalışan ortamlardır.
- Yapılan işlerden edinilecek sonuçlar belirgin değildir.
- Geliştirilen **yazılımlar genellikle sınırlı sayıda kullanılır** ve kullanım bittikten sonra işe yaramaz hale gelir ve atılır.
- Model-zaman-fiyat kestirimi olmadığı için **sabit fiyat sözleşmelerinde uygun değildir.**

# Örnek

- En Hızlı Çalışan asal sayı test programı!
- En Büyük asal sayıyı bulma programı!
- Satranç programı!



# Formal Sistem Geliştirme

(Formal System Development)

---

- Cleanroom yazılım geliştirme
- Matematiksel belirtimin farklı gösterim şekilleri ile çalıştırılabilir programa dönüştürülmesine dayalıdır.
- Formal belirtim, tasarım ve geçerleme kullanarak yazılımda doğruluğun geliştirilmesini vurgular.
- Yazılım artımlarla geliştirilir.
- Sürekli tümlestirme vardır ve fonksiyonellik tümlestirilen yazılım artımları ile artar.
- Felsefesi pahalı hata ayıklama işlemini engellemek için kodu ilk yazarken doğru yazmak ve test aşamasından doğruluğunu sağlamak
  - Formal yöntemler
  - Z dili

# Formal Sistem Geliştirme

## **Problemleri**

- Teknikleri uygulayabilmek için eğitim ve özel beceriler gerekmektedir
- Kullanıcı arayüzü gibi sistemin bazı kısımlarını formal olarak belirtmek zordur.

## **Uygulanabilirliği**

- Sistem kullanıma konmadan emniyet ve güvenlik durumlarını sağlanması gereken kritik sistemler.

# Bileşen-Tabanlı Model (Component-Based Model)

---

Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tümleştirilmesi esasına dayanır.

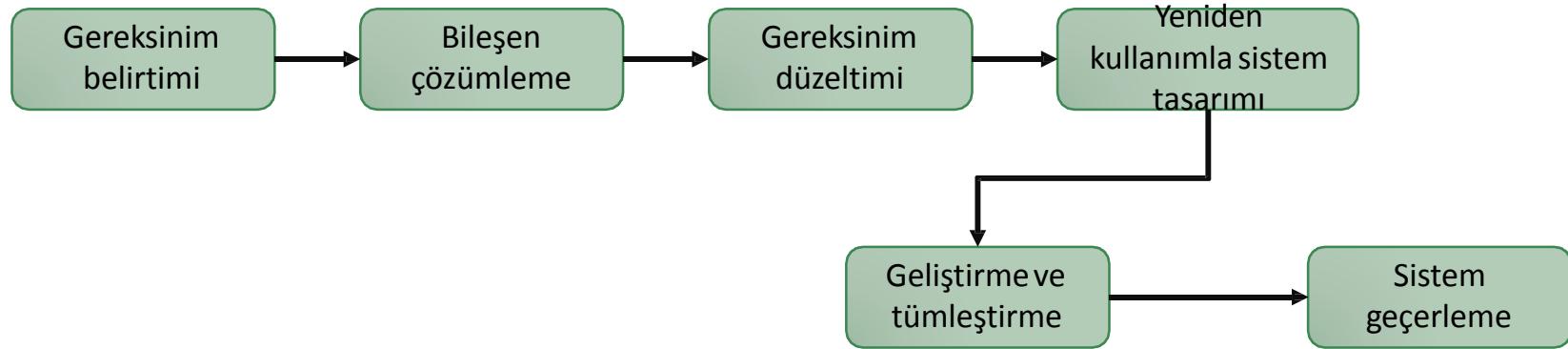
## **Süreç adımları:**

- Bileşen analizi
- Gereksinim günleme
- Bileşenler kullanarak sistem tasarıımı
- Geliştirme ve tümleştirme

Bu yaklaşım, bileşen standartlarındaki gelişmeler ilerledikçe daha yaygın olarak kullanılmaya başlanmıştır. ama halen kullanımı limitlidir.

# Bileşen-Tabanlı Model - Adımlar

---



# Metodolojiler

---

- **Metodoloji:** Bir BT projesi ya da yazılım yaşam döngüsü aşamaları boyunca kullanılacak ve birbirleriyle uyumlu yöntemler bütünü.
  
- Bir metodoloji,
  - bir süreç modelini ve
  - belirli sayıda belirtim yöntemini içerir
  
- Günümüzdeki metodolojiler genelde Çağlayan ya da Helezonik modeli temel almaktadır

# Bir Metodolojide Bulunması Gereken Temel Bileşenler (Özellikler)

- Ayrıntılandırılmış bir süreç modeli
- Ayrıntılı süreç tanımları
- İyi tanımlı üretim yöntemleri
- Süreçlerarası arayüz tanımları
- Ayrıntılı girdi tanımları
- Ayrıntılı çıktı tanımları
- Proje yönetim modeli
- Konfigürasyon yönetim modeli
- Maliyet yönetim modeli
- Kalite yönetim modeli
- Risk yönetim modeli
- Değişiklik yönetim modeli
- Kullanıcı arayüz ve ilişki modeli
- Standartlar

# Bir Metodoloji Örneği

- Yourdan Yapısal Sistem Tasarımı Metodolojisi.
  
- Kolay uygulanabilir bir model olup, günümüzde oldukça yaygın olarak kullanılmaktadır.
  
- Çağlayan modelini temel almaktadır.
  
- Bir çok CASE aracı tarafından doğrudan desteklenmektedir.

# Yourdon Yapısal Sistem Tasarım Metodolojisi

---

Aşama	Kullanılan Yöntem ve Araçlar	Ne için Kullanıldığı	Çıktı
Planlama	Veri Akış Şemaları, Süreç Belirtimleri, Görüşme, Maliyet Kestirim Yöntemi, Proje Yönetim Araçları	Süreç İnceleme  Kaynak Kestirimi  Proje Yönetimi	Proje Planı
Analiz	Veri Akış Şemaları, Süreç Belirtimleri, Görüşme, Nesne ilişki şemaları Veri	Süreç Analizi  Veri Analizi	Sistem Analiz Raporu
Analizden Tasarıma Geçiş	Akısa Dayalı Analiz, Süreç belirtimlerinin program tasarım diline dönüştürülmesi, Nesne ilişki şemalarının veri tablosuna dönüştürülmesi	Başlangıç Tasarımı  Ayrıntılı Tasarım  Başlangıç Veri tasarımları	Başlangıç Tasarım Raporu
Tasarım	Yapısal Şemalar, Program Tasarım Dili, Veri Tabanı Tabloları	Genel Tasarım  Ayrıntılı Tasarım Veri Tasarımı	Sistem Tasarım Raporu

# Özet

---

Yazılım yaşam döngüsü, herhangi bir yazılımin, üretim aşaması ve kullanım aşaması birlikte olmak üzere geçirdiği tüm aşamalar biçiminde tanımlanır.

Yazılım Yaşam Döngüsü temel adımları

- Planlama
- Çözümleme
- Tasarım
- Gerçekleştirme
- Bakım

Belirtim Yöntemleri (Software Specification Methods) - bir çekirdek süreçin fonksiyonları yerine getirmek amacıyla kullanılan yöntemler.

Süreç Modelleri (Software Process Models) - yazılım yaşam döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlayan modeller kullanılır.

**Süreç Modelleri**, Yazılım Yaşam Döngüsünde belirtilen süreçlerin geliştirme aşamasında, hangi düzen ya da sırada, nasıl uygulanacağını tanımlar.

Barok Modeli: Belgelemeyi ayrı bir süreç olarak ele alır, ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür.

# Özet

---

## Çağlayan Modeli Aşamaları:

- Gereksinim Tanımlama
- Sistem ve Yazılım Tasarımı
- Gerçekleştirme ve Birim Test
- Birleştirme ve Sistem Testi
- İşlem ve Bakım

## V süreç modelinin temel çıktıları;

- Kullanıcı Model
- Mimari Model
- Gerçekleştirim Modeli

## Helezonik Model Aşamaları:

- Planlama
- Risk Analizi
- Üretim
- Kullanıcı Değerlendirmesi

## İki çeşit evrimsel geliştirme vardır:

- Keşifçi geliştirme (exploratory development)
- Atılacak prototipleme (throw-away prototyping)

# Özet

---

- Artırımsal Geliştirme Modeli-Öğrencilerin bir dönem boyunca geliştirmeleri gereken bir programlama ödevinin 2 haftada bir gelişiminin izlenmesi (bitirmetezleri).
- Formal Sistem Geliştirme: Matematiksel belirtimin farklı gösterim şekilleri ile çalıştırılabilir programa dönüştürülmesine dayalıdır.
- Bileşen-Tabanlı Model : Sistemin COTS (“commercial-off-the-shelf”) adı verilen hazır bileşenler kullanılarak tımlaştırılması esasına dayanır.
- Bileşen-Tabanlı Model – Adımlar
  1. Gereksinim belirtimi
  2. Bileşen çözümleme
  3. Gereksinim düzeltimi
  4. Yeniden kullanımıla sistem tasarıımı
  5. Geliştirme ve tımlaştırma
  6. Sistem geçerleme

# Sorular

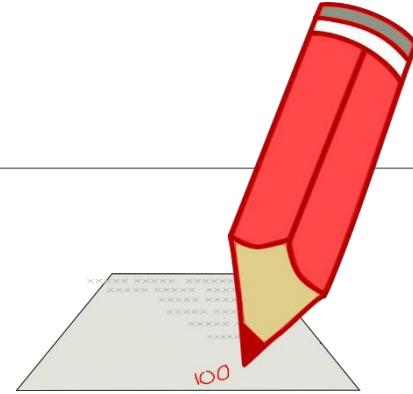
---

1. Yazılım yaşam döngüsünün temel adımlarını açıklayınız.
2. Süreç modelleri ve belirtim yöntemlerinin önemini belirtiniz.
3. Süreç modelleri ile belirtim yöntemleri arasındaki farklılıklarını belirtiniz.
4. Barok modeli tanımlayınız, yararlarını ve aksak yönlerini belirtiniz.
5. Çağlayan modeli tanımlayınız, yararlarını ve aksak yönlerini belirtiniz.
6. Helezonik modeli tanımlayınız, ayırıcı özelliklerini belirtiniz. Yararlarını ve aksak yönlerini açıklayınız.
7. V model kullanılarak geliştirilecek örnek bir proje tanımı yapınız.
8. VP süreç modeli ile geliştirilecek bir projede uygulanabilecek üç prototipleme örneği veriniz.
9. Evrimsel Geliştirme süreç modelinde Konfigürasyon yönetimi ve değişiklik denetimi neden sorundur?
10. Artımsal geliştirme süreç modelini tanımlayınız, yararlı ve aksak yönlerini belirtiniz.
11. Artımsal geliştirme süreç modeli kullanılarak geliştirilecek bir proje örneği veriniz. Gerekçenizi açıklayınız.
12. Araştırma tabanlı süreç modeli için uygun proje örnekleri veriniz.
13. Metodolojiyi tanımlayınız. Bildiğiniz metodoloji örneklerini listeleyiniz.
14. Yourdon Yapısal Sistem Geliştirme Metodolojisini tamamlayınız.
15. Süreç modelleri, belirtim yöntemleri ile metodolojiler arasındaki ilişkiyi belirtiniz.

# Ödev

---

1. Çevik Yazılım Hakkında Araştırma Yapınız.
2. Çevik Yazılım Geliştirmenin projelerdeki başarısı hakkında edindiğiniz bilgileri rapor haline getirin.



# Kaynaklar

---

- “Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.
- “Software Engineering” (8th. Ed.), Ian Sommerville, 2007.
- “Guide to the Software Engineering Body of Knowledge”, 2004.
- “Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.
- “Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.
- Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.
- Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)
- Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.
- Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN  
<http://blog.alisuleymantopuz.com/2014/08/30/yazilim-mimarisi-ve-tasarimi-nedir/>  
<http://www.akifsahman.com/?p=175>  
<https://ece.uwaterloo.ca/~se464/08ST/index.php?src=lecture>  
<http://info.psu.edu.sa/psu/cis/azzarrad/se505.htm>  
<http://www.metinakbulut.com/YAZILIM-MIMARISI/>  
[http://ceng.gazi.edu.tr/~hkaracan/source/YPY\\_H3.pdf](http://ceng.gazi.edu.tr/~hkaracan/source/YPY_H3.pdf)  
<http://iiscs.wssu.edu/drupal/node/3399>  
<http://www.cs.toronto.edu/~sme/CSC340F/slides/21-architecture.pdf>  
<http://www.users.abo.fi/lpetre/SA10/>  
<http://sulc3.com/model.html>  
<http://salyangoz.com.tr/blog/2013/11/23/digerleri/yazilim-gelistirme-surec-modelleri-3/>

# Sorularınız

---





# YMT 412-Yazılım Kalite Ve Güvencesi Çevik Yazılım Geliştirme

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bölüm-2

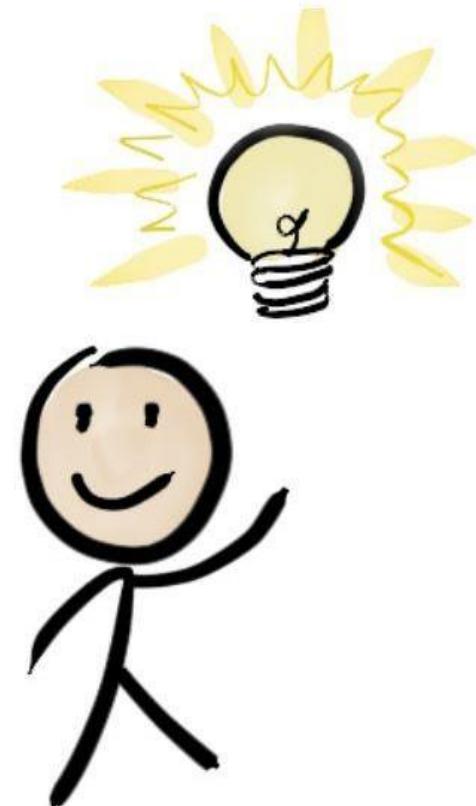
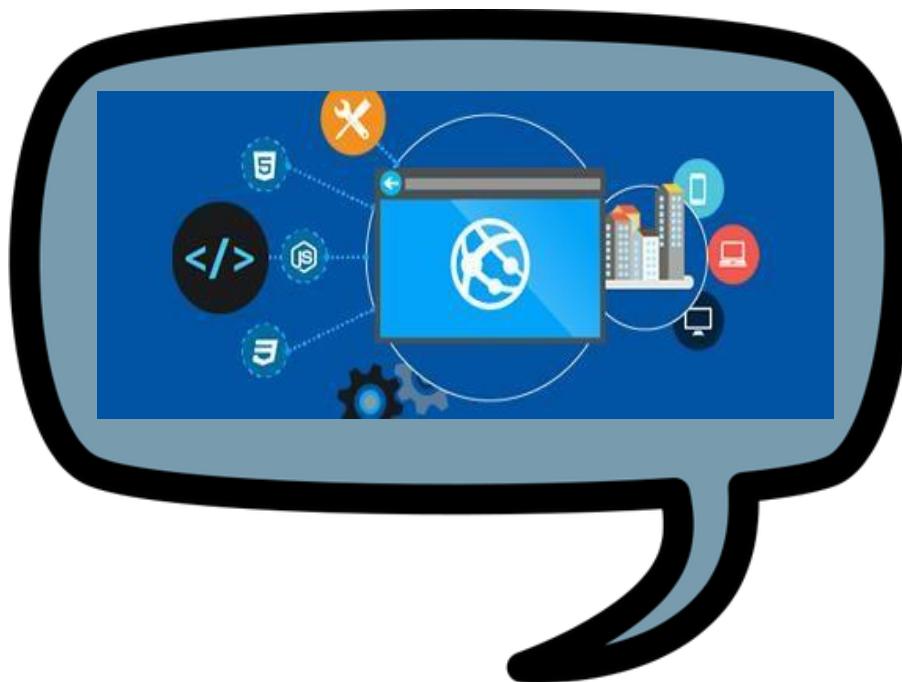
# İçindekiler

1	Günümüzde Yazılım Projelerinin Durumu.....	3
2	Çevik Yazılım Geliştirme Yöntemi.....	9
3	Geleneksel Model vs. Agile.....	17
4	Değerlendirme.....	26
5	Çevik Yazılım Şemsiyesi.....	28
6	Scrum Modeli.....	29

# 1.Günümüzde Yazılım Projelerinin Durumu

---

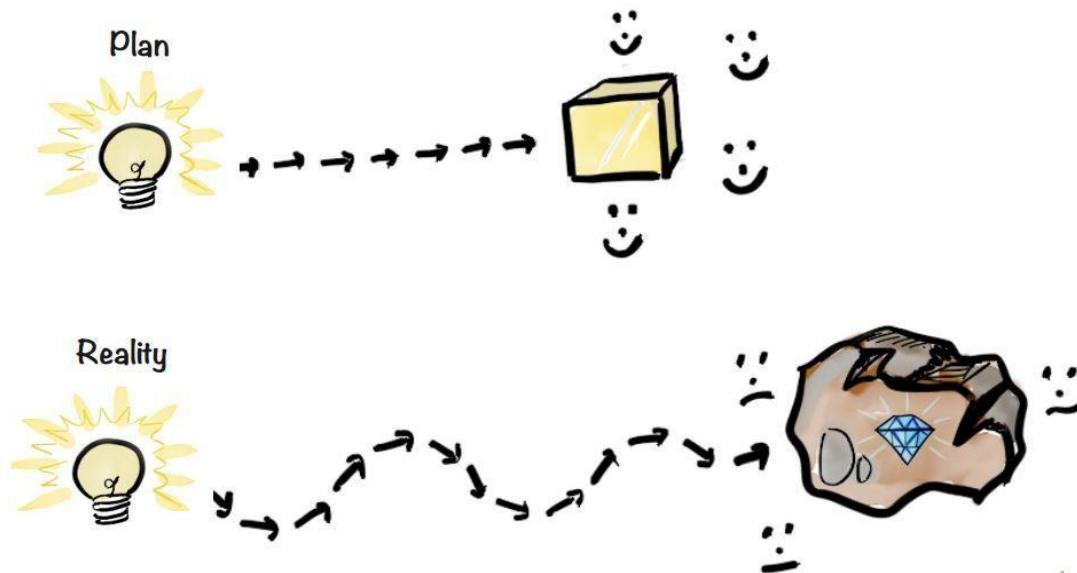
- Birçok proje harika bir fikir ile başlar!



# 1. Günümüzde Yazılım Projelerinin Durumu

---

- Bu projelerin büyük bir kısmının başarısız olması muhtemeldir!



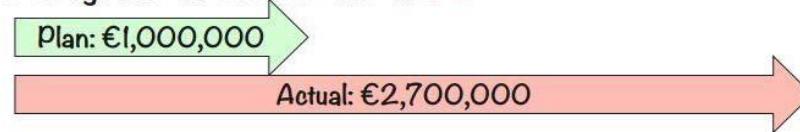
# 1. Günümüzde Yazılım Projelerinin Durumu

---

- Birçok Bilgi Teknolojisi projesi başarısız olmuş veya gecikmiştir. The Standish Group, 10 yıl içerisinde 40.000'den fazla proje üzerinde çalışmıştır.

IT project success rate 1994: 15%

Average cost & time overrun: ≈170%



IT project success rate 2004: 34%

Average cost & time overrun: ≈70%



# 1.Günümüzde Yazılım Projelerinin Durumu

---

Ülkemizde durum nasıl?

Durum	Oran
Tam başarılı	%4-5
Kısmen başarılı	%45-50
Çöpe gidenler	%50

Agile Turkey

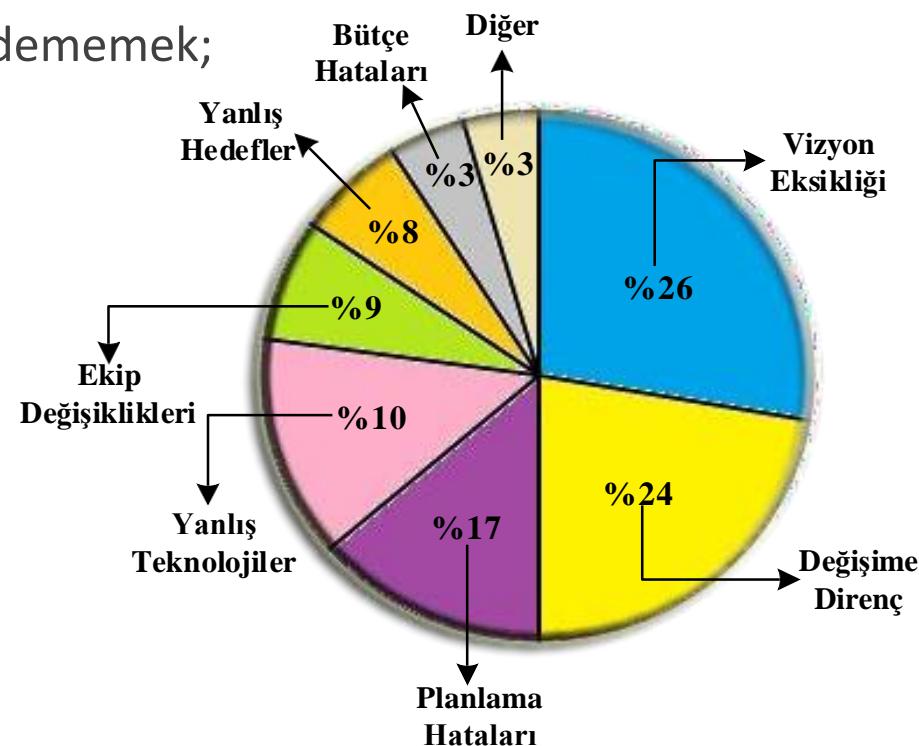
Örneğin, ülkemizde geliştirilen projelerin başarıya ulaşmasına katkı sağlamak amacıyla, Araştırma Destek Programları Başkanlığı (ARDEB) tarafından desteklenen projelerin çıktı, sonuç ve etkilerini nicelik ve nitelik olarak artırmak amacıyla yüksek başarı ile sonuçlanan projelerin yürütücü ve araştırmacılarını ödüllendirmek için TÜBİTAK tarafından belirlenen ölçütler ve değerlendirme yöntemine göre hesaplanarak, proje ekibine (yürüttü ve araştırmacılara) TÜBİTAK Proje Performans Ödülü (PPÖ), denilen bir teşvik ödülü verilmektedir.

# 1. Günümüzde Yazılım Projelerinin Durumu



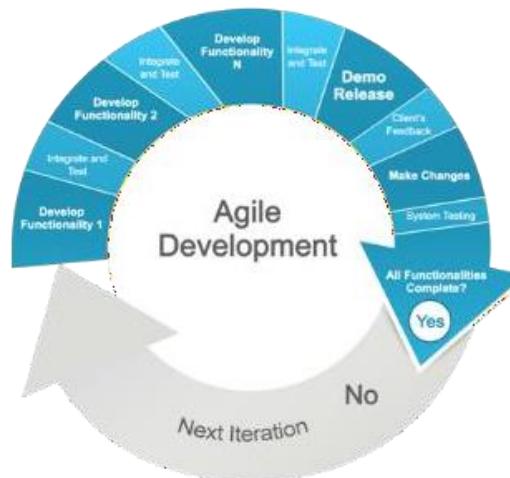
## ➤ Başarısızlığın ana sebepleri:

- Müşterinin isteklerini doğru analiz edememek;
- Proje için uygun ekibi kuramamak;
- Yanlış teknoloji ve mimari seçimleri;
- Geleneksel yöntemlerin eksiklikleri;
- Müşteriyle iletişimden kaçınmak vs.



# 1. Günümüzde Yazılım Projelerinin Durumu

## Peki ne yapmalıyız?



# 2.Çevik Yazılım Yöntemi



# 2.Çevik Yazılım Yöntemi



Hızlı, devamlı ve kullanışlı yazılım üreterek müşteri memnuniyeti sağlamayı amaçlar.

Geliştiriciler ile iş adamları arasında günlük ve yakın işbirliği bulunmalıdır.

Çalışan yazılım gelişiminin en önemli ölçüsüdür.

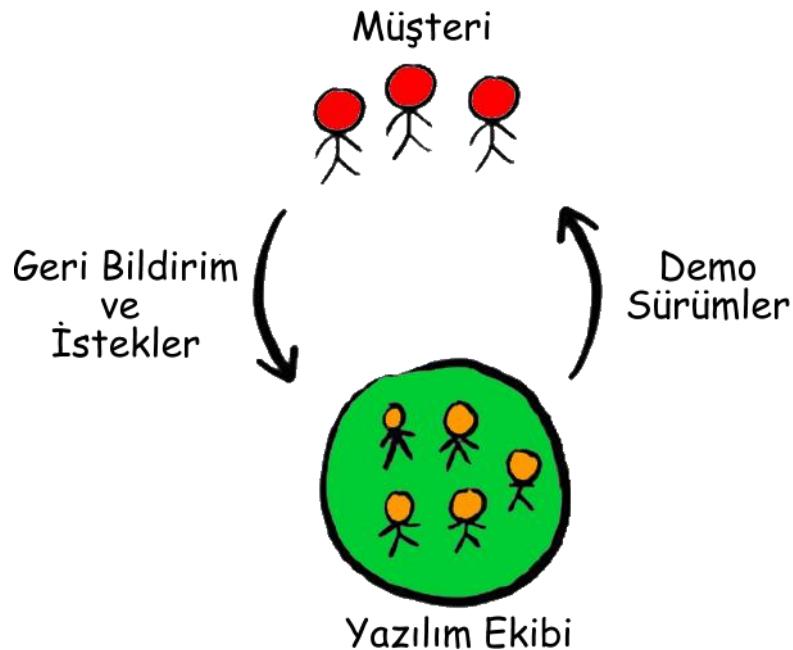
Taleplerdeki geç değişikliklerin de memnuniyetle karşılanması.

Yüz yüze görüşme iletişimini en güzel yoludur.

Kendi kendini organize eden takım yapısı gereklidir.

Basitlik önemlidir.

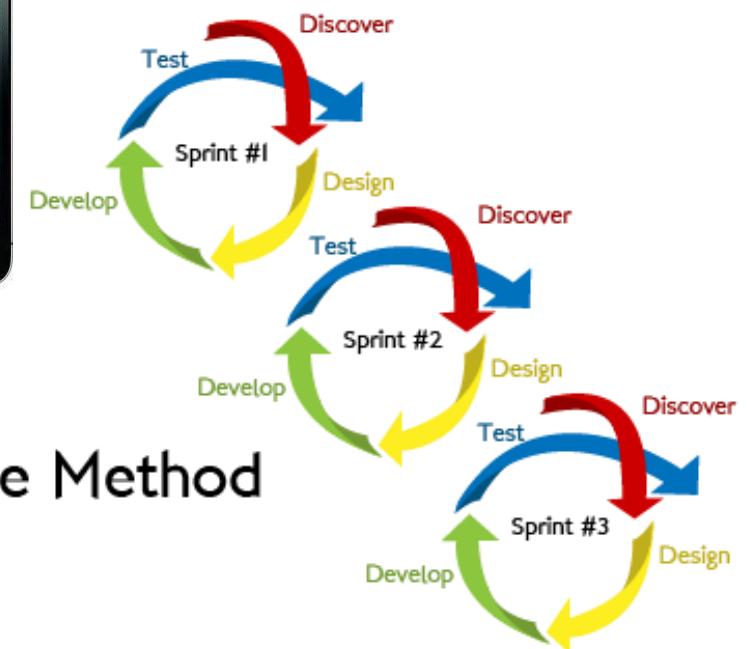
## 2.Çevik Yazılım Yöntemi



➤ Çevik yazılım metodu, kısa vadeli planlar ve küçük parçalar halinde yazılımın geliştirilmesini ön görür. Yazılımın geliştirilmesindeki geri dönüş (**feedback**) ve değişikliklere uyum sağlamak son derece önemlidir. Her yapılan yineleme yazılımı hedeflenen adıma bir adım daha yakınlaştırır. İstenilen sonuca ulaşmak adına birden çok yineleme gereklidir.

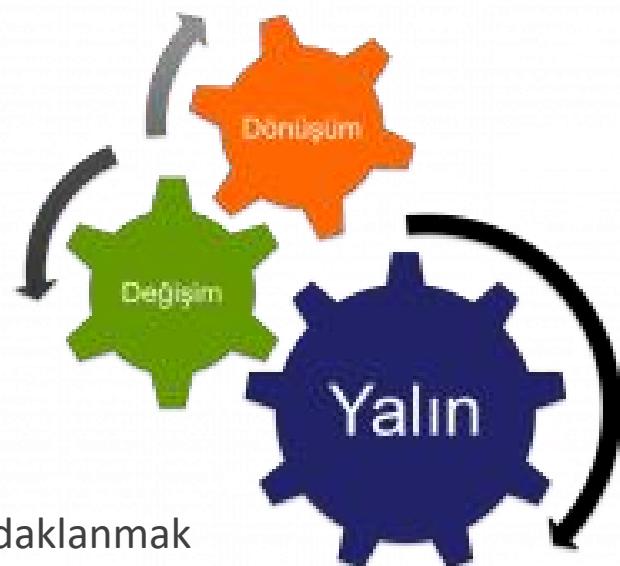
# 2.Çevik Yazılım Yöntemi

➤ Örnek:

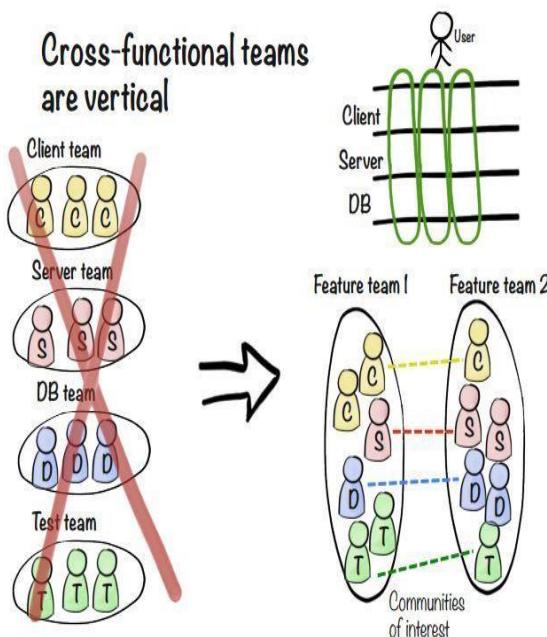


## 2.Çevik Yazılım Yöntemi

- Temel prensipler:
- Müşteriyi memnun etmek
- Değişen ihtiyaçları karşılamak
- Sık aralıklarla ürün teslimi yapmak
- Yüz yüze iletişime önem vermek
- Sürdürülebilir gelişmeyi desteklemek
- Teknik mükemmeliyete, iyi dizayna ve sadeliğe odaklanmak
- **Kendi kendine organize olan takımlar kurmak ?**



## 2.1.Çevik Model Takımları



- Biraraya gelmiş,
- Kendi kendilerine organize olan,
- Çapraz fonksiyonlu,
- İşine odaklanmış,
- Hedefleri net olan,
- Teslim edilebilecek düzeyde ürün ortaya koyabilen
- Küçük(3-7 kişilik) gruplar.

## 2.1.Çevik Model Takımları

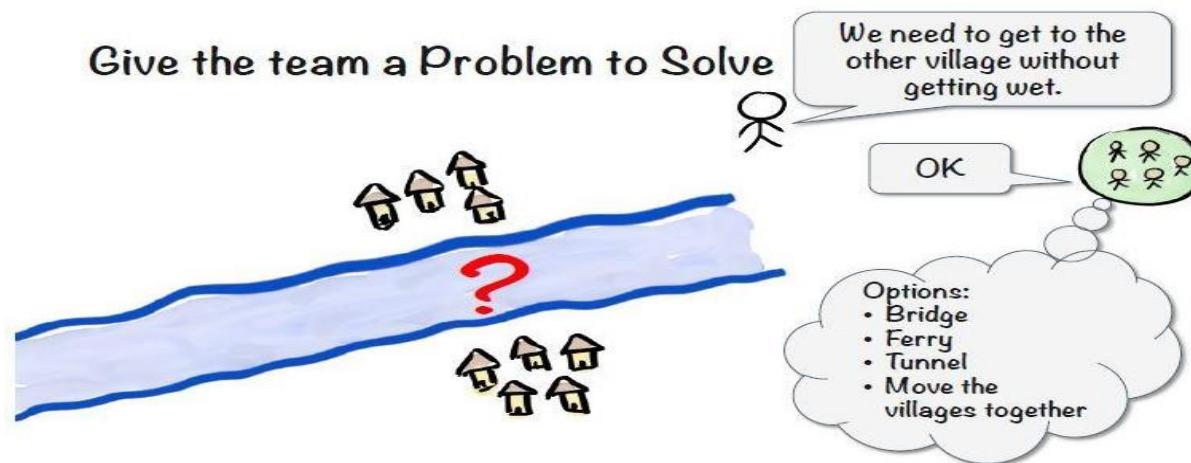
Don't give the team a Solution to Build



Takımlara çözümü söylemeyin!

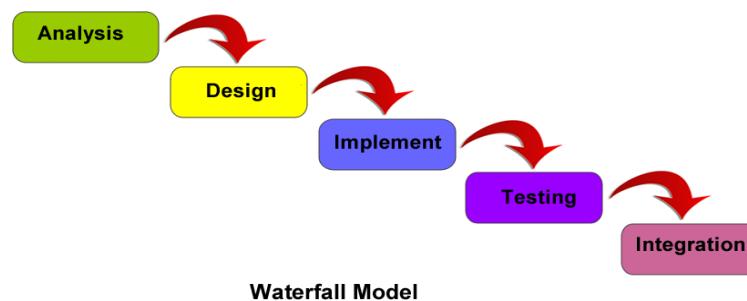
## 2.1.Çevik Model Takımları

Sorunu söyleyin, onlar çözümü üretsin!



### 3. Geleneksel Model vs. Agile

➤ **Çağlayan modeli** 2008 yılında dahi geçerliliğini koruyan bir modeldir ve çevik modellemeden farklılık gösterir. Bu model yazılım projesini baştan sona planlar. Gelişim, sunulabilir işler açısından ölçülür: talep açıklamaları, tasarım dokümanları, test planları, kod incelemeleri vb. Bu durum belli aralıklara bölünmeye uygun değildir ve ilerideki değişikliklere uyum gösterilemez.



# 3. Geleneksel Model vs. Agile



- Geleneksel Yöntemler
- Müşteriler ne istediğini iyi bilir.
- Geliştiriciler neyi, ne şekilde üreteceklerini iyi bilir.
- Bu yol boyunca hiç birşey değişmeyecektir.

# 3. Geleneksel Model vs. Agile

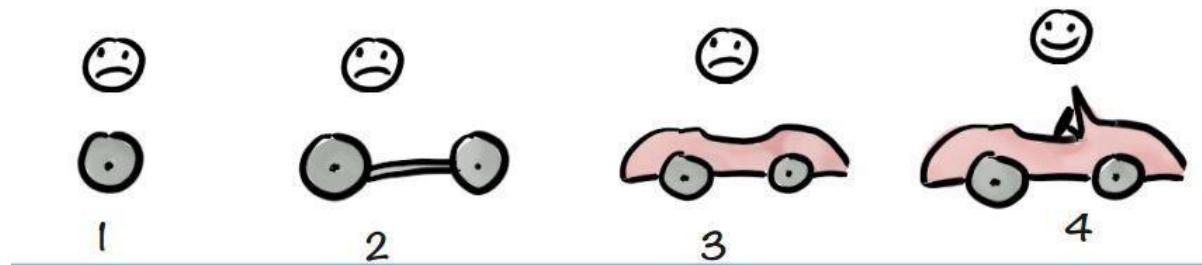
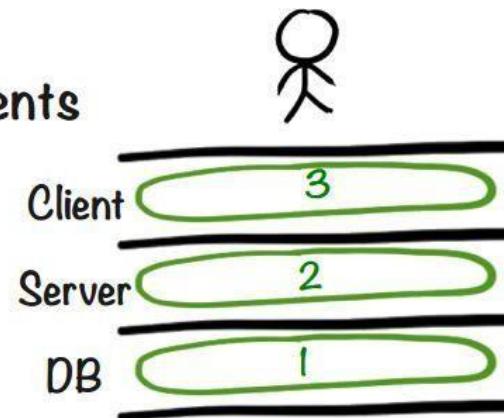
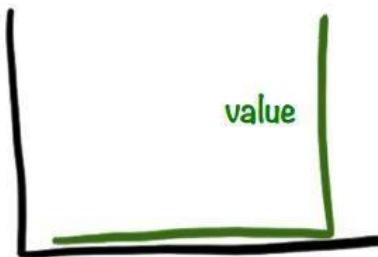


- Çevik Yöntemler
- Müşteriler ne istediğini keşfeder.
- Geliştiriciler neyi nasıl üreteceğini keşfeder.
- Bu yol boyunca bir çok değişiklik yapılabilir.

# 3. Geleneksel Model vs. Agile

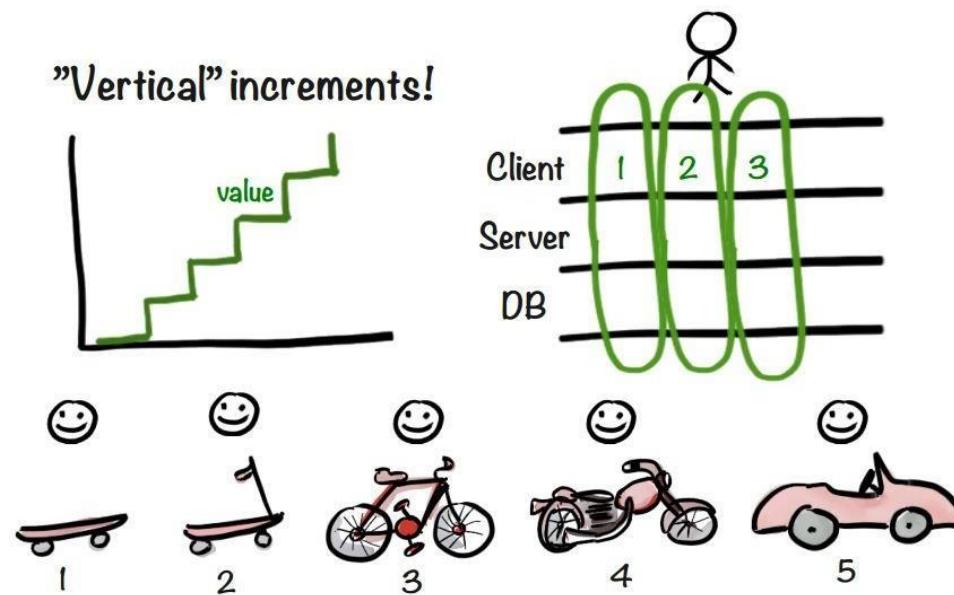
Geleneksel Yöntemler

Not "horizontal" increments

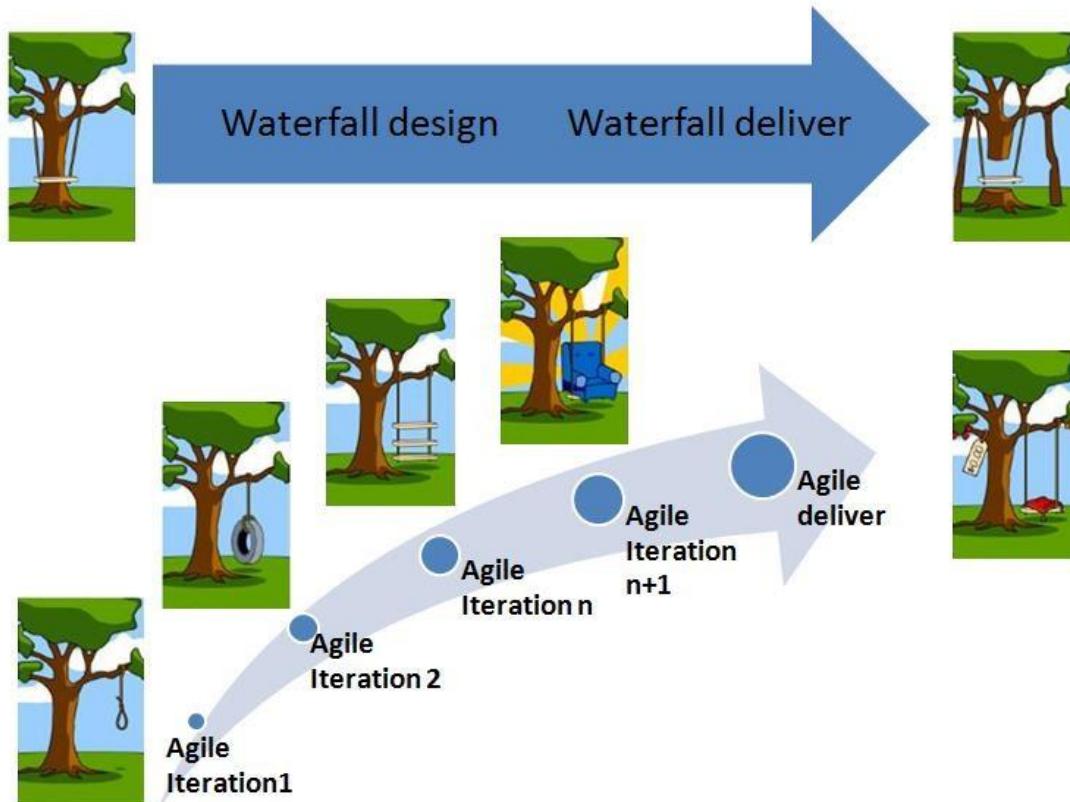


# 3. Geleneksel Model vs. Agile

## Çevik Yöntemler

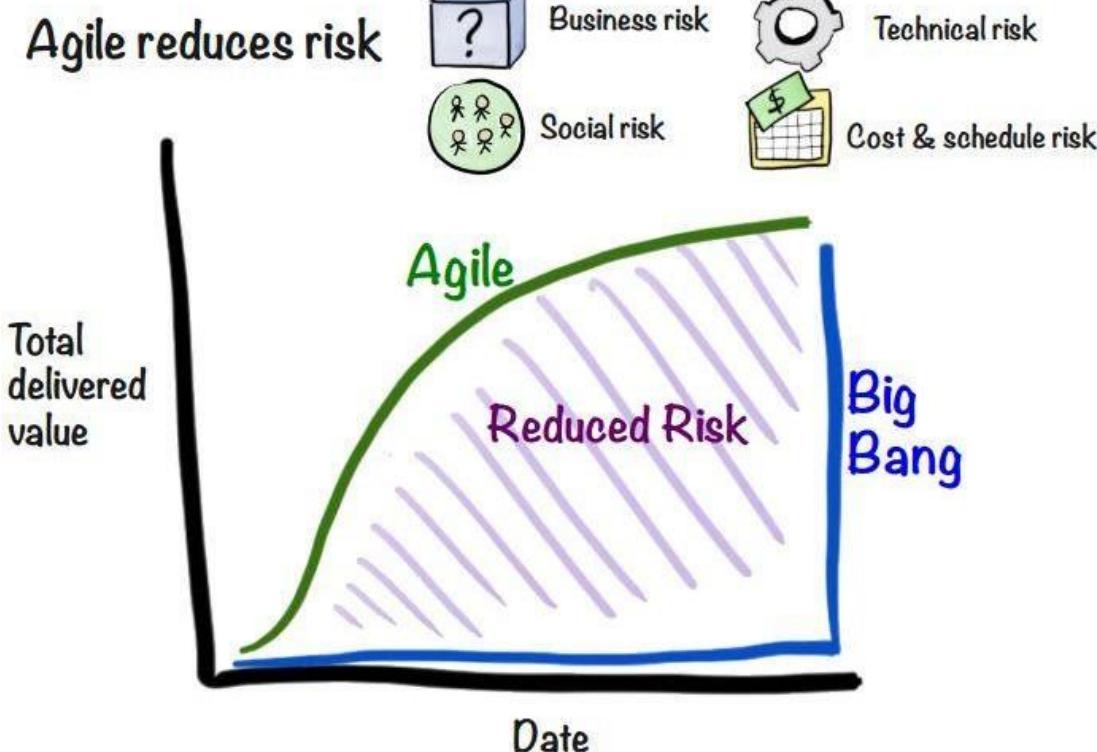


### 3. Geleneksel Model vs. Agile

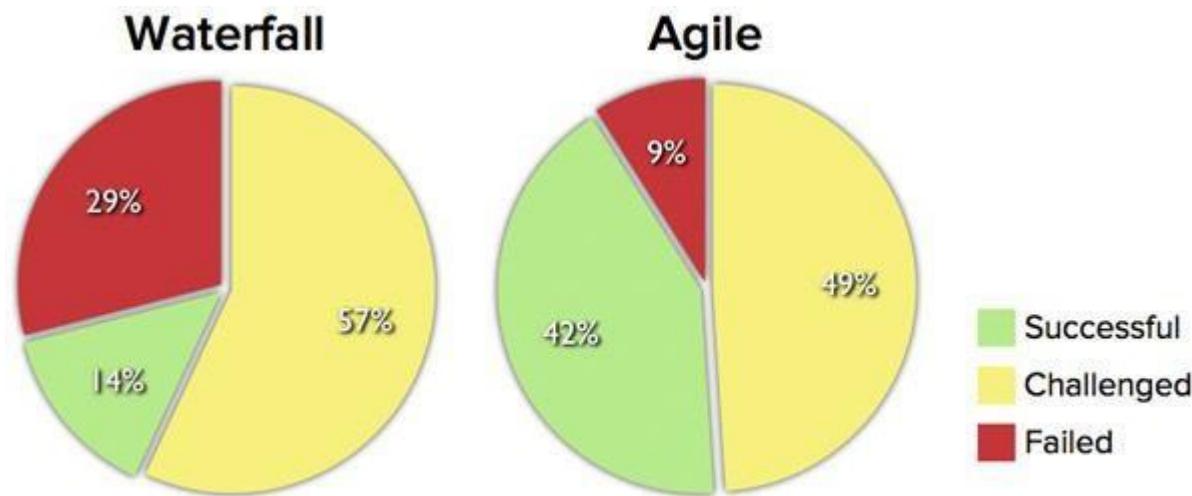


### 3. Geleneksel Model vs. Agile

Çevik modeller riski azaltır!



### 3. Geleneksel Model vs. Agile



Source: The CHAOS Manifesto, The Standish Group, 2012.

# 3. Geleneksel Model vs. Agile

Ölçüm	Çevik Modelleme	Çağlayan Modeli
Planlama ölçeği	Kısa dönemlik	Uzun dönemlik
Müşteri ile geliştirici arasındaki mesafe	Kısa	Uzun
Özelleştirme ve uygulama arasındaki zaman	Kısa	Uzun
Sorunları keşfetmek için zaman	Kısa	Uzun
Proje tamamlanma riski	Düşük	Yüksek
Değişikliklere uyum yeteneği	Yüksek	Düşük

# 4. Değerlendirme

Büyük Projeler



Büyük Özellikler



Büyük Takımlar



Büyük Dönüşümler



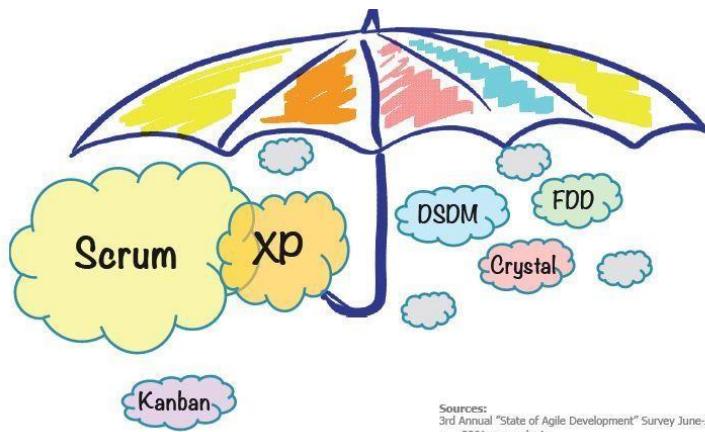
## Sonuç

➤ Büyüklük kötüdür, onu parçalara ayırin. Böylece daha başarılı projeler geliştirebilirsiniz.

# 4. Değerlendirme

- 3 somut değişiklik yapın!
- 1. Gerçek takımlar oluşturun
  - Küçük, çapraz fonksiyonlu, kendi kendine organize olabilen
- 2. Sık sık teslimat yapın
  - Normal olarak ortalama her 3 haftanın sonunda
  - Ek olarak projenin tüm çeyreklerinin sonunda
- 3. Gerçek kullanıcıları dahil edin.
  - Takım ve kullanıcılar arasında doğrudan ve hızlı geri dönüşler

# 5. Çevik Yazılım Şemsiyesi



**FDD: Feature-Driven Development**

**RUP : Rational Unified Process**

**DSDM : Dynamic System Development Method**

# FDD TANIMI

➤ FDD, Avustralyalı Jeff De Luca tarafından geliştirilmiş ve Singapur projesinde beraber çalıştıkları Peter Coad tarafından modifiye edilmiştir. Singapur projesi 50 kişi ile 15 ayda tamamlanan bir proje olmuştur. Daha sonra FDD'nin uygulandığı başka bir proje ise 250 kişi ile 18 ayda tamamlanmıştır. FDD değişik boyutlara büyüyebilin, tekrarlanabilir bir süreçtir. Aşağıdaki noktalara odaklanır.

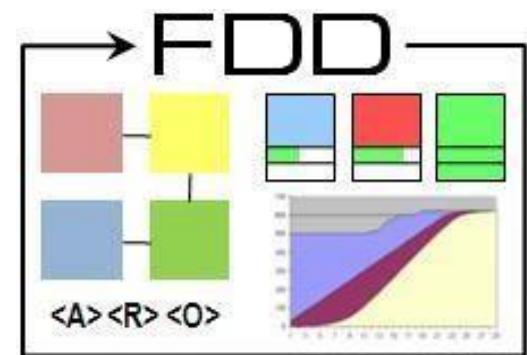
1. Sistemi hazırlamak için gereken sistem büyüyebilir olmalıdır. Büyük projeler için de kullanılabilir olmalıdır.

2. Basit iyi tanımlanmış sistem iyi çalışır.

3. Süreç adımları basit olmalıdır.

4. İyi süreç arka plana saklanır ve insanlar sonuçlara odaklanabilir.

5. Kısa, iteratif, özellik yaklaşımı yaşam döngüleri en iyi sonucu verir

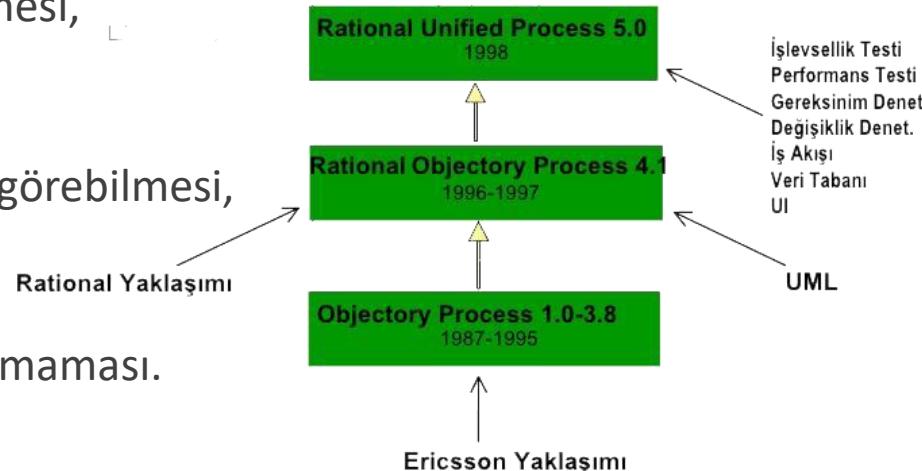


# RUP ("Rational Unified Process")

- 2003 yılından beri IBM'in bir bölümü tarafından oluşturulan bir iteratif yazılım geliştirme süreci çerçevesidir. Başarısız bir yazılımdaki sorunların aşılıp başarılı yazılım oluşturmak için gerekli adımları saptayarak oluşturulmuş bir süreçtir.
- Başarısız bir yazılımdaki özelliklerini yazımızın devamında okuyabilirsiniz. RUP şirketlere yazılım geliştirme aşamasında bir yön sağlar. RUP use-case ve nesne teknolojileri tabanlı; tekrarlanan (iterative) yazılım geliştirme ve ış modelleme yöntemidir. RUP'un verebileceği özellikler şunlardır;

## RUP'un Gelişimi

1. Müşteriyi ve yazılımcıyı organize edebilmesi,
2. Standart tanımlı adımları olması,
3. Oluşacak yazılımdaki sık değişiklikleri öngörebilmesi,
4. Basit olması,
5. Proje yönetim aktivitelerinin çok fazla olmaması.



# Uç Programlama (Extreme Programming XP) Nedir?

---

➤ **Uç Programlama (XP)**, yazılım geliştirme süreci boyunca son derece kaliteli olmak koşuluyla çalıştırılabilir kod üretmeye odaklanmış bir yazılım geliştirme metodolojisidir. Yazılım geliştirme sürecinin en temel, en önemli ve final çıktısı ya da ürünü çalıştırılabilir kod olduğundan, XP metodolojisi sürecin en başından itibaren çalıştırılabilir kodu sürecin merkezinde tutmaktadır. İşte bu yüzden bu metodolojinin adı XP'dir.

**Uç Programlama'nın özünde aşağıdaki uygulamalar yer alır:**

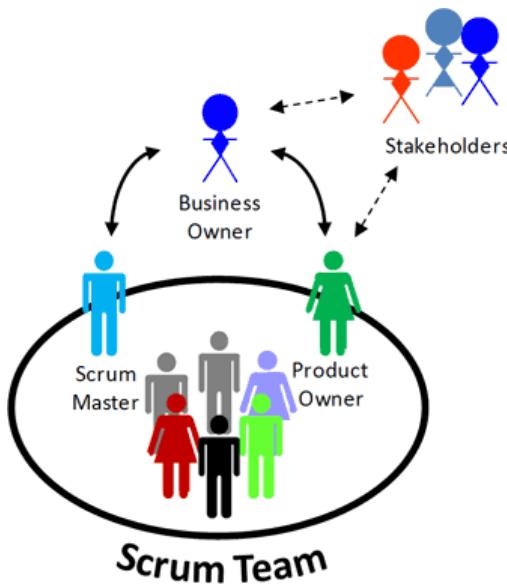
- Planlama
- Sık ve küçük sürümler
- Basit tasarım:
- Önce test
- Refactor etme
- Haftada 40 saat çalışma
- Müşteriyle yakın iletişim
- Kodlama standartları

# 6. Scrum Modeli

---



# 6. Scrum Modeli



➤ **Scrum Takımı:** Ürün Sahibi, Geliştirme Ekibi ve Scrum Master'dan oluşur. Takım kendi kendini örgütler. Böylece kendi içerisinde uyum içinde olan takımlar daha başarılı sonuçlar alırlar. Scrum takım modeli esneklik, yaratıcılık ve verimliliği optimize etmek için tasarlanmıştır.

# 6. Scrum Modeli

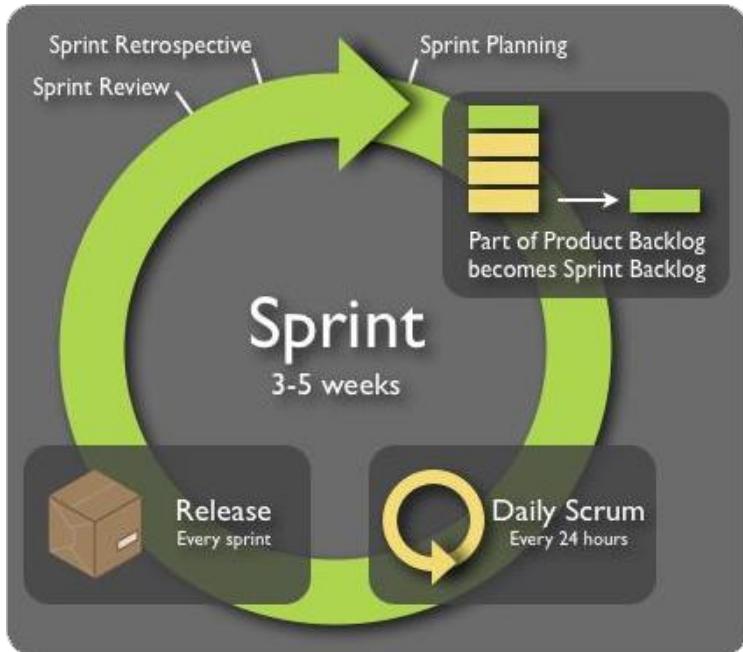


## Backlog:

- Müşteriden ve son kullanıcıdan gelen gereksinimleri içerir.
- "Ne yapacağız" sorusunun yanıtını içerir.
- Herkese açık ve herkes tarafından müdahale edilebilir.
- Risk, iş değeri, zaman gibi kavrlamlara göre ürün sahibi tarafından sıralandırılır.
- User Story'lerden oluşur.

# 6. Scrum Modeli

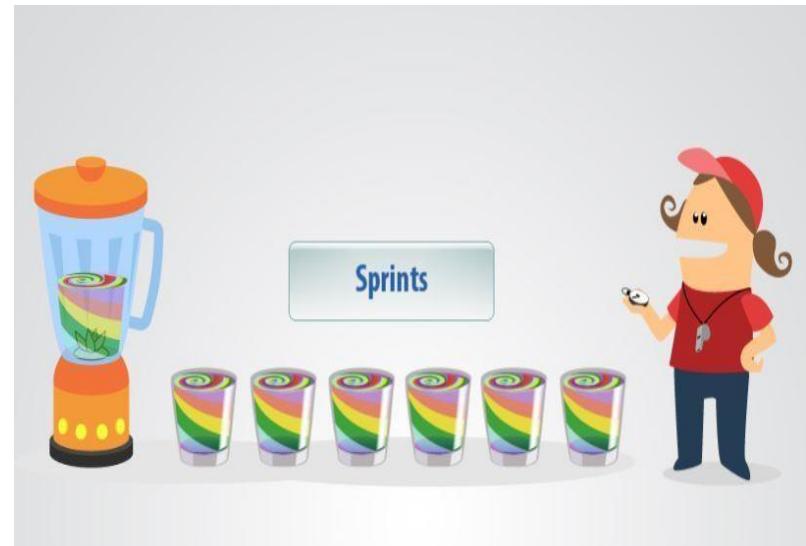
---



## ➤ Sprint

- Belirli bir süreye sahiptir.
- Sonunda ortada değeri olan bir çıktı olmalıdır.
- Toplantılarla içerik belirlenir.
- Sprint süresi boyunca her gün toplantılar yapılır.

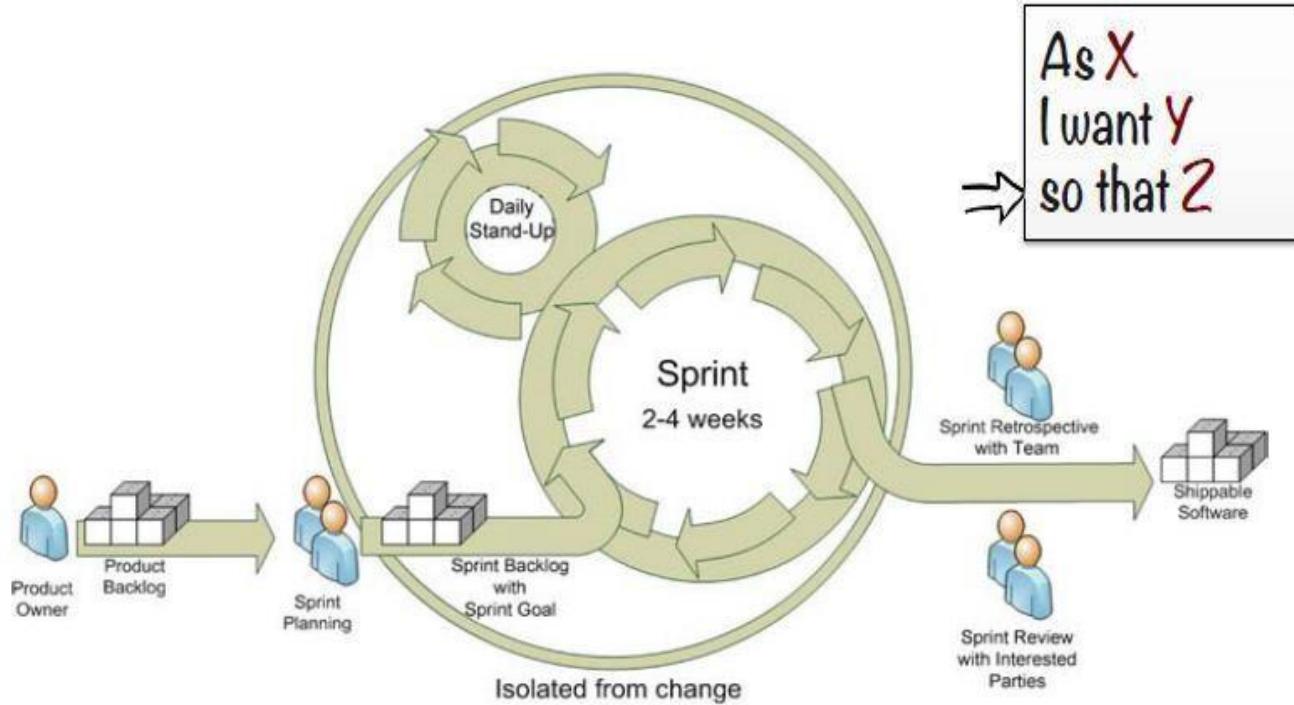
# 6. Scrum Modeli



Sprint Gösterimi

# 6. Scrum Modeli

➤ **User Story:** Müşteri, son kullanıcı veya ürün sahibi için değerli olan ve anlam ifade eden genellikle fonksiyonel özelliklerin belirtildiği ifadelerdir.



# 6. Scrum Modeli

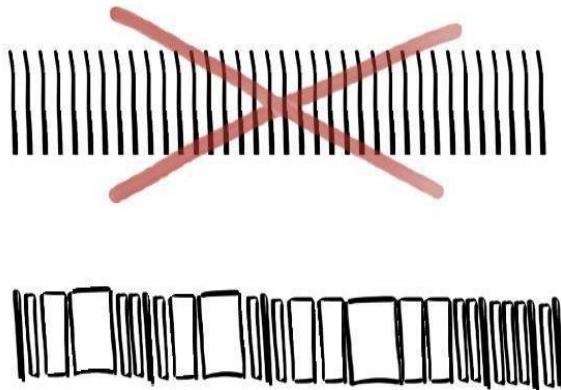
➤ **Örnek User Story:** Online alışveriş yapan biri **olarak**, alışverişe daha sonra devam edebileyim **diye**, alışveriş kartımın kaydedilmesini **istiyorum**.

As online buyer  
I want to save my shopping cart  
so that I can continue shopping later

# 6. Scrum Modeli

---

Fact: Features have different sizes

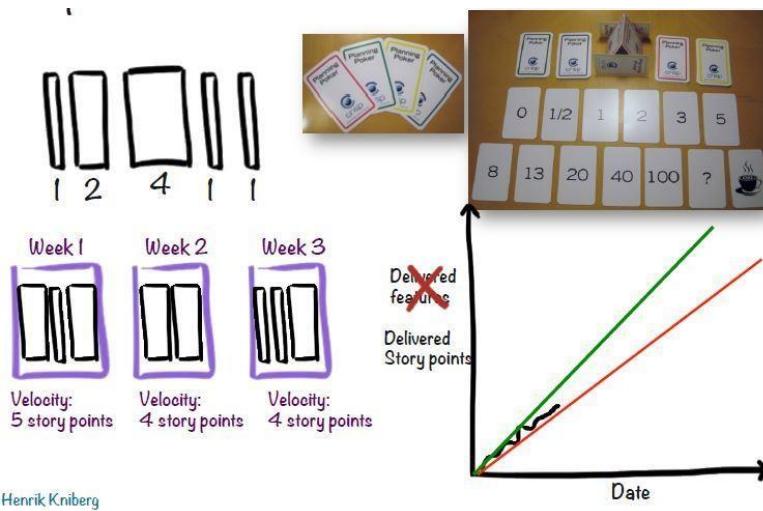


➤ Her bir user story farklı bir boyuttadır. Somut olarak bakarsak, bir projedeki her bir gereksinim için gereken iş gücü ve zaman aynı değildir. Bu sebeple ürün backlogları sprintlere bölünürken, user storylerin boyut ve öncelikleri göz önünde bulundurulur. Örneğin bir sprint 3 user story içerirken diğerleri daha küçük boyutlarda 5 user story içerebilir. **Peki boyutları nasıl belirleyeceğiz?**

İş göründüğü gibi değil!

# 6. Scrum Modeli

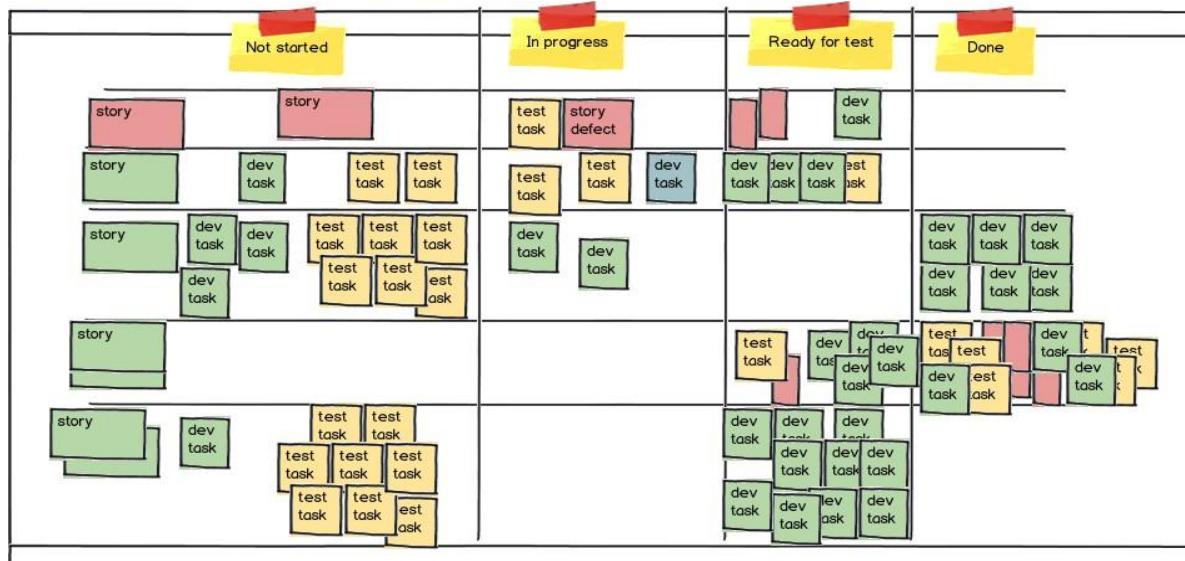
---



➤ **Poker Kartları:** Scrum takım üyeleri bir araya gelir. Scrum master bir user story okur. Takımdaki her bir üye user story için uygun gördüğü poker kartlarından birini seçer. Herkes kartları seçikten sonra tüm kartlar açılır ve değerlendirilir. Böylece herkesin ortak görüşü sonunda user story'lerin büyüklüğü belirlenir.

# 6. Scrum Modeli

---



# 6. Scrum Modeli

---

## SCRUM

- Sprint (2 hafta-1 ay)
- Sprintler en son halini aldıktan, toplantı yapıldıktan sonra değişmez.
- Özellikler geliştiriciler tarafından derecelendirilir.
- Herhangi bir mühendislik pratiği tanımlamaz.

## XP (EXTREME PROGRAMMING)

- Sprint (1 yada 2 hafta)
- Sprintler değişebilir.
- Özellikler ürün sahibi tarafından derecelendirilir.
- Mühendislik pratikleri tanımlar. Eşli programlama, otomatik test, basit dizayn vs.

# Early delivery of business value

Less bureaucracy

## The Agile: Scrum Framework at a glance

Inputs from Executives,  
Team, Stakeholders,  
Customers, Users



Product Owner



The Team



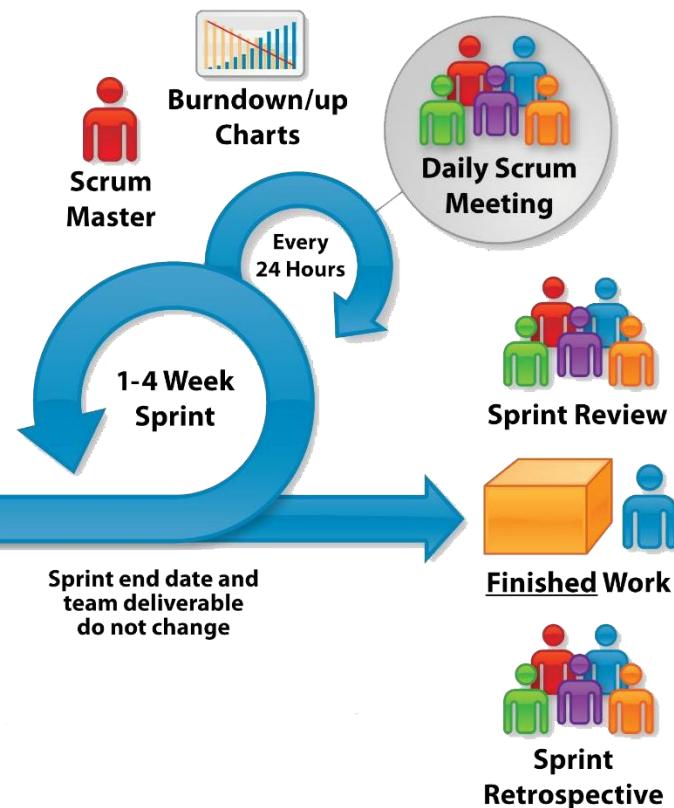
Product  
Backlog

Team selects starting at top as much as it can commit to deliver by end of Sprint

Sprint  
Planning  
Meeting



Sprint  
Backlog



# Çalışma Soruları

---

1. Agile modelinin geliştirilmesine neden ihtiyaç duyulmuştur? Açıklayınız.
2. Agile takımları kimlerden oluşur ve iş paylaşımıları nasıldır?
3. Agile modelinin diğer modellerden farkları nelerdir? Üzerinde durduğu temel noktalar nelerdir?
4. User story, backlog ve sprint kavramlarını açıklayınız.
5. Agile ile geliştirilmiş büyük yazılım projelerine örnekler veriniz.
6. Scrum haricindeki diğer çevik yazılım geliştirme yöntemlerini açıklayınız.

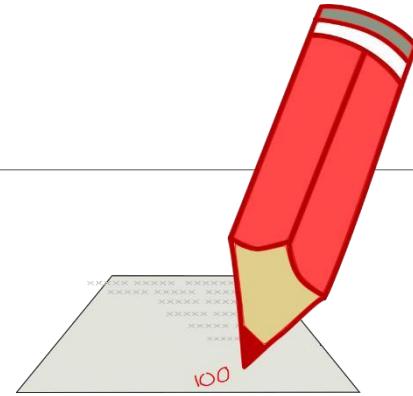
# Kaynaklar

---

- 1 Martin, Micah, and Robert C. Martin. *Agile principles, patterns, and practices in C#*. Pearson Education, 2006.
- 2 Kniberg, Henrik. "Scrum and XP from the Trenches." *Lulu. com* (2007).
- 3 Kniberg, Henrik. "What is agile", 2013
- 4 [http://en.wikipedia.org/wiki/Agile\\_software\\_development](http://en.wikipedia.org/wiki/Agile_software_development)
- 5 <http://www.mshowto.org/microsoft-visual-studio-team-foundation-server-nedir.html>
- 6 [www.kurumsaljava.com/download/10/](http://www.kurumsaljava.com/download/10/)
- 7 <https://www.tubitak.gov.tr/tr/destekler/akademik/uygulamalar-ve-yonergeler/icerik-proje-performans-odulu-ppo-uygulamasi>
- 8 [http://images.slideplayer.biz.tr/8/2395426/slides/slide\\_27.jpg](http://images.slideplayer.biz.tr/8/2395426/slides/slide_27.jpg)
- 9 <http://www.bayramucuncu.com/wp-content/uploads/2013/04/Ads%C4%B1z.png>
- 10 <http://antasya.com/Images/is-alanlarimiz/yazilim-sistemleri/agile-method.png>
- 11 <http://volkansel.com/wp-content/uploads/2014/07/agile-scrum.jpg>

# Ödev

---



# Sorularınız

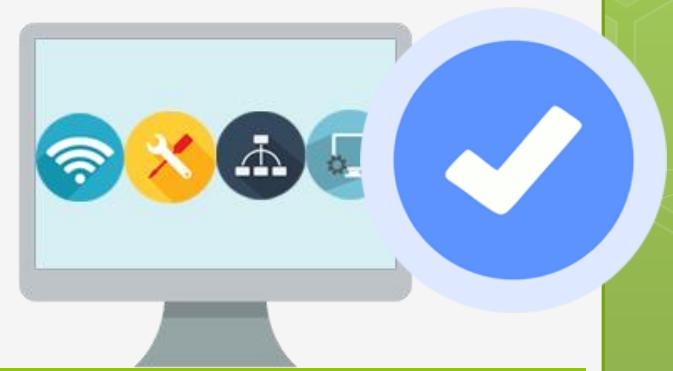
---



# YAZILIM MÜHENDİSLİĞİNİN TEMELLERİ

## 8.Hafta

### Yazılım Doğrulama ve Geçerleme



# Bölüm Hedefi

- Yazılım üretimi boyunca, "Doğru Yazılımı mı üretiyoruz?" ve "Yazılımı doğru olarak üretiyor muyuz?" sorularının yanıtlarını araştıran doğrulama ve geçerleme yöntemleri bu bölümde açıklanmaktadır.



# Giriş

- Yazılım belirtimlerinin ve proje yaşam sürecindeki her bir etkinlik sonunda alınan çıktıların, tamam, doğru, açık ve önceki belirtimleri tutarlı olarak betimler durumda olduğunu doğrulanması.
- Proje süresince her bir etkinlik ürününün teknik yeterliliğinin değerlendirilmesi ve uygun çözüm elde edilene kadar aktivitenin tekrarına sebep olması.

# Giriş

- Projenin bir aşaması süresince geliştirilen anahtar belirtimlerin önceki belirtimlerle karşılaştırılması.
- Yazılım ürünlerinin tüm uygulanabilir gerekleri sağladığının gerçekleşmesi için sınamaların hazırlanıp yürütülmesi biçiminde özetlenebilir.

# Doğrulama / Geçerleme

## Doğrulama

Doğu ürünü mü üretiyoruz?

Ürünü kullanacak kişilerin isteklerinin karşılanıp karşılanmadığına dair etkinliklerden oluşur.

## Geçerleme

Ürünü doğru olarak mı üretiyoruz?

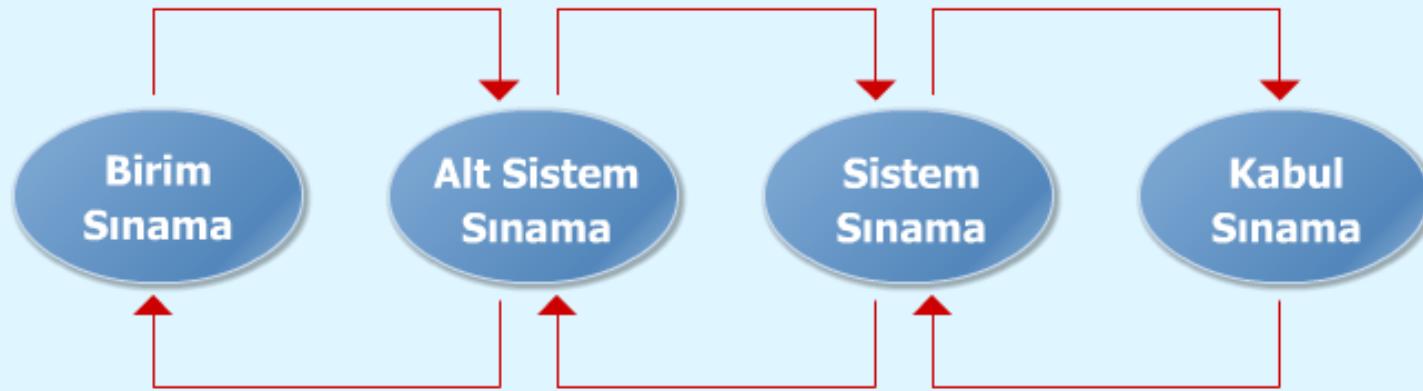
Ürünün içsel niteliğine ilişkin izleme ve denetim etkinliklerinden oluşur.

# Sınamalar Kavramları

- Sınamalar ve bütünlendirme işlemlerinin bir strateji içinde gerçekleştirilmesi, planlanması ve tekniklerin seçimi gerekmektedir.
- Bütünlendirme işleminde, en küçük birimlerden başlanarak sistem düzeyine çıkmaktadır. Bu değişik düzeylere hitap edecek sınamalar yöntemleri olmalıdır.



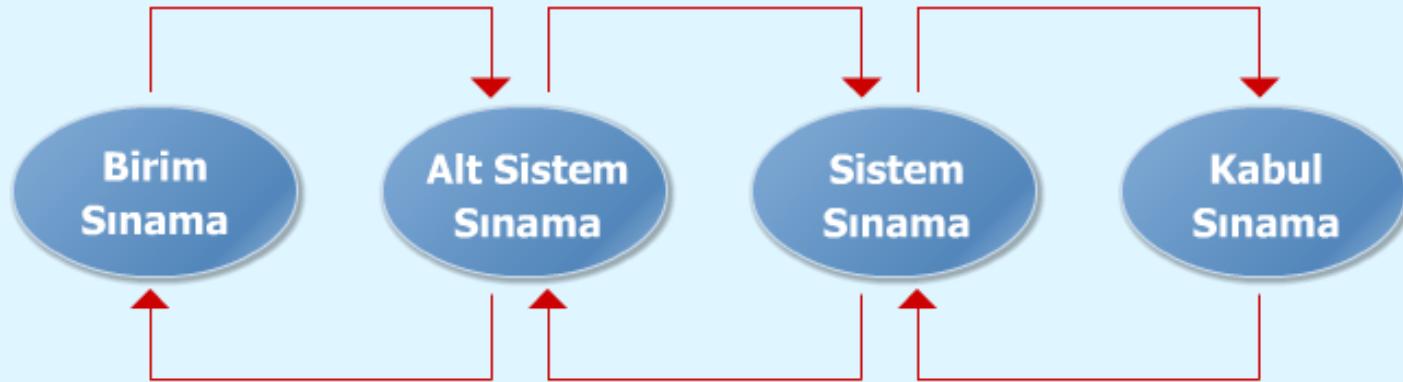
# Sınamalar Kavramları



## Birim Sınaması

Bağılı oldukları diğer sistem unsurlarından bütünüyle soyutlanmış olarak birimlerin, doğru çalışmalarının belirlenmesi amacıyla yapılır. Birimler, ilişkili yapıtaşlarının bütünlüğünün oluşturulmasına katkıda bulunur.

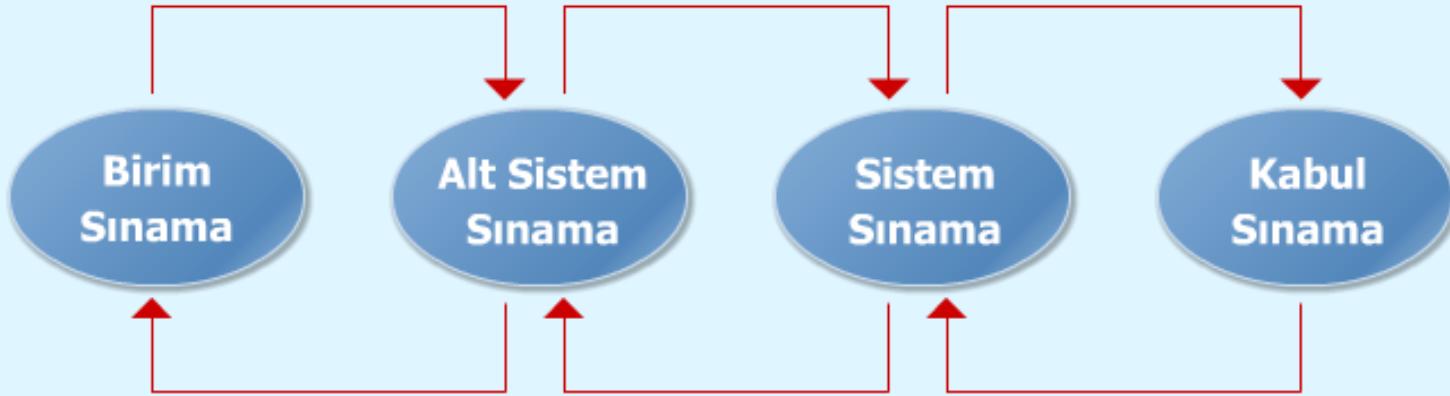
# Sınamalar Kavramları



## Alt-Sistem Sınaması

Alt sistemler ise, modüllerin bütünlüğü ile ortaya çıkar. Yine bağımsız olarak sınamaları yapılmalıdır. Bu düzeyde en çok hata arayüzlerde bulunmaktadır, arayüz hatalarına yönelik sınamalara yoğunlaşılmalıdır.

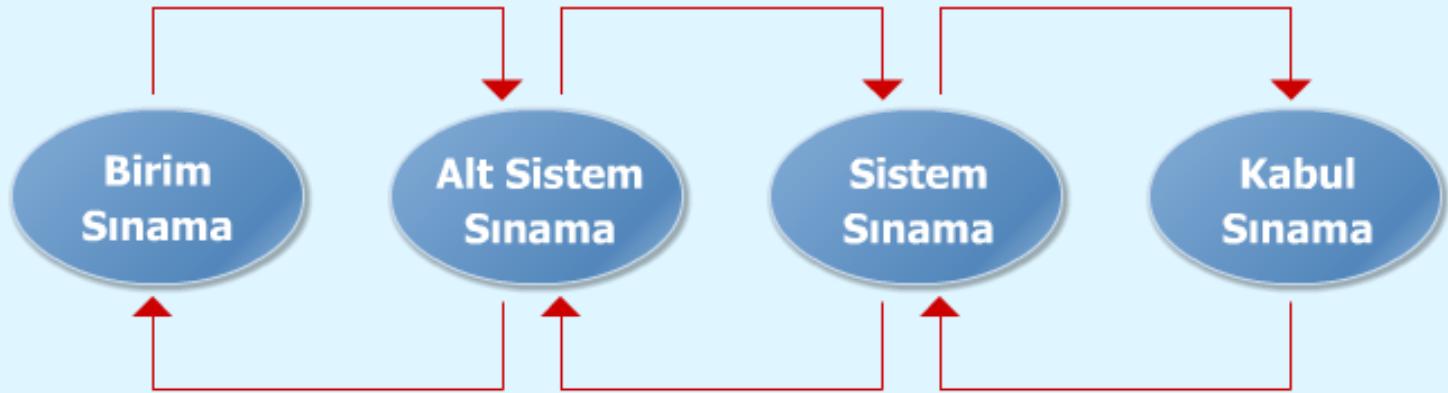
# Sınamalar Kavramları



## Sistem Sınaması

Üst düzeyde bileşenlerin sistem ile olan etkileşimlerinde çıkacak hatalar aranmaktadır. Ayrıca belirtilen ihtiyaçların doğru yorumlandıkları da sınanmalıdır.

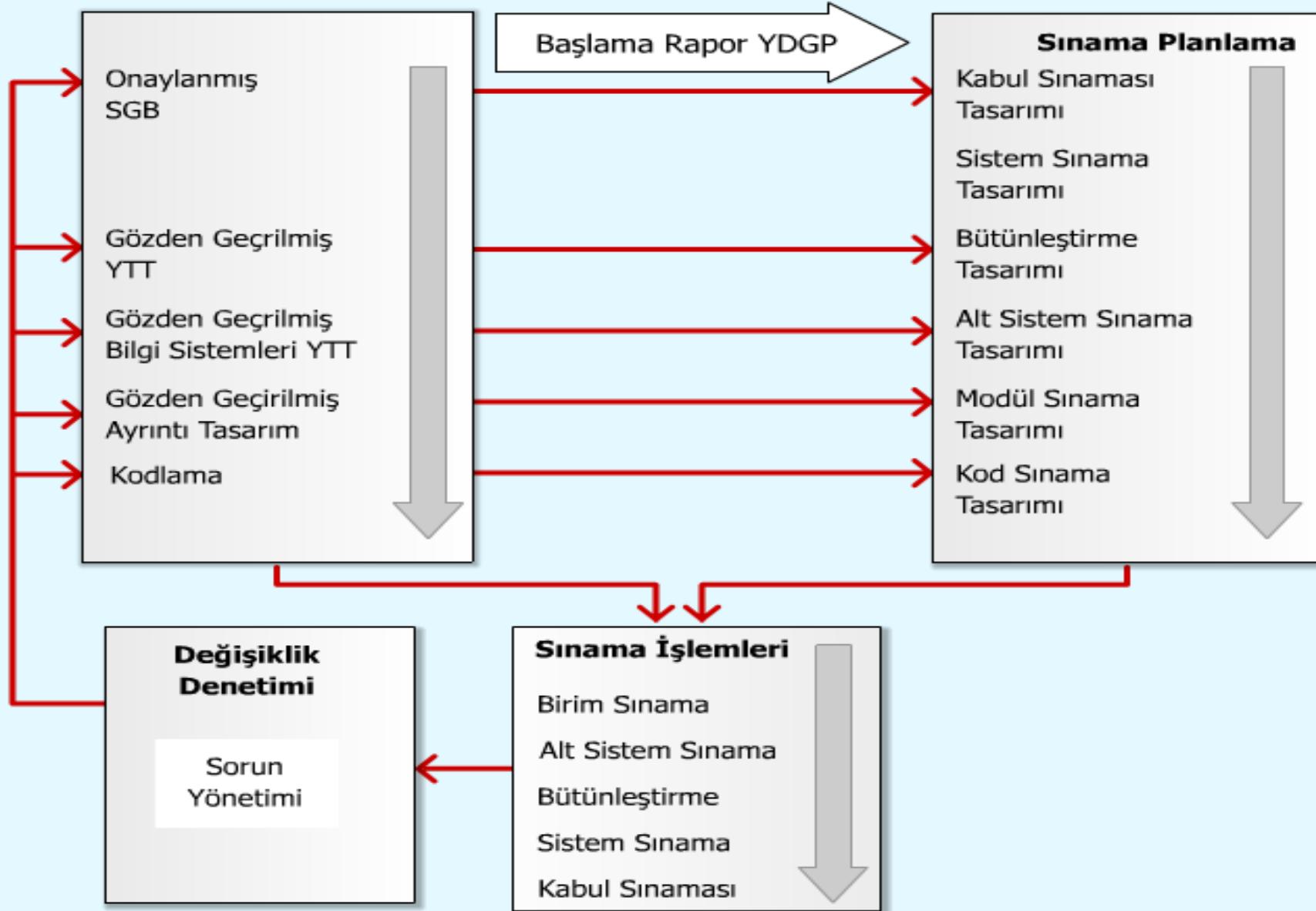
# Sınamalar Kavramları



## Kabul Sınaması

Çalıştırılmadan önce sistemin son sınamasıdır. Artık yapay veri yerine gerçek veriler kullanılır. Bu sınama türü alfa sınaması ve beta sınaması olarak da bilinir. Alfa sınamada, tanımında, sınamanın geliştirici organizasyonun yerleşkesinde, kullanıcıların da gelerek katkıda bulunması içerilir. Daha sonra ürünün pazarlama işlemi sırasında beta sınaması denilen, sınamalar, kullanıcının kendi yerleşkesinde, geliştirici gözetiminde yapılır.

# Doğrulama ve Geçerleme Yaşam Döngüsü



# Sınama Yöntemleri

- Sınama işlemi, geliştirmeyi izleyen bir düzeltme görevi olmak ile sınırlı değildir. Bir "sonra" operasyonu olmaktan çok, geliştirme öncesinde planlanan ve tasarıımı yapılması gereken bir çaba türüdür.
- Her mühendislik ürünü, iki yoldan biri ile sınanır: Sistemin tümüne yönelik işlevlerin doğru yürütüldüğünün (**kara kutu - black box**) veya iç işlemlerin belirtimlere uygun olarak yürütüldüğünün bileşenler tabanında sınanması (**beyaz kutu - white box**).
- Kara kutu sınamasında sisteme, iç yapısı bilinmeksizin gelişigüzel girdiler verilerek sınama yapılır. Sonraki sayfalarda beyaz kutu sınamasına ilişkin bilgiler verilmektedir.

# Beyaz Kutu Sınaması

- Beyaz kutu sınavası tasarlarken, birimin süreç belirtiminden yararlanılır.
- Yapılabilecek denetimler arasında:
  - Bütün bağımsız yolların en azından bir kere sınanması,
  - Bütün mantıksal karar noktalarında iki değişik karar için sınavaların yapılması,
  - Bütün döngülerin sınır değerlerinde sınanması,
  - İç veri yapılarının denenmesi bulunur.

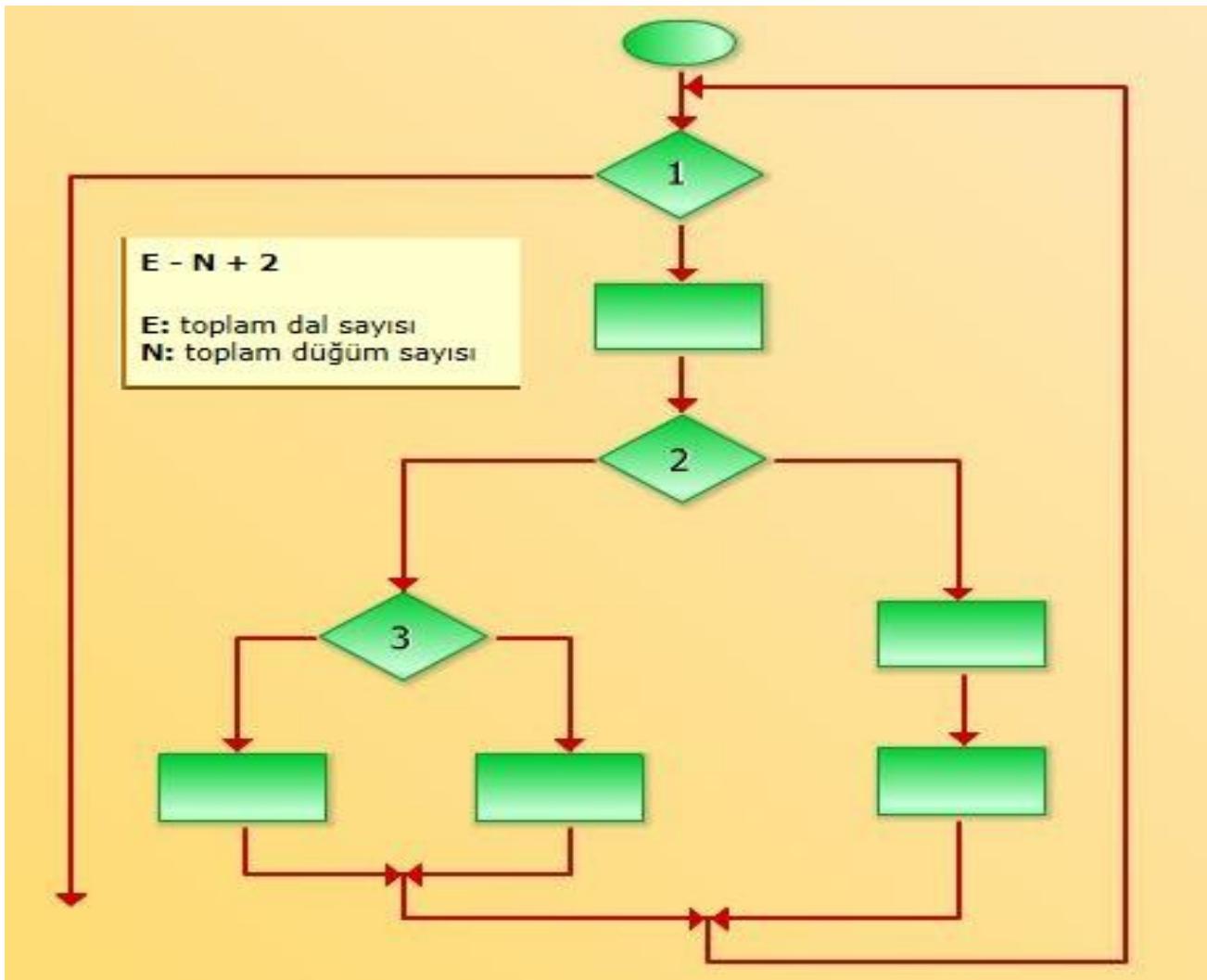
# Beyaz Kutu Sınaması

- Sınamaları yürütürken sınırlı çabamızı yerinde kullanmamız gereklidir. Bunun için hataların bazı özelliklerinin bilinmesinde yarar vardır:
- Bir program kesiminin uygulamada çalıştırılma olasılığı az ise o kesimde hata olması ve bu hatanın önemli olması olasılığı fazladır.
- Çoğu zaman, kullanılma olasılığı çok az olarak kestirilen program yolları, aslında çok sıkça çalıştırılıyor olacaktır.
- Yazım hataları rasgele olarak dağılırlar. Bunlardan bazılarını derleyiciler bulur, bazıları da bulunmadan kalır.

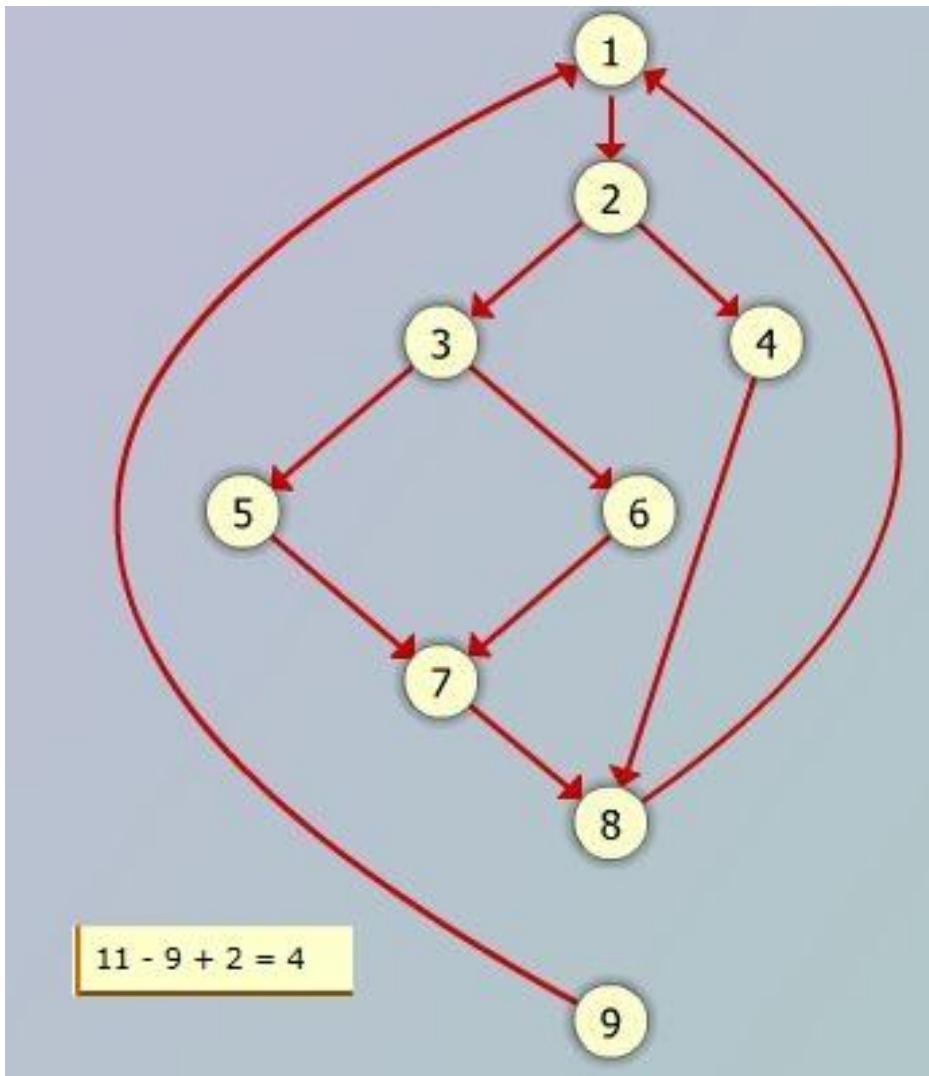
# Temel Yollar Sınaması

- Daha önce çevrimsellik karmaşıklığı konusunda gördüğümüz hesap yöntemi ile bir programdaki bağımsız yollar bulunduktan sonra, bu kadar sayıda sınavma yaparak programın her birimini bir şekilde sınavmalara dahil etmiş oluruz.
- Bağımsız yolların saptanması için önce, program çizgesel bir biçimde çevrilir.
- Bunu yapmak için ise, program iş akış şemaları diyagramları iyi bir başlangıç noktasıdır.

# Temel Yollar Sınaması



# Temel Yollar Sınaması



# SINAMA ve BÜTÜNLEŞTİRME STRATEJİLERİ

- Genellikle sınavma stratejisi, bütünlendirme stratejisi ile birlikte değerlendirilir. Ancak bazı sınavma stratejileri bütünlendirme dışındaki tasaları hedefleyebilir.
- Örneğin, yukarıdan aşağı ve aşağıdan yukarı stratejileri bütünlendirme yöntemine bağımlıdır. Ancak işlem yolu ve gerilim sınavmaları, sistemin olaylar sırasında değişik işlem sıralandırmaları sonucunda ulaşacağı sonuçların doğruluğunu ve normal şartların üstünde zorlandığında dayanıklılık sınırlını ortaya çıkarır.

# Yukarıdan Aşağı Sınama ve Bütünleştirme

- Yukarıdan aşağı bütünləştirməde, önce sistemin en üst düzeylerinin sınanması ve sonra aşağıya doğru olan düzeyleri, ilgili modüllerin takılarak sınamaları söz konusudur.
- En üst noktadaki bileşen, bir birim/modül/alt sistem olarak sıandıktan sonra alt düzeye geçilmelidir.

# Yukarıdan Aşağı Sınama ve Bütünleştirme

- Ancak bu en üstteki bileşenin tam olarak sınanması için alttaki bileşenlerle olan bağlantılarının da çalışması gereklidir.
- Alt bileşenler ise bu stratejiye göre henüz hazırlanmış olamazlar. Bunların yerine üst bileşenin sınavması için kullanılmak üzere 'koçan' programları yazılır.

# Yukarıdan Aşağı Sınama ve Bütünleştirme

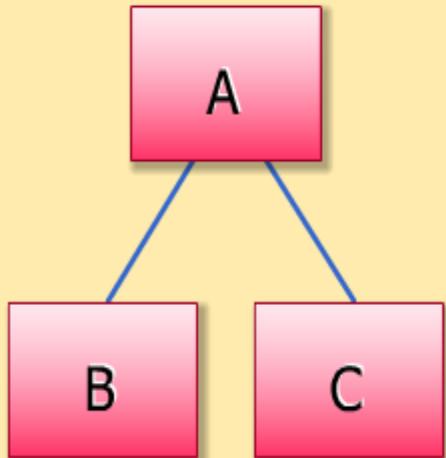
- Koçanlar, bir alt bileşenin, üst bileşen ile arayüzünü temin eden, fakat işlevsel olarak hiç bir şey yapmayan, boş çerçeve programlarıdır.
- Üst bileşenin sınanması bittikten sonra bu koçanlar, içleri doldurularak kendi kodlama ve birim sınavma işlemlerini tamamladıktan sonra üst bileşen ile yeniden sınanırlar.

# Yukarıdan Aşağı Sınama ve Bütünleştirme

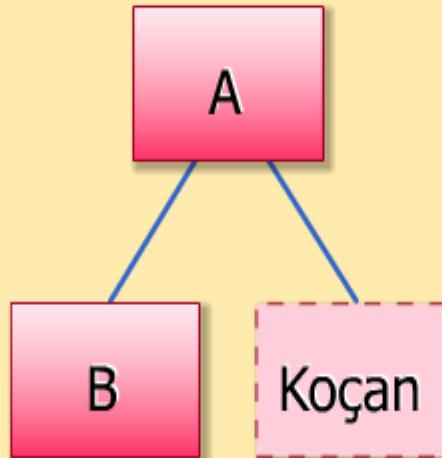
## Örnek:

A, B, C birimlerinden oluşan ve birim şeması şekil 7.5(a)'da belirtilen bir sistemin bu tür koçan kullanılarak sınanması şekil 7.5(b) ve şekil 7.5(c)'de belirtilmektedir.

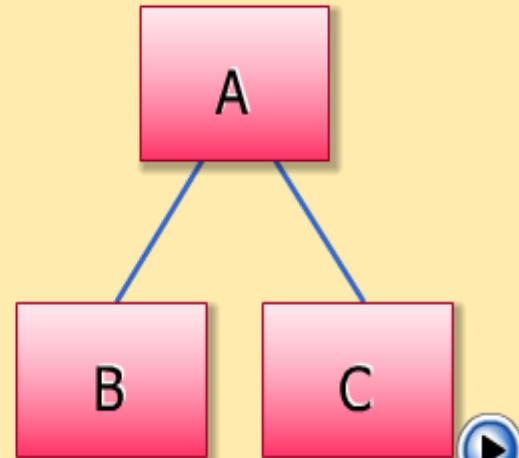
**(a)** Sınanacak Sistem  
Bütünleştirme



**(b)** A-B Bütünleştirme



**(c)** A-B-C



**Şekil 7.5:** Bütünleştirme Sınamasında "koçan" Kullanımı

İlk adımda A ve B birimleri bütünleştirilir; C için bir "koçan" yazılır. İkinci adımda ise "koçan" kaldırılır ve C ile yer değiştirilerek A-B-C bütünleştirilir.

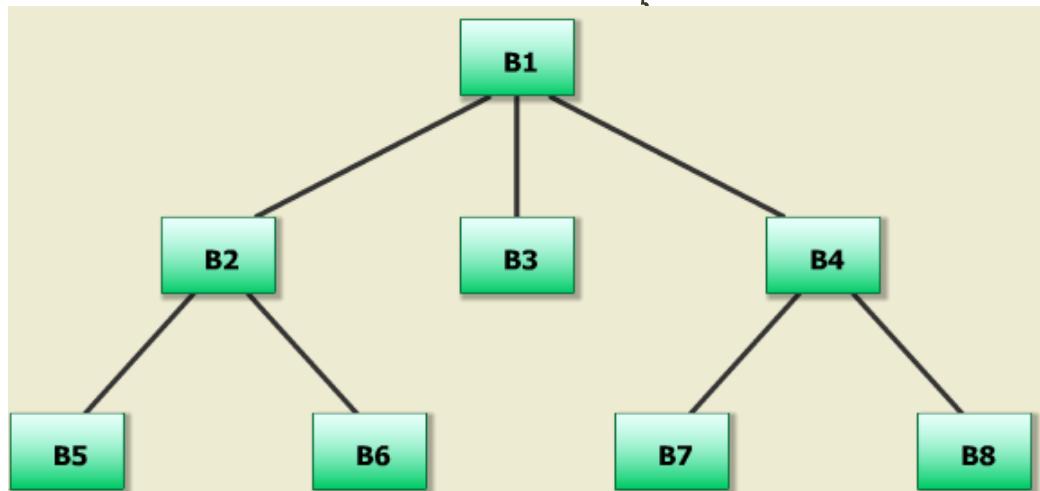
# **Yukarıdan Aşağı Sınama ve Bütünleştirme**

Yukarıdan aşağıya doğru bütünleştirme işleminde iki yaklaşım izlenebilir:

- 1. Yaklaşım: Düzey Öncelikli Bütünleştirme
- 2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

# 1. Yaklaşım: Düzey Öncelikli Bütünleştirme

- En üst düzeyden başlanır, öncelikle aynı düzeylerdeki birimler bütünleştirilir.



**1. Adım:** B1-B2 ( KoçanB3 - KoçanB4- KoçanB5 - KoçanB6)

**2. Adım:** B1-B2-B3 ( KoçanB4- KoçanB5 - KoçanB6)

**3. Adım:** B1-B2-B3-B4 ( KoçanB7 - KoçanB8- KoçanB5 - KoçanB6)

**4. Adım:** B1-B2-B3-B4-B5 ( KoçanB7 - KoçanB8- KoçanB6 )

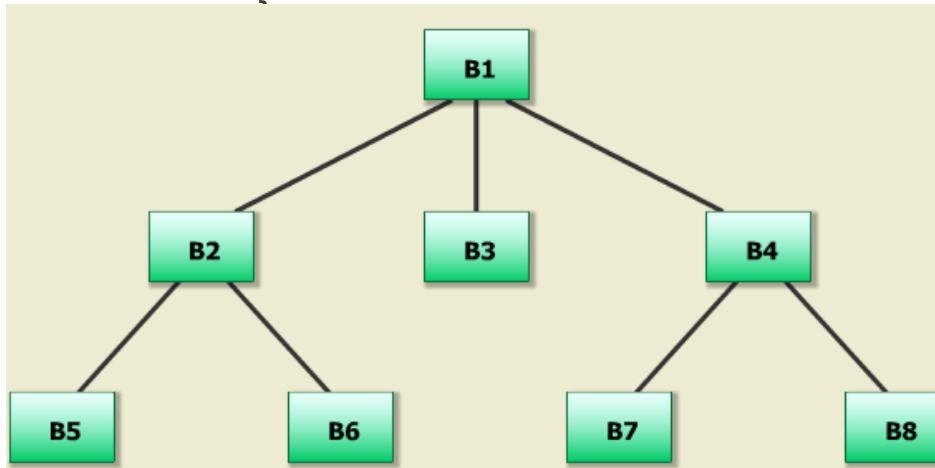
**5. Adım:** B1-B2-B3-B4-B5-B6 ( KoçanB7 - KoçanB8 )

**6. Adım:** B1-B2-B3-B4-B5-B6-B7 ( KoçanB8 )

**7. Adım:** B1-B2-B3-B4-B5-B6-B7-B8

## 2. Yaklaşım: Derinlik Öncelikli Bütünleştirme

- En üst düzeyden başlanır. Birim şemasında bulunan her dal soldan sağa olma üzere ele alınır. Bir dala ilişkin bütünlüğüne bitirildiğinde diğer dalın bütünlüğünü başar.



- 1. Adım:** B1-B2 ( KoçanB3 - KoçanB4- KoçanB5 - KoçanB6)
- 2. Adım:** B1-B2 - B5 (KoçanB3 - KoçanB4 - KoçanB6)
- 3. Adım:** B1-B2-B5-B6 (KoçanB3 - KoçanB4)
- 4. Adım:** B1-B2-B5-B6-B3 (KoçanB4 )
- 5. Adım:** B1-B2-B5-B6-B3-B4 ( KoçanB7 - KoçanB8 )
- 6. Adım:** B1-B2-B5-B6-B3-B4-B7 (KoçanB8 )
- 7. Adım:** B1-B2-B5-B6-B3-B4-B7-B8

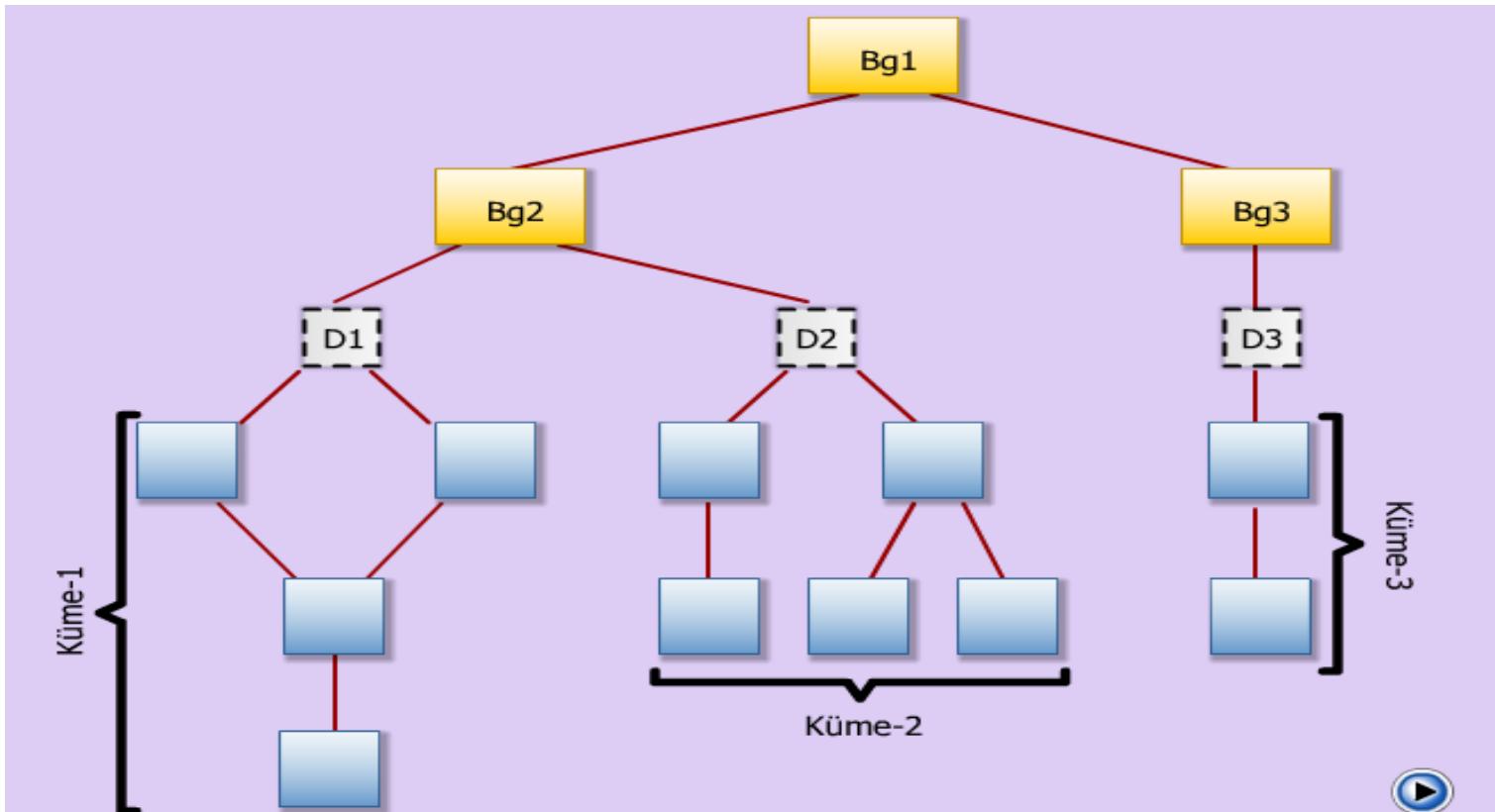
# Aşağıdan Yukarıya Sınama ve Bütünleştirme

- Once en alt düzeydeki işçi birimleri sınanır ve bir üstteki birimle sınanması gerekiğinde bu üst bileşen, bir '**sürücü**' ile temsil edilir.
- Yine amaç, çalışmasa bile arayüz oluşturacak ve alt bileşenin sınanmasını sağlayacak bir birim edinmektir.

# Aşağıdan Yukarıya Sınama ve Bütünleştirme

- Bu kez kodlama, bütünllestirme ve sınama aşağı düzeylerden yukarı düzeylere doğru gelişir ve yukarı düzeylerde önce sürücü olarak yazılan birimler sonra gerçekleriyle yer değiştirerek o düzeyin birimleri/alt sistemleri olurlar.

# Aşağıdan Yukarıya Sınama ve Bütünleştirme



Şekil 7.7: Aşağıdan Yukarı BüTÜnleştirme

Sekilde;

1. Belirli bir yazılım alt işlevini gören alt düzey birimler **kümeler** biçiminde oluşturulurlar,
2. Denetim amaçlı bir **sürücü** programı sınama işlemi için girdi ve çıktı oluşturmak amacıyla yazılır,
3. Sürücüler aşağıdan yukarı kaldırılır ve gerçek birim ya da birim kümeleriyle değiştirilerek sınama işlemi sürdürülür.



Tekrar

# SINAMA PLANLAMASI

- Sınaması işlemi çok kapsamlıdır. Bir plan güdümünde gerçekleştirilmelidir. Böyle bir planın temel bileşenleri önceki sayfalarda belirtilmiştir.
- Yazılım yaşam döngüsünün süreçlerine koşut olarak, farklı ayrıntı düzeylerinde birden fazla sınav planı hazırlanır.

**Giriş**

**Amaç**  
**Tanım ve Kısıtlamalar**  
**Referanslar**

# SINAMA PLANLAMASI

## Sınamaya Yönetime

Sınamaya Konusu

Sınamaya Etkinlikleri ve Zamanlama

Temel Sınamaya Etkinlikleri

Destek Etkinlikler

Kaynaklar ve Sorumluluklar

Personel ve Eğitim Gereksemeleri

Sınamaya Yaklaşımı

Riskler ve Çözümler

Onaylar

# SİNAMA PLANLAMASI

## Sınama Ayrıntıları

Sınanacak Sistemler

Girdiler ve Çıktılar

Sınamaya Başlanma Koşulları

Girdilerin Hazır Olması

Ortam Koşulları

Kaynak Koşulları

Sınama Tamamlama Kısıtları

Sınama Geçme-Kalma Kısıtları

Sınama Askıya Alınma Kısıtları ve Südürme Gerekleri

Sınama Sonuçları

- Sınama planları; Birim (Modül) Sınama Planı, Alt Sistem Sınama Planları, Bütünleştirme Sınama Planları, Kabul Sınama Planları, Sistem Sınama Planları biçimindedir.

# SİNAMA BELİRTİMLERİ

- Sınamaların belirtimleri, bir sınavın işleminin nasıl yapılacağına ilişkin ayrıntıları içerir. Bu ayrıtlar temel olarak:
- sınavanın program modülü ya da modüllerinin adları,
- sınavın türü, stratejisi (beyaz kutu, temel yollar vb.),
- sınavın verileri,
- sınavın senaryoları  
türündeki bilgileri içerir.

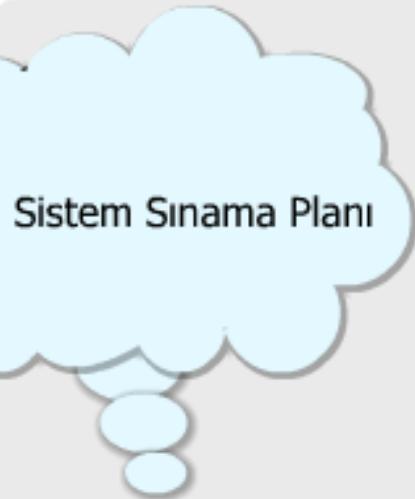
# SİNAMA BELİRTİMLERİ

- Sınamaya verilerinin elle hazırlanması çoğu zaman kolay olmayabilir ve zaman alıcı olabilir. Bu durumda, otomatik sınavlama verisi üreten programlardan yararlanılabilir.
- Sınamaya senaryoları, yeni sınavlama senaryosunu üretebilmeye yardımcı olacak biçimde hazırlanmalıdır. Zira sınavlama belirtimlerinin hazırlanmasındaki temel amaç, etkin sınavlama yapılması için bir rehber oluşturmasıdır.

# SİNAMA BELİRTİMLERİ

- Sinama işlemi sonrasında bu belirtimlere,
  - sinamayı yapan,
  - sinama tarihi,
  - bulunan hatalar ve açıklamaları türündeki bilgiler eklenerek sinama raporları oluşturulur.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ



Planlama

Çözümleme

Tasarım

Geçerleştirm

Kurulum

Planlama aşamasında genel sınavma planı oluşturulur. Söz konusu plan tüm sınavma etkinliklerini çok genel hatlarıyla tanımlar ve sınavma planlamasında verilen bilgileri içerir.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ

Alt Sistem Sınama Planları



Çözümleme aşamasında, sistemler ve alt sistemler ortaya çıkarılır ve sınama planı alt sistemler bazında ayrıntılılandırılır.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ

- Modül Sınaması Planı
- Sınamaya İlgili Belirtimler
- Sınamaya İlgili Eğitim Kılavuzları



Tasarım aşaması, tüm yazılım modüllerinin ortaya çıkarıldığı aşamadır. Bu aşamanın başlangıcında yazılım modülleri için sınav planı detaylandırılır ve sınav belirtimleri hazırlanır. Söz konusu belirtimler, kullanıcı kılavuzları ile birlikte sınavlara ilişkin eğitim için temel bilgileri oluşturur.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ

- Modül Sınaması
- Bütünleştirme Sınaması
- Sınayıcı Eğitimi

Planlama

Çözümleme

Tasarım

Gerçekleştirim

Kurulum

Gerçekleştirim aşamasında teknik sınavmalar yapılır, sınavma raporları hazırlanır ve kullanıcı sınavıcıları eğitilir.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ

- Kullanıcı Sınama
- Sınama Raporları



Kurulum aşamasından hemen sonra kullanıcı sınavları yapılarak sınav raporları hazırlanır.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ

- Sınamada bulunan her hata için, değişiklik kontrol sistemine (DKS), "Yazılım Değişiklik İsteği" türünde bir kayıt girilir. Hatalar, DKS kayıtlarında aşağıdaki gibi gruplara ayrılabilir:
  - ✖ **Onulmaz Hatalar:** BT projesinin gidişini bir ya da birden fazla aşama gerileten ya da düzeltilmesi mümkün olmayan hatalardır.
  - ✖ **Büyük Hatalar:** Projenin kritik yolunu etkileyen ve önemli düzeltme gerektiren hatalardır.

# YAŞAM DÖNGÜSÜ BOYUNCA SİNAMA ETKİNLİKLERİ

✖ **Küçük Hatalar:** Projeyi engellemeyen ve giderilmesi az çaba gerektiren hatalardır.

✖ **Şekilsel Hatalar:** Heceleme hatası gibi önemsiz hatalardır.

# BİR UYGULAMA: Görsel Yazılım Geliştirme Ortamında Sınama

- Bu kısımda, gerçek yaşam ortamında, **Oracle Designer CASE** aracı ve **Developer** görsel yazılım geliştirme platformu kullanılarak geliştirilen yazılım modüllerinin sınanması işleminin nasıl yapılacağı ve buraya kadar açıklanan sınama yöntemlerinin nasıl uygulandıkları bir örnek üzerinde anlatılmaktadır.

# BİR UYGULAMA: Görsel Yazılım Geliştirme Ortamında Sınama

- Oracle Developer kullanılarak geliştirilen her yazılım **formlardan** oluşur.
- Bir form, bir ekran ve bu ekranada yapılan işlemlere karşılık gelen PL/SQL kodları biçiminde tanımlanır.
- Bu örnekte elimizde, sınavma işlemine koşulacak ve uygulamanın çeşitli işlevlerine ilişkin bir dizi form olduğunu düşünebiliriz.

# BİR UYGULAMA: Görsel Yazılım Geliştirme Ortamında Sınama

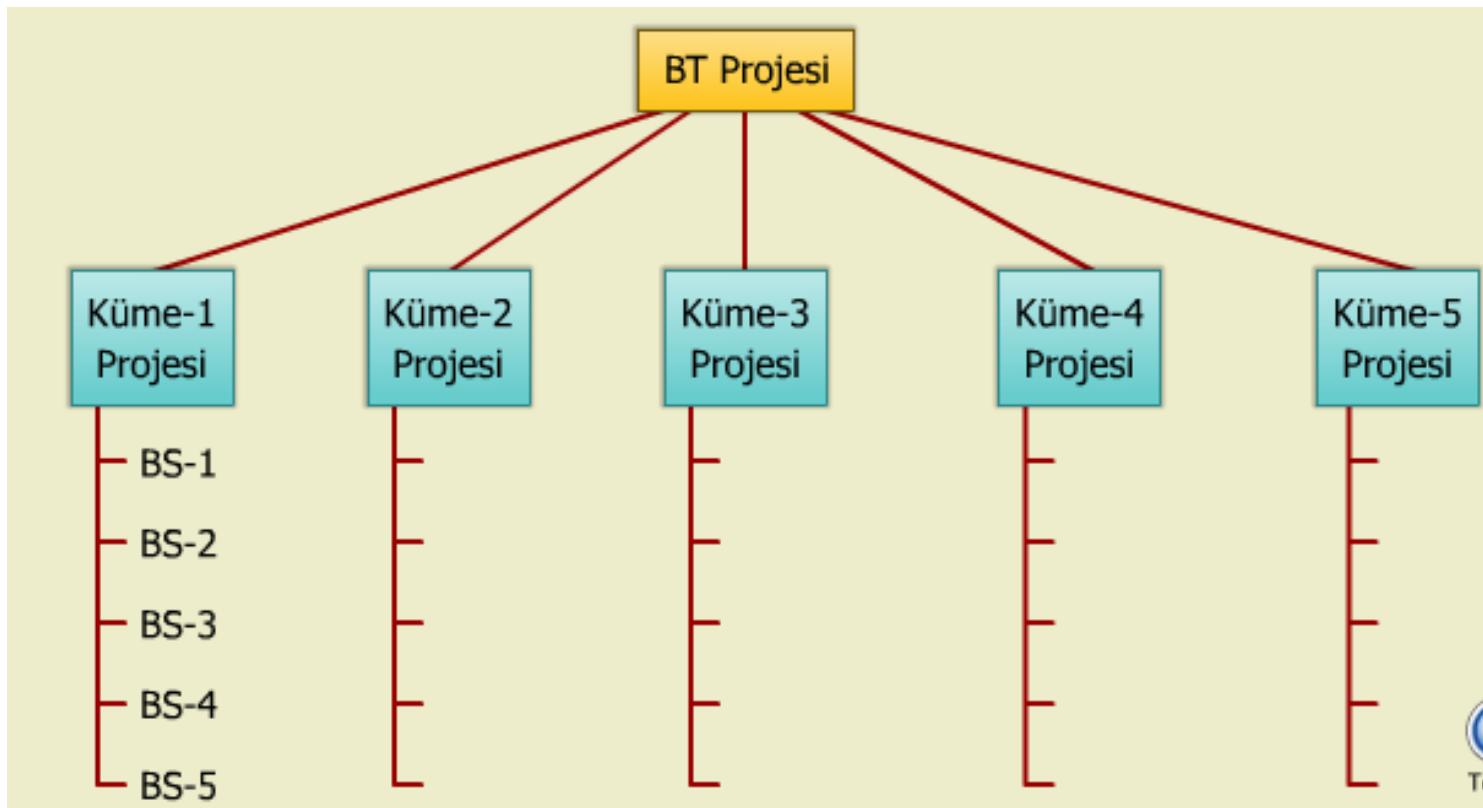
- Bu örnekte söz edilen uygulama, 2000'den fazla kullanıcısı olan, ülkenin çeşitli yörelerine dağılmış birimlerde çalışacak biçimde tasarlanmış ve 1000'den fazla Developer formundan oluşmaktadır.

# BİR UYGULAMA: Görsel Yazılım Geliştirme Ortamında Sınaması

- Uygulamanın sınavınamasına gelmesi, 2 yıllık bir süre ve yaklaşık 100 kişi-yıl'lık bir iş gücü gerektirmiştir.
- Uygulama beş ana kümeye bölünmüştür ve her kümeye belirli sayıda bilgi sistemini içermektedir.
- Toplam olarak 30 bilgi sistemi bulunmaktadır. Uygulama sıra düzeni Şekil 7.10'da verilmektedir.

# BİR UYGULAMA:

## Görsel Yazılım Geliştirme Ortamında Sınama



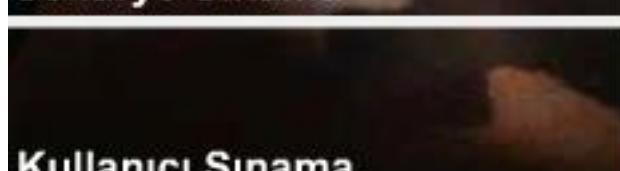
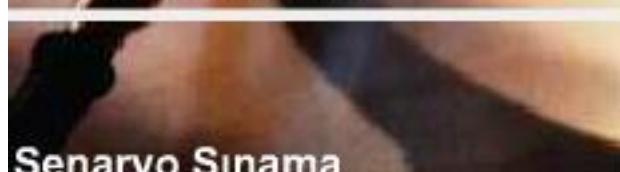
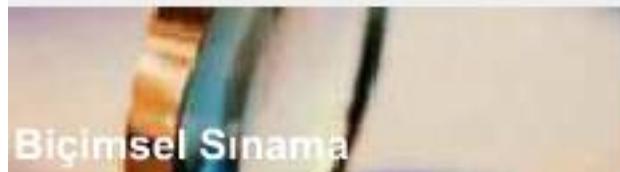
Tek

# Sınaması Ortamı Oluşturulması

- Üretimin etkilenmemesi amacıyla, yalnızca sınayıcıların kullanacakları ve ayrı bilgisayarlardan oluşan bir sınavma ortamı oluşturuldu.
- Oluşturulan sınavma ortamı ile üretim ortamının birebir aynı olması sağlandı.
- Üretimi biten yazılım parçaları, bir kayıt düzeni içerisinde sınavma ortamına alındı.



# Sınamaya Yöntemlerine Karar Verilmesi



## Teknik Sınamaya

Üretim ortamında yapılacak sınamaya olarak karar verildi. Bu sınamaya, modül sınaması ve bütünlendirme sınaması olarak üretim ekipleri tarafından gerçekleştirildi. Modül sınamaya yöntemi olarak “beyaz kutu” sınamaya yöntemi uygulandı. Tüm program deyimleri en az bir kez, tüm döngüler en az 10 kez yinelenecek biçimde sınamaya yapıldı.

Bütünlendirme sınamaya yöntemi olarak, “yukarıdan aşağıya sınamaya yöntemi” ve “derinlik öncelikli bütünlendirme” stratejisi uygulandı.

# Sınama Yöntemlerine Karar Verilmesi



## Biçimsel Sınaması

Üretim ekiplerinden bağımsız olarak, sınama ekipleri tarafından yapılan sınamadır. Bu sınama, Developer formları üzerinde görsel olarak yapıldı. Amaç, formların, önceden kararlaştırılan standartlara uygunluğunun saptanmasıydı. Örneğin, form alanları, kararlaştırılan uzunlukta mı? Başlıklar istenilen gibi koyu mu? Yardım düğmesi hep aynı yerde mi vb.

Sınamalar, formlar işletilmeden yapılır. Tüm formlar tek tek incelenir ve standartlara uygun olmayanlar belirlenip, düzeltilmek üzere üretim ekibine geri iletilir. Biçimsel sınamaların yapılması amacıyla denetim listeleri hazırlanır ve sınama sırasında bu listeler kullanılır. Listelere kaydedilen her sonuç DKS'ye aktarılır. Bu yolla üretim ekiplerinin performansı izlenebilir.

# Sınama Yöntemlerine Karar Verilmesi



## İşletimsel Sınama

Üretim ekiplerinden bağımsız olarak, sınama ekipleri tarafından yapılan sınamadır. Büçimsel sınama işlemi bittikten sonra yapılır. Bu sınamada her form ayrı ayrı çalıştırılarak işlem yapılır. Amaç, formun çalışıp çalışmadığının belirlenmesidir. Form alanlarının sınır değerlerle çalışıp çalışmadığı, aykırı değer verildiğinde uygun hata iletisi alınıp alınmadığı vb belirlenmeye çalışılır.

# Sınamaya Yöntemlerine Karar Verilmesi



## Senaryo Sınaması

Sınamaya ekipleri tarafından yapılan sınamadır. Ancak, senaryoların hazırlanması sırasında üretim ekipleri ile birlikte çalışılır.

Amaç, birden fazla formun bir arada sıyanmasıdır. Bu amaçla, "senaryo"lar hazırlanır. Her senaryo, çözümleme aşamasında belirlenen bir iş fonksiyonuna karşılık gelecek biçimde, hazırlanır.

# Sınamaya Yöntemlerine Karar Verilmesi



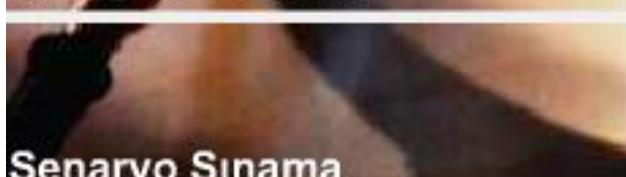
Teknik Sınaması



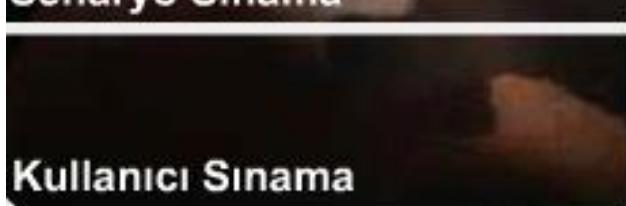
Biçimsel Sınaması



İşletimsel Sınaması



Senaryo Sınaması



Kullanıcı Sınaması

## Kullanıcı Sınaması

Kullanıcılar tarafından yapılması öngörülen sınamadır. Senaryo sınamasının kullanıcı tarafından yapılan biçimini olarak düşünülebilir.

# Kullanıcı Sınaması Eğitimi

- Sınamaların nasıl yapılacağına ilişkin eğitim verilmesi gerekmektedir.
- Eğitim kitapçıklarının hazırlanması amacıyla, senaryo sınamalarında kullanılan "senaryo"lar ve kullanıcı kitapçıkları kullanılır.

# Sınamaların Yapılması

Sınamalar sırasıyla:

- Teknik Sınaması,
  - Biçimsel Sınaması,
  - İşletimsel Sınaması,
  - Senaryo Sınaması,
  - Kullanıcı Sınaması
- biçiminde yapılır.

# Alınan Dersler 1

- Bir taraftan üretimin yapıldığı, öte yandan ise kullanıcı tarafından sınamaların yapıldığı bir ortamda, üretim ekipleri bir yandan yeni yazılım parçaları geliştirme, öte yandan ise sınavma sonucu bildirilen hataları düzeltme durumu ile karşı karşıyadır.

# Alınan Dersler 1

- Bu durum, zaman zaman üretim ekiplerinde dirençlere neden olmaktadır.
- Büyük projeler için, kaçınılmaz olan bu tür durumların iyi izlenmesi ve planlanması gerekmektedir.
- Bu anlamda Kalite ekibi oldukça önem kazanmaktadır.

## Alınan Dersler 2

- Sınamaya yapılacak ortam ile üretim ortamının, fiziksel olarak birbirinden ayrı olarak düzenlenmesi çok önemidir.
- Aksi durumda, sınav sırasında, sistemlerin kilitlenmesi ve veri tabanının zarar görmesi vb. sorunlarla karşılaşılır.
- Yapılan işlerin izlenmesi zorlaşır.

# Alınan Dersler 3

- Kullanıcı sınayıcı eğitimlerinin zaman zaman yinelelenmesi gereklidir.
- Kurumlardaki eleman değişiminin fazla olması sonucu, eğitim almamış kullanıcıların sistemi sınavası gibi durumlarla karşılaşılır ki bu da projeyi olumsuz olarak etkiler.

# Alınan Dersler 4

- Kullanıcı sinayıcıları, kendi işlerinin yoğunluğunu öne sürerek, sınama işlemine gereken önemi gösterememektedir.
- Bu durumda Yerinde Destek ekiplerine önemli görevler düşmektedir.

## Sorular

- 1.** Doğrulama ile Geçerleme arasındaki farklılıklarını belirtiniz. Birer örnekle açıklayınız.
- 2.** Sınamaya Yöntemlerini açıklayınız.
- 3.** "Beyaz Kutu" sınaması ile "Temel Yollar Sınaması" yöntemleri arasındaki farklılıklarını belirtiniz.
- 4.** Sınamaya Yöntemleri ile sınamaya belirtimleri arasındaki farkı belirtiniz.
- 5.** Yukarıdan aşağıya doğru bütünlendirme ve aşağıdan yukarıya bütünlendirme yöntemlerinin zorluklarını ve kolaylıklarını belirtiniz.
- 6.** Sınamaya belirtimlerinin önemi nedir.
- 7.** Kullanıcı sınavası sırasında yaşanabilecek sorunları belirtiniz.

# Genel

- Ders Kitabı: Yazılım Mühendisliği  
Erhan Sarıdoğan- papatya Yayıncılık  
(kitapyurdu.com)
- Diğer Kaynaklar:
  - Ders Notları.
  - Ali Arifoğlu, Yazılım Mühendisliği. SAS bilişim  
Yayınları
  - Internet, UML Kaynakları
  - Roger S. Pressman, Software Engineering –  
Practitioner's Approach



# YMT 312-Yazılım Tasarım Ve Mimarisi Planlama ve Sistem Çözümleme

Fırat Üniversitesi Yazılım Mühendisliği Bölümü



# Bu Haftaki Konular

Proje Planlama Aşamaları için Gerekli Koşullar .....	4
Proje Maliyet Kestirim Yöntemleri .....	14
Sistem Çözümleme .....	40
Gereksinim Çözümleme Çalışması .....	44
Gereksinim Veri Toplama Yöntemleri .....	55
Veri Modelleme Yöntemleri .....	66
Veri Akış Diyagramı .....	71
Süreç Tanımlama Dili .....	77

# Amaçlar

---

- Yazılımda Planlamanın nasıl yapılacağını öğrenmek,
- Planlama aşamasında gerekli olan yöntemleri öğrenmek,
- Planlama aşamasında kaynakların kullanımını görmek,
- Proje yapımında maliyet kestirimlerini öğrenip projede uygulamak,
- COCOMO Modeli hakkında bilgi sahibi olmak,
- Sistem Çözümleme aşamasını öğrenmek,
- Sistem çözümleme aşamasında kullanılan yapıları kavramak,
- Ayrıntılı bir sistemi çözümleyerek yeni bir sistem oluşturmak,
- Proje için gereksinimlerin ne anlama geldiğini öğrenmek.
- Veri akış diyagramlarının ne tür amaca göre oluşturulduğunu görmek,
- Genel olarak projemizde Tasarım aşamasından önceki Planlama ve Sistem Çözümleme aşamalarını kavramak.



# Planlama

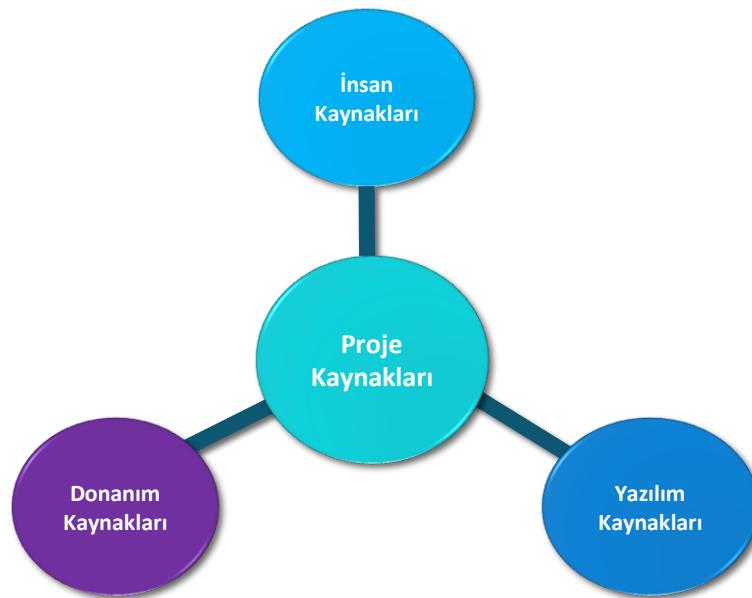
---

- Yazılım geliştirme sürecinin ilk aşaması,
- Başarılı bir proje geliştirebilmek için projenin tüm resminin çıkarılması işlemi,
- Proje planlama aşamasında yapılan işlemler
  - Proje Kaynaklarının Belirlenmesi
  - Proje Maliyetlerinin Kestirilmesi
  - Proje Ekip Yapısının Oluşturulması
  - Ayrıntılı Proje Planının Yapılması
  - Projenin İzlenmesi
- Proje planı tüm proje süresince sürekli olarak kullanılacak, güncellenecek ve gözden geçirilecek bir belgedir.



# Proje Kaynakları

---



**Planlama;** bu kaynakların tanımını yapar ve zaman kullanımı, görev süreleri, edinilme zamanlarını planlar.

# İnsan Kaynakları

---

- **Planlama;** hangi tür elemanların, hangi süre ile ve projenin hangi aşamalarında yer alacağını belirler.

Proje Yöneticisi	Donanım Ekip Lideri
Yazılım Ekip Lideri	Donanım Mühendisi
Web Tasarımcısı	Ağ Uzmanı
Sistem Tasarımcısı	Yazılım Destek Elemanı
Programcı	Donanım Destek Elemanı
Sistem Yöneticisi	Eğitmen
Veri Tabanı Yöneticisi	Denetleyici
Kalite Sağlama Yöneticisi	Çağrı Merkezi Elemanı

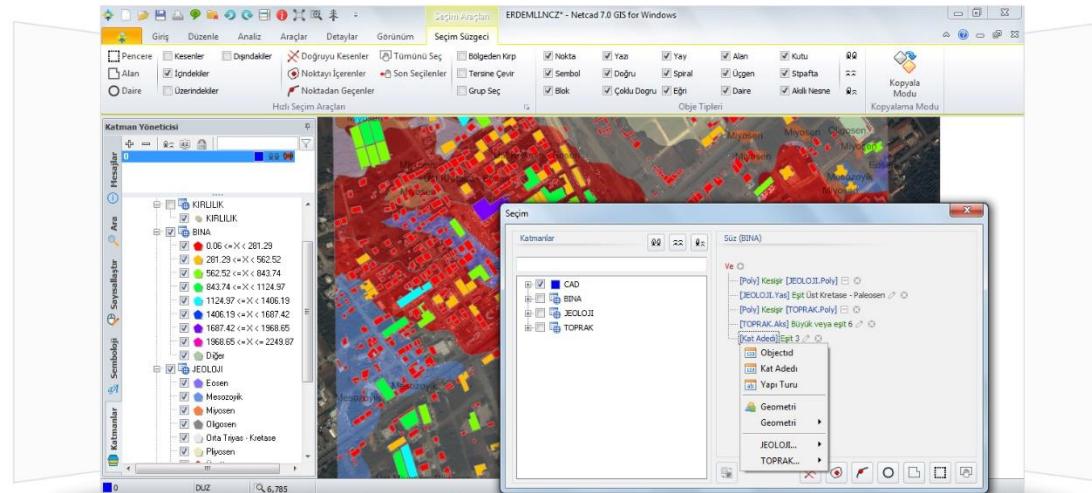
# Donanım Kaynakları

---

- Günümüzde giderek açık sistem mimarisine dönüştürmektedir.
- Donanım Kaynakları:
  - Ana Bilgisayarlar
  - Sunucular (Web, E-posta, Veri Tabanı)
  - Kullanıcı Bilgisayarları (PC)
  - Yerel Alan Ağı (LAN) Alt Yapısı
  - Geniş Alan Ağı (WAN) Alt Yapısı
- Yazılımın geliştirileceği ortam, gerçek kullanım ortamı dışında olmalıdır.
- Öte yandan, geliştirme ve uygulama ortamlarının aynı konfigürasyonda olmaları, ileride kurulum sırasında ortaya çıkabilecek taşıma sorunlarını büyük ölçüde giderecektir.

# Yazılım Kaynakları

- Büyük ölçekte otomatik hale getirilmiş ve bilgisayar destekli olarak kullanılmaktadır.
- **Bilgisayar Destekli Tasarım (CAD) ve Bilgisayar Destekli Mühendislik (CASE) araçları olarak bilinmektedirler.**



# Yazılım Kaynakları

---

## ■ İş sistemleri planlama araçları

- İş akış yapısının üst modelinin üretilmesinde kullanılır.
- Bilgi akışı, bilgi yapısı iş birimlerindeki tıkanıklıklar bu araçlar kanalıyla ortaya çıkarılır.

## ■ Proje yönetim araçları

- Yönetici tarafından, projede yapılan işlerin izlenmesi, kaynak ataması, proje iş yapısının üretilmesi, gözlenen değerlerin işlenmesini sağlayan araçlar.

## ■ Analiz ve tasarım araçları

- Kullanılan modelleme tekniklerini ayrı ayrı ya da bütünlük olarak uygulayan araçlar. Üretilen modelin kalitesinin ölçülmesi

## ■ Programlama araçları

- Derleyiciler, nesne-tabanlı programlama araçları, görsel programlama platformları.

# Yazılım Kaynakları

---

## ■ Test araçları

- Yazılımı doğrulama ve geçerleme işlemlerinde kullanılır. Test verisi üreticiler, otomatik test yordamları, ...

## ■ Prototipleme ve simülasyon araçları

- Geliştirmenin erken aşamalarında kullanıcıya, sonuç ürünün çalışması ile ilgili fikir veren ve yönlendiren araçlar.

## ■ Bakım araçları

- Programın bakımını kolaylaştıran, bir kaynak koddan program şemalarının üretilmesini, veri yapısının ortaya çıkarılmasını sağlayan araçlar.

## ■ Destek araçları

- İşletim sistemleri, ağ yazılımları, e-posta ve ortam yönetim araçları.

# Proje Maliyetleri

---

- **Maliyet kestirimi;** bir bilgi sistemi ya da yazılım için gerekebilecek iş gücü ve zaman maliyetlerinin üretimden önce belirlenebilmesi için yapılan işlemlerdir.

- **Kullanılan Unsurlar**

- Geçmiş projelere ilişkin bilgiler
- Proje ekibinin deneyimleri
- İzlenen geliştirme modeli

birden çok kez uygulanabilir



# Proje Maliyetleri

---

Maliyet yönetimi sayesinde;

- Gecikmeler önlenir
- Bilgi sistemi geliştirme süreci kolaylaştırılır
- Daha etkin kaynak kullanımı sağlanır
- İş zaman planı etkin olarak gerçekleştirilir
- Ürün sağlıklı olarak fiyatlandırılır
- Ürün zamanında ve hedeflenen bütçe sınırları içerisinde bitirilir



# Gözlemlenebilecek değerler

---

- Projenin toplam süresi
- Projenin toplam maliyeti
- Projede çalışan eleman sayısı, niteliği, çalışma süresi
- Toplam satır sayısı
- Bir satırın maliyeti (ortalama)
- Bir kişi/ay'da gerçekleştirilen satır sayısı
- Toplam işlev sayısı
- Bir işlevin maliyeti
- Bir kişi/ay'da gerçekleştirilen işlev sayısı
- Bir kişi/ay'da maliyeti



# Maliyet Kestirim Yöntemleri

---

## 1. Projenin boyut türüne göre

- Proje büyüklüğünü kestiren yöntemler
- Proje zaman ve işgücünü kestiren yöntemler

## 2. Projelerin büyüklüğüne göre

- Makro yöntemler (büyük boyutlu projeler 30 kişi-yıl)
- Mikro Yöntemler (orta ve küçük boyutlu projeler)

## 3. Uygulanış biçimlerine göre

- Çok yalın düzeyde
- Orta ayrıntılı düzeyde
- Çok ayrıntılı düzeyde

## 4. Değişik aşamalarda kullanılabilirlik

- Planlama ve analiz aşamasında kullanılabilen
- Tasarım aşamasında kullanılabilen
- Gerçekleştirim aşamasında kullanılabilen yöntemler

## 5. Yöntemlerin yapılarına göre

- Uzman deneyimine gereksinim duyan
- Önceki projelerdeki bilgileri kullanan yöntemler



# İşlev Noktaları Yöntemi

---

- İşlev noktaları geliştirmenin erken aşamalarında (analiz aşamasında) saptanan bir değerdir.
- Sistemin oluşturulduğu ortamdan bağımsız elde edilir.
- Problem tanımı girdi olarak alınarak üç temel adım izlenir:
  - Problemin bilgi ortamının incelenmesi
  - Problemin teknik karmaşıklığının incelenmesi
  - İşlev noktası hesaplama

# Problemin bilgi ortamının incelenmesi

---

- **Kullanıcı Girdileri:** personel sicil bilgileri, personel izin bilgileri gibi
- **Kullanıcı Çıktıları:** her türlü mantıksal çıktı; raporlar, ekran çıktıları, hata iletileri,...
- **Kullanıcı Soruları:** personel sicil bilgilerinin sorgulaması, personel maaş bilgilerinin sorgulaması
- **Dosyalar:** Her türlü mantıksal bilgi yiğini, tablolar, veri tabanları
- **Dışsal ara yüzler:** Başka programlarla veri传递. import/export

Bunların ağırlık faktörleriyle çarpımları toplanarak, Ayarlanmamış İşlev Nokta (AİN) sayısı hesaplanır.

# Problem Bilgi Ortamı Bileşenleri

---

Ölçüm Parametresi	Sayı	Ağırlık Faktörü			=	
		Yalın	Ortalama	Karmaşık		
Kullanıcı Girdi sayısı	?	3	4	6	=	
Kullanıcı Çıktı sayısı	?	4	5	7	=	
Kullanıcı Sorğu Sayısı	?	3	4	6	=	
Kütük Sayısı	?	7	10	15	=	
Dışsal Araryüz Sayısı	?	5	7	10	=	
<b>Toplam Sayı</b>					=	

# Problemin teknik karmaşıklığının incelenmesi

---

1. Uygulama, güvenilir yedekleme ve kurtarma gerektiriyor mu?
2. Veri iletişim gerektiriyor mu?
3. Dağıtılmış işlemler var mı?
4. Performans kritik mi?
5. Girdiler, çıktılar, dosyalar ya da sorgular karmaşık mı?
6. İçsel işlemler karmaşık mı?
7. Tasarlanacak kod yeniden kullanılabilir mi?
8. Dönüşürme ve kurulun tasarımda dikkate alınacak mı?



Cevaplar 0 ile 5 arasında puanlandırılır.

Bunlar hesaplanıp toplanarak **Teknik Karmaşıklık Faktörü (TKF)** elde edilir.

# İşlev Noktası Sayısı Hesaplama

---

- $\dot{I}N = A\dot{I}N * (0,65 * 0,01 * TKF)$

Değişik amaçlarla kullanılabilir

- **Üretkenlik** =  $\dot{I}N / \text{Kişi-Ay}$
- **Kalite** =  $\text{Hatalar} / \dot{I}N$
- **Maliyet** =  $\$ / \dot{I}N$

# Satır Sayısı Kestirimi

---

Assembly	300
Cobol	100
Fortran	100
Pascal	90
C	90
Ada	70
Nesne Kökenli Diller	30
4. Kuşak Dilleri	20
Kod Üreticiler	15

$iN=300$  ise ve Nesne Tabanlı bir dil (SmallTalk) kullanılıyor ise

$\text{Satır Sayısı} = 300 * 30$

olarak hesaplanır.

# Etkin Maliyet Modeli

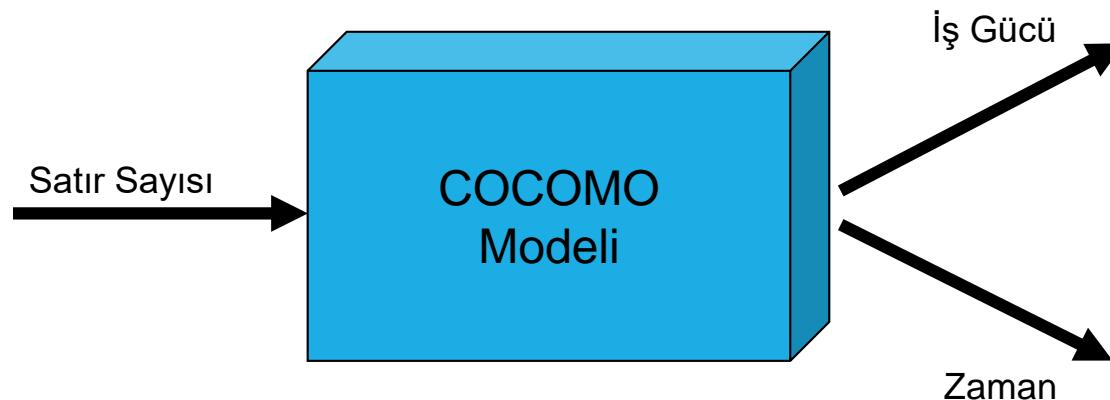
---

- COCOMO 1981 Boehm
- Mikro maliyet kestirim modeline örnektir.
- Kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabilir:
  - Temel Model
  - Ara Model
  - Ayrıntı Model



# COCOMO Modeli

---



# COCOMO Formülleri

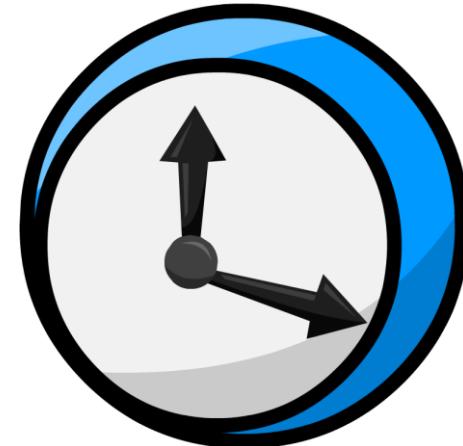
---

İş Gücü (K)  $K=a*S_b$

Zaman (T)  $T=c*K_d$

a,b,c,d : her bir model için farklı katsayılar

S : bin türünden satır sayısı



# Proje Sınıfları

---

## ■ **Ayrık Projeler:**

- Boyutları küçük,
- Deneyimli personel tarafından gerçekleştirilmiş
- LAN üzerinde çalışan insan kaynakları yönetim sistemi gibi

## ■ **Yarı Gömülü:**

Hem bilgi boyutu hem donanım sürme boyutu olan projeler

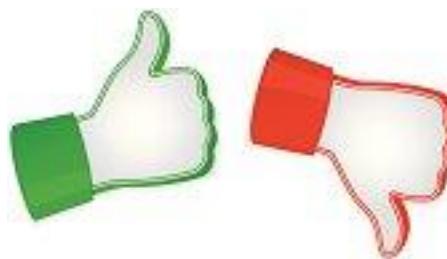
## ■ **Gömülü Projeler:**

Donanım sürmeyi hedefleyen projeler (pilotsuz uçağı süren yazılım - donanım kısıtları yüksek)

# Temel Model

---

- Küçük-orta boy projeler için hızlı kestirim yapmak amacıyla kullanılır
- **Dezavantajı:** Yazılım projesinin geliştirileceği ortam ve yazılımı geliştirecek ekibin özelliklerini dikkate almaz
- **Avantajı:** Hesap makinesi ile kolaylıkla uygulanabilir



# Temel Model

---

## ■ Ayrık Projeler

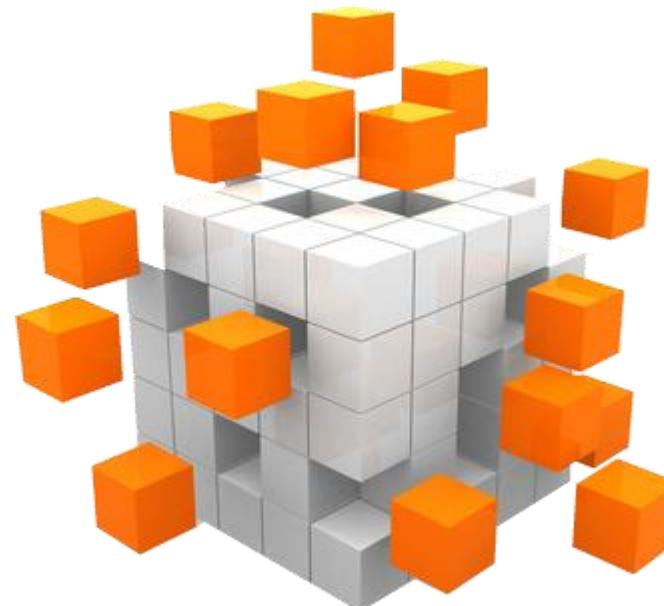
- İş Gücü  $K=2.4*S^{1,05}$
- Zaman  $T=2.5*K^{0,38}$

## ■ Yarı Gömülü Projeler

- İş Gücü  $K=3,0*S^{1,12}$
- Zaman  $T=2.5*K^{0,35}$

## ■ Gömülü Projeler

- İş Gücü  $K=3,6*S^{1,20}$
- Zaman  $T=2.5*K^{0,32}$



# Ara Model

---

- Temel modelin eksikliğini gidermek amacıyla oluşturulmuştur.
- Bir yazılım projesinin zaman ve iş gücü maliyetlerinin kestiriminde;
  - Proje ekibinin özelliklerini,
  - Proje geliştirmede kullanılacak araçları, yöntem ve ortamı dikkate alır.
- Üç Aşamadan oluşur:
  - İş gücü hesaplama
  - Maliyet çarpanı hesaplama
  - İlk iş gücü değerini düzeltme



# İş Gücü Hesaplama

---

- Ayrık Projeler

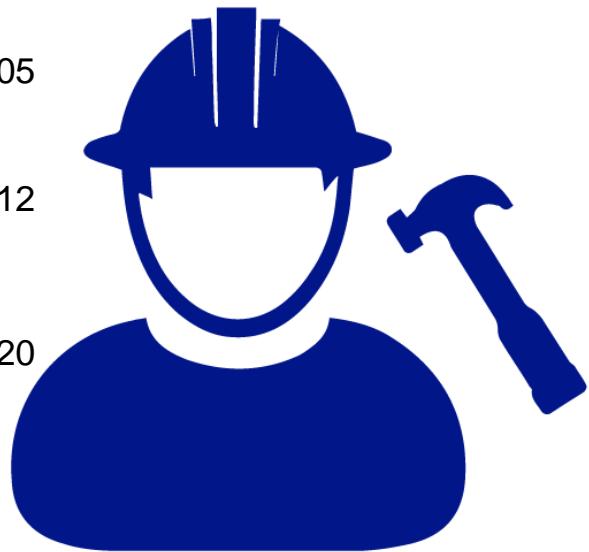
$$K=3.2 \cdot S^{1,05}$$

- Yarı Gömülü Projeler

$$K=3,0 \cdot S^{1,12}$$

- Gömülü Projeler

$$K=2.8 \cdot S^{1,20}$$



# Maliyet Çarpanı Hesaplama

---

- Maliyet Çarpanı 15 maliyet etmeninin çarpımı sonucudur.

$$C = C_1 * C_2 * C_3 * \dots * C_{15}$$



# Maliyet Etmenleri

---

Maliyet etmeni		Seçenekler					
		Çok Düşük	Düşük	Normal	Yüksek	Çok Yüksek	Oldukça Yüksek
Ürün Özellikleri	<b>RELY</b>	0,75	0,88	1,00	1,15	1,40	-
	<b>DATA</b>	-	0,94	1,00	1,08	1,16	-
	<b>CPLX</b>	0,70	0,85	1,00	1,15	1,30	1,65
Bilgisayar Özellikleri	<b>TIME</b>	-	-	1,00	1,11	1,30	1,66
	<b>STOR</b>	-	-	1,00	1,06	1,21	1,56
	<b>VIRT</b>	-	0,87	1,00	1,15	1,30	-
	<b>TURN</b>	-	0,87	1,00	1,07	1,15	-
Personel Özellikleri	<b>ACAP</b>	1,46	1,19	1,00	0,86	0,71	-
	<b>AEXP</b>	1,29	1,13	1,00	0,91	0,82	-
	<b>PCAP</b>	1,42	1,17	1,00	0,86	0,70	-
	<b>VEXP</b>	1,21	1,10	1,00	0,90	-	-
	<b>LEXP</b>	1,14	1,07	1,00	0,95	-	-
Proje Özellikleri	<b>MODP</b>	1,24	1,10	1,00	0,91	0,82	-
	<b>TOOL</b>	1,24	1,10	1,00	0,91	0,83	-
	<b>SCED</b>	1,23	1,08	1,00	1,04	1,10	-

# Ürün Özellikleri

---

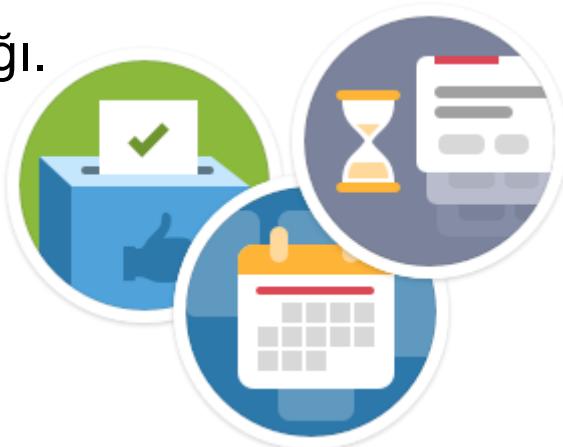
- **Rely:** Yazılımın güvenirliği
- **Data:** Veri Tabanının Büyüklüğü.  
Burada program büyüğününe oranı dikkate alınır.
- **Cplx:** Karmaşıklığı.



# Bilgisayar Özellikleri

---

- **Time:** İşletim zamanı kısıtı
- **Stor:** Ana Bellek Kısıtı
- **Virt:** Bilgisayar Platform Değişim Olasılığı.  
Bellek ve Disk kapasitesi artırımı,  
CPU Upgrade
- **Turn:** Bilgisayar İş Geri Dönüş Zamanı.  
Hata düzeltme süresi.



# Personel Özellikleri

---

- **Acap:** Analist Yeteneği:

Deneyim, Birlikte çalışabilirlik.

- **Aexp:** Uygulama Deneyimi.

Proje ekibinin ortalama tecrübe.

- **Pcap:** Programcı Yeteneği.

- **Vexp:** Bilgisayar Platformu Deneyimi.

Proje ekibinin geliştirilecek platformu tanıma oranı.

- **Lexp:** Programlama dili deneyimi.



# Proje Özellikleri

---

## ■ **Modp:** Modern Programlama Teknikleri.

- Yapısal programlama,
- Görsel programlama,
- Yeniden kullanılabılırlik.

## ■ **Tool:** Yazılım Geliştirme araçları kullanımı.

- CASE araçları
- Metin düzenleyiciler
- Ortam yönetim araçları

## ■ **Sced:** Zaman Kısıtı.



# İlk İşgücü değerini Düzeltme

---

■  $K_d = K * C$

$K_d$ =Düzeltilmiş İşgücü

\* Temel Formüldeki Zamanla formülü kullanılarak zaman maliyeti hesaplanır.



# Ayrıntı Modeli

---

Temel ve ara modele ek olarak iki özellik taşıır.

- Aşama ile ilgili işgücü katsayıları: her aşama için (planlama, analiz, tasarım, geliştirme, test etme) farklı katsayılar, karmaşıklık belirler
- Üç düzey ürün sıra düzeni: yazılım maliyet kestiriminde
  - Modül
  - Altsistem
  - Sistem

Sıra düzenini dikkate alır.



# Proje Ekip Yapısı Oluşturma

---

- PANDA proje Ekip yapısı temel olarak her proje biriminin doğrudan proje yönetimine bağlı olarak çalışması ve işlevsel bölümlenme esasına göre oluşturulur. Temel bileşenler
  - Proje Denetim Birimi
  - Proje Yönetim Birimi
  - Kalite Yönetim Birimi
  - Proje Ofisi
  - Teknik Destek Birimi
  - Yazılım Üretim Eşgüdüm Birimi
  - Eğitim Birimi
  - Uygulama Destek Birimi



66 Yeni bir ekip nüansı  
başarıya ulaşmak için yoktur! 99

# Yüklenici Proje Ekip Yapısı

---

- **Proje Denetim Birimi:** En üst düzey yönetimlerin proje ile ilgisinin sürekli sıcak tutulması ve onların projeye dahil edilmesi
- **Proje Yönetim Birimi:** Proje yönetiminden en üst düzeyde sorumlu birim. Proje boyutuna göre bir yada daha çok yöneticiden oluşur.
- **Kalite Yönetim Birimi:** Projenin amacına uygunluğunu üretim süreci boyunca denetler ve onaylar
- **Proje Ofisi:** Her türlü yönetimsel işlerden(yazışma, personel izleme) sorumlu birimdir.

# Yüklenici Proje Ekip Yapısı

---

- **Teknik Destek Birimi:** Donanım, İşletim sistemi, Veri tabanı gibi teknik destek
- **Yazılım Üretim Eşgündüm Birimi:** Yazılım Üretim Ekiplerinden oluşur(4-7 kişilik sayı fazla artmaz). Eğer birden fazla yazılım Üretim Ekibi varsa Ortak uygulama yazılım parçalarının geliştirilmesinden sorumlu Yazılım Destek Ekibi de olur.
- **Eğitim Birimi:** Proje ile ilgili her türlü eğitimden sorumludur.
- Uygulama Destek Birimi: Uygulama anında destek. (mesela telefonla)

# İş Sahibi Proje Ekip Yapısı

---

- Proje Eşgündüm Birimi
- Kalite Yönetim Birimi
- Proje Ofisi
- Teknik Altyapı İzleme Birimi
- Yazılım Üretim İzleme Birimi
- Eğitim İzleme Birimi
- Kullanıcı Eşgündüm Birimi



# Sistem Çözümleme

---

- Üretim sürecinin başlangıcıdır.
- Temel olarak mevcut sistemin nasıl çalıştığını araştırır.
- Temel hedef gereksinimlerinin saptanmasıdır.
- Mantıksal bir model
- Mutlaka bir model/yöntem kullanma zorunluluğu vardır.
- Yöntemler; veri modelleme ve süreç modelleme olmak üzere ikiye ayrılır.
- Çözümleme değişik açılardan değerlendirilir.

# Gereksinim Nedir?

---

- Sistemin amaçlarını yerine getirme yeteneği olan bir özellik ya da belirtim olarak tanımlanmaktadır.
- Gereksinim sistemin yada işlevlerinin nasıl yerine getirileceği ile ilgili değildir. **Ne olduğu ile ilgilidir.**
  - hangi veri tabanı,
  - hangi tablolar,
  - ne kadar bellek kullanılıyor,bunlar tasarım ve gerçekleştirim aşamasında ele alınır.
- Gereksinim, kullanıcı ve tasarımcı ya da yazılım mühendisi ile ilgili olarak iki amaca yönelik olacak biçimde tanımlanmalıdır;
  - **Kullanıcılar**, geliştirilecek sistemin amaçları istenilen ölçüde tanımlanmış mı sorusuna yanıt ararken,
  - **Tasarımcılar** ise gereksinimlerin tasarıma dönüştürülebilme uygunluğunu ararlar.

# Gereksinim Çeşitleri



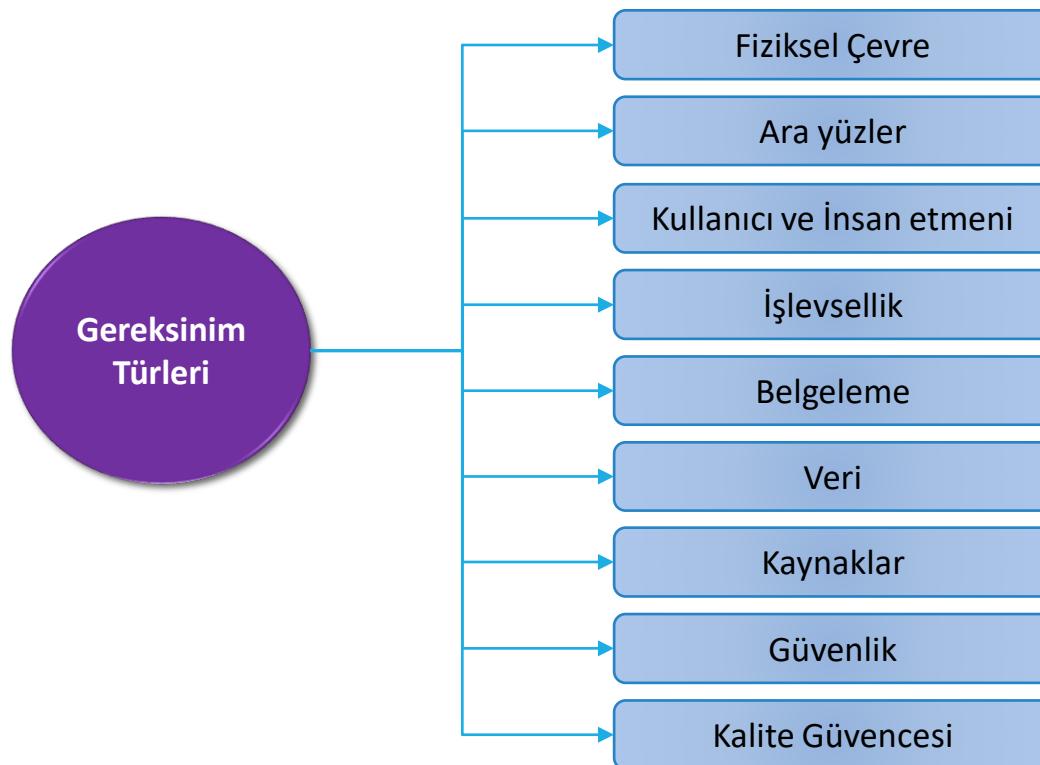
- Sistem ile çevresi arasındaki iletişimini belirleyen gereksinimler işlevsel gereksinim olarak tanımlanır.
- İşlevsel gereksinimler ayrıca sistemin herhangi bir durum karşısındaki davranışını belirler..
- Ör: Maaş çeki hazırlama da, işlevsel gereksinimler;
  - Maaş çekinin ne zaman hazırlanacağı,
  - Çek hazırlamak için ne tür girdiler verilmesi gereği,
  - Hangi koşullarda bir kişiye maaş verilmeyeceği gibi gereksinimler işlevsel gereksinimlerdir.



- Kullanıcının sorunundan bağımsız olarak çözülmesi gereken sorunlar işlevsel olmayan gereksinim olarak tanımlanır.
- Bir çok kaynacta bu terim yerine "Sistem Kısıtları" terimi de kullanılmaktadır.
- Ör: İşlevsel olmayan gereksinimler
  - Kullanılacak bilgisayar türü,
  - yazılım geliştirme ortamı,
  - kullanılacak veri tabanı yönetim sistemi vb.  
bu tür gereksinimlere örnek verilebilir.

# Gereksinim Türleri

---



# Fiziksel Çevre

---

- İşlevlerin geliştirileceği, işletileceği aygıtlar nerededir.
- Sistem tek bir yerde mi olacak? birden çok ve fiziksel olarak birbirinden ayrılmış yerler söz konusu mu?
- Sıcaklık nem oranı veya manyetik etkileşim gibi çevresel kısıtlamalar var mı?



# Ara yüzler

---

- Girdiler bir mi yoksa birden çok sistemden mi geliyor?
- Çıktılar bir mi yoksa birden çok sisteme mi gidiyor?
- Verilerin nasıl biçimlendirileceğine ilişkin bir yol var mı?
- Verilerin kullanılacağı önerilen bir ortam var mı?



# Kullanıcı ve İnsan etmeni

---

- Sistemi kim kullanacak?
- Farklı tiplerde kullanıcılar olacak mı?
- Her bir kullanıcı tipinin yetenek düzeyi nedir?
- Her kullanıcı tipi için ne tür eğitimler gerekli?
- Bir kullanıcının sistemi kötü amaçlı kullanması ne ölçüde zordur?

# İşlevsellik

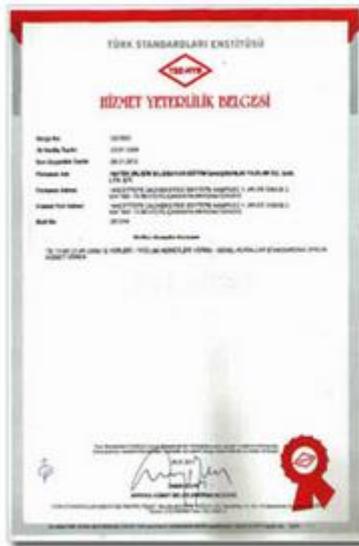
---

- Sistem ne yapacak?
- Sistem bunu ne zaman gerçekleştirecek?
- Sistem nasıl ve ne zaman değiştirilebilir ve/veya güçlendirilebilir?
- Çalışma hızı, yanıt süresi ya da çıktı üzerinde kısıtlayıcı etmenler var mı?



# Belgeleme

- Ne kadar belgeleme gereklidir?
- Belgeleme hangi kullanıcı kitlesini hedeflemektedir?



# Veri

---

- Hem giriş hem çıkış için verinin biçimi ne olmalıdır?
- Bu veri ne sıklıkla alınacak veya gönderilecektir?
- Bu verinin doğruluk (kesinlik) ölçüsü ne olmalıdır?
- Hesaplamalar hangi duyarlık derecesine kadar yapılandırılacaktır?
- Sistemde ne kadar veri akışı olacaktır?
- Veri belirli bir zaman süresince kaynağında saklanacak mı?



# Kaynaklar

---

- Sistemi kurmak, kullanmak ve bakımını yapmak için ne kadar malzeme, personel ve diğer kaynaklara ihtiyaç var?
- Geliştiriciler hangi yeteneklere sahip olmalı?
- Sistem ne kadar fiziksel yer kaplayacak?
- Güç, ısıtma ve soğutma için kısıtlar nelerdir?
- Geliştirim için tavsiye edilen bir zaman çizelgesi var mı?

# Güvenlik

---

- Sisteme ya da bilgiye erişim denetlenmeli midir?
- Bir kullanıcının verisi diğerinden nasıl ayrılacaktır?
- Kullanıcı programları, diğer program ve işletim sisteminden nasıl ayrı tutulacaktır?
- Sistem hangi sıklıkla yedeklenecektir?
- Yedek kopyaları başka yerde saklanacak mıdır?
- Yangın ve hırsızlığa karşı ne tür önlemler alınacaktır?
- Internet erişimi var mı? Güvenlik kullanılıyor mu?



# Kalite Güvencesi

---

- Güvenirlilik için gereksinimler nelerdir?
- Sistemin özellikleri insanlara nasıl aktarılmalıdır?
- Sistem çökmeleri arasında öngörülen zaman aralığı nedir?
- Kaynak kullanımı ve yanıt süresine ilişkin verimlilik ölçütleri nelerdir?
- Hataları kendisi bulup gidermeli mi?
- Çökmeler?
- Tasarımda yapılacak değişiklikler
- Kaynak kullanımı ve yanıt süresine ilişkin verimlilik ölçütleri?
- Sistemi bir bilgisayardan diğerine aktarmalı mı?

# Gereksinim Özellikleri

---

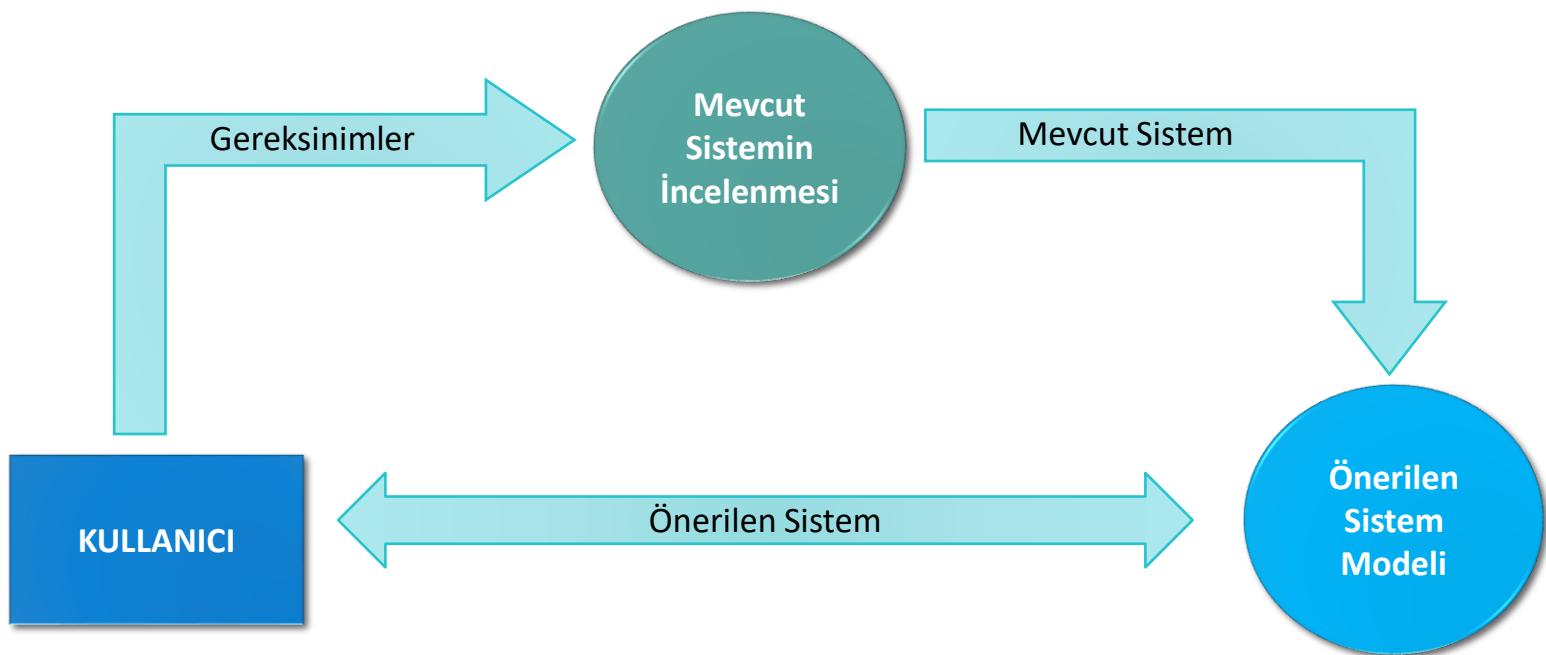
Gereksinimler üç amaca hizmet eder. Bunlar;

1. Geliştiricilerin, müşterilerin sistemin nasıl çalışmasını istediklerini anlamalarını sağlar.
2. Gereksinimler, sonuç sistemin ne özellikte ve işlevsellikte olacağını söyler.
3. Gereksinimler sınama ekibine, kullanıcıyı, sunulan sistemin istenen sistem olduğuna ikna etmek için neler göstermeleri gerektiğini söyler.



**What Do You Need?**

# Gereksinim Çözümleme Çalışması



# Gereksinim Çözümleme Çalışması

---

## Mevcut sistemin incelemesi

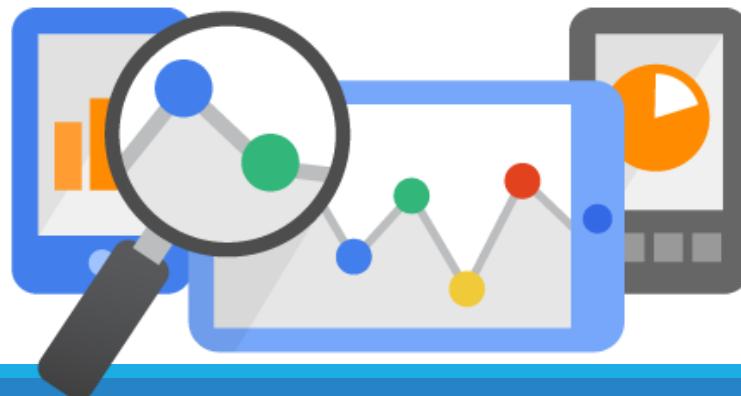
- Amaç: Yazılım geliştirilecek sistemin tanınmasıdır.
- Girdi, İşlev ve çıktı analizi yapılır.
- Kanun, yönerge ve yönetmenlikler incelenir.
- Elde yürütülen işlerde kullanılan form, defter ve yazışma örnekleri incelenir.

# Gereksinim Çözümleme Çalışması

---

## Önerilen Sistemin Modellenmesi

- Önerilen sistemin işlevsel yapısını, veri yapısını ve kullanıcı ara yüzünü oluşturur.
- Bu model daha çok bilgi sistemini geliştirecek teknik personele yönelikir.
- **Mantıksal model** olarak da tanımlanır.



# Doğrulama Süreci

---

1. Gereksinimler doğru oluşturulmuş mu?
2. Gereksinimler tutarlı mı?
3. Gereksinimler tam mı? (Dışsal tamlık / İçsel tamlık)
4. Gereksinimler gerçekçi mi?
5. Her gereksinim kullanıcı tarafından istenen bir şeyi mi tanımlamaktadır?
6. Gereksinimler doğrulanabilir mi?
7. Gereksinimler izlenebilir mi?



# Örnek

---

- Görev planlaması için kesinlik (doğruluk) yeterli olacaktır.
- Pozisyon hatası, yörünge boyunca 50 metreden, yörünge dışında 30 metreden az olacaktır.
- Sistem sorgulamaları gerçek zamanlı olarak yanıtlanmalıdır.
- Sistem kişi sorgulamaları en çok iki saniye içinde verilmelidir.

# Gereksinim Verisi Toplanması

---

1. Sorma Yöntemi
  - Karşılıklı görüşme
  - Anket
2. Psikolojik Türetme Teknikleri
3. İstatistiksel Teknikler



# Sorma Yöntemi

---

- Sorma yöntemi, gereksinim verilerinin toplanması sırasında kullanılan en önemli yöntemlerden biridir.
- Karşılıklı görüşme ya da anket yolu ile uygulanır.



# Sorma Yöntemi

---

## Karşılıklı Görüşme

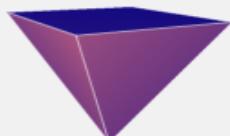
- Karşılıklı görüşme sırasında, gereksinimleri ilişkin amaçlar, düşünceler, remi olmayan yöntemler, duygular ve düşünceler araştırılır.
- Karşılıklı görüşme, veri toplama için en etkin yollardan biridir.
- Sorma yönteminde açık uçlu sorular (yanıtları kişiden kişiye değiŞebilen, yorumla açık sorular) ya da kapalı uçlu sorular (yanıtları evet hayır ya da bir sayı biçiminde kesin olan sorular) sorulabilir.
- Çözümleme kolaylığı, verinin güvenliği ve duyarlığı açısından kapalı uçlu sorular tercih edilmelidir.
- Sorular, dağınık olarak değil, yapısal bir biçimde sorulmalıdır. Bu neden bu üç tarzdan biri seçilmelidir.
  - **Piramit Tarzı**
  - **Koni Tarzı**
  - **Elmas Tarzı**

# Karşılıklı Görüşme Yapısal Soru Türleri



Piramit Tarzı

**Piramit Tarzı:** Özel sorularla başlayıp, giderek genel sorularla sürdürme.



Koni Tarzı

**Koni Tarzı:** Genel sorularla başlayıp özel sorularla sürdürme.



Elmas Tarzı

**Elmas Tarzı:** Özel sorularla başlayıp, genel sorularla sürdürme, tekrar özel sorularla sona erdirme.

Sorular sırasında dikkat edilecek önemli konu; **yönlendirici** sorular ve **iki nesneli** sorulardan kaçınmak olmalıdır.

# Sorma Yöntemi

## Anket Yöntemi

Kullanıcı sayısının fazla olduğu durumlarda eğilimleri ve davranış biçimlerini saptamak için kullanılır.

Genelde yazılı test biçiminde hazırlanır.

Bir anket sorusu temel olarak soru kısmı ve yanıt kısmı olmak üzere iki kısımdan oluşur. Yanıt kısmı da tanımlama ve ölçek kısmı olmak üzere iki bölümden oluşur.



# Psikolojik Türetme Teknikleri

---

- Özellikle belirsizliğin fazla olduğu ve zayıf yapılı ortamlarda, bilgi edinebilmek amacıyla insan psikolojisine dayalı teknikler kullanılır.
- Bu teknikler temelde görüşme ve ankete dayalı tekniklerdir.
- Diğerlerinden farkı, bilgi üretmek için psikolojide bilinen "üçleme" tekniğini kullanmasıdır.
- Bazı diğer psikolojik türetme yöntemleri, karar verme ortamlarında bilgi gereksinimlerini saptamak amacıyla algılama haritaları ve neden-etki çizgeleri kullanmaktadır.

# İstatistiksel Teknikler

---

- Veri yoğun ve veri hacmi yüksek ortamlarda verinin özelliklerini belirlemek amacıyla kullanılır.
  - Bu yöntemlerden en çok bilinen ikisi **Örnekleme Yöntemi** ve **PIRA Modelidir.**
- **Örnekleme yöntemi**, bir topluluk içerisinde, sistematik yolla temsil edici örnek alma biçiminde tanımlanır.
- Amaç, veri toplama hızını artırmak ve verilerdeki çelişkileri önlemektir.



# Veri Modelleme Yöntemleri

---

Önerilen sistemin mantıksal modelinde veri yapısını açıklamak amacıyla 'Veri Modelleme' yöntemleri kullanılmaktadır.

Bu yöntemler, veri yapısını çeşitli düzeylerde tanımlama (en soyut düzeyden en ayrıntı düzeye kadar) amacını güder.

Sistem Çözümleme aşamasında en yaygın olarak kullanılan veri modelleme yöntemleri:

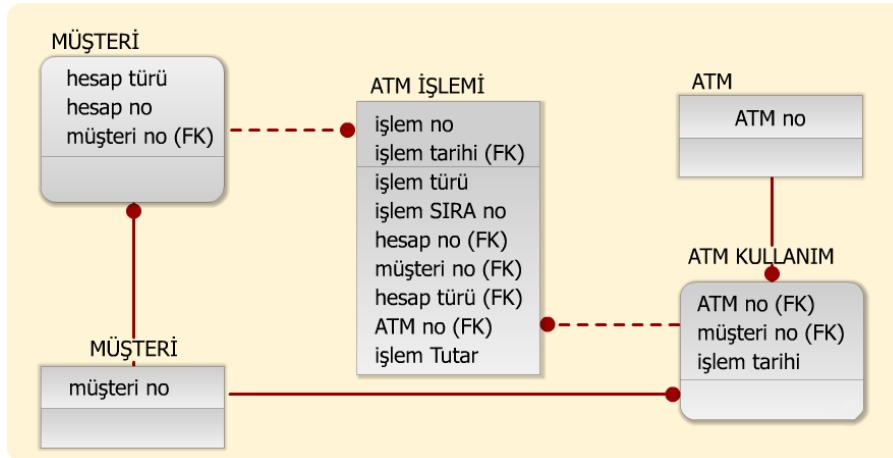
- **Nesne İlişki Şemaları**
- **Veri Sözlüğü**

olarak bilinmektedir.

**Nesne-İlişki Şemaları**, veri yapısını en soyut düzeyde tanımlamak amacıyla kullanılır.

**Veri Sözlüğü** ise veri yapısına ilişkin ayrıntı bilgileri içerir. Günümüzde Bilgisayar Destekli Yazılım Geliştirme Araçlarında bu yöntemler bütünlük olarak kullanılmaktadır.

# Nesne-İlişki Şemaları



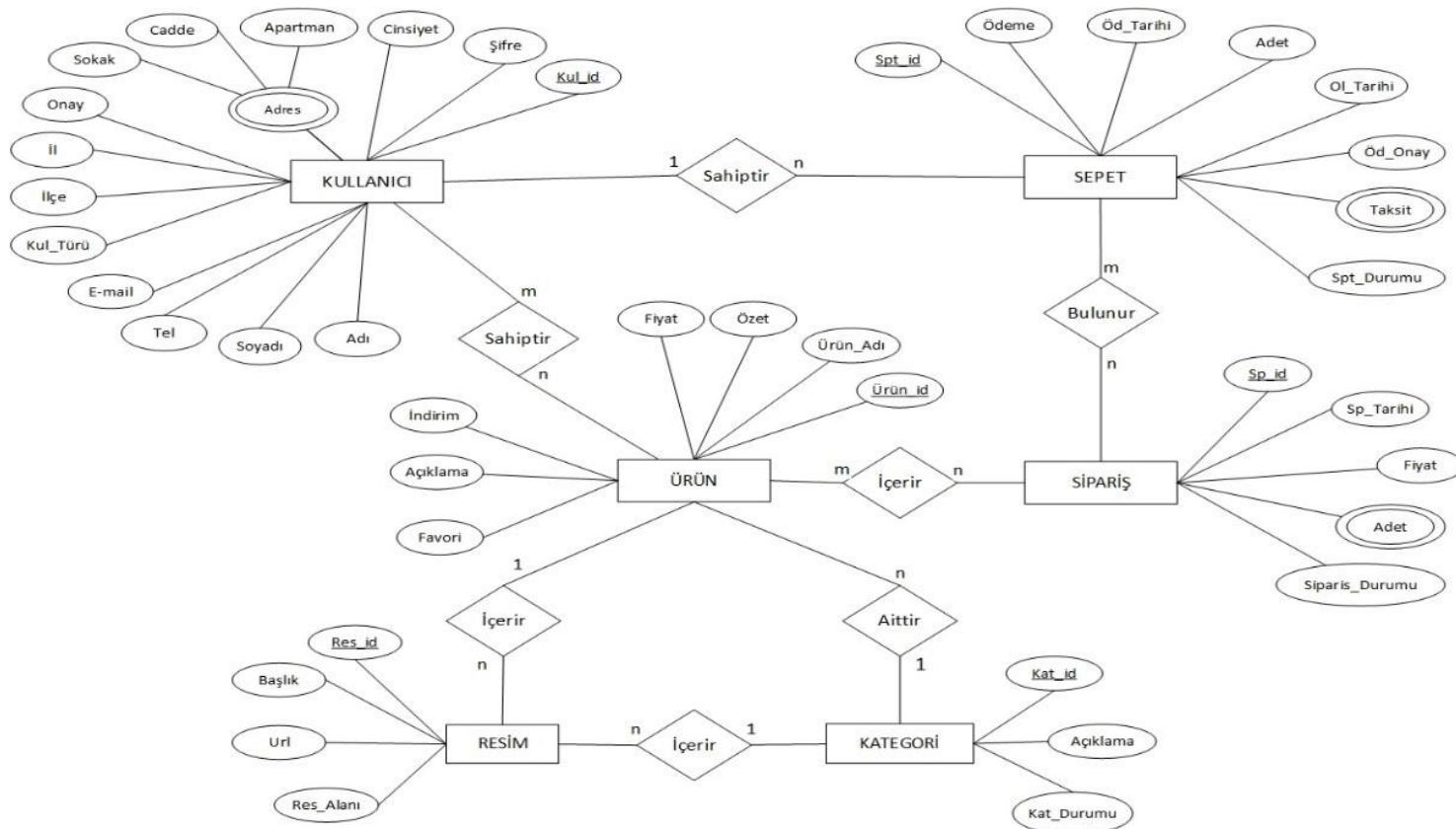
- Nesne ilişki şemaları, veri tasarıımı açısından çok önemlidir.
- Geliştirilecek sistemin kullanacağı ana veri nesneleri ve aralarındaki ilişkileri belirtir.
- Günümüzde birçok CASE aracı nesne ilişki şemalarını otomatik olarak veri tabanı tablo yapılarına dönüştürmektedir.

**Bir veri nesnesi, üç temel özelliği ile bilinir:**

1. Veri nesnesi varlığının adı: Veri nesnesi varlığını tanımlayan özelliklerdir.
2. Veri nesnesi varlığının özellikleri
3. Veri nesnesi varlığının diğer veri nesnesi varlıklarına referansı: Veri nesnesi varlığının diğer veri nesneleri ile olan ilişkisinin belirtilmesi amacıyla kullanılır. Bu amaçla her bir veri nesnesini tek olarak belirleyen bir belirteç (anahtar) kullanılır. Söz konusu belirteç veri nesnesi varlığının ad özelliklerini arasında yer alır.

# Veri Modelleme ER diyagramı

---



# Veri Sözlüğü

---

➤ Nesne İlişki şemalarında belirtilen nesne özelliklerinin ayrıntılı tanımları Veri Sözlüğünde yer alır. Söz konusu ayrıntılı tanımlar genel olarak:

- Veri Adı
- Veri Eş-adı (Aynı veri için kullanılan diğer ad)
- Nerede/nasıl kullanıldığı
- İçerik tanımı

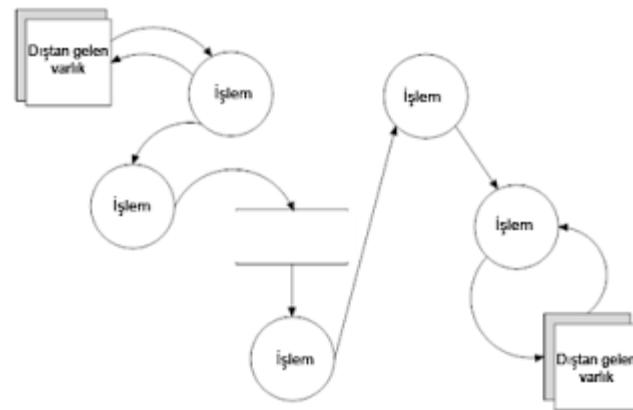
türünde bilgileri içerir. Veri nesnelerinin sırasının tanımlanması, birden çok veri nesnesi arasından seçim ve veri nesnelerinin yinelemeli gruplamaları için özel gösterim biçimleri kullanılarak karmaşık veri nesneleri tanımlanır.

Nitelik	Tanım	Etki alanı	Gerekli	Tür
<i>kat_id</i>	Bu alan adı Primary key ve otomatik olarak artış gösterir.	Primary key ve değiştirilemez. Integer olarak tanımlıdır.	✓	Int(11)
<i>kat_adi</i>	Kategori adı	Kategori seçeneklerinden biri, adı.	✓	Varchar(150)
<i>aciklama</i>	Sipariş tarihi	Boş geçilebilir.		text
<i>kat_durumu</i>	Kategori aktif/pasif	Kategori durumu	✓	Enum('1','0')

Kategoriler Tablosu Veri Sözlüğü

# Süreç/İşlem Modelleme Yöntemleri

- Geliştirilecek sistemin süreç ya da işlemlerini ve bu süreçler arasındaki ilişkileri tanımlamak amacıyla kullanılan yöntemlerdir.
  - Veri Akış Diyagramları (DFD)
  - Süreç Tanımlama Dili (PDL)



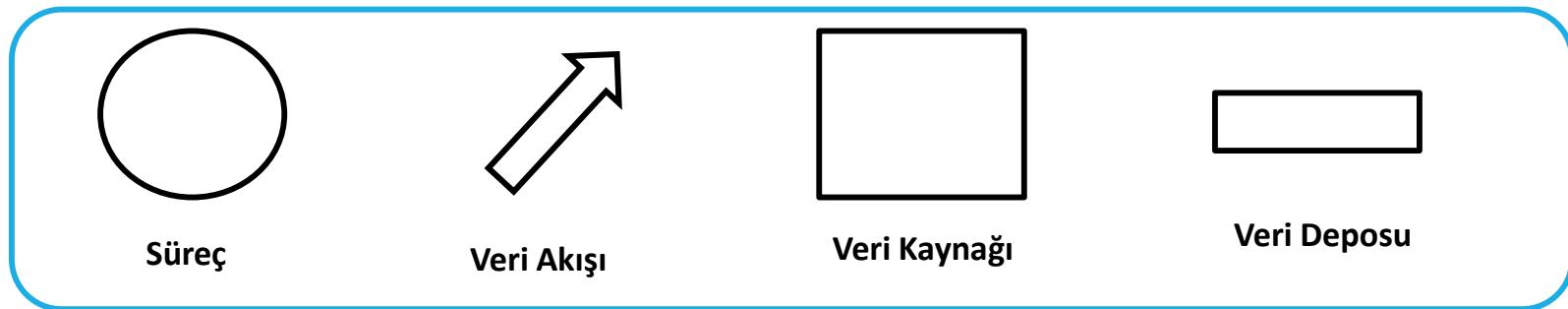
# Veri Akış Diyagramı

---

- VAD kullanılarak, geliştirilecek sistemin mantıksal modeli, '**Yukarıdan Aşağıya**' bir yaklaşımıyla oluşturulur.
- Sistem önce en genel biçimde ele alınır, yalnızca dışsal ilişkileri incelenir.
- Daha sonra, sistemin iç yapısındaki süreçler ve bu süreçler arasındaki ilişkiler belirlenen bir ayrıntı düzeyine kadar modellenir.
- Yapısal sistem geliştirme metodolojilerinde kullanılan tek veri modelleme yöntemi, **Veri Akış Diyagramları (VAD)** yöntemidir.
- VAD günümüzde oldukça yaygın olarak kullanılmaktadır. Hemen hemen bütün CASE araç ve ortamları VAD yöntemini içermektedir.
- VAD'nin oldukça yaygın olarak kullanılmasındaki temel etmen, hem bilgisayara yabancı olan kullanıcılar, hem de yazılım geliştiriciler için oldukça kolay algılanması ve kullanılabilirliğidir.

# VAD Yönteminde Kullanılan Semboller

VAD, 4 temel sembol kullanılarak oluşturulur.



- **Süreç** ya da işlemleri belirtmek amacıyla çember şekli kullanılmaktadır. Verilerin alındığı, ya da gönderildiği **dış birimleri** göstermek amacıyla **dikdörtgenler** kullanılmaktadır.
- Mantıksal veri yiğinlarını göstermek amacıyla **ucu açık dikdörtgenler** kullanılmaktadır.
- Gerek süreçler arası gerekse süreçler ile veri kaynakları ve veri depoları arasındaki veri akış ilişkileri göstermek amacıyla **oklar** kullanılmaktadır.

# Veri Akış Diyagramı

## Kapsam Diyagramı

Geliştirilecek bilgi sisteminin dışsal ilişkilerinin göstermek amacıyla kullanılır. Yalnızca bir çember ve gerekli sayıda veri kaynağı ile aralarındaki veri akışlarının belirtimlerinden oluşur.

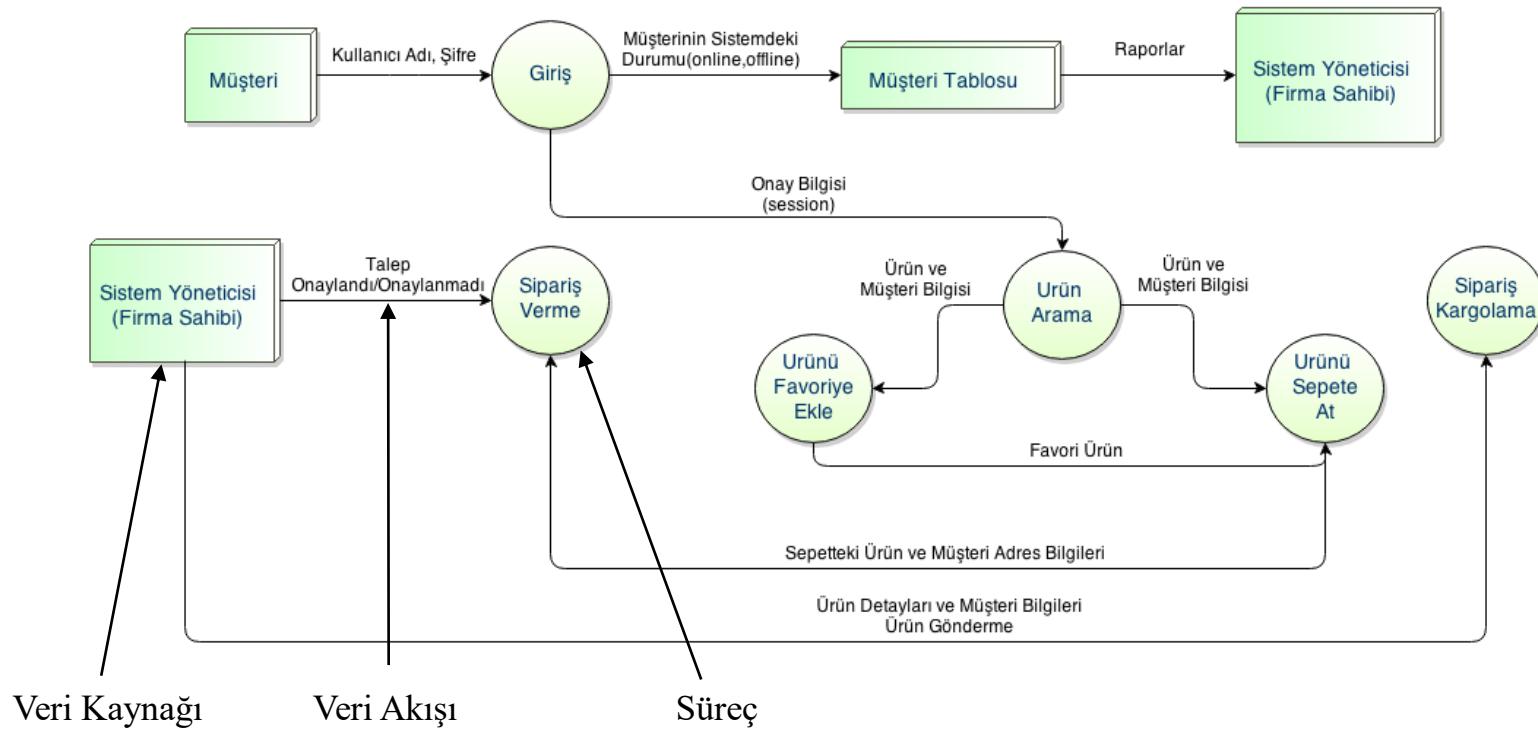
## Genel Bakış Diyagramı

Bir Bilgi sisteminin ana işlevlerini, bu işlevlere ilişkin veri kaynaklarını, veri depolarını ve işlemler arasındaki ilişkileri içeren VAD, Genel Bakış Diyagramı olarak tanımlanmaktadır. Genel Bakış Diyagramı, kapsam diyagramının ayrıstırılmış yada "patlatılmış" biçimidir. Bir bilgi sistemi için, bilgi sisteminin büyüklüğüne göre en çok birkaç Genel Bakış Diyagramı çizilir.

## Ayrıntı Diyagramı

Geliştirilecek bilgi sisteminin dışsal ilişkilerinin göstermek amacıyla kullanılır. Yalnızca bir çember ve gerekli sayıda veri kaynağı ile aralarındaki veri akışlarının belirtimlerinden oluşur.

# Veri Akış Diyagramı



# Veri Akış Diyagramı Neyi Gösterir

---

- Bilgi sisteminin durağan yapısını,
- Bilgi sisteminin süreçlerini ve bu süreçler arasındaki veri akış ilişkisini,
- Bilgi sistemi ile ilişkili olan kurum birimlerini ya da dış birimleri kaynak olarak,
- Bilgi sistemi için gerekli olan ana veri depolarının neler olduğunu ve hangi süreçler tarafından kullanıldığını,
- Bilgi sisteminin süreçlerini yukarıdan-aşağıya ayrıştırma ile gösterir.



# Veri Akış Diyagramı Neyi Göstermez

---

- Bilgi sistemi süreçlerinin zamana ilişkin durumunu ve bu duruma ilişkin bilgileri göstermez.
- Bilgi sistemi süreçlerinin kendi aralarındaki karar ilişkisini göstermez.
- Gerek bilgi sistemi süreçleri, gerekse akışları ve veri kaynakları ve depoları için ayrıntı içermez.



# Süreç Tanımlama Dili

---

- Bilgi sistemi süreçlerinin iç yapılarını belirtmek amacıyla; kullanılan araç, yöntem ya da gösterim biçimleridir.
- Üç farklı yaklaşım izlenir:
  - Düz Metin
  - Şablon
  - Yapısal İngilizce



# Düz Metin

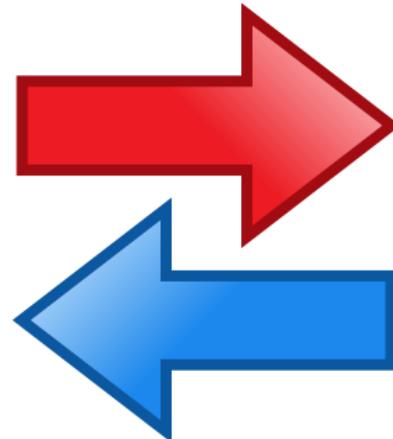
---

- Üçgeni inceler, üçgenin kenar boyutlarını A,B,C) girdi olarak alır.
- Süreç önce bütün bu değerlerin pozitif olup olmadığını denetler.
- Eğer değerlerden biri negatif ise hata verir.
- Süreç tüm kenar uzunlıklarının bir üçgeni belirleyecek şekilde geçerli olup olmadığını denetler.
- Eğer geçerli ise eşkenar, ikizkenar veya çeşitkenar olduğunu belirler.

# Şablon

---

- Süreç : Üçgeni İncele
- Girdi : Üçgenin kenar boyutları
- Çıktı : Üçgen türü, hata iletisi
- İşlem : A,B,C değerlerinin pozitif olup/olmadığını denetle. Negatif ise hata iletisi ver. A,B,C değerlerinin geçerli olup olmadıklarını denetle. Eğer geçerli değerler ise üçgenin türünü belirle (eşkenar, ikizkenar veya çeşitkenar). Değilse hata iletisi ver



# Yapısal İngilizce

---

## ■ Procedure : Üçgeni İncele

Üçgenin kenar boyutlarını oku

If herhangi bir boyut negatif then HATA

If en büyük kenar diğer iki kenar toplamından küçük then

begin

    eşit kenar sayısını belirle

    If 3 kenar eşit then eşkenar

    If 2 kenar eşit then ikiz kenar

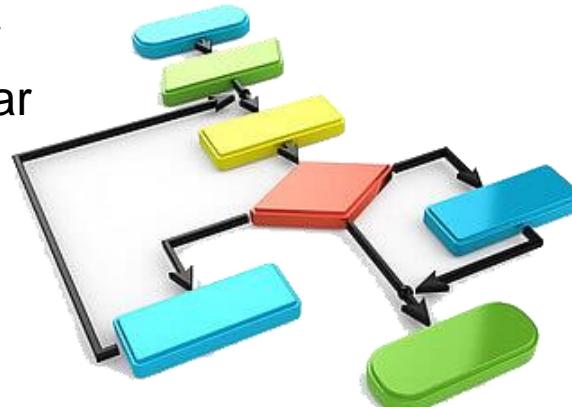
    If 1 kenar eşit çeşitkenar

    Üçgen türünü yaz.

end

else değerler üçgen belirtmiyor

Endproc.



# Kullanıcı Arayüz Prototipleme (KAP)

---

- Ekran tasarımı için kullanıcıdan onay alınması esastır.
- Geleneksel yaklaşımlarda bilgi sistemi girdi ve çıktılarının tanımları el ile kağıt üzerinde yapılır ve kullanıcılarından bu biçimde onay alınmaya çalışılır.
- Gereksinimlerin kesinleştirilmesini kolaylaştırır.



# KAP Özellikleri

---

- Ayrılan zaman sistem analizi için ayrılan zamanın %5'ini aşmamalıdır.
- Her özellik bir kez gösterilmelidir.
- Hiç bir içsel işlem içermemelidir.



# KAP Raporları

---

- Raporların bir kod numarası olmalıdır.
- Her rapor için örnek çıktı yapısı ayarlanır. Word dokümanında örnek yapı hazırlanır. İlgili çıktı gönderilirken bu çıktı gönderilir.



# Sistem Analiz Raporu

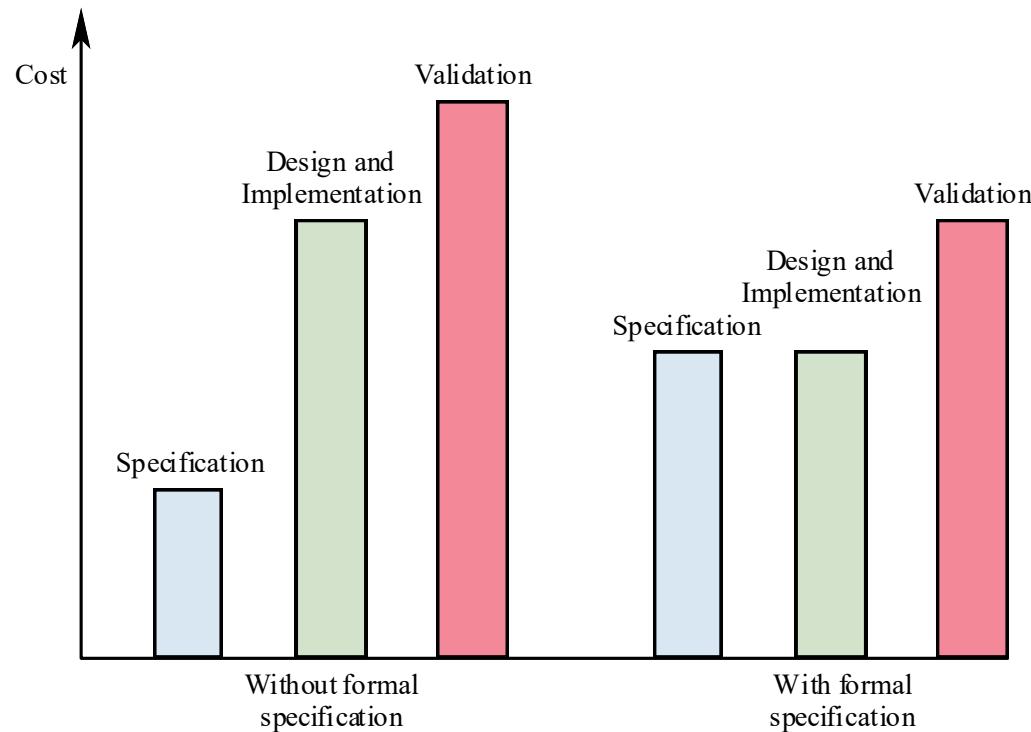
---

- Sistem analiz çalışması sonucunda alınan rapordur (şartname). Söz Konusu rapor çalışmanın tüm ayrıntılarını içerir.
- 5 ana bölümde incelenebilir.
  - Giriş
  - Mevcut sistemin incelenmesi
  - İstenen sistem mantıksal modeli
  - Arayüz gerekleri
  - Belgeleme gerekleri



# Geliştirim Masrafları Karşılaştırması

---



# Özet

---

- **Planlama**, yazılım geliştirme sürecinin ilk aşamasıdır.
- **Proje Kaynakları**: İnsan, donanım ve yazılım olarak 3 kategoride incelenir.
- **Maliyetkestirimi**; bir bilgi sistemi ya da yazılım için gerekebilecek iş gücü ve zaman maliyetlerinin üretimden önce belirlenebilmesi için yapılan işlemlerdir.
- Problemin bilgi ortamının incelenmesinde, **kullanıcı girdileri**, **kullanıcı çıktıları**, **kullanıcı sorguları**, **dosyalar** ve **dışsal ara yüzler** mevcuttur.
- **Proje Sınıfları**: Ayrık, Yarı Gömülü ve Gömülü Projeler olmak üzere 3 şekildedir.
- Maliyet Çarpanı hesaplanması 15 maliyet etmeninin çarpımı sonucunda bulunur.
- COCOMO Modeli kullanılacak ayrıntı düzeyine göre üç ayrı model biçiminde yapılabılır bunlar: Temel, Ara ve Ayrıntı Modelidir.
- Gereksinim, sistemin amaçlarını yerine getirme yeteneği olan bir özellik ya da belirtim olarak tanımlanmaktadır.

# Özet

---

- Gereksinim, kullanıcı ve tasarımcı ya da yazılım mühendisi ile ilgili olarak iki amaca yönelik olacak biçimde tanımlanmalıdır;
- Gereksinim çeşitleri ikiye ayrılır; **İşlevsel** ve **İşlevsel olmayan** gereksinimler.
- Genel olarak 9 tane gereksinim türü bulunmaktadır.
- Gereksinim çözümleme çalışmasında, **mevcut sistem** ve **önerilen sistem** önemli bir gereksinim incelemesidir.
- Gereksinim Verisi Toplanması yöntemi 3 şekilde yapılır. Bunlar; **Sorma**, **psikolojik türetme teknikleri** ve **istatiksel yöntemdir**.
- Veri Modelleme Yöntemleri: **Nesne-ilişki şemaları** ve **veri sözlüğü** olarak iki tiptedir.
- Süreç/İşlem Modelleme Yöntemleri: Veri Akış Diyagramları ve Süreç Tanımlama Dili
- Veri Akış Diyagramı, yapısal sistem geliştirme metodolojilerinde kullanılan tek veri modelleme yöntemidir.
- Veri Akış Diyagramı: **Kapsam**, **Genel Bakış** ve **Ayrıntı diyagramı** olarak 3 çeşittir.

# Sorular

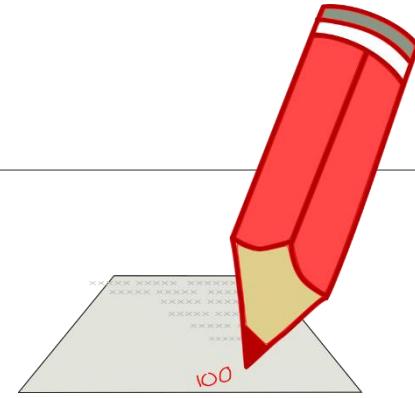
---

1. Kaç çeşit proje kaynağı mevcuttur?
2. Maliyet kestirimi nedir ve ne için yapılmaktadır?
3. Proje sınıfları nelerdir?
4. COCOMO nedir?
5. Yazılımda Sistem çözümlemesi aşamasını açıklayınız.
6. Gereksinim nedir ve ne amaçla yapılmaktadır?
7. Kaç çeşit gereksinim türü vardır ? Bunları açıklayınız.
8. Gereksinim çözümleme aşamasında yapılan incelemeleri söyleyiniz.
9. Gereksinim verisi toplarken hangi yöntemlere başvurulur?
10. Veri modelleme yöntemlerinden Nesne-İlişki şemasını açıklayınız.
11. Veri modelleme yöntemlerinden Veri sözlüğü ne anlama gelmektedir. Örnek veriniz.
12. Süreç modelleme yöntemleri nelerdir?
13. VAD ne anlama gelmektedir?
14. Kaç çeşit VAD diyagramı mevcuttur? Açıklayınız.

# Ödev

---

Günümüzde kullanılan maliyet kestirim yöntemlerini araştırınız.



# Kaynaklar

---

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN



# YMT 312-Yazılım Tasarım ve Mimarisi Yazılım Mühendisliği’ne Giriş

Bölüm-1

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

# Bu Haftaki Konular

Yazılım Nedir?.....	4
Yazılım Mühendisliği.....	15
Yazılımların Sınanması.....	20
Yazılım Maliyetleri.....	22
Yazılım Sistemlerinin Sınıflandırılması.....	23
Yazılımda Kalite.....	27

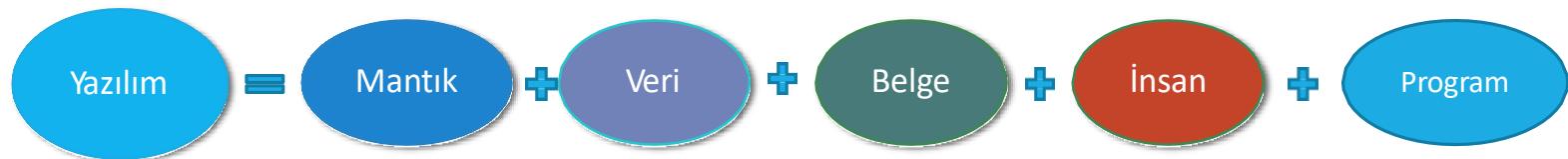
# Amaçlar

---

- Yazılım nedir?
- Yazılım ve Donanım karşılaştırılması?
- Yazılım Mühendisliği nedir?
- Yazılım Mühendisi kime denir?
- Yazılımda Hatalar?
- Yazılım Maliyetleri?
- Yazılımda Kalite?



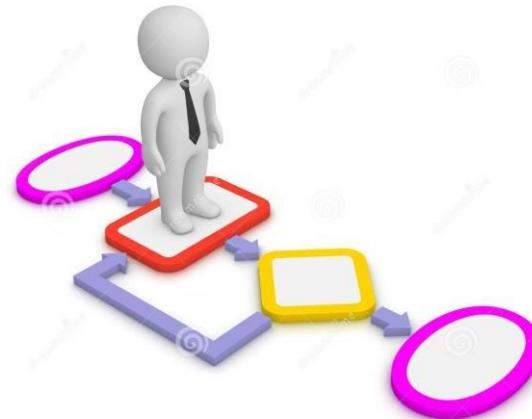
# Yazılım Nedir?



- Bileşenlerinin, belirli bir üretim amacına yönelik olarak bir araya getirilmesi, yönetilebilmesi için kullanılabilecek ve üretilen, yöntem, araç, bilgi ve belgelerin tümünü içerir.
- Yazılım en yalın biçiminde, "**Bir sistemin donanım bileşenleri dışında kalan her şey**" olarak tanımlanabilir.

# Mantık (Algoritma)

- Mantık, bilginin yapısını incelerken, kesin sonuca ulaşmak için doğru ile yanlış arasındaki akıl yürütme ayrimı yapmaktadır.
- Bilgisayarlaştırılmak istenen işin mevcut mantığı yazılıma yansıtılmak durumundadır.
- Bu nedenle mantık (algoritma) bileşeni yazılımın en önemli bileşenlerinden biridir.



# Veri (Bilgi)

- Veri (ing. data), işlenmemiş bilgi veya bilginin ham halidir. Bilgi ise, en basit anlamda verinin işlenmiş şeklidir. Her tür yazılım mutlaka bir veri üzerinde çalışmak durumundadır. Veri dış ortamdan alınabileceğ이 gibi, yazılım içerisinde de üretilebilir.
- Yazılımın temel amacı “veri” yi “bilgi” ye dönüştürmektir.



# Belge (Dokümanlar)

- Yazılım üretimi bir mühendislik disiplini gerektirir. Mühendislik çalışmalarında izlenen yol ya da kullanılan yaklaşımlar yazılım üretimi için de geçerlidir.
- Yazılım üretimi sırasında, birçok aşamada yapılan ara üretimlere ait bilgiler (planlama, analiz, tasarım, gerçekleştirim, vb. bilgileri) belli bir düzende belgelenmelidirler.



# İnsan (Kullanıcı, geliştirici)

- İki boyutludur; yazılımı geliştirenler ve kullananlar.
- Günümüzde artık tek kişi ile yazılım geliştirmekten söz edilmemektedir.
- Yazılım üretimi için bir takım oluşturulmakta ve takımın uyumlu çalışabilmesi için çeşitli yöntemler geliştirilmektedir.

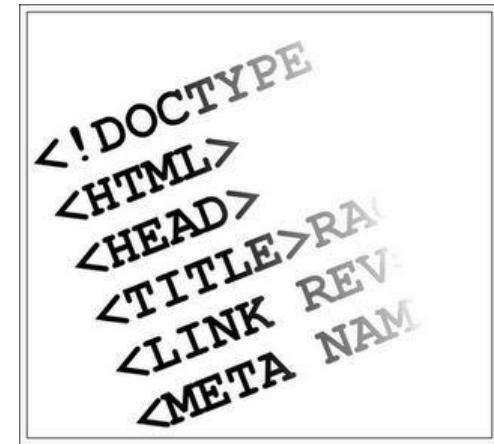


# Program (Kod)

- Yazılımın ana çıktısı sonuçta bir bilgisayar programıdır.
- Program işletime alındıktan sonra bakım çalışmaları sürekli olarak gündeme gelir.

## Bunun iki temel nedeni:

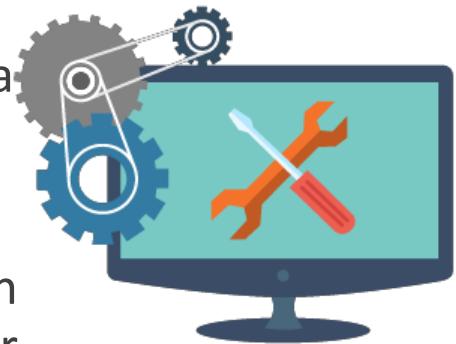
- Hiç bir program bütünüyle her olasılık göz önüne alınarak test edilemez.
- İşletmeler doğaları gereği dinamik bir yapıya sahiptir ve zaman içerisinde sürekli olarak yeni istek ve gereksinimler ortaya çıkabilmektedir.



# Yazılım Donanım Karşılaştırması

---

- Yazılım geliştirilir & donanım üretilir. (fabrika ortamında seri üretim)
- Donanım bileşenleri dışarıdan temin edilebilir, ancak yazılımı oluşturan parçalar için bu çoğu zaman mümkün değildir (günümüzde “yeniden kullanılabilir yazılım” %1-2).



# Yazılım Donanım Karşılaştırması

---

- Yazılım eskimez.
- Oysa, her donanımın belli bir ömrü vardır. Ömrünü tamamlayan donanım yenisi ile değiştirilir.
- Yazılımın eskimesi ortaya çıkabilecek yeni ihtiyaçları karşılayamaması, kullandığı teknolojinin eskimesi olarak tanımlanabilir.
- Yeni gereksinimler yazılıma ekler yaparak yansıtılır.

# Yazılım Donanım Karşılaştırması

---

- Yazılım en az donanım kadar önemlidir.
- Diyaliz makinelerinde kullanılan yazılımların 2000 yılı uyumsuzluğundan ötürü, bir çok diyaliz makinesi çalışmamış ve böbrek hastaları zor durumda kalmıştır.
- Japonya'da telefon yazılımində ortaya çıkan bir yazılım hatası onbinlerce abonenin saatlerce telefon konuşması yapamamasına neden olmuştur.

# Yazılım Donanım Karşılaştırması

---

- Yazılım kopyalama ve donanım kopyalama farklıdır.
- Hata toleransı amacıyla, hayatı olan bir donanımın sistemde bir kopyası daha bulundurulur ve sistemde biri arızalandığında diğerinin çalışmaya devralabilir.
- Oysa, bir yazılımı sistemde iki ayrı bilgisayar üzerine kopyalamak oluşabilecek hatalara çözüm olmayacaktır. Belki, sisteme aynı işi yapan iki farklı eş yazılım yüklenmesi çözüm olabilir (**kritik yazılım sistemleri-uçak avionics**).

# Yazılım Üretim Ortamı

---

- Değişik yetenekte bir çok personel (analist, programcı, test uzmanı, vs.)
- Yazılım çıktısı ile ilgilenen kullanıcılar
- Yeniliğe tepki gösteren kullanıcılar ve yöneticiler !
- Yeterince tanımlanmamış kullanıcı bekłentileri
- Personel değişim oranının yüksekliği
- Yüksek eğitim maliyetleri
- Dışsal ve içsel kısıtlar (zaman, maliyet, işgücü, vs)
- Standart ve yöntem eksiklikleri
- Verimsiz kaynak kullanımı
- Mevcut yazılımlardaki kalitesizlik
- Yüksek üretim maliyeti



# Yazılım Mühendisliği

## [IEEE Tanımı \(1993\)](#)

“Yazılım Mühendisliği:

Sistemli, düzenli, ölçülebilir bir yaklaşımın yazılım geliştirmede, yazılımın işlenilmesinde ve bakımında uygulanmasıdır.

Diger bir deyişle mühendisliğin yazılıma uygulanmasıdır.



# Yazılım Mühendisliği

- Yazılım üretiminin mühendislik yöntemleriyle yapılmasını öngören ve bu yönde;
  - yöntem,
  - araç
  - teknik ve
  - metodolojiler
- üreten bir disiplindir.



# Yazılım Mühendisliği

---

- Yazılım mühendisliği bir yöntemler, teknikler ve araçlar kümesi olarak değerlendirilebilir.
- Yazılım mühendisliğinin hedefi; yazılım üretimindeki karmaşıklıkları gidermektir.
- Geçmişte kullanılan iş akış şemaları gibi yöntemler günümüzde yetersiz kalmaktadır.
- Ayrıca, yazılım üretimi işi tek kişinin başarabileceği boyuttan çok büyük ve bir takım işi biçimine dönüşmüştür.

# Yazılım Mühendisi

---

- Yazılım Mühendisliği İşini yapan kişidir.
- Temel hedefi; üretimin en az maliyet ve en yüksek nitelikte yapılmasını sağlamaktır.
- Programcı değildir. Ancak programcının tüm yeteneklerine sahiptir.
- Yazılımın daha çok mantıksal boyutuyla ilgilenir ve işi insanlarla ilişkisi gerektirir.
- Sistem analisti de değildir. Farkı; analist sadece sistemin analiz aşaması ile ilgilenirken, yazılım mühendisi tüm aşamaların içindedir.

# Yazılım Hataları

---

- Yazılım hataları, yazılım yaşam döngüsünde çok önemli yer tutan unsurlardan biridir.
- Yazılım geliştirme de karşılaşılan en sıkıcı durumdur.
- Hatalar yüzünden yazılım maliyeti artmaktadır.
- Yazılım geliştirme sürecini uzatmaktadır.
- Çözümü erken bulunmayan hatalar bazen uyulması gereken sistemi zor durumda bırakarak zamansal problemler oluşturmaktadır



# Yazılımların Sınaması

- Bir programı tüm ayrıntıları ile test etmek teorik olarak mümkün olmakla birlikte, uygulamada bu mümkün değildir.
- Yazılım ancak sınırlı sayıda veri ile sınanabilir.

Mantıksal Tasarım	%20
İşlevsel Tasarım	%15
Kodlama	%30
Belgeleme ve diğerleri	%35

# Hataların “Yayılma” Özelliği

## ➤ Yazılımda Hata Düzeltme Maliyetleri:

- Yazılım üretimindeki hatalar yayılma özelliği gösterir.
- Bu nedenle, hata düzeltme maliyetleri ilerleyen aşamalarda giderek artar.

<b>Analiz</b>	1
<b>Tasarım</b>	5
<b>Kodlama</b>	10
<b>Test</b>	25
<b>Kabul Testi</b>	50
<b>İşletim</b>	100



# Yazılım Maliyetleri

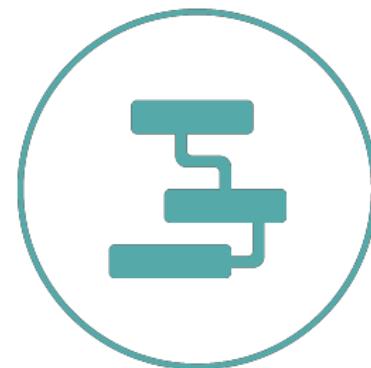
- Günümüzde yazılım maliyetlerindeki artışlar giderek artmaktadır. "Yazılımımızı alırsanız yanında donanımı ücretsiz olarak sağlayacağız" deyişi zaman içerisinde giderek doğrulanmaktadır.
- Örneğin, günümüzde bir kopyası yüz bin dolar dolayında satılan kurumsal kaynak planlama yazılımlarının bulunduğu gözlemlenmektedir.
- Öte yandan bir kişisel bilgisayar ise 1000 ABD dolarının altında satılmaktadır.
- Yazılımın kopyalanma maliyeti ile donanım kopyalama maliyetinin arasındaki farklılık dikkate alındığında, yazılım maliyetlerinin, donanım maliyetlerine oranla oldukça yüksek olduğu ortaya çıkar.



# Yazılım Sistemlerin Sınıflandırılması

---

- İşlevlerine göre sınıflandırma
- Zamana dayalı özelliklere göre sınıflandırma
- Boyuta göre sınıflandırma



# İşlevlerine Göre Sınıflandırma

<b>Hesaplama</b>	Mühendislik Çözümleme
<b>Veri İşleme</b>	Bankacılık
<b>Süreç Temelli</b>	Gömülü Sistemler
<b>Kural Temelli</b>	Robotik, Yapay Zekâ
<b>CAD</b>	Sinyal İşleme

# Zamana Dayalı Özelliklere Göre Sınıflandırma

Toplu (Çevrim-Dışı)	Çevrim-İçi
Gerçek Zamanlı	

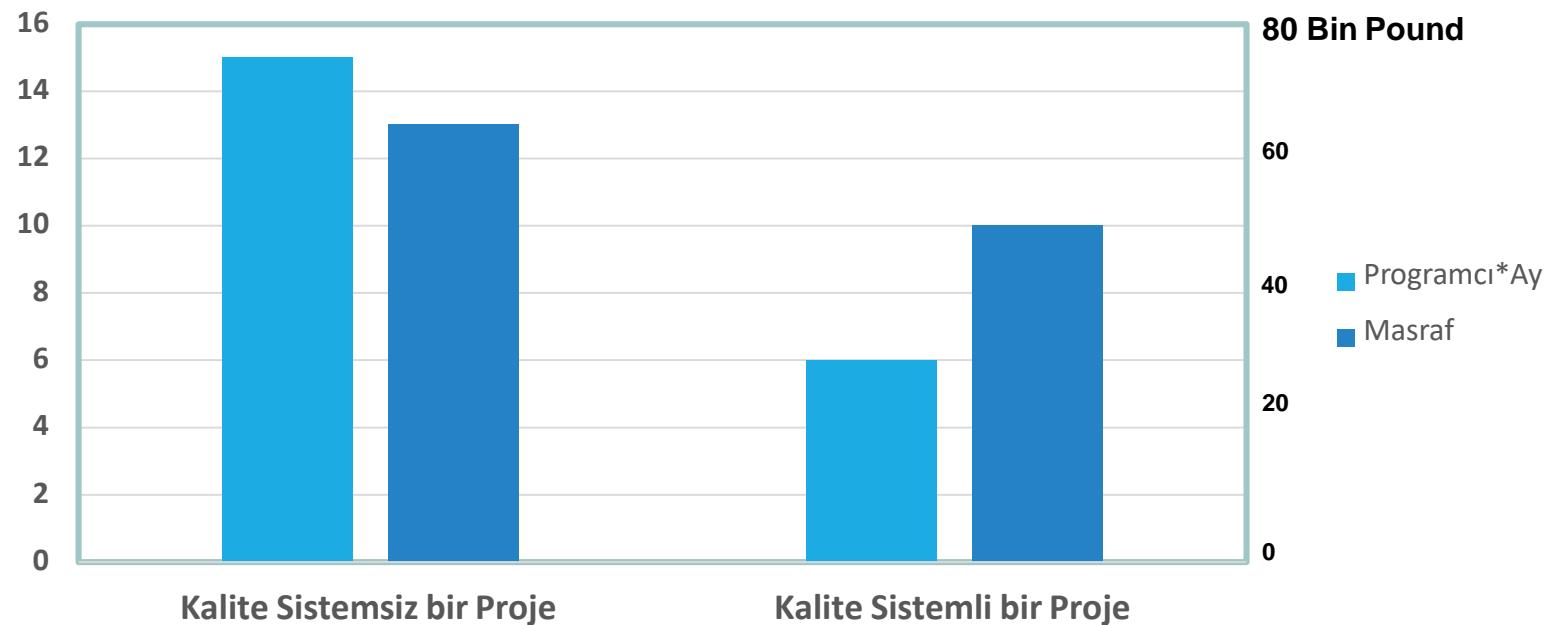


# Boyuta Göre Sınıflandırma

<b>Küçük (SS&lt;2000)</b>	PC Oyunları Öğrenci Projeleri
<b>Orta (2000&lt;SS&lt;100,000)</b>	CAD BDE Yazılımları
<b>Büyük(100,000&lt;SS&lt;1 Milyon)</b>	İşletim Sistemleri
<b>Çok Büyük (SS&gt;1 Milyon)</b>	Komuta Kontrol Sistemleri Hava Tahmini Sistemleri Yıldız Savaşları Sistemleri

# Yazılımda Kalite

---



# Yazılımda Kalite

➤ Üretim Süreci Boyunca ara ürünlere ilişkin kalite standartlarının geliştirilmesi ve geliştirme işlemlerinin bu standartlara uygunluğunun denetlenmesidir.

Yazılım kalite sağlama etkinlikleriyle;

- Yazılım maliyetleri düşürülür,
- Yazılım üretiminin yönetimi kolaylaşır,
- Belgeleme ve standart sorunları giderilir.



# Yazılımda Kalite

---

Ekonomi	Tamlık	Yeniden Kullanılabilirlik	Etkinlik	Bütünlük
Güvenirlilik	Modülerlik	Belgeleme	Kullanılabilirlik	Temizlik
Değiştirilebilirlik	Geçerlik	Esneklik	Genellik	Sınanabilirlik
Taşınabilirlik	Bakılabilirlik	Anlaşılabilirlik	Birlikte Çalışabilirlik	

# Özet

---

- **Yazılım**= Mantık, veri, belge, insan ve program bileşenlerinin, belirli bir üretim amacına yönelik olarak bir araya getirilmesi, yönetilebilmesi için kullanılabilecek ve üretilen, yöntem, araç, bilgi ve belgelerin tümünü içerir.
- **Yazılım Mühendisliği:** Sistemli, düzenli, ölçülebilir bir yaklaşımın yazılım geliştirmede, yazılımın işlenilmesinde ve bakımında uygulanmasıdır. Diğer bir deyişle mühendisliğin yazılıma uygulanmasıdır.
- **Yazılım mühendisliğinin hedefi;** yazılım üretimindeki karmaşıklıkları gidermektir.
- **Yazılım kalite sağlama etkinlikleriyle;**
  - Yazılım maliyetleri düşürülür,
  - Yazılım üretiminin yönetimi kolaylaşır,
  - Belgeleme ve standart sorunları giderilir.

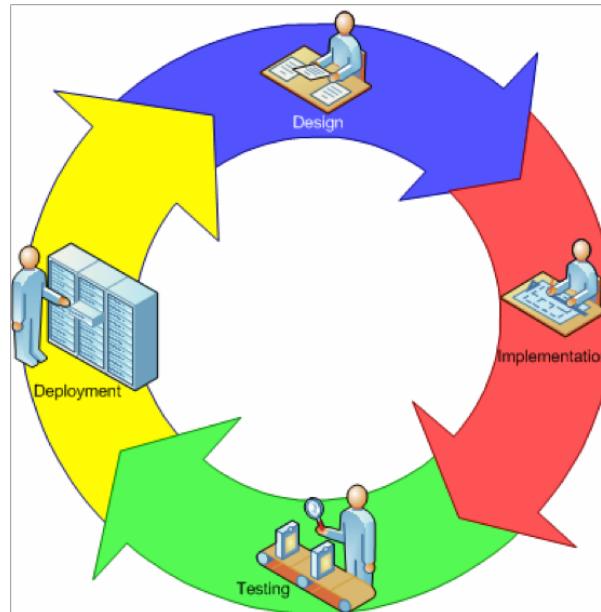
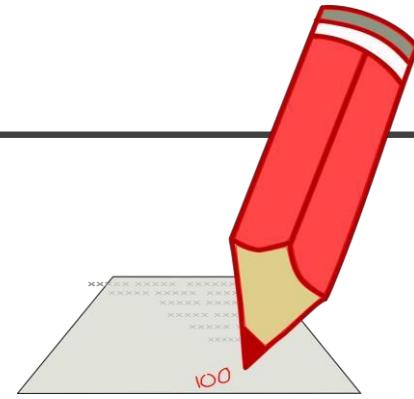
# Sorular

---

1. Yazılım ile program arasındaki farklılığı belirtiniz.
2. Yazılım ile donanım arasındaki farklılıkları belirtiniz.
3. Neden yazılım giderek pahalılaşırken, donanım ucuzlamaktadır?
4. Yazılım mühendisliği ile diğer mühendislik disiplinlerini karşılaştırınız.
5. Bu bölümde verilen yazılım sınıflandırmasını dikkate alarak, her sınıflandırma için bir yazılım örneği veriniz.
6. Yazılım mühendisi ile programcı arasındaki farklılığı belirtiniz.
7. Yazılım Hatalarında Yayıılma neyi ifade etmektedir?
8. Yazılım Maliyetinde hangi Sınıflandırma ağır basmaktadır?
9. İşlevlerine Göre Sınıflandırma da Kural Temelli 'ye Örnek veriniz.
10. Yazılım Kalitesini Tanımlayınız.

# Ödev

1. Yazılım Yaşam Döngüsünü Araştırınız.
2. Yazılım Süreç Modellerini Araştırınız.



# Kaynaklar

---

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

<http://blog.alisuleymantopuz.com/2014/08/30/yazilim-mimarisi-ve-tasarimi-nedir/> \*

<http://www.akifsahman.com/?p=175>

<https://ece.uwaterloo.ca/~se464/08ST/index.php?src=lecture> \* <http://info.psu.edu.sa/psu/cis/azarrad/se505.htm>

<http://www.metinakbulut.com/YAZILIM-MIMARISI/> \* [http://ceng.gazi.edu.tr/~hkaracan/source/YPY\\_H3.pdf](http://ceng.gazi.edu.tr/~hkaracan/source/YPY_H3.pdf)

<http://iiscs.wssu.edu/drupal/node/3399> \* <http://www.cs.toronto.edu/~sme/CSC340F/slides/21-architecture.pdf>

<http://www.users.abo.fi/lpetre/SA10/> \* <http://sulc3.com/model.html>

<http://salyangoz.com.tr/blog/2013/11/23/digerleri/yazilim-gelistirme-surec-modelleri-3/>



# YMT 312-Yazılım Tasarım Ve Mimarisi Yazılım Tasarımı

Fırat Üniversitesi Yazılım Mühendisliği Bölümü



# Bu Haftaki Konular

Yazılım Tasarımının Önemi.....	4
Tasarım Kavramları.....	8
Yapısal Tasarım.....	16
Tasarlanması Gereken Ortak Alt Sistemler.....	29
Kullanıcı Arayüz Tasarımı.....	36
Tasarım Kalite Ölçütleri.....	43
Yapışıklık.....	50

# Amaçlar

- Tasarımın ne olduğunu ve çeşitli tasarım türlerinin ürünün farklı yönleriyle nasıl ilgilendiğini açıklamak
- Tasarımı bir problem çözme etkinliği olarak sunmak, soyutlama ve modellemenin tasarımdaki rolünü ortaya koymak
- Yazılım yaşam döngüsünde tasarımın yerini belirlemek
- Yazılım mühendisliğinde tasarım metodlarını incelemek



# Yazılım Tasarımının Önemi

- Tasarlanmış (designed) bir dünyada yaşıyoruz.
- Tasarım ekonomik olarak öneme sahiptir ve yaşam kalitemizi doğrudan etkiler.
- Yazılım son derece yaygın hale gelmektedir.
- Yazılım tasarımının kalitesinin önemli sonuçları olmaktadır ve yazılım tasarımcıları bunların farkında olmalı, bunları ciddiye almalıdır.

# Giriş

- Tasarım, Sistem Analizi çalışması sonucunda üretilen Mantıksal Modelin Fiziksel Modele dönüştürülme çalışmasıdır.
- Fiziksel Model geliştirilecek yazılımın;
  - hangi parçalardan oluşacağını,
  - bu parçalar arasındaki ilişkilerin neler olacağını,
  - parçaların iç yapısının ayrıntılarını,
  - gerekecek veri yapısının fiziksel biçimini (dosya, veri tabanı, hash tablosu, vektör, vs.)tasarımını içerir.

# Yazılım Ürünleri

Bir yazılım ürünü, müşterinin gereksinim ve isteklerini karşılayan bir veya daha fazla programdan, verilerden, ve destekleyici materyal ve hizmetlerden oluşan bir varlıktır. Bu ürün, tek başına bir ürün olabileceği gibi başka bir ürünün temel bileşeni de olabilir.

# Yazılım Tasarımı Nedir?

- Yazılım tasarımcıları da temelde diğer disiplinlerdeki tasarımcıların yaptığı işi yapar.
- Tasarlanan şey bir yazılım ürünüdür.

**Yazılım tasarımı**, müşterinin gereksinim ve isteklerini karşılayan yazılım ürününün doğasını ve bileşimini belirleme etkinliğidir.

# Tasarım Kavramları

---

## Soyutlama (abstraction):

- **Soyutlama (abstraction):** Detayları gizleyerek yukarıdan bakabilme imkanı sağlanır.



Soyutlama kavramı veri, işlev ve yapısal açılar için geçerlidir. Örneğin bir kapı nesne olarak ele alındığında onun kulpu, rengi menteşeleri, malzemesi gibi detayları düşünmeden kapıyı bir ev mimarisi içinde değerlendirebiliriz. Aksi taktirde diğer detaylara yoğunlaşan bir tasarımcı 'oda' düzeyinde görsel canlandırmalara hakim olamaz.

# Tasarım Kavramları

## İyileştirme (enhancement):

- **İyileştirme (enhancement):** Soyutlama düzeyinde irdeleme bittikten sonra, daha alt seviyelere inilerek tanımlamalarda ayrıntı, bazen de düzeltme yapılarak tasarımın daha kesinlik kazanması sağlanır.

Kulp	Menteşe
Renk	Malzeme

Soyutlama kavramı veri, işlev ve yapısal açılar için geçerlidir. Örneğin bir kapı nesne olarak ele alındığında onun kulpu, rengi menteşeleri, malzemesi gibi detayları düşünmeden kapıyı bir ev mimarisi içinde değerlendirebiliriz. Aksi takdirde diğer detaylara yoğunlaşan bir tasarımcı 'oda' düzeyinde görsel canlandırmalara hakim olamaz.

# Tasarım Kavramları

## Modülerlik (modularity):

- **Modülerlik (modularity):** Sistemi istenen kalite faktörleri ışığında parçalara ayrıştırma sonucu elde edilir. Bir işlev için sistemin tümü değil, ayrılmış bir kısmı üzerinde çalışma yapabilme olanağı sağlar.

Yuvarlak Köşeli Uzun Kısa ...	Yay Dil Vidalar ...
Ahşap Metal Cam ...	Beyaz Metalik Kahverengi ...

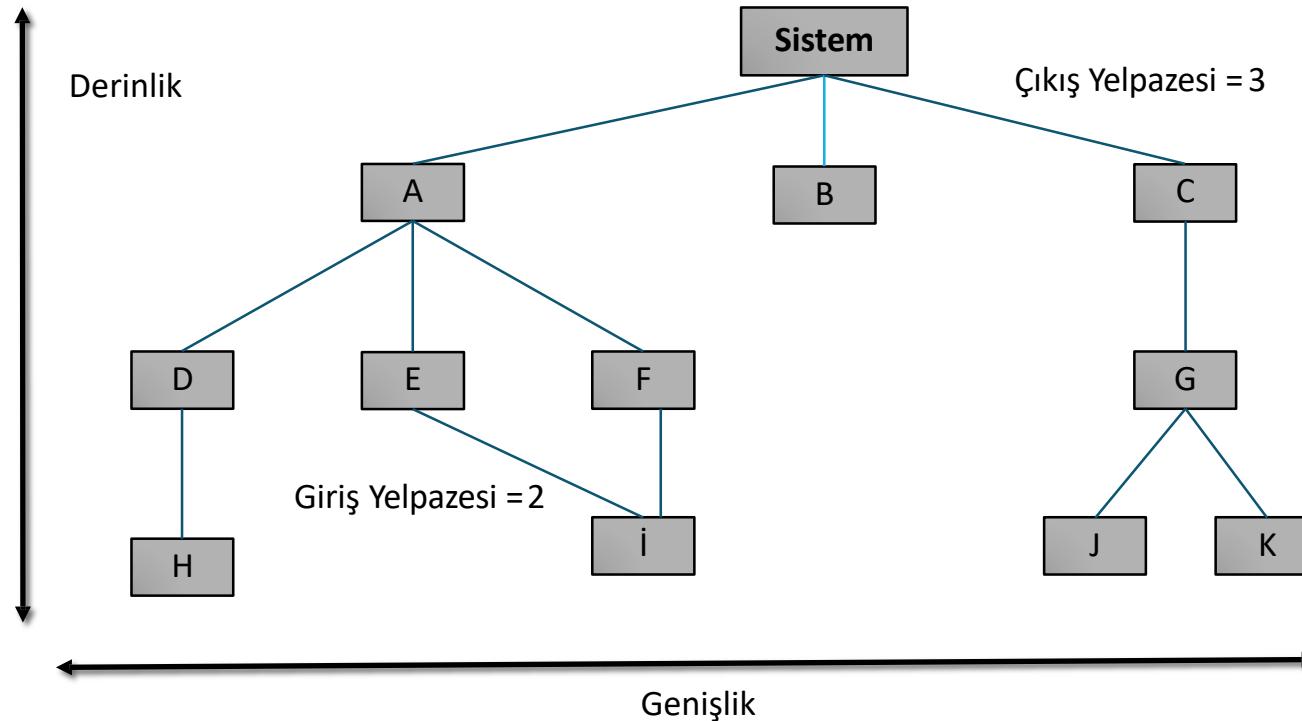
Kapı ve pencerenin de kendi ayrıntılarını, birer kelime ile soyutladığımız isimleri içerisinde saklamaları, onları birer neşene olarak bir oda içerisinde ‘modüler’ bir yapı düzende olabilmelerini sağlar. Bir pencerenin yerini değiştirmek, tasarım esnasında onun camı, menteşesi ve malzemesi gibi detayından bağımsız, aynı zamanda da odadaki diğer ‘modüllerden’ hemen hemen bağımsız olarak ele alınabilir.

# Modülerlik

- Bütün karmaşıklığın tek bir modülde toplanması yerine, anlaşılabilir ve dolayısıyla projenin zihinsel kontrol altında tutulması için sistem bir çok module ayrıılır.
- Modüller, isimleri olan tanımlanmış işlevleri bulunan ve hedef sistemi gerçekleştirmek üzere tümleştirilen birimlerdir.



# Sistem ve Modülleri



# İşlevsel Bağımsızlık

---

- Modüllere parametre ile veri gönderilir ve sonuç değer alınır. Bu modülü çağıran program parçası sadece bu sonucu kullanabilir. Çağrılan modülün işlevsel olarak yaptıkları ile ilgili değildir.



# Veri Tasarımı

- Yapı Tasarımı, arayüz tasarımı ve süreç tasarımından önce yapılması gereken ilk tasarım veri tasarımidır.
- Bilgi saklama ve soyutlama bu işlem için önemli kavamlardır.



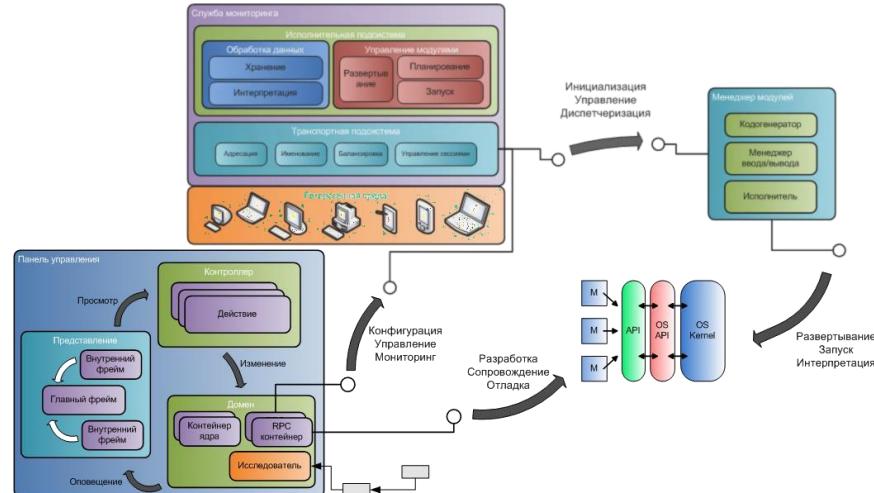
# Veri Tasarımında Dikkat Edilecek Konular

---

- Değişik veri yapıları değerlendirilmelidir.
- Bütün veri yapıları ve bunlar üzerinde yapılacak işlemler tanımlanmalıdır.
- Alt düzeyde tasarım kararları tasarım süreci içerisinde geciktirilmelidir.
- Bazı çok kullanılan veri yapıları için bir kütüphane oluşturulmalıdır.
- Kullanılacak programlama dili soyut veri tiplerini desteklemelidir.

# Yapısal Tasarım

- Yapısal Tasarımın ana hedefi modüler bir yapı geliştirip modüller arasındaki kontrol ilişkilerini temsil etmektir.
- Ayrıca yapısal tasarım bazen de veri akışlarını gösteren biçimde dönüştürülebilir.
- Veri Akışları Üç parça da incelenebilir
  - Girdi Akışı
  - Çıktı Akışı
  - İşlem Akışı



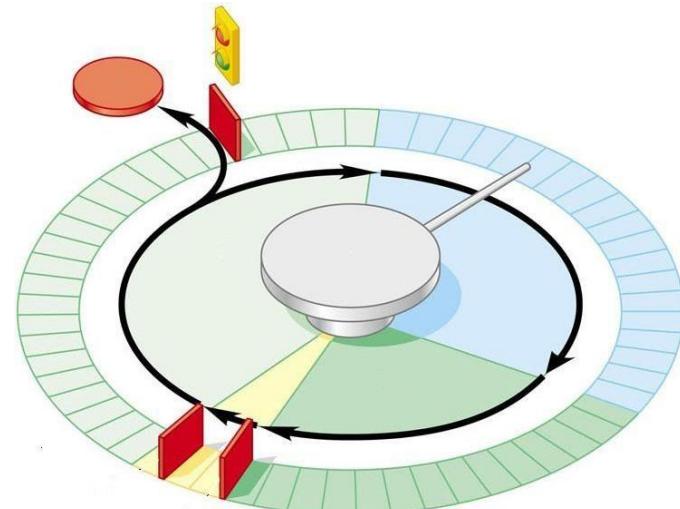
# Ayrıntı Tasarım- Süreç Tasarımı

- Süreç tasarımları; veri, yapı ve ara yüz tasarımlarından sonra yapılır.
- İdeal şartlarda bütün algoritmik detayın belirtilmesi amaçlanır.
- Ayrıca süreç belirtiminin tek anlamı olması gereklidir, değişik şahıslar tarafından farklı yorumlanmamalıdır.
- Doğal diller kullanılabilir (açıklamalarda, çünkü doğal dil tek anlamlı değildir)
- Süreç Tanımlama Dili (PDL)



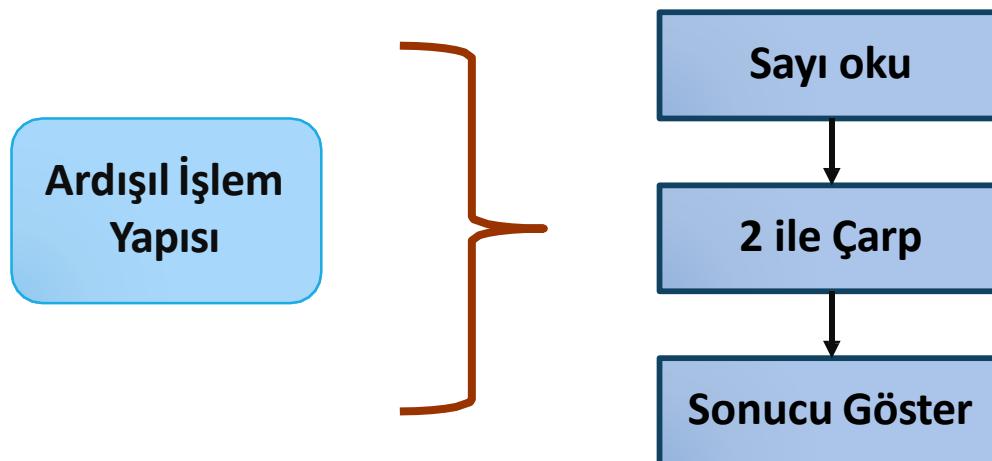
# Yapısal Program Yapıları

- Yapısal programlamanın temel amacı;
  - program karmaşıklığını en aza indirmek,
  - program anlaşılabilirliğini artırmaktır.
- Bu amaçla şu yapıları kullanılır;
  - Ardışıl İşlem yapısı
  - Koşullu işlem yapısı
  - Döngü yapısı
- GOTO kullanımı uygun değildir.



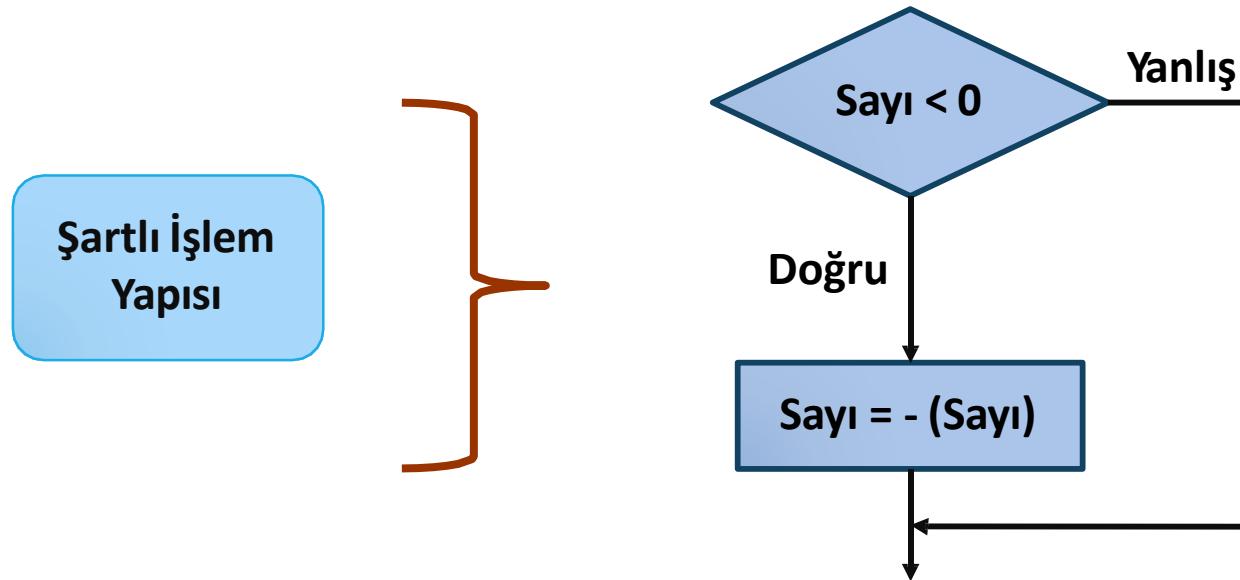
# Program Akış Diyagramı Yapıları

---

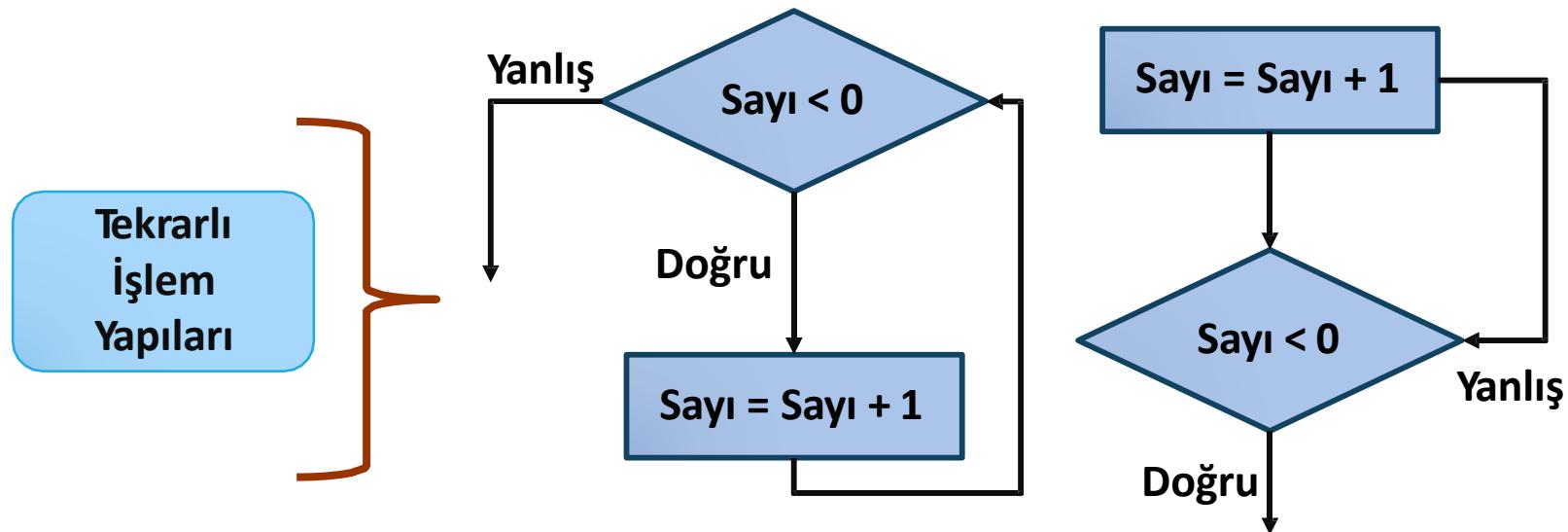


# Program Akış Diyagramı Yapıları

---



# Program Akış Diyagramı Yapıları

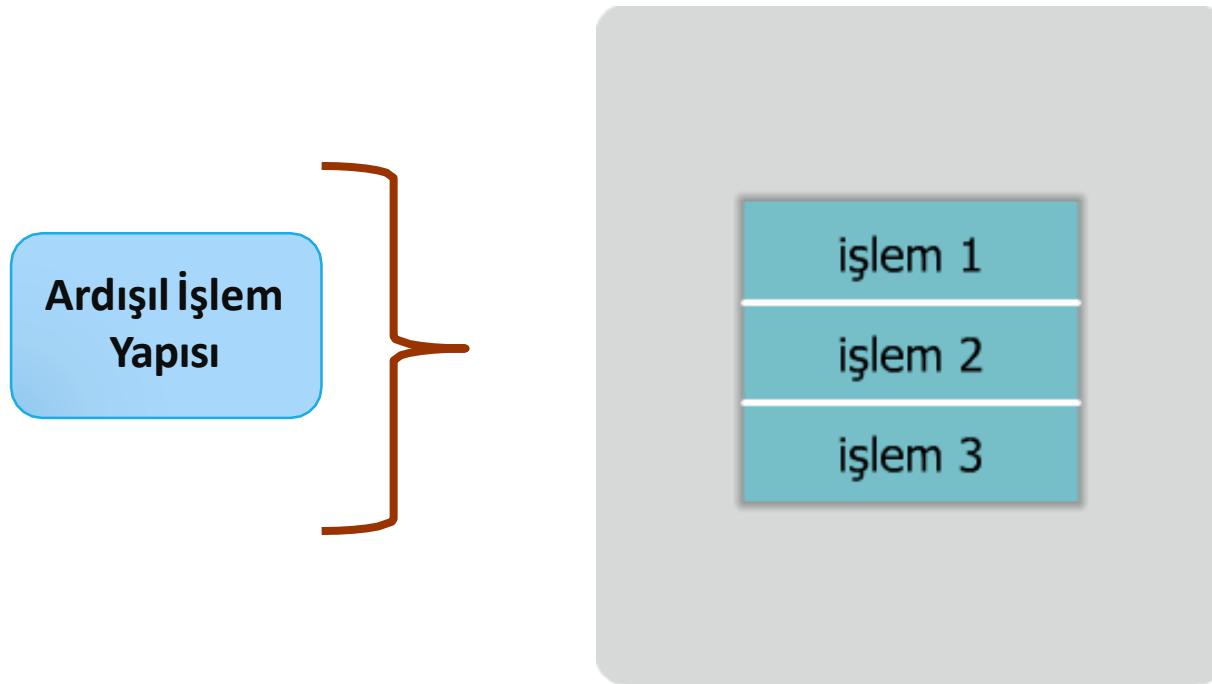


# Program Akış Diyagramları

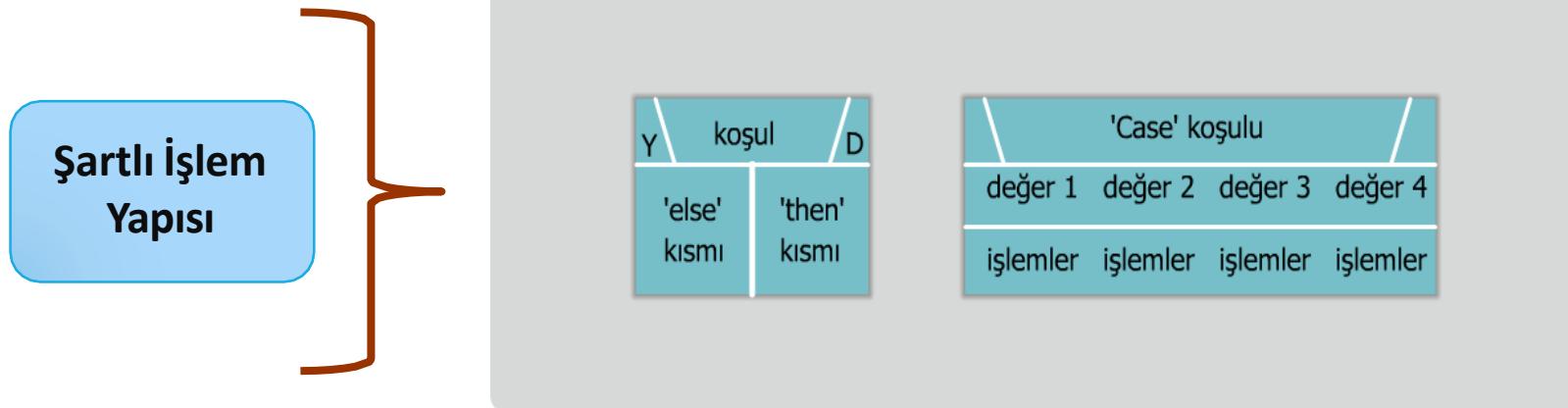


# Kutu Diyagramları

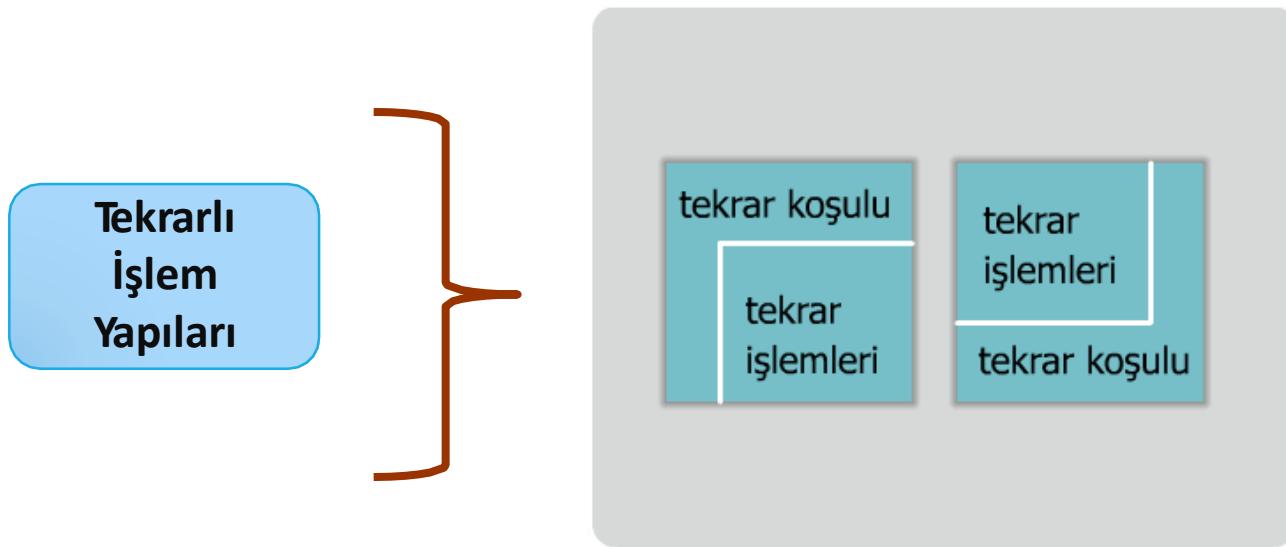
---



# Kutu Diyagramları

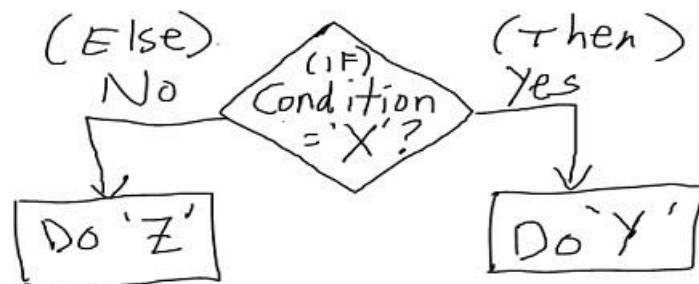


# Kutu Diyagramları



# Karar Tabloları

- Bazen karmaşık koşul değerlendirmeleri yapmak gereklidir. Bunların düzenli bir gösterilimi karar tablolarında yapılabilir.
- Öncelikle, bütün işlemler saptanmalı, sonra ön koşullar belirlenmelidir.
- Belirli işlemler ile belirli koşulları birleştirerek tablo oluşturulur.
- Alt tarafta ise işlemler benzer satırlar olarak gösterilir.



# Karar Tabloları

---

Kurallar	1	2	3	4
Hesap Geçerli	✓	✓	✓	✓
Şifre Geçerli	✓		✓	
Yeterli Nakit Var	✓		✓	✓
Hesapta Yeterli Bakiye Var	✓			✓
Bakiye Bildirme İşlemi	✓			
Para Çekme İşlemi		✓		
Para Yatırma İşlemi			✓	
Para Gönderme İşlemi				

# Program Tasarım Dili

- Program Tasarım Dilleri süreç belirtiminde doğal dillerin programlama dili ile sentezlenmesi şeklinde ortaya çıkmıştır.
- Hangi programlama dilinin kullanılacağından bağımsız özellikler bulunmalıdır.

DO

Hesap Numarasını Oku

IF (hesap numarası geçerli değil) başlangıça dön  
işlem türünü iste

IF (para yatırma işlemi) { para\_yatir(); Başlangıça dön}

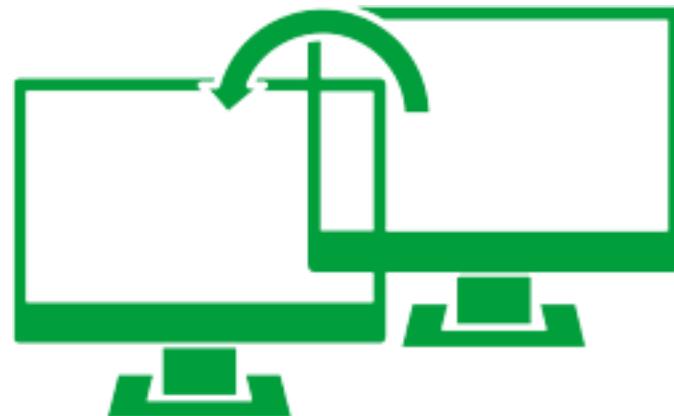
IF (yeterli bakiye yok) başlangıça dön

WHILE

# Tasarlanması Gereken Ortak Alt Sistemler

---

- Yetkilendirme altsistemi
- Güvenlik altsistemi
- Yedekleme altsistemi
- Veri transferi altsistemi
- Arşiv altsistemi
- Dönüştürme altsistemi



# Yetkilendirme Alt Sistemi

- Özellikle kurumsal uygulamalarda farklı kullanıcıların kullanabilecekleri ve kullanamayacakları özellikleri ifade eder.
  - İşlev bazında yetkilendirme
  - Ekran bazında yetkilendirme
  - Ekran alanları bazında yetkilendirme
- Oracle veri tabanına erişim konusunda yetkilendirme yapmaktadır.



# Güvenlik Alt Sistemi

---

- Güvenlik alt sistemi, bilgi sisteminde yapılan işlerin ve yapan kullanıcıların izlerinin saklanması ve gereken durumlarda sunulması ile ilgilidir.
- Bir çok yazılım geliştirme ortamı ve işletim sistemi, bu amaca yönelik olarak, "**sistem günlüğü**" olanakları sağlamaktadır.
- Sistem günlüğü ile sunulanın olanaklar yeterli olmadığı durumlarda ek yazılımlar geliştirilmesi gerekmektedir.
- LOG files (Sistem günlüğü)

# Yedekleme Alt Sistemi

- Her bilgi sisteminin olağanüstü durumlara hazırlıklı olmak amacıyla kullandıkları veri tabanı (sistem) yedekleme ve yedekten geri alma işlemlerinin olması gerekmektedir.
- Günümüzde tüm veri tabanı yönetim sistemi geliştirme platformları, oldukça zengin yedekleme ve yedekten geri alma olanakları sağlamaktadır.
- Bu konuda, tasarım bağlamında yapılması gereken, yedekleme işleminin düzenlenmesini tasarlamak olmalıdır.
- Yedeklemenin hangi sıklıkla yapılacağı, ne zaman, elle ya da otomatik olarak yapılip yapılmayacağı gibi planlamalar, tasarım aşamasında yapılmalıdır.

# Veri İletişim Alt Sistemi

- Coğrafi olarak dağıtılmış hizmet birimlerinde çalışan makineler arasında veri akışının sağlanması işlemleri
- **Çevirim içi veri传递 (real-time):** Verinin bir birimden diğerine anında iletilmesi olarak tanımlanır.
  - Bu tür veri传递, gerçek zamanlı sistemler için oldukça önemlidir.
  - Bilgi sistemi uygulamalarında, - zamansal kritiklik, gerçek zamanlı uygulamalara oranla daha az olduğu için - bu tür传递 çok yaygın değildir.
- **Çevirim dışı veri传递 (disketler, teypler):** Bilgilerin传递 hatları kanalıyla değil, çevrim dışı ortamlar (teyp, disket, cd vb.) aracılığı ile传递mesi çevrim dışı传递 olarak tanımlanmaktadır.
  - Kısacası, "kargo" olarak tanımlanan传递inin, el ile yapılan türüdür.
  - Ağ传递inin sağlanamadığı durumlarda kullanılan bir yöntemdir.
  - Kullanımı giderek azalmaktadır.

# Arşiv Alt Sistemi

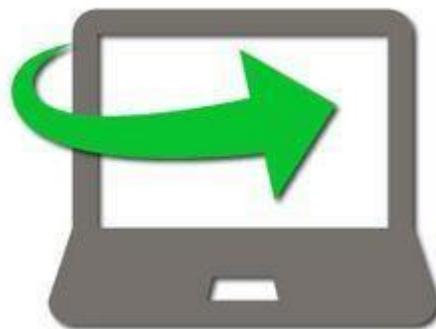
---

- Belirli bir süre sonrasında sık olarak kullanılmayacak olan bilgilerin ayrıılması ve gerektiğinde bu bilgilere erişimi sağlayan alt sistemlerdir.
- Örneğin, insan kaynakları yönetimi bilgi sisteminde, emekli olan bir kişiye ilişkin bilgilerin, çevrim-içi olarak tutulan veri tabanından alınarak, çevrim dışı bir ortama alınması ve aradan örneğin beş yıl geçtikten sonra, pasaport işlemleri için gerek duyulabilecek kişi bilgilerine erişilmesini sağlayan işlemler arşiv alt sistemleri tarafından gerçekleştirilmektedir.
- İşlem türü olarak ortak bir çok özellik içeren arşiv alt sistemleri, uygulama bazında az da olsa farklılaşabilmektedir.
- Aktif veri tabanı.

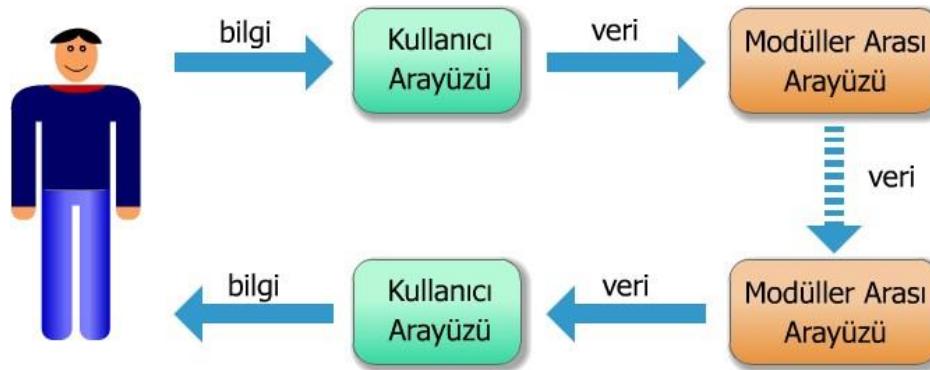
# Dönüştürme Alt Sistemi

---

- Geliştirilen bilgi sisteminin uygulamaya alınmadan önce veri dönüştürme (mevcut sistemdeki verilerin yeni bilgi sistemine aktarılması) işlemlerine ihtiyaç vardır.
- Mevcut uygulamalardaki bilgisayar ortamında saklanan bilgilerin ortam çeşitliliği, dönüştürme işlemlerini zorlaştırır.



# Kullanıcı Arayüz Tasarımı



- Kullanıcı ile ilişkisi olmayan arayüzler
  - Modüller arası arayüz
  - Sistem ile dış nesneler arası arayüz
  
- Kullanıcı arayüzleri
  - Kullanım kolaylığı ve öğrenim zamanı esastır.
  - Program = arayüz yaklaşımı vardır.

# Genel Prensipler

- Komut seçimi, veri giriş formlarının şekli gibi bir çok konuda tutarlı bir yapı izlenmedir.
- Önemli silmelerde teyit alınmalıdır.
- Yapılan çoğu işlem kolayca geri alınabilmelidir
- İşlemler arasında ezbere tutacak bilgi miktarı azaltılmalıdır.
- Kullanıcı hareketleri, düşünme ve algılamasında verimlilik sağlanmalıdır.
- Hataların affedilmesi, yanlış giriş olduğunda program korunmalı ve düzeltme şansı verilmelidir.
- İşlemleri sınıflandırıp ekran geometrisi buna uygun olarak kullanılmalıdır.
- Komut isimleri kısa ve basit olmalıdır.
- Menülerin ve diğer etkileşimli araçların standart yapıda tasarlanmalıdır.

# Bilgi Gösterimi

- Yalnızca içinde bulunulan konu çerçevesi ile ilgili bilgi gösterilmeli
- Veri çokluğu ile kullanıcı bunaltılmamalı, grafik ve resimler kullanılmalı
- Tutarlı başlık, renkleme ve kısaltma kullanılmalı
- Hata mesajları açıklayıcı ve anlaşılır olmalı
- Değişik tür bilgiler kendi içinde sınıflandırılmalı
- Rakamsal ifadelerde analog görüntü verilmeli (%89 değil)



# Veri Girişi

---

- Kullanıcı hareketleri en aza indirilmelidir. Yazma yerine ekranın listelerden seçme, bir komuta en az sayıda fare tıklamasıyla erişme gibi.
- Gösterim ve girdi sahaları birbirinden ayırt edecek biçimler (renk, büyülüklük, yerleşim vb.) tutarlı olarak kullanılmalıdır.
- Kullanıcı uyarlamasına izin verilmelidir: kullanıcı bazı özellikleri tanımlayabilir, bazı uyarı mesajlarını istemeyebilir.
- Kullanılan konu ile ilgili gereksiz komutlar geçici olarak etkisizleştirilmelidir.
- Bütün girdiler için yardım kolaylıklarını olmalıdır.



# Kullanıcı Arayüz Prototipi

- Tasarım çalışması sonucunda, daha önceden gereksinim çalışması sırasında hazırlanmış olan kullanıcı arayüz prototipi, ekran ve rapor tasarımları biçimine dönüşür. Ekranlar son halini alır, raporlar kesinleşir. Kullanıcıya gösterilerek onay alınır.
- Tüm programın tek elden çıktıının ifade edilebilmesi açısından tüm ekranların aynı şablon üzerine oturtulması önerilmektedir.
  - Menü Çubuğu
  - Araç Çubuğu
  - **Gövde (Değişebilir)**
  - Durum Çubuğu

# Başlangıç Tasarım Gözden Geçirme

---

- Yapılan tasarım çalışmasının bir önceki geliştirme aşaması olan analiz aşamasında belirlenen gereksinimleri karşılayıp karşılamadığının belirlenmesidir.
  - Sistem gereksinimlerine yardımcı olan kullanıcılar
  - Sistem analizini yapan çözümleyiciler
  - Sistemin kullanıcıları
  - Tasarımcılar
  - Yönlendirici
  - Sekreter
  - Sistemi geliştirecek programcılardan oluşan bir grup tarafından yapılır.

# Ayrıntılı Tasarım Gözden Geçirme

---

- Başlangıç tasarımları gözden geçirme çalışmasının başarılı bir biçimde tamamlanmasından sonra, tasarımın teknik uygunluğunu belirlemek için **Ayrıntılı Tasarım Gözden Geçirme** çalışması yapılır. Bu çalışmada;
  - Çözümleyiciler
  - Sistem Tasarımcıları
  - Sistem Geliştiriciler
  - Sekreterden oluşan bir ekip kullanılır.



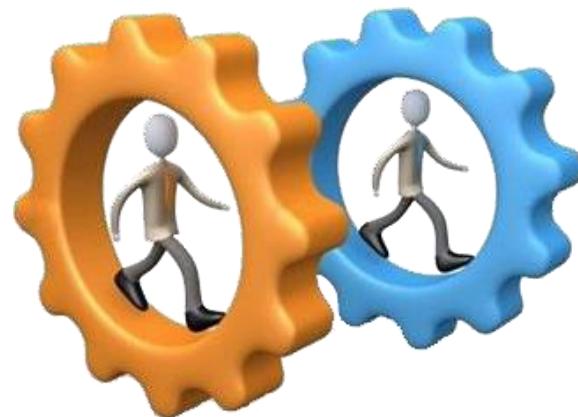
# Tasarım Kalite Ölçütleri

- Bağlaşım (Coupling)

Tasarımı oluşturan modüller arası ilişki ile ilgiliidir.

- Yapışıklık (Cohesion)

Modüllerin iç yapısı ile ilgiliidir.



# Bağlaşım

---

- Modüller arası bağılılığın ölçülmesi için kullanılan bir ölçütür.
- Yüksek kaliteli bir tasarımda bağlaşım ölçümü az olmalıdır.
- Bağlaşımın düşük olması
  - Hatanın dalgasal yayılma özelliğinin azaltılması
  - Modüllerin bakım kolaylığı
  - Modüller arası ilişkilerde karmaşıklığın azaltılmasınedenleri ile istenmektedir

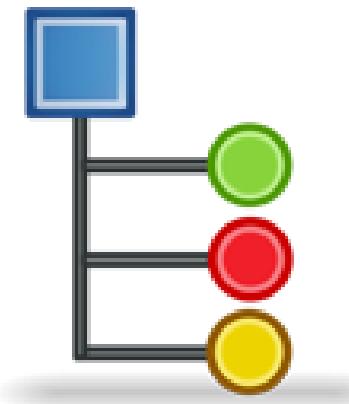
# Yalın Veri Bağlaşımı

- Herhangi iki modül arası iletişim yalın veriler (tamsayı, karakter, boolean, vs) aracılığı ile gerçekleştiriliyorsa bu iki modül yalın veri bağılılığıdır şeklinde tanımlanır.



# Karmaşık Veri Bağlaşımı

- Herhangi iki modül arasındaki iletişimde kullanılan parametrelerin karmaşık veri yapısı ([kayıt](#), [dizi](#), [nesne](#), [vs](#)) olması durumunda modüller karmaşık veri paylaşımı olarak tanımlanır.



# Denetim Bağlaşımı

- İki Modül arasında iletişim parametresi olarak **denetim verisi** kullanılıyorsa bu iki modül denetim bağlısımlı olarak tanımlanır.



# Ortak Veri Bağlaşımı

- Eğer iki modül ortak bir alanda tanımlanmış verilere ulaşabiliyorsa bu iki modül **ortak veri bağımlı** olarak tanımlanır.
- Verilerin ortak veri bağımlı olmaları şu nedenlerden dolayı fazla istenmez;
  - Ortak veri alanını izlemek zordur.
  - Ortak veri kullanan modüllerde yapılan değişiklikler diğer modüller etkiler.
  - Ortak veri üzerinde yapılacak değişikliklerde bu veriyi kullanacak bütün modüller göz önüne alınmalıdır.

# İçerik Bağlaşımı

---

- Modüllerin iç içe tasarlanması sonucu, bir modülün başka bir modül içerisinde tanımlanmış veri alanına erişebilmesi olanaklaşır ve bu durum **İçerik bağlaşımına** yol açar.



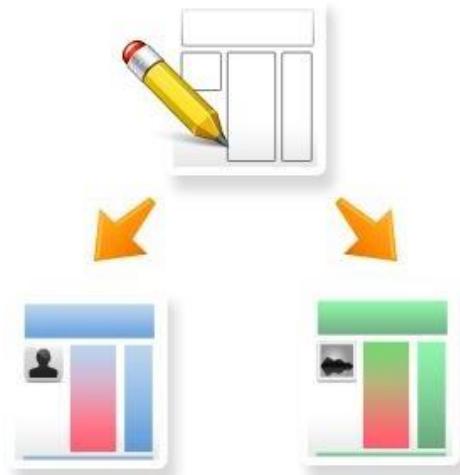
# Yapışıklık

---

- Bir modülün kendi içindeki işlemler arasındaki ilişkilere ilişkin bir ölçütür. **Modül gücü** olarak da tanımlanır.
- Tasarımda **yapışıklık** özelliğinin yüksek olması tercih edilir.
- Yapışıklık ile Bağlaşım ters orantılıdır.

# İşlevsel Yapışıklık

- İşlevsel Yapışık bir modül, tek bir iş problemine ilişkin sorunu çözen modül olarak tanımlanır.
- Maas\_Hesapla, Alan\_Hesapla gibi



# Sırasal Yapışıklık

- Bir modülün içindeki işlemler incelediğinde, bir işlemin çıktısı, diğer bir işlemin girdisi olarak kullanılıyorsa bu modül **sırasal yapışık** bir modül olarak adlandırılır.

Ham\_Veri\_Kaydını\_Düzelt

Duzeltlimis\_Ham\_Veri\_Kaydini\_Dogrula

Dogrulanmis\_Kaydi\_Gonder

# İletişimsel Yapışıklık

- Bir modülün içindeki farklı işlemler aynı girdi ya da çıktıyı kullanıyorlarsa bu modül **İletişimsel yapışık** bir modül olarak adlandırılır.

Sicil\_No\_yu\_Ai

Adres\_Bilgisini\_Bul

Telefon\_Bilgisini\_Bul

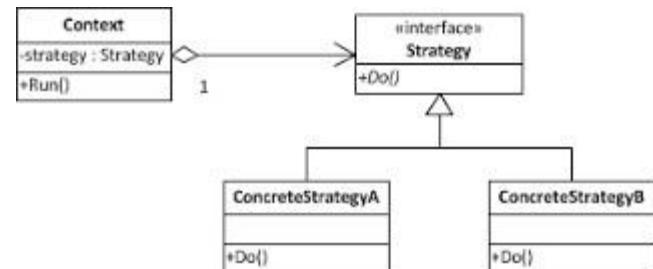
Maas\_Bilgisini\_Bul



# Yordamsal Yapışıklık

- Yordamsal Yapışık modüldeki işlemler arasında denetim ilişkisi bulunmaktadır.
- İşlemlerin birbirleri ile veri ilişkisi yoktur, ancak işlem sırası önemlidir.

Ekran\_Goruntusunu\_Yaz  
Giris\_Kaydini\_Oku



# Zamansal Yapışıklık

- Bir modül içindeki işlemlerin belirli bir zamanda uygulanması gerekiyor ve bu işlemlerin kendi aralarında herhangi bir ilişkisi yok, yani işlemlerin sırası önemli değil ise, **zamansal yapışıklık** vardır.

Alarm\_Zilini\_Ac

Kapiyi\_Ac

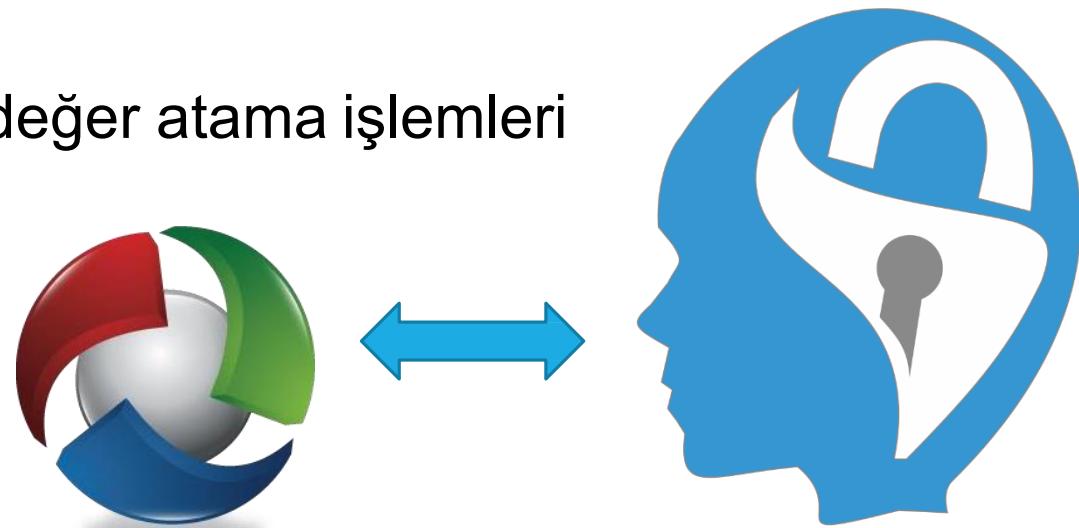
Kamerayı\_Calistir



# Mantıksal Yapışıklık

- Mantıksal olarak aynı türdeki işlemlerin bir araya toplandığı modüller **mantıksal yapışık** olarak adlandırılır.

Dizilere değer atama işlemleri



# Gelişigüzel Yapışıklık

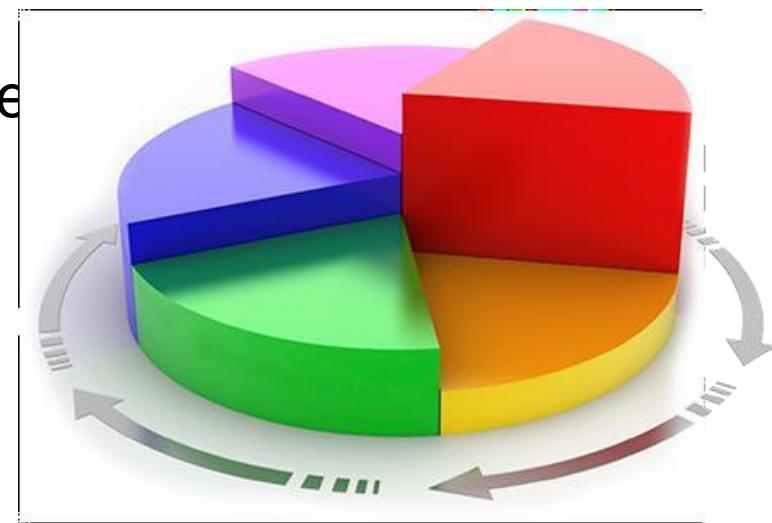
- İşlemler arasında herhangi bir ilişki bulunmaz.

Ara\_Kayit\_Oku

B\_dizisine\_baslangic\_de

Stok\_kutugu\_oku

Hata\_iletisi\_yaz



# Özet

---

- Tasarım, Sistem Analizi çalışması sonucunda üretilen Mantıksal Modelin Fiziksel Modele dönüştürülme çalışmasıdır.
- Tasarım kavramları: Soyutlama, İyileştirme ve Modülerlik olmak üzere 3 çeşittir.
- Yapı Tasarımı, arayüz tasarımı ve süreç tasarımından önce yapılması gereken ilk tasarım veri tasarımıdır.
- Veri Akışları Üç parçada incelenebilir: Girdi Akışı, Çıktı Akışı ve İşlem Akışı
- Süreç tasarımı; veri, yapı ve ara yüz tasarımından sonra yapılır.
- Program Akış Diyagramları: Tekrarlı, ardışıl ve koşullu şeklindedir.

Tasarlanması Gereken Ortak Alt Sistemler;

- Yetkilendirme altsistemi
- Güvenlik altsistemi
- Yedekleme altsistemi
- Veri transferi altsistemi
- Arşiv altsistemi
- Dönüştürme altsistemi      şeklindedir.

# Sorular

---

1. Yazılım tasarım sürecinin temel işlemlerini sayınız. E-R diyagramı çizerek ilişkilerini gösteriniz.
2. Geliştireceğiniz bir uygulama için ekran şablonunuzu belirleyiniz.
3. Bağlaşım ve yapışıklık kavramlarını açıklayınız. Program bakımı ile ilişkilerini belirtiniz.
4. Bir sistem tümüyle bağımsız biçimde tasarlabilir mi? Yani sistemin tüm modülleri arasında hiç bağlaşım olmadan tasarım yapılabilir mi?
5. Tümüyle işlevsel yapışık modüllerden oluşan bir sistem tasarlabilir mi? Neden yapılabilir? Neden yapılamaz?
6. Bağlaşım ile Yazılım Taşınabilriliği arasındaki ilişkiyi belirtiniz.
7. Tasarım gözden geçirmenin önemi nedir? Yapılmaması ne tür sonuçlara yol açar?
8. Arşiv alt sistemi ile yedekleme alt sistemi arasındaki benzerlikleri ve farklılıklarını belirtiniz.
9. Tasarım ile sınama arasındaki ilişkiyi belirtiniz.
10. 13. Geliştirdiğiniz bir uygulama tasarımlı için
  - (a) Bir birim sınama belirtimi
  - (b) İki sistem sınama belirtimi (senaryo) hazırlayınız.

# Kaynaklar

---

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş ”, TBİL-211, Dr. Ali Arifoğlu.

” Yazılım Mühendisliği ” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönerek Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

# Kaynaklar

---

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

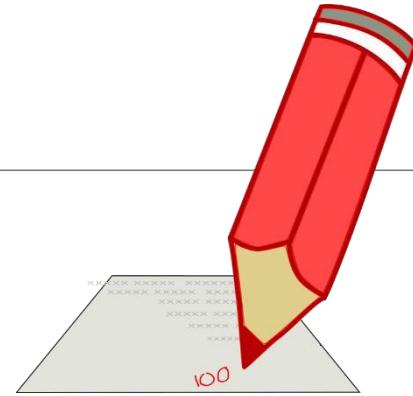
Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

# Ödev

---

Yazılım Tasarım Kalıpları hakkında araştırma yapınız.





# YMT 312-Yazılım Tasarım Ve Mimarisi Mimari Tasarım

Fırat Üniversitesi Yazılım Mühendisliği Bölümü



# Bu Haftaki Konular

Genel Bir Yazılım Mühendislik Tasarımı Süreci.....	7
Mimari Tasarıma Etki Eden Faktörler.....	11
Mimari Tasarım Süreci.....	12
Mimari Spesifikasyon Notasyonları.....	17
Kutu-ve-çizgi diyagramları .....	25
Ortak UML Notasyonları.....	28
Sağlanan ve Gereksinilen Arabirimler.....	39
Mantıksal ve Fiziksel Mimari.....	44

# Amaçlar

Mimari tasarım, ürün tasarımı ve ayrıntılı tasarım arasındaki ilişkileri tartışmak

Mimari tasarıma etki eden faktörleri listelemek

Mimari tasarım sürecini gözden geçirmek

Mimari Tasarım Dokümanı'nın (SAD) içeriğini sunmak

Kalite niteliklerini ve bunların mimari tasarımdaki rolünü açıklamak

Mimari tasarım spesifikasyonu notasyonlarını ve özellikle arabirimler (interface) için olanlarını araştırmak



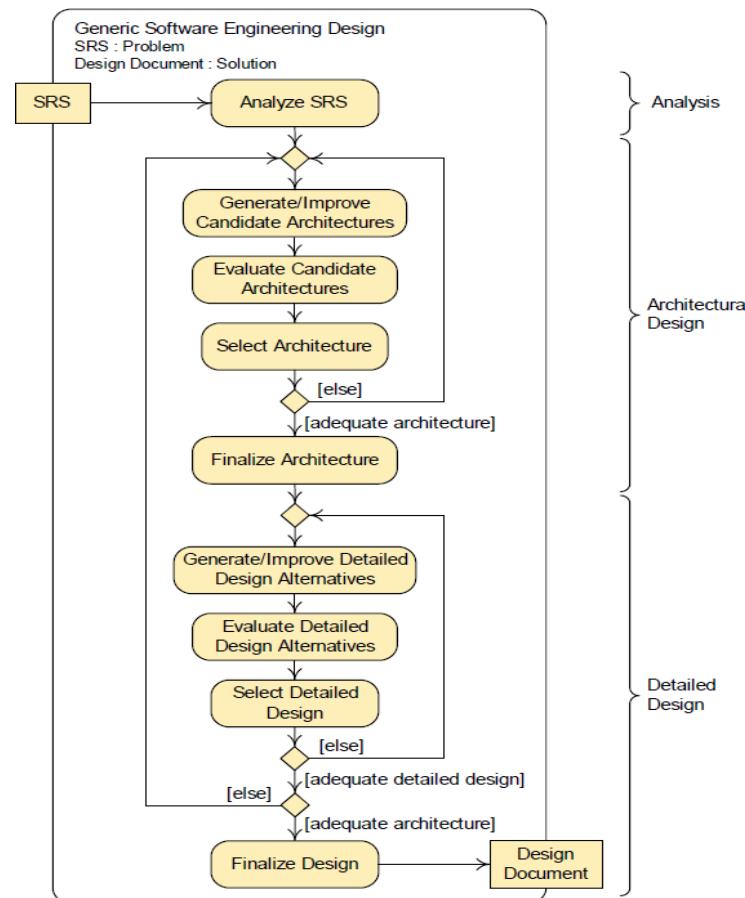
# İçerik

---

- Mimari tasarım, ürün tasarımı, ve ayrıntılı tasarım
- Mimari tasarımı etkileyen faktörler
- Mimari tasarım süreci
- Kalite nitelikleri
- Mimari tasarım spesifikasyonları



# Genel Bir Yazılım Mühendislik Tasarımı Süreci



# Mimari Tasarım (Architectural Design)

---



**Mimari tasarım**, programın büyük parçalarının, bunların sorumluluklarının, özelliklerinin, ve arabirimlerinin; ve bu parçalar arasındaki ilişki ve etkileşimlerin belirlenmesi aktivitesidir.

# Ürün Tasarımında Mimari Tasarım

---

Mimariye ürün tasarımları sırasında da gereksinim duyulur.

- Fizibiliteyi değerlendirmek için
- Paydaşları (stakeholders) gereksinimlerinin karşılanabileceğine ikna etmek için
- Fayda-maliyet analizi yapmak için
- Projeyi planlamak için



# Mimari Tasarım ve Ayrıntılı Tasarım

---

- Mimari tasarım ve ayrıntılı tasarım arasındaki ayırım bazen yeterince belirgin olmayabilir.
- “Büyük” bir program parçası ne demek?
  - Mimari spesifikasyonları ne kadar soyut (abstract) olmalı?
  - Çok küçük bir programın mimarisi nedir/nasıldır?



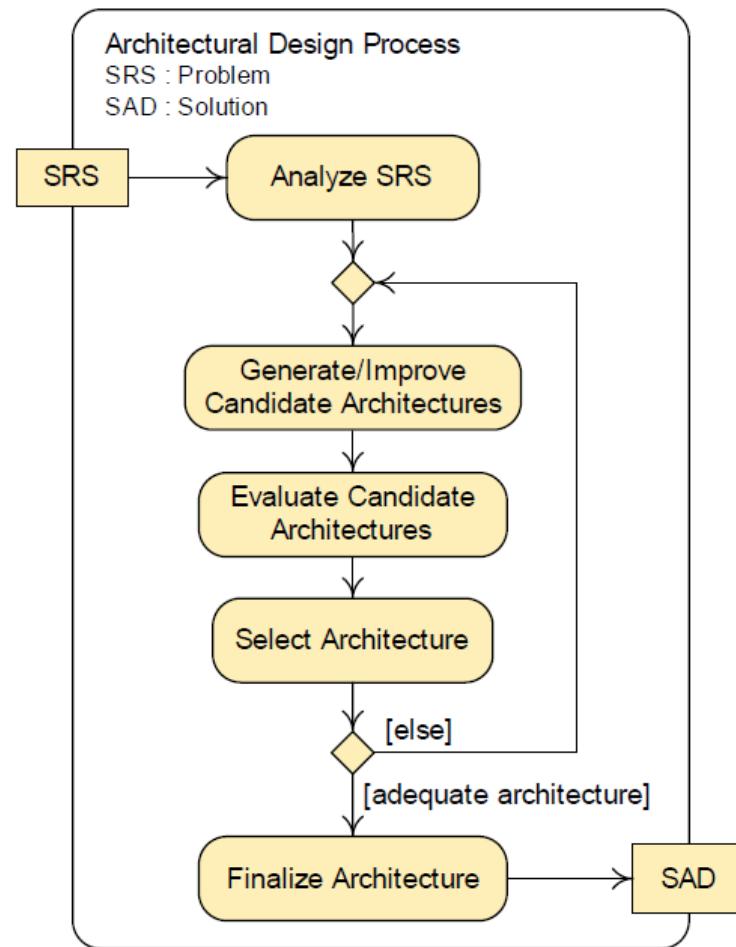
# Mimari Tasarıma Etki Eden Faktörler

---

- Kod kütüphaneleri ve diğer kullanılabilecek varlıklar (assets)
- Kurumsal yapı (Organizational structure)
- Tasarımcıların bilgi ve deneyimi
- Mimariler de kişileri ve kurumları etkileyebilir.



# Mimari Tasarım Süreci



# Mimari Tasarım Dokümanı (SAD)

---

- **Ürüne Genel Bakış (Product Overview)**—Ürün vizyonu, paydaşlar, hedeflenen pazar, vs.
- **Mimari Modeller (Architectural Models)**—Statik ve dinamik çeşitli modellere ilişkin spesifikasyonlar,
  - DeSCRIPTR
- **Modellerin Nasıl Eşleştiği (Mapping Between Models)**—Modellere ilişkin tablolar ve yazılı bilgiler
- **Mimari Tasarımın Gerekçesi (Architectural Design Rationale)**—Zor, çok önemli, kafa karıştırıcı, ve değiştirilmesi zor tasarım kararlarına dair açıklamalar

# Kalite Nitelikleri (Quality Attributes)

---

Bir **kalite niteliği**, müşteri gereksinimlerinin karşılanması bakımından önemli olan işlevlerinden bağımsız olarak, bir yazılım ürününün bir karakteristiği veya özelliğidir.

- İşlevsel olmayan gereksinimler
- Mimarilerin kalite nitelikleri üzerinde büyük etkisi vardır
- Geliştirmeye veya operasyona yönelik nitelikler
- Kalite niteliklerini kullanmak için teknikler (daha sonra)

# Geliştirme Nitelikleri (Development Attributes)

---

- **Bakım Yapılabilirlik (Maintainability)**—Ürünün hatalarının düzeltilebilme, iyileştirilebilme ve başka platforma taşınabilme (port) kolaylığı,  
Genellikle altböülümlere ayrıılır.
- **Yeniden Kullanılabilirlik (Reusability)**—Ürünün parçalarının başka ürünlerin geliştirilmesinde kullanılabilirliğinin derecesi
- Diğer



# Operasyonel Nitelikler (Operational Attributes)

---

- **Performans (Performance)**—Ürün işlevlerinin zaman ve kaynak limitleri içerisinde sağlanabilme becerisi
- **Uygunluk (Availability)**—Kullanıma hazır bulunma durumu
- **Güvenilirlik (Reliability)**—Normal çalışma koşullarında gereksinimlere uygun davranış bilme becerisi
- **Güvenlik (Security)**—Kötü niyetli uygulamalar ve etkiler karşısında zarar görmeye ya da zarar vermeye karşı koyabilme becerisi
- Diğer



# Mimari Spesifikasyon Notasyonları

Spesifikasyon Türü	Kullanışlı Notasyonlar
Decomposition	Box-and-line diagrams, class diagrams, package diagrams, component diagrams, deployment diagrams
States	State diagrams
Collaborations	Sequence and communication diagrams, activity diagrams, box-and-line diagrams, use case models
Responsibilities	Text, box-and-line diagrams, class diagrams
Interfaces	Text, class diagrams
Properties	Text
Transitions	State diagrams
Relationships	Box-and-line diagrams, component diagrams, class diagrams, deployment diagrams, text

# Arabirimler (Interfaces)

Bir **arabirim** varlıklar arasındaki iletişim sınırıdır.

Bir **arabirim spesifikasyonu**, bir varlığın içinde bulunduğu ortam ile haberleşmek için kullandığı mekanizmayı tanımlar.



# Arabirim Spesifikasyonları

- Sözdizim (Syntax)—İletişim ortamının elemanları ve bunların mesaj oluşturmak için nasıl kombine edildiği
- Semantik (Semantics)—Mesajların anlamları
- Pragmatik (Pragmatics)—Mesajların ilgili bağlam içerisinde görevleri yerine getirmek için nasıl kullanıldığı

Arabirim spesifikasyonları bir modül ve onun içinde bulunduğu ortamla yaptığı iletişimın sözdizim, semantik, ve pragmatiklerini içermelidir.

# Arabirim Spesifikasyonu Şablonu

---

## 1. Sağlanan Servisler (Services Provided)

Sağlanan her bir servis için belirtilmelidir:

- a) Sözdizim (Syntax)
- b) Semantik (Semantics)
- c) Pragmatik (Pragmatics)

## 2. Gereksinilen Servisler (Services Required)

Gereksinilen her bir servis adıyla belirtilmelidir.

Servis açıklaması da eklenebilir.

## 3. Kullanım Kılavuzu (Usage Guide)



Kılavuz

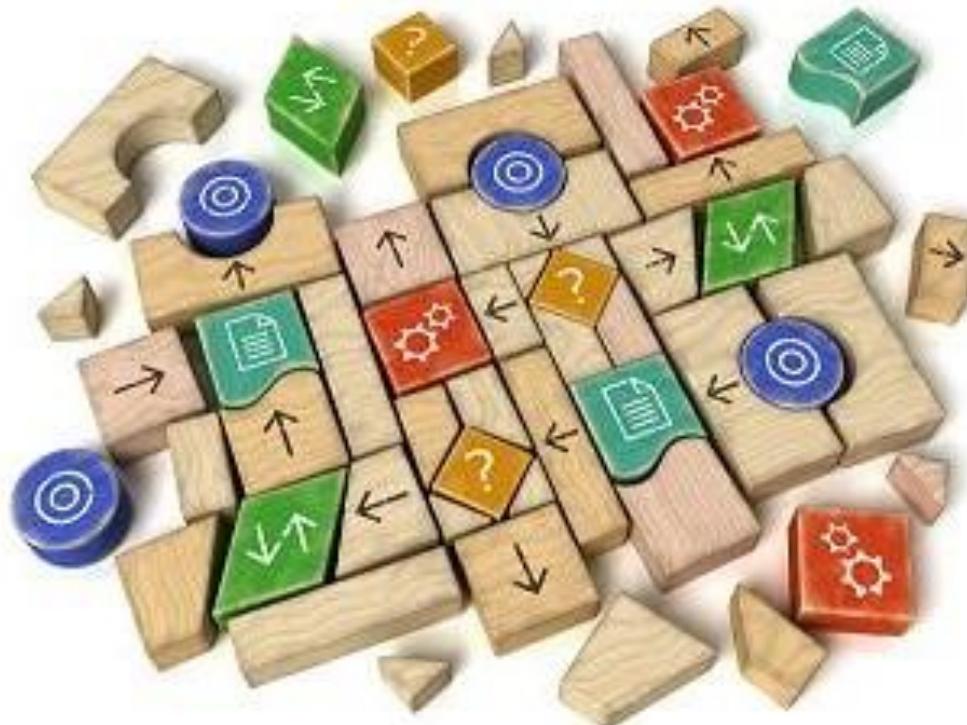
## 4. Tasarım Gerekçesi (Design Rationale)

# Semantik (Semantic) Spesifikasyonu

---

- Bir **önkoşul (precondition)** bir aktivite veya operasyonun başlangıcında sağlanması (doğru olması) gereken bir koşuldur.
- Bir **sonkoşul (postcondition)** bir aktivite veya operasyonun bitiminde sağlanması (doğru olması) gereken bir koşuldur.
- Önkoşullar ve sonkoşullar bir operasyon gerçekleştiğinde ne olması gerektiğini yani operasyonun semantiklerini belirtirler.

# Mimari Modelleme Notasyonları



# Amaçlar

---

Mimari modellemede kullanılan çeşitli notasyonları göstermek

- Box-and-line diagrams
- UML package diagrams
- UML component diagrams
- UML deployment diagrams

Ortak UML notasyonlarını göstermek

- Notes
- Constraints
- Properties
- Stereotypes



# İçerik

---

- Kutu-ve-çizgi diyagramları (Box-and-line diagrams)
- Ortak UML notasyonları
- Paketler ve paket diyagramları (Packages and package diagrams)
- Bileşenler ve bileşen diyagramları (Components and component diagrams)
- Nodes, artifacts, and deployment diagrams

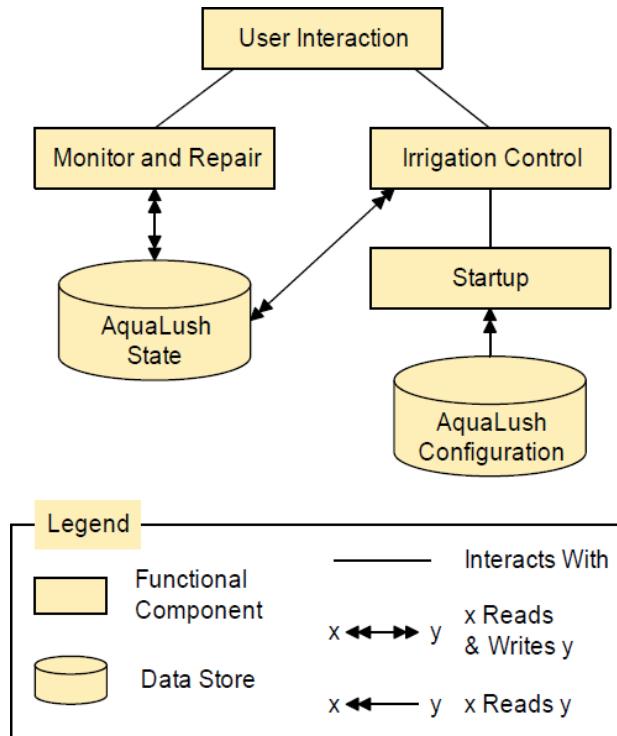


# Kutu-ve-Çizgi Diyagramları (Box-and-Line)

---

- Çizgilerle birleştirilen simgeler (kutular, icon'lar) şeklindedir
- Oluşuma ilişkin kurallar yoktur
- Hem statik hem de dinamik modelleme için kullanılırlar
- Lejant (gösterge) eklenmesi iyi olur

# Kutu-ve-Çizgi Diyagramı Örneği



# Kutu-ve-Çizgi Diyagramı Kuralları

---

- Kutu-ve-çizgi diyagramlarını yalnızca standart notasyonların yetersiz kaldığı durumlarda kullanın.
- Kutuları ve çizgileri basit ve sade tutun.
- Farklı şeyler için farklı semboller/simgeler kullanın.
- Farklı diyagamlarda sembollerini tutarlı bir şekilde kullanın.
- Elemanları isimlendirirken gramer kurallarına uyın.
- Statik ve dinamik elemanları birlikte kullanmayın.

# UML'de Notlar ve Kısıtlar

**Not (Note)**—Model elemanlarına kesik çizgiyle bağlanan, bir köşesi kıvrılmış bir kutu

- İsteğe bağlı herhangi bir metin içerebilir
- Yorumlar ve belirtimler için kullanılır

**Kısıt (Constraint)**—Model elemanları tarafından belirtilen varlıklar için sağlanması gereken koşulu belirten bir ifade

- Küme parantezi içinde yazılır { }
- Model elemanlarının yanında
- Çeşitli model elemanlarını bağlayan kesik çizgilerin yanında

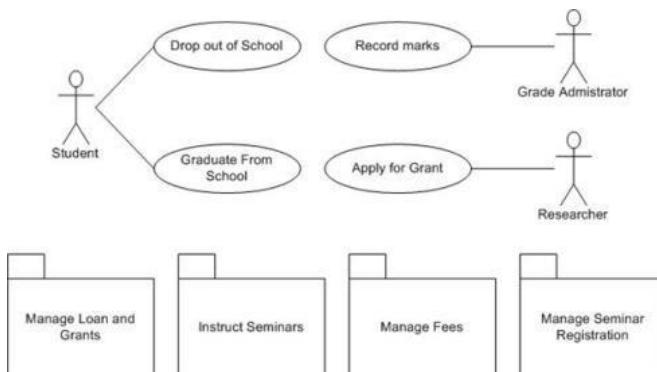
# UML'de Özellikler ve Stereotipler

**Özellik (Property)**—Bir model elemanı tarafından belirtilen bir varlığın karakteristiği

- Küme parantezi içinde etiketli değerlerin listesi
- Etiketli değer: etiket = değer
- True olan Boolean özellikler için değer ve eşittir simgesi yazılmayabilir.

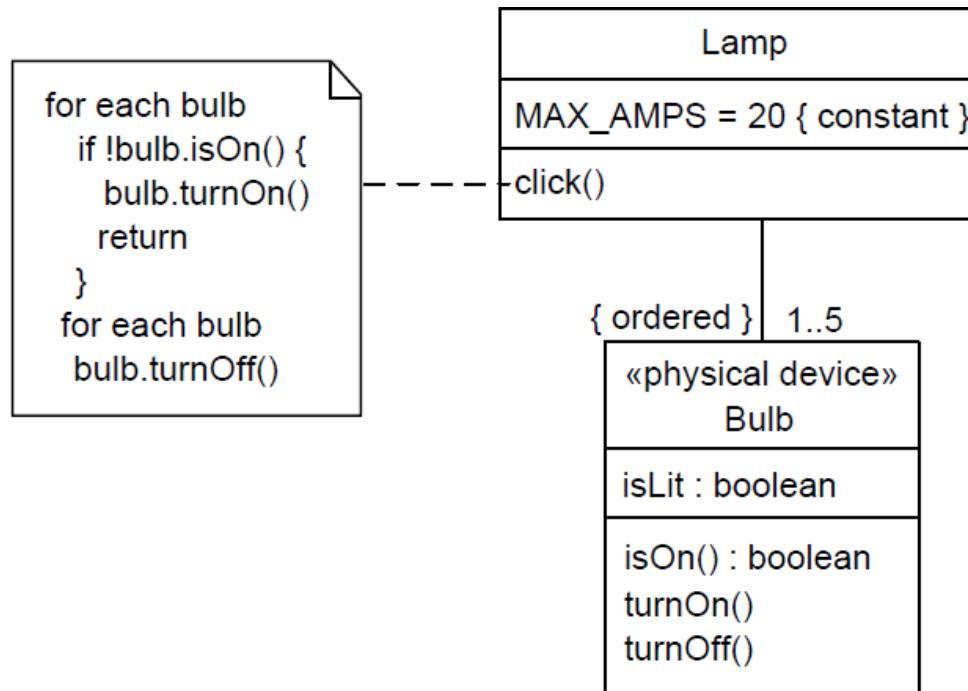
**Stereotip (Stereotype)**—Daha spesifik anlam verilen bir model elemanı

- Simgeler, renkler, ve grafiklerle gösterilir
- Stereotip anahtar sözcükleri özel çift tırnaklar arasında yazılır, örneğin «interface»



# Ortak Elemanlara Örnek

---



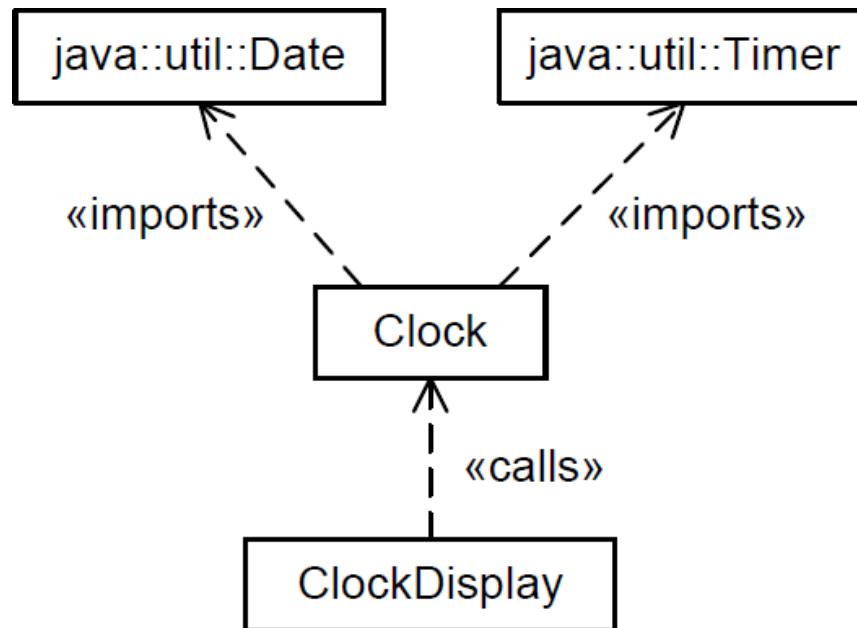
# UML Bağımlılık (Dependency) İlişkileri

---

*D* ve *I* iki varlık olmak üzere; *I*'da (bağımsız - independent) olan bir değişiklik *D*'yi (bağımlı - dependent) etkiliyorsa, *D* ve *I* arasında bir **bağımlılık ilişkisi** (dependency relation) vardır.

- Örneğin: *D* *I*'yı kullanır, *D* derlenebilmek için *I*'ya bağımlıdır, *D* *I*'yı import eder.
- **Bağımlılık okları**yla (dependency arrow) temsil edilir: stereotipli kesik çizgili oklar.

# Bağımlilik İlişkisine Örnek



# UML Paketleri (Packages)

- Bir UML paketi (package), paket üyeleri (package member) olarak adlandırılan model elemanlarının bir koleksiyonudur.
- Paket simgesi olarak dosya klasörü kullanılır.
  - Eğer gövde kısmı doluysa paket adı sekmeye, gövde dolu değilse gövde içinde değilse gövdeye yazılır
  - Üyeler gövde içinde ya da bir kapsama simbolü (çember içinde artı işaret) kullanılarak gösterilir
  - Genellikle import veya export bağımlılık oklarıyla bağlanırlar.

# Paket Diyagramları

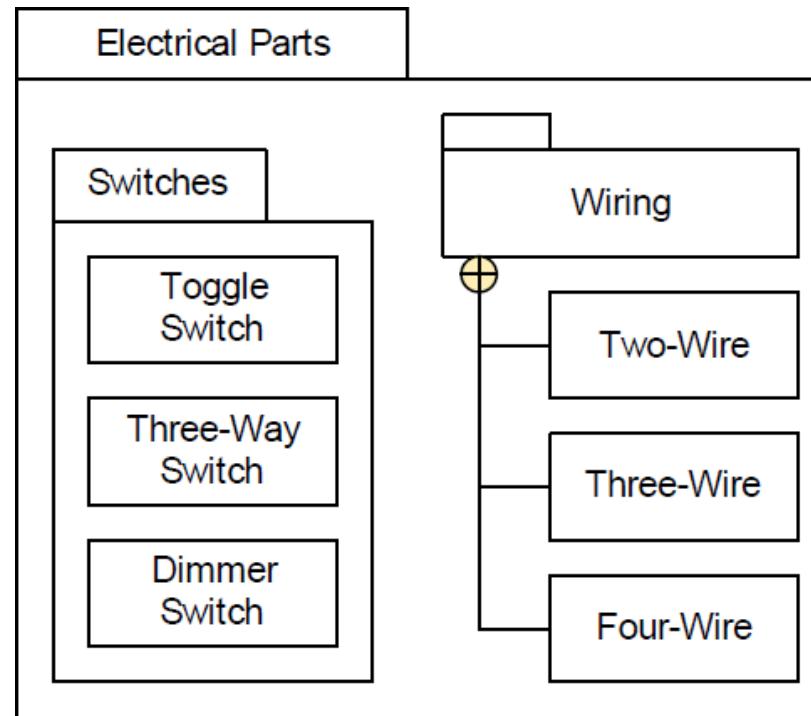
---

- Temel olarak paket sembollerini kullanılarak oluşturulan diyagramlara UML paket diyagramı denir.
- Kullanım alanı:
  - Modüllerin, parçalarının, ve onların ilişkilerinin statik modellerinin gösterimi
  - Mimari modelleme



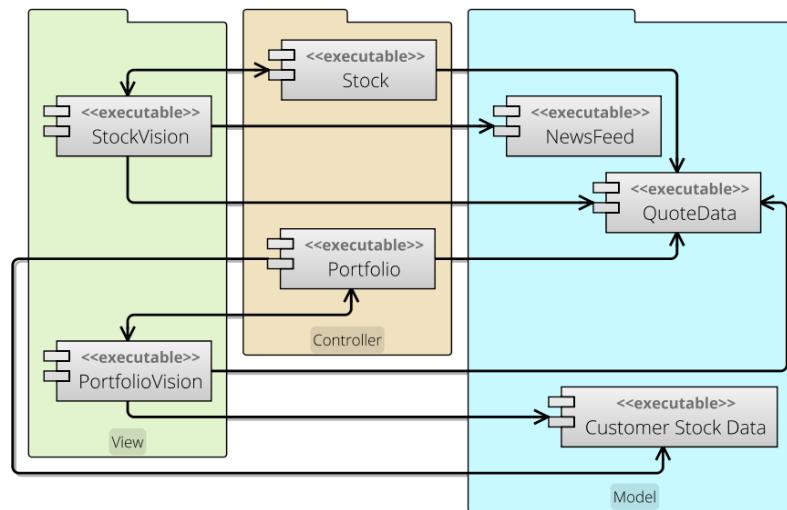
# Paket Diyagramı Örneği

---



# Yazılım Bileşenleri (Components)

- Bir yazılım bileşeni yeniden kullanılabilir ve değiştirilebilir bir yazılım parçasıdır.
- **Bileşen-tabanlı yazılım geliştirme**, ürünlerin satınalınabilir ya da özel olarak geliştirilmiş yazılım bileşenleri kullanılarak tasarlandığı ve geliştirildiği bir yaklaşımındır.



# UML Bileşen Diyagramları

- Bir UML bileşeni, iyi tanımlanmış arabirimlere sahip, modüler, ve başkasıyla değiştirilebilir (replacable) bir birimdir.
  - Bileşen simgesi isim içeren bir dikdörgendir
  - «component» ile stereotipli veya sağ üst köşesinde bileşen simgesi olabilir
- Bir UML bileşen diyagramı bileşenleri, ortamlarıyla ilişkilerini, ve iç yapılarını gösterir.

# UML Arabirimleri (Interfaces)

Bir UML **arabiri**mi (interface) public tanımlı özellikler ve soyut (abstract) operasyonların adlandırılmış bir koleksiyonudur.

- Stereotipli bir sınıf sembolüyle temsil edilir (daha sonra)
- Özel top ve soket sembolleriyile temsil edilir.

Not: arabirim (interface) sözcüğünün buradaki anlamı daha önce gördüğümüz iletişim sınırı anlamından farklıdır.

# Sağlanan ve Gereksinilen Arabirimler

---

➤ Bir sınıf veya bileşen bir arabirimin tüm özelliklerini kendine dahil eder ve arabirimin tüm operasyonlarını implemente ederse, bu sınıf veya bileşen bu arabirimini **gerçekliyor/gerçekleştiriyor** (realize) demektir.

**Sağlanan (Provided) arabirim**—Sınıf veya bileşen tarafından gerçekleştirir

- Bir top veya lolipop sembolüyle temsil edilir

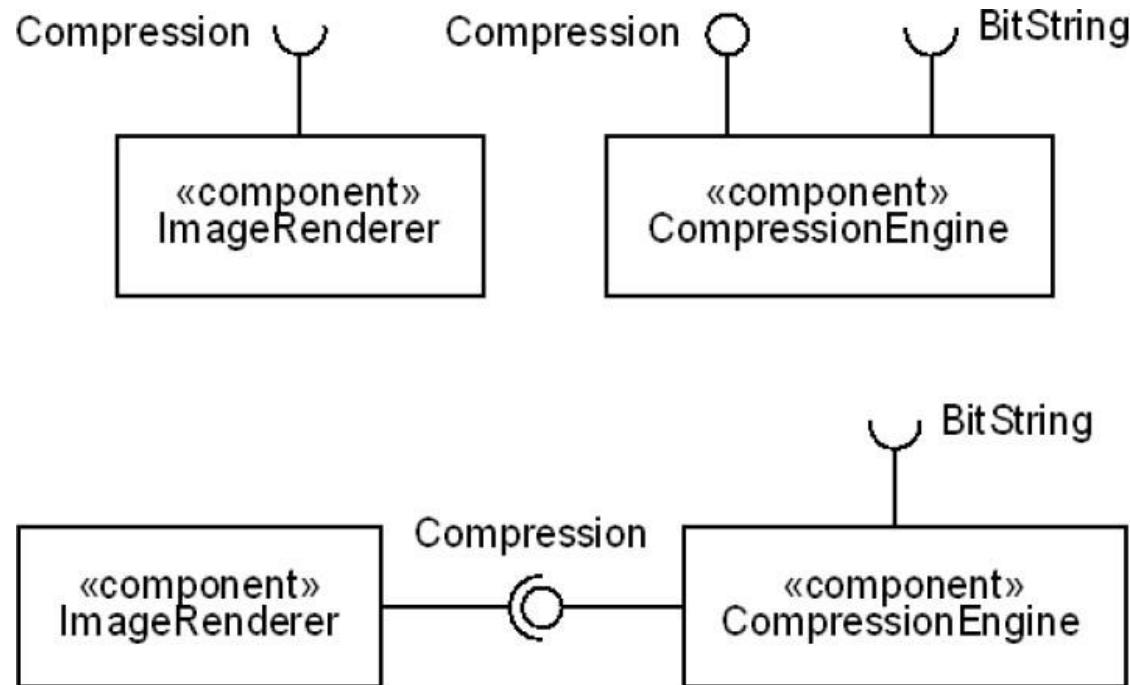
**Gereksinilen (Required) arabirim**—Bir sınıf veya bileşen tarafından ihtiyaç duyulur

- Bir soket sembolüyle temsil edilir

**Montaj konnektörü** (assembly connector) arabirimleri birbirine bağlar.

# Arabirim Sembollerine Örnekler

---

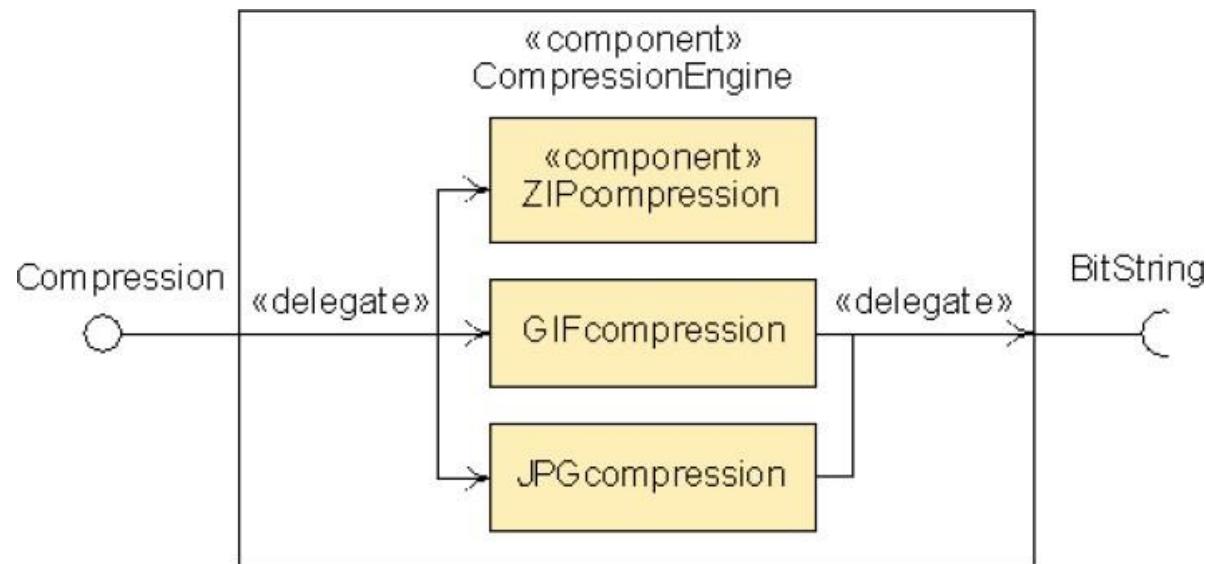


# Bileşenlerin İç Yapısı

- Bileşenler başka bileşenleri veya sınıfları içerebilirler ve içsel yapılarını gösterebilirler.
- Bir **delegasyon konnektörü** (delegation connector), bir bileşen arabirimini bu arabirimini gerçekleyen veya kullanan bir veya daha fazla içsel sınıfı veya bileşene bağlar.
  - Düz çizgili oklar
  - «**delegate**» ile stereotipli



# Bileşenlerin iç Yapısına Örnek

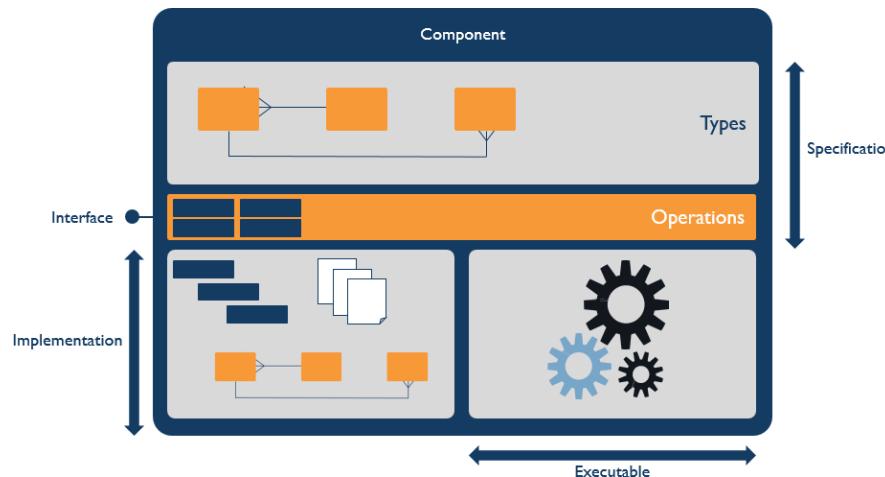


# Bileşen Diyagramı Kullanımı

- Yazılım bileşenlerinin statik modellerinin gösteriminde (yeniden kullanılabilir ve başkasıyla değiştirilebilir parçalar)
- Program bileşenlerinin modellenmesinde
  - Mimari modeller
  - Ayrıntılı tasarım modelleri
  - Ortam ile ilişkiler
- Bileşenlerin içsel yapısının modellenmesinde kullanılır.

# Mantıksal ve Fiziksel Mimari

- Mantıksal (Logical) mimari—Bir ürünün temel parçalarının ve onların ilişkilerinin çalışan kod halinde gerçek bir makinede implementasyonundan daha soyut bir konfigürasyonu
- Fiziksel (Physical) mimari—Bir ürünün işlemsel kaynaklar üzerinde kod ve veri dosyaları halinde bulunmasının ve çalışmasının gerçekleştirilmesi
- UML kurulum (deployment) diyagramı fiziksel mimariyi modeller.



# UML Artefaktları (Artifacts)

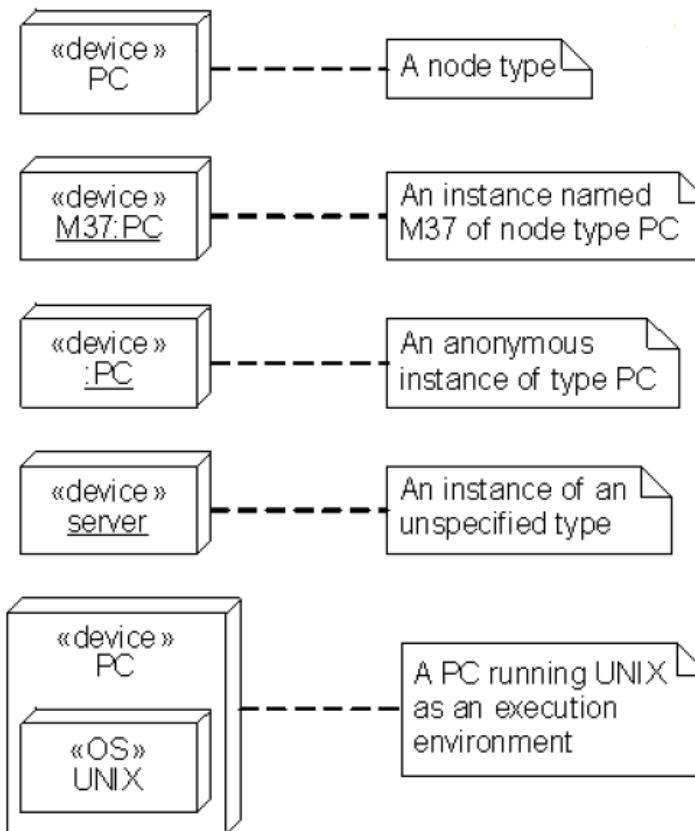
- Bir UML artefaktı (artifact) geliştirme veya işletim sırasında kullanılan veya üretilen verinin herhangi bir fiziksel temsilidir.
  - Örneğin: dosyalar, dokümanlar, program kodları, veritabanı tabloları, vb.
- Artefaktların tipleri ve örnekleri (instance) vardır
  - İsim içeren dikdörtgenle temsil edilir
  - «**artifact**» ile stereotiplidir veya sağ üst köşesinde artefakt sembolü vardır
  - Örneklerin isimlerinin altı çizgilidir, tiplerin isimlerini altı çizgili değildir
- Artefaktlar mantıksal varlıkların gerçekleşmiş şekilleridir (sınıflar, bileşenler, vb.)

# UML Düğümleri (Nodes)

- Bir UML **düğümü** (node) işlemsel (computational) bir kaynaktır.
  - **Cihaz (Device)**—Bir bilgisayar gibi fiziksel bir işlem birimidir
  - **İşletim ortamı (Execution environment)**—İşletim sistemi veya bir dil yorumlayıcısı gibi bir sanal makineyi implemente eden bir yazılım sistemidir
- UML'de kutu veya kütük (slab) sembolüyle temsil edilir
  - «device» veya «execution environment» ile stereotiplidir
  - Tipler ve örnekler (instance)
  - Tiplerin isimleri vardır
  - Örneklerin name : type formunda altı çizgili etiketleri vardır
  - İsim veya tipten biri yazılmayabilir, ama ikisi birden değil

# Düğüm Sembolü Örnekleri

---



# Kurulum (Deployment) Diyagramları

---

➤ Bir UML kurulum diyagramı, işlemsel kaynakları, aralarındaki iletişim yollarını, ve üzerlerinde bulunan ve işletilen artefaktları modeller.

Kullanım yeri:

- Sistemde kullanılan gerçek ve sanal makineleri göstermek
- Makineler arasındaki iletişim yolunu göstermek
- Sistemi oluşturan program ve veri dosyalarını göstermek
  - Yerleşim (Residence)
  - İşletim (Execution)

# Kurulum Diyagramı Kuralları

İşlemsel kaynaklar düğümleridir

İletişim yolları düğümler arasındaki düz çizgilerdir

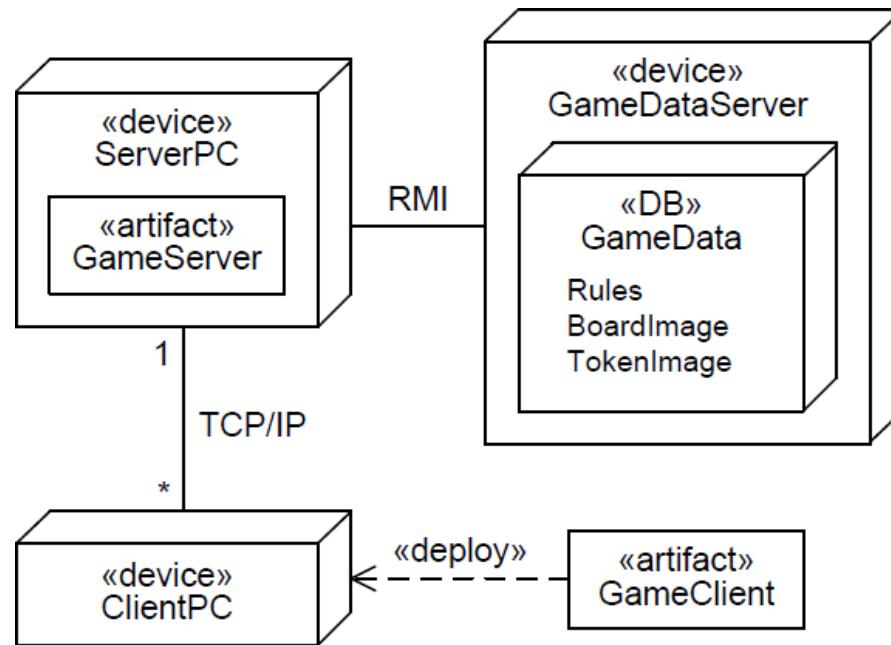
- Etiketlenebilir
- Çoklukları ve rol isimleri olabilir

Artefakt sembollerı

- Düğüm sembollerinin içinde görünebilir
- Düğüm sembollerinin içinde listelenebilir
- «**deploy**» stereotipli bağımlılık oklarıyla düğüm sembollerine bağlanabilir

# Kurulum Diyagramı Örneği

---



# Özet

---

- Mimari tasarım ürünün fizibilitesinin değerlendirilebilmesi amacıyla genellikle ürün tasarıımı sırasında başlamalıdır.
- Mimari tasarımdan ayrıntılı tasarıma doğru soyutlamadan (abstraction) derecesi azalır.
- Bir mimarı tasarım dokümanı (SAD) ürüne genel bir bakış, mimari spesifikasyonlar, ve tasarım gereklisi gibi bölümlerden oluşur.
- Mimari kararların kalite nitelikleri üzerinde büyük etkisi vardır.
- Mimari modelleme için kullanılan çeşitli notasyonlar mevcuttur.
- Arabirim spesifikasyonları, iletişim ortamına dair tanımlamalar içermelidir
  - Sözdizim (Syntax),
  - Semantik (Semantics), ve
  - Pragmatik.

# Özet

- Kutu-ve-çizgi diyagramları statik ve dinamik mimari model oluşturmak için kullanılır.
- Notlar, kısıtlar, özellikler, ve stereotipler herhangi bir UML diyagramında kullanılabilir.
- Paket diyagramları modulleri ve modüllerin parçalarını modellemek için kullanılır.
- Bileşen diyagramları yazılım bileşenlerini modellemek için kullanılır.
- Kurulum diyagramları fiziksel mimarileri modellemek için kullanılır.

# Kaynaklar

---

“Software Engineering A Practitioner’s Approach” (7th. Ed.), Roger S. Pressman, 2013.

“Software Engineering” (8th. Ed.), Ian Sommerville, 2007.

“Guide to the Software Engineering Body of Knowledge”, 2004.

” Yazılım Mühendisliğine Giriş”, TBİL-211, Dr. Ali Arifoğlu.

”Yazılım Mühendisliği” (2. Basım), Dr. M. Erhan Sarıdoğan, 2008, İstanbul: Papatya Yayıncılık.

Kalıpsız, O., Buharalı, A., Biricik, G. (2005). Bilgisayar Bilimlerinde Sistem Analizi ve Tasarımı Nesneye Yönelik Modelleme. İstanbul: Papatya Yayıncılık.

Buzluca, F. (2010) Yazılım Modelleme ve Tasarımı ders notları (<http://www.buzluca.info/dersler.html>)

Hacettepe Üniversitesi BBS-651, A. Tarhan, 2010.

Yazılım Proje Yönetimi, Yrd. Doç. Dr. Hacer KARACAN

YZM211 Yazılım Tasarımı – Yrd. Doç. Dr. Volkan TUNALI

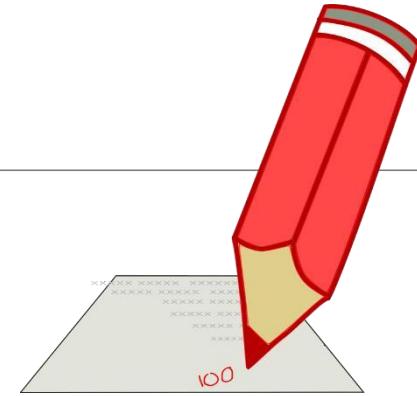
[http://www.cclub.metu.edu.tr/bergi\\_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme](http://www.cclub.metu.edu.tr/bergi_yeni/e-bergi/2008/Ekim/Cevik-Modelleme-ve-Cevik-Yazilim-Gelistirme)

[http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE\\_517\\_Fall\\_2011/ch6\\_6d\\_sk](http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2011/ch6_6d_sk)

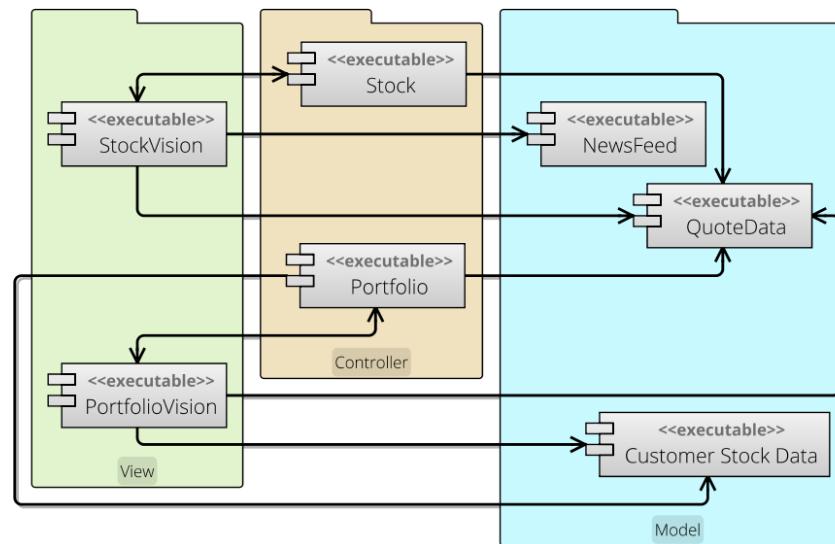
<http://dsdmofagilemethodology.wikidot.com/>

<http://caglarkaya.piquection.com/2014/07/01/244/>

# Ödev



Mimari Tasarım Hakkında Araştırma Yapınız.  
Yazılım Mimarısında Kullanılan Stilleri Araştırınız.





# YMT 312-Yazılım Tasarım Ve Mimarisi Geçerlekstirim

Fırat Üniversitesi Yazılım Mühendisliği Bölümü

Bölüm-6

# Bu Haftaki Konular

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

# Amaçlar

---

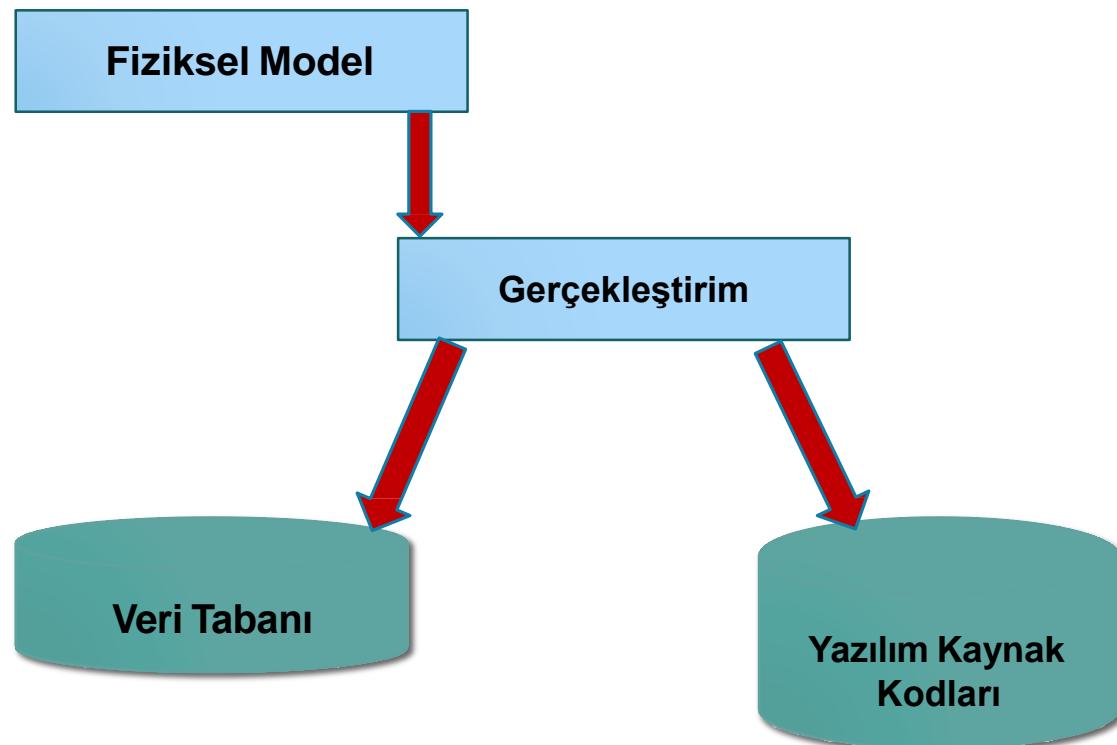
- Yazılım Gerçekleştirimini Kavramak
- Yazılım Geliştirme Ortamları Hakkında Bilgilenmek
- Veri Tabanı Sistemlerini Anlamak
- Veri Modelleme Yöntemlerini Öğrenmek
- Kodlama Stilleri Hakkında Bilgi Edinmek
- Yapısal Programlama Yapılarını Öğrenmek
- Program Karmaşıklığı Hesaplamasında Kullanılan Yöntemler



# Giriş

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

# Gerçekleştirme Çalışması



# Yazılım Geliştirme Ortamları

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

Programlama Dili

Veri Tabanı Yönetim Sistemi

Hazır Program Kitapçıkları

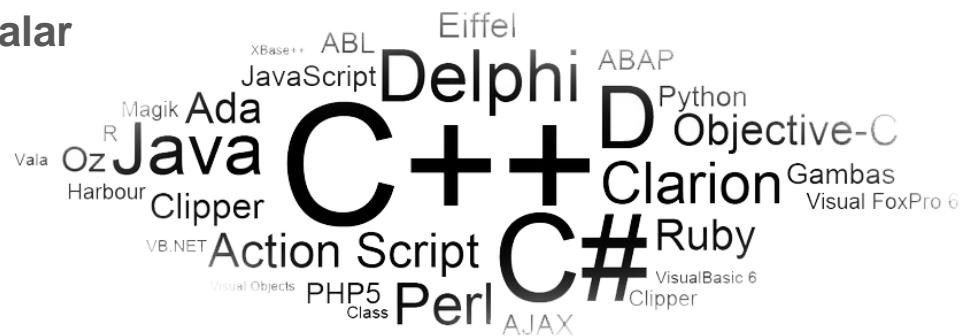
CASE Araçları

bileşenlerinden oluşur.

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

# Programlama Dilleri

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

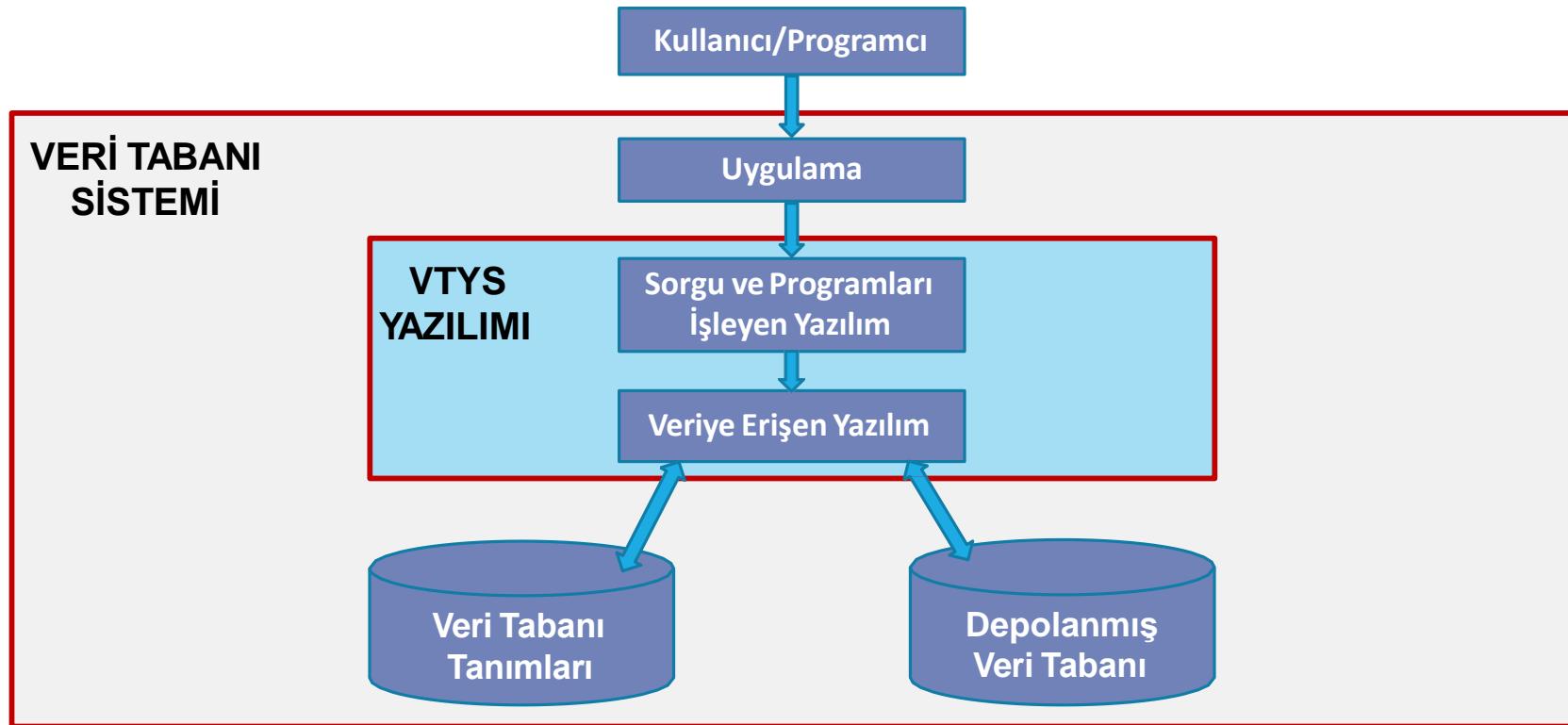


# Veri Tabanı Yönetim Sistemleri

---

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

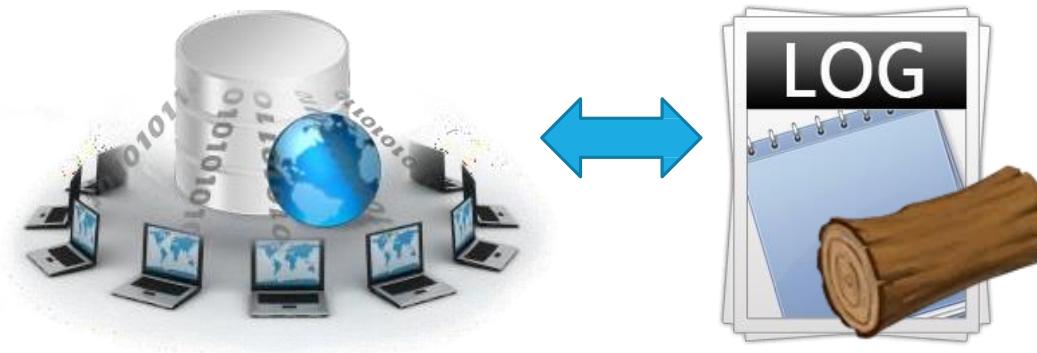
# Veri Tabanı Sistemleri



# Veri tabanı Yaklaşımının Geleneksel Kütük Kavramından Ayıran Özellikler

---

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



# Veri Sözlüğü

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



fields in Student file

primary key

field name

data type for State field

default value

metadata about State field

Field Name	Data Type	Description
Student ID	AutoNumber	Student's ID Number
First Name	Text	Student's First Name
Last Name	Text	Student's Last Name
Address	Text	Student's Address
City	Text	City Student Lives
State	Text	State Student Lives
Postal Code	Text	Student's Postal Code
E-mail Address	Hyperlink	Student's E-mail
Date Admitted	Date/Time	Date Student Admitted to School
Major	Text	Student's Major Code
Photo	Attachment	Digital Photo of Student

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

# Veri Soyutlama

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



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

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

# Program-Veri ve Program-İşlem Bağımsızlığı

---

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

# Birden Çok Kullanıcı Desteği

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

## Verinin Birden Fazla İşlem Arasında Paylaşımı

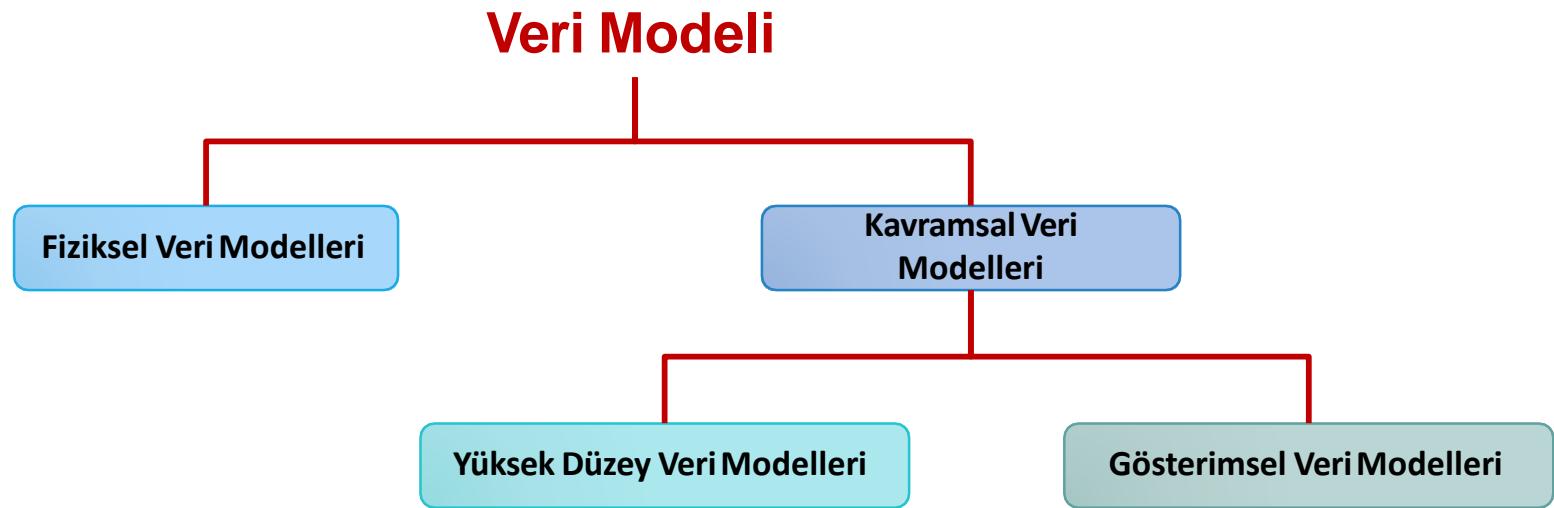
---

- Çok kullanıcılı VTYS yazılımlarının temel rolü eş zamanlı işlemlerin karışıklık olmadan doğru bir şekilde yapılmasına olanak tanıtmak işlemlerin doğru olarak yapılmasını garanti etmektir.
- Bu özellik veri tabanı (VT) kavramını, dosya işleme kavramından ayıran en önemli özellikleştir.
- Bu kontrol aynı anda birden çok kullanıcının aynı veriyi güncellemeye çalışmasını, güncellemenin doğru olması açısından garanti eder.

# VTYS Kullanımının Ek Yararları

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

# Veri Modelleri



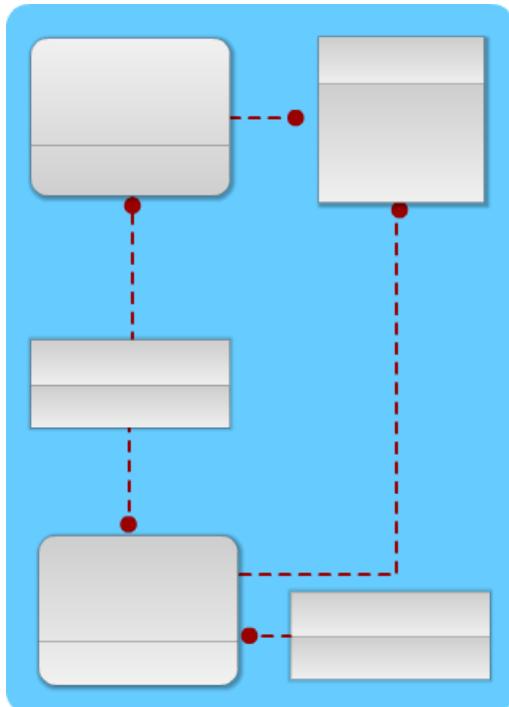
# Veri Modelleri

---

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

# Şemalar

---



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

# VTYS Mimarisi

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

## **İçsel Düzey**

- Veri tabanının fiziksel saklama yapısını açıklar.

## **Kavramsal Düzey**

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

## **Dışsal Düzey**

- Dış şemalar ve kullanıcı görüşlerini içerir.

# Veritabanı Dilleri ve Arabirimleri

---

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

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

# Veritabanı Dilleri ve Arabirimleri

---

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

# VTYS nin Sınıflandırılması

Sınıflandırmalar kullanılan Veri Modeline göre yapılır.

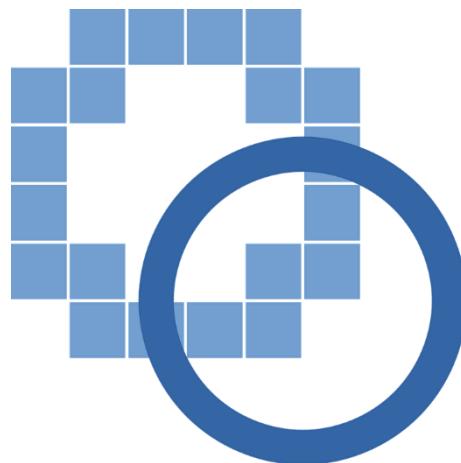
- **İlişkisel Model:** Veri tabanı tablo yiğinından oluşmuştur. Her bir tablo bir dosya olarak saklanabilir.
- **Ağ Modeli:** Veriyi kayıt ve küme tipleri olarak gösterir.
- **Hiyerarşik Model:** Veri ağaç yapısında gösterilir.
- **Nesne Yönelimli Model:** Veri tabanı nesneler, özellikleri ve işlemleri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

# Hazır Program Kütüphaneleri

- Hemen hemen tüm programlama platformlarının kendilerine özgü hazır kütüphaneleri bulunmaktadır.
  - Pascal            \*.tpu
  - C                 \*.h
  - Java              \*.jar
- Günümüzde bu kütüphanelerin temin edilmesi internet üzerinden oldukça kolaydır.

# CASE Araç ve Ortamları

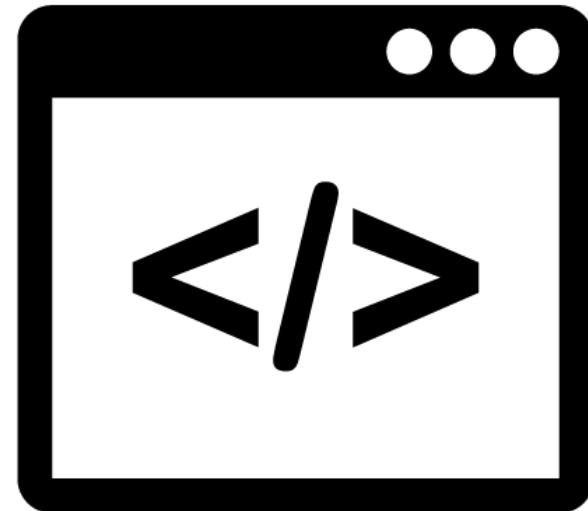
- Günümüzde bilgisayar destekli yazılım geliştirme ortamları oldukça gelişmiştir.
- CASE araçları yazılım geliştirme sürecinin her aşamasında üretilen bilgi ya da belgelerin bilgisayar ortamında saklanması ve bu yolla kolay erişilebilir ve yönetilebilir olmasına olanak sağlar.



# Kodlama Stili

---

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



# Açıklama Satırları

---

**İyi:**

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

---

**KÖTÜ:**

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

---

**KÖTÜ:**

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

---

- Modülün başlangıcına

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

gibi genel bilgilendirici açıklamaları  
yapılır.

- Aynı zamanda modülün kritik bölümleri  
vurgulanarak açıklanır.

# Kod Biçimlemesi

---

**İyi:**

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

**KÖTÜ:**

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

**İyi:**

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

**KÖTÜ:**

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

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

# Anlamlı İsimlendirme

---

## İyi:

KodlamaStandartlari.csproj  
namespace KodlamaStandartlari

KodlamaStandartlari.Models.dll  
namespace KodlamaStandartlari.Models

KodlamaStandartlari.csproj projesinde Models dizini  
namespace KodlamaStandartlari.Models

## KÖTÜ:

KodlamaStandartlari.csproj  
namespace Kodlama

KodlamaStandartlari.Models.dll  
namespace KodlamaStandartlari.Modeller

KodlamaStandartlari.csproj projesinde Models dizini  
namespace KodlamaStandartlari.Modeller

## “İsim Alanı” İsimlendirme

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

# Yapısal Programlama Yapıları

- Program kodlarının, okunabilirlik, anlaşılabilirlik, bakım kolaylığı gibi kalite etmenlerinin sağlanması ve program karmaşıklığının azaltılması amacıyla "yapısal programlama yapıları" kullanılarak yazılması önemlidir.
- Yapısal Programlama Yapıları, temelde, içinde "go to" deyimi bulunmayan, "tek giriş ve tek çıkışlı" öbeklerden oluşan yapılardır.
- Teorik olarak herhangi bir bilgisayar programının, yalnızca Yapısal Programlama Yapıları kullanılarak yazılabileceği kanıtlanmıştır.
- Üç temel Yapısal Programlama Yapısı bulunmaktadır:
  - Ardisil işlem yapıları
  - Koşullu işlem yapıları
  - Döngü yapıları

# Yapısal Programlama Yapıları

## Ardışıl İşlem Yapıları

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

## Koşullu İşlem Yapıları

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

# Yapısal Programlama Yapıları

## Döngü Yapıları

- Döngü yapıları, belirli bir koşula bağlı olarak ya da belirli sayıda, bir ya da dah çök kez yinelenen işlemler için kullanılan yapılardır. Temelde üç tür döngü yapısı bulunmaktadır. Bunlar:
  1. Belirli sayıda yinelenen işlemler için kullanılan yapılar (for yapısı),
  2. Bir koşula bağlı olarak, sıfır yada birden çok kez yinelenen işlemler için kullanılan yapılar (while-end yapısı),
  3. Bir koşula bağlı olarak, bir ya da daha çok kez yinelenen işlemler için kullanılan yapılar (repeat-until yapısı).

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

# Program Karmaşıklığı

---

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

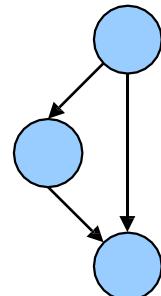
# Programın Çizge Biçimine Dönüşürülmlesi-1

## Sıradan İşlemler:

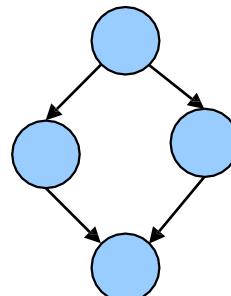


Bir ya da birden fazla ardışık işlem

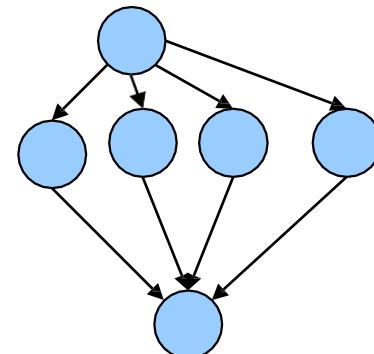
## Koşullu İşlemler:



If-Then işlemi



If-Then-Else işlemi

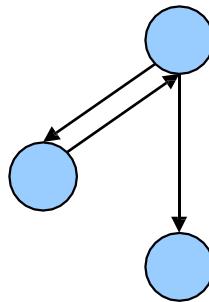


Case işlemi

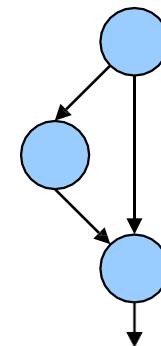
# Programın Çizge Biçimine Dönüşürtlmesi-2

---

## Döngü İşlemleri:



While Döngüsü



Repeat Döngüsü

# McCabe Karmaşıklık Ölçütü

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

**K:** Diyagramdaki kenar çizgi sayısı

**D:** Diyagramdaki düğüm sayısı

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

# Olağanüstü Durum Çözümleme

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



# Kod Gözden Geçirme

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

# Gözden Geçirme Sürecinin Düzenlenmesi

---

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

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

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

# Gözden Geçirme Sırasında Kullanılacak Sorular

---

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



# Öbek Arayüzü

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

# Giriş Açıklamaları

---

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

# Veri Kullanımı

---

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

# Sunuş

1. Her satır, en fazla bir deyim içeriyor mu?
2. Bir deyimin birden fazla satıra taşması durumunda, bölünme anlaşılabilirliği kolaylaştıracak biçimde anlamlı mı?
3. Koşullu deyimlerde kullanılan mantıksal işlemler yalın mı?
4. 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?

# Sorular

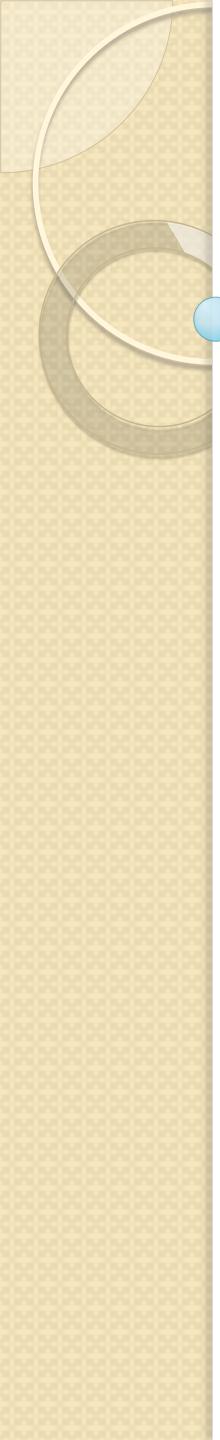
---

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 stilini 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 gerçekleştirileceğini belirtiniz.
9. Verilen bir N doğal sayısının asal olup olmadığını belirleyen bir programı dört değişik biçimde yazınız. Programlar arasındaki zaman farklılıklarını ölçünüz.
10. Program geliştirirken kullandığınız olağan dışı durum çözümleme yöntemlerini açıklayınız?
11. Geliştirdiğiniz bir program için, bölüm içerisinde verilen gözden geçirme sorularını yanıtlayınız. Programınızın niteliği hakkında ne söyleyebilirsiniz?

# Kaynaklar

---

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



# Yazılım Mühendisliği

## Ders 8: Yazılım Bakım

# Genel Bakış

- Giriş
- Tanım
- Bakım Süreç Modeli
  - Sorun Tanımlama Süreci
  - Çözümleme Süreci
  - Tasarım Süreci
  - Gerçekleştirme Süreci
  - Sistem Sınama Süreci
  - Kabul Sınaması Süreci
  - Kurulum Süreci

# Giriş

Sınaması tamamlanmış ve bitirilen yazılımın kullanıcıya yüklenmesi ve uygulamanın başlatılması gerekmektedir. Yazılım kullanıma geçtikten sonra, yaşam döngüsünün en önemli ve hiç bitmeyecek aşaması olan "bakım" aşaması başlar.

# Yazılım Bakımı nedir ?

- Bakım, işletime alınan yazılımın sağlıklı olarak çalışması ve ayakta tutulabilmesi için yapılması gereken çalışmalar bütünü olarak tanımlanır.
- Bakım yazılım yaşam döngüsünün en önemli ve en maliyetli aşamalarından biridir.
- Bakım süreci, yazılım yaşam döngüsünde "buzdağının görünmeyen kısmı" olarak adlandırılır. Bakım maliyetleri, zaman zaman üretim maliyetlerinin %60'ını geçer.

# Bakım Türleri

## ○ Düzeltici Bakım:

- Teorik olarak bir yazılımin tümüyle sınanabilmesi olası olsa bile, pratikte bu sağlanamaz.
- Çalışan bir yazılımda her an hata ile karşılaşma olasılığı vardır. Bu nedenle zaman zaman çalışan yazılımda ortaya çıkan hataların düzeltilmesi gereklidir.
- Bu tür düzeltme çalışmaları “Düzeltici Bakım olarak adlandırılır.”

# Bakım Türleri

## ○ Uyarlayıcı Bakım:

- Uygulama yazılımları, işletme ya da kuruluşların günlük yaşamlarında yaptıkları işleri bilgisayar ortamında yapmalarını sağlayan araçlardır.
- Her kuruluş ya da işletme canlı bir varlık gibi düşünülebilir. Hiçbir işletme durağan değildir. Süreç içinde değişiklik gösterir.
- İşletme ya da kuruluşlardaki değişiklik, yapılan işlerin yapılması tarzının değişmesi, yeni iş türlerinin ortaya çıkması biçiminde kendini gösterebilir.
- Yaşanan değişiklıkların o kuruluşun uygulama yazılımlarına da yansıtılması gereklidir.
- Bu yansıtma işlemi “Uyarlayıcı Bakım” olarak tanımlanır.

# Bakım Türleri

- **En İyileyici Bakım:**
  - Zaman zaman uygulama yazılımlarının çalışma performanslarının iyileştirilmesi amacıyla yapılan çalışmalar “en iyileyici bakım” olarak tanımlanır.

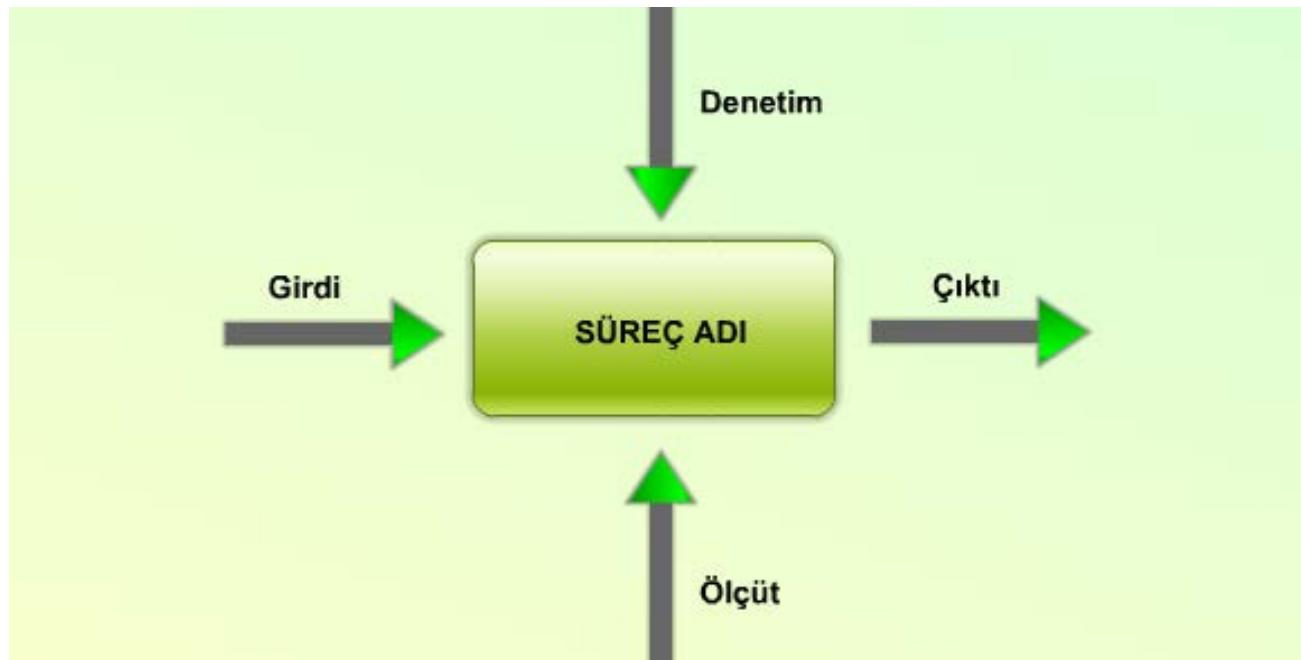
# Bakım Türleri

- **Önleyici Bakım:**

- Herhangi bir yanlış veri girişi veya hesaplama sonucu oluşabilecek potansiyel bir hatayı önlemeye yönelik tedbir alınması, güvenirlilik ve bakılabilirliğin arttırılması çalışmalarını kapsamaktadır.

# Bakım Süreç Modeli

IEEE 1219 standarı tarafından önerilen bakım süreç modeli şekilde belirtilen şablonu kullanarak bakım süreçlerini tanımlamaktadır.



# Sorun Tanımlama Süreci



# Sorun Tanımlama Süreci

## Girdi

Sorun tanımlama sürecinin temel girdisi, Bakım isteğidir. Herhangi bir bakım isteğine örnek olarak:

Sistemde beklenen ve yeni düzenlemelere ilişkin değişiklikler,

- Yeni işlev istekleri,
- Yazılımda bulunan yanlışların düzeltilme istekleri,
- Performans artımına ilişkin istekler,
- Yeni iş yapma türlerine ilişkin istekler ve
- Teknolojinin zorlaması sonucu oluşan istekler  
türündeki istekler verilebilir.

# Sorun Tanımlama Süreci

## İşlem/Süreç

Yazılım bakım isteği oluştuklarında yapılması gereken işlemler:

- a. Değişiklik isteğine bir tanım numarası verilmesi,
- b. Değişiklik türünün sınıflandırılması,
- c. Değişiklik isteğin kabul, red ya da daha ayrıntılı incelenmesi yönünde karar alınması,
- d. Değişiklik ile ilgili zaman/boyut/ışgücü kestirimi yapılması,
- e. Değişiklik isteğin önceliklendirilmesi
- f. Değişiklik isteğin diğerleri ile birlikte zaman ve iş planına kaydedilmesi

biçiminde özetlenebilir. Bu işlemlerin birçoğunda, değişiklik isteğini yapan kişi, kullanıcı temsilcileri, yazılım mühendisleri ve konu uzmanları birlikte çalışıp karar verirler.

# Sorun Tanımlama Süreci

## Denetim

Sorun tanımlama aşamasında, değişiklik isteğinin daha önceden yapılip yapılmadığı denetlenmeli ve tek olduğu belirlenmelidir. Bu amaçla, daha önceki değişiklik istekleri taranır.

## Çıktı

Bu sürecin temel çıktısı, doğrulanmış ve geçerlenmiş ve karar verilmiş Bakım İsteğidir. Bir veri tabanında saklanan bu isteğin ayrıntıları:

Sorun ya da yeni gereksinimin tanımı,

•Sorun ya da gereksinimin değerlendirmesi,

•Başlangıç öncelik,

•Geçerleme verisi (Düzeltilen bakım için),

•Başlangıç kaynak gereksinimi,

•Mevcut ve gelecekte kullanıcılar üzerindeki etkileri,

•Yararlı ve aksak yönler

# Sorun Tanımlama Süreci

## Ölçüt

Sorun tanımlama sırasında kullanılabilen ölçütler:

- Bakım isteklerinde kabul edilmeyen madde sayısı,
- Gelen bakım istekleri sayısı ve
- Sorun geçerleme için harcanan kaynak ve zaman

birimindedir.

# Çözümleme Süreci



Çözümleme süreci, veri tabanında saklanmış ve geçerlenmiş bakım isteğini girdi olarak alır, projeye ilişkin bilgi ve belgeleri kullanarak söz konusu isteğin yerine getirilmesi için gerekli genel planı yapar

# Çözümleme Süreci

## Girdi

Çözümleme sürecinin girdileri:

- Geçerlenmiş bakım isteği,
- Başlangıç kaynak gereksinimleri ve diğer veriler ve
- Mevcut proje yada sistem bilgi ve belgeleri

biçimindedir

# Çözümleme Süreci

## İşlem/Süreç

Çözümleme süreci temel olarak iki aşamadan oluşur: Olurluk aşaması ve ayrıntılı çözümleme aşaması. Mevcut sisteme ya da projeye ilişkin yapısal belgelerin bulunmadığı durumlarda tersine mühendislik yöntemi kullanılır.

Olurluk çalışması,

- Değişikliğin etkisi,
- Prototiplemeyi içeren seçenek çözümler,
- Dönüştürme gereksinimlerinin çözümlemesi,
- Güvenlik ve emniyet zorunlulukları
- İnsan faktörleri,
- Kısa ve uzun erimli maliyetler ve
- Değişikliği yapmanın yararları bilgilerini içerir.

Ayrıntılı çözümleme çalışmasında, değişiklik isteği için ayrıntılı gereksinim tanımaması yapılır. Bu çalışmada etkilenen yazılım öğeleri (yazılım tanımları, yazılım gereksinimleri, tasarım, kod, vb) belirlenir. Yazılım öğelerinin değişmesi gereken kısımları belirlenir. En az üç düzeyli sınama stratejisi (birim sınaması, bütünlendirme sınaması ve kabul sınaması) tanımlanır.

# Çözümleme Süreci

## Denetim

Çözümleme çalışmasının denetiminde aşağıdaki işlemler yapılır.

- a.** Gerekli proje yada sistem bilgi/belgelerine erişimin sağlanması (Ortam denetim organizasyonundan)
- b.** Önerilen değişikliklerin ve çözümleme çalışmasının teknik ve ekonomik olurluğunun gözden geçirilmesi,
- c.** Güvenlik ve emniyet konularının tanımlanması,
- d.** Önerilen değişikliğin, mevcut yazılımla bütünlendirilmesinin dikkate alınması,
- e.** Proje belgelerinin düzgün olarak günlendiğinin denetimi,
- f.** Çözümleme belgelerinin düzgün olarak hazırlanmasının sağlanması,
- g.** Sınama stratejilerinin uygun olarak belirlenmesi.

# Çözümleme Süreci

## Çıktı

Çözümleme çalışmasının çıktıları:

- Değişiklik isteklerine ilişkin olurluk çalışması,
- Ayrıntılı çözümleme raporu,
- İzlenebilirlik listesini içeren günlenmiş gereksinim tanımları,
- Başlangıç değişiklik listesi,
- Sınama stratejisi,
- Gerçekleştirim planı

## Ölçüt

Çözümleme çalışmasında kullanılabilen ölçütler:

- Gereksinimlerdeki değişiklik sayısı,
- Belgeleme hata oranı,
- Her işlev alanı için gerekecek işgücü ve
- Toplam zaman

# Tasarım Süreci



Tasarım aşamasında, değişiklikten etkilenebilecek tüm proje bilgi ve belgeleri üzerinde çalışma yapılip söz konusu bilgi ve belgeler değişiklikle ilgili olarak günlenir.

# Tasarım Süreci

## Girdi

Tasarım çalışmasının girdileri:

- Çözümleme çalışması çıktıları
  1. Ayrıntılı çözümleme,
  2. Günlenmiş gereksinim tanımları,
  3. Başlangıç değişiklik listesi,
  4. Sınama stratejisi,
  5. Gerçekleştirim planı
- Sistem ve proje belgeleri ve
- Varolan kaynak kodları açıklamalar ve veritabanları

biçimindedir.

# Tasarım Süreci

## İşlem/Süreç

Tasarım için gerekli temel işlemler aşağıda belirtilmektedir.

- Etkilenen yazılım modüllerinin tanımlanması,
- Yazılım modül belgelerinin değiştirilmesi,
- Yeni tasarım için, güvenlik ve emniyet konularını da içeren sınama senaryolarının hazırlanması,
- İlişki sınamalarının tanımlanması,
- Kullanıcı belgelerinin günleme gereksinimlerinin tanımlanması
- Değişiklik listesinin günlenmesi

# Tasarım Süreci

## Denetim

Tasarım çalışmasında aşağıdaki denetim ortamı kurulmalıdır.

Tasarımın belirlenen standartlara uygunluğunun denetlenmesi,

- Güvenlik ve emniyet konularını içeren yeni tasarım belgesinin tasarım belgesinin ve bilgilerinin oluşturulmasının sağlanması,
- Sınama bilgilerinin günlenmesinin sağlanması,
- Gereksinimlerden tasarıma izlenebilirliğin sağlanması.

# Tasarım Süreci

## Çıktı

Bakım tasarımı çalışmasının çıktıları:

- Gözden geçirilmiş değişiklik listesi,
- Günlenmiş tasarım
- Günlenmiş sınama planları,
- Günlenmiş ayrıntılı çözümleme,
- Günlenmiş gereksinimler,
- Gözden geçirilmiş gerçekleştirim planı
- Risk ve kısıtlar listesi

biçimindedir.

# Tasarım Süreci

## Ölçüt

Tasarım çalışması için kullanılabilen ölçütler aşağıda verilmektedir.

- ✓ Yazılım karmaşıklığı,
- ✓ Tasarım değişiklikleri
- ✓ Her işlev alanı için gerekecek işgücü,
- ✓ Toplam zaman,
- ✓ Sınama yönerge ve plan değişiklikleri,
- ✓ Önceliklendirmedeki hata oranları,
- ✓ Varolan kodda, eklenen, çıkarılan ve değiştirilen satır sayısı,
- ✓ Uygulama sayısı.

# Gerçekleştirim Süreci



Gerçekleştirim süreci, temel olarak tasarım çıktılarını ve kaynak kodları girdi olarak almakta ve değişiklik isteğini gerçekleştiren kod parçaları ile günlenmiş yazılım kodlarını üretmektedir. Günlenmiş yazılıma ilişkin sınama bilgi ve belgelerinin ve eğitim belgelerinin üretimi de bu süreçte yapılmaktadır

# Gerçekleştirim Süreci

## Girdi

Gerçekleştirim sürecinin girdileri:

- Tasarım çalışması sonuçları,
- Varolan kaynak kodlar, açıklamalar, belgeler ve
- Proje ve sistem belgeleri,  
biçimindedir.

# Gerçekleştirim Süreci

## İşlem/Süreç

Gerçekleştirim sürecinin dört ana işlemi vardır:

- Kodlama ve birim sınama
- Bütünleştirme, Risk çözümleme
- Sınama hazırlığı gözden geçirme
- Kodlama işleminde, değişiklik isteğini karşılayan yazılım kodları, varolan yazılıma eklenmektedir. İşlem sonucunda elde edilen yeni, değişmiş modüllere birim sınama uygulanmaktadır. Birim sınama işlemini, bütürleştirme sınama izlemekte, tüm sistem yeniden sınanmaktadır.Uygulamadaki riskleri gidermek amacıyla, gerçekleştirim aşamasında sürekli risk çözümleme yapılmaktadır.

# Gerçekleştirim Süreci

## Denetim

Gerçekleştirim sürecinde oluşturulacak denetim yapısı, aşağıdaki özellikleri sağlamalıdır:

- Belirlenen standartlara uygun olarak kod ve yazılım gözden geçirmeleri yapılması,
- Birim ve bütünlendirme sınavları ile ilgili bilgilerin derlenmesi ve kaydedilmesinin sağlanması,
- Sınamaya belgelerinin günlenmesi ve oluşturulmasının sağlanması,
- Sınamaya hazırlık gözden geçirmeleri sırasında risk çözümleminin sağlanması,
- Yeni yazılımın, yazılım ortam yönetimi altında kaydedilmesi ve denetlenmesinin sağlanması,
- Teknik ve eğitim belgelerinin günlenmesinin sağlanması,
- Tasarımdan koda izlenebilirliğin sağlanması

# Gerçekleştirim Süreci

## Çıktı

Gerçekleştirim süreci aşağıdaki çıktıları vermelidir:

- Günlenmiş Yazılım,
- Günlenmiş tasarım bilgi/belgeleri,
- Günlenmiş sınama belgeleri,
- Günlenmiş kullanıcı belgeleri,
- Günlenmiş eğitim kılavuzları,
- Riskler ve kullanıcılara etkileri,
- Sınama hazırlığı gözden geçirme rapor.

## Ölçüt

Gerçekleştirim çalışmasında kullanılabilecek ölçütler:  
Değişiklik oranı ve Hata oranı biçimindedir.

# Sistem Sınaması Süreci



Değişikliklerin varolan yazılıma yansıtılmasından sonra elde edilen yeni yazılım sürümünün belirlenen standartlara uygun olarak tamamen bütünlük sistemi üzerinde sınamaların yapılması gerekmektedir. Sistem sınamalarının, kullanıcı ve üretici ekiplerin tanıklığında bağımsız bir yapı tarafından gerçekleştirilmeleri önerilmektedir.

# Sistem Sınaması Süreci

## Girdi

Sistem sınavama sürecinin girdileri:

- a.** Sınamaya hazırlık raporu
- b.** Belgeler

- Sistem sınavama planları,
- Sistem sınavamları,
- Sistem sınavama yönergeleri,
- Kullanıcı kılavuzları,
- Tasarım

- c.** Günlenmiş sistem biçimindedir.

# Sistem Sınaması Süreci

## İşlem/Süreç

Sistem sınavama, tümüyle bütünsel bir sistem üzerinde yapılmalıdır. Bu aşamada, işlevsel sistem sınavama, arayüz sınavama, regresyon sınavama ve sınavama hazırlık raporunun gözden geçirilmesi işlemleri yapılır.

Denetim: Sistem sınavaları, üretici ve kullanıcılarından bağımsız bir grup tarafından gerçekleştirilmelidir. Yazılım kodları ve her türlü bilgi belge, yazılım ortam yönetimi tarafından saklanır.

# Sistem Sınaması Süreci

## Denetim

Gerçekleştirim sürecinde oluşturulacak denetim yapısı, aşağıdaki özelliklerini sağlamalıdır:

- Belirlenen standartlara uygun olarak kod ve yazılım gözden geçirmeleri yapılması,
- Birim ve bütünlendirme sınavları ile ilgili bilgilerin derlenmesi ve kaydedilmesinin sağlanması,
- Sınamaların günlenmesi ve oluşturulmasının sağlanması,
- Sınamaların hazırlık gözden geçirmeleri sırasında risk çözümlemenin sağlanması,
- Yeni yazılımin, yazılım ortam yönetimi altında kaydedilmesi ve denetlenmesinin sağlanması,
- Teknik ve eğitim belgelerinin günlenmesinin sağlanması,
- Tasarımdan koda izlenebilirliğin sağlanması

# Sistem Sınaması Süreci

## Çıktı

Gerçekleştirim süreci aşağıdaki çıktıları vermelidir:

- Günlenmiş Yazılım,
- Günlenmiş tasarım bilgi/belgeleri,
- Günlenmiş sınavma belgeleri,
- Günlenmiş kullanıcı belgeleri,
- Günlenmiş eğitim kılavuzları,
- Riskler ve kullanıcılar etkileri,
- Sınamaya hazırlığı gözden geçirme rapor.

## Ölçüt

Gerçekleştirim çalışmasında kullanılabilen ölçütler:

- Değişiklik oranı ve
- Hata oranı  
biçimindedir.

# Kabul Sınaması Süreci

- Kabul sınavası süreci, kullanıcılar ya da kullanıcı temsilcileri tarafından gerçekleştirilen bir süreçtir. Kullanıcıların, değişiklikleri içeren yeni yazılımı sınavaları ve kabul etmeleri beklenmektedir



# Kabul Sınaması Süreci

## Girdi

Kabul sınavma sürecinin girdileri:

- a. Gözden geçirilmiş sınavma hazırlık raporu,
- b. Tümüyle bütünsel sistem,
- c. Kabul sınavma planları,
- d. Kabul sınavları ve
- e. Kabul sınavma yönergeleri biçimindedir.

# Kabul Sınaması Süreci

## İşlem/Süreç

Kabul sınavası işlemleri:

- a. İşlevsel kabul sınavlarının yapılması,
- b. Birlikte çalışabilirlik sınavası,
- c. Regresyon sınavası  
biçimindedir.

# Kabul Sınaması Süreci

## Denetim

Kabul sınavları sırasında denetimi aşağıdaki işlemleri içermektedir.

- a. Kabul sınavlarının uygulanması,
- b. Sınamaların raporlanması,
- c. İşlevsel denetim yapılması,
- d. Yeni sistemin oluşturulması,
- e. Kabul sınavı belgelerinin yazılım konfigürasyonuna yerleştirilmesi.

# Kabul Sınaması Süreci

## Çıktı

Kabul sınavlarının çıktıları, yeni sistem, işlevsel konfigürasyon denetim raporu ve kabul sınavası raporudur.

## Ölçüt

Bu aşamada kullanılabilecek ölçütler: üretilen ve düzeltlen hata oranlarıdır.

# Kurulum Süreci

- Kurulum süreci, geliştirilen ya da değiştirilmiş yeni yazılım sürümünün, uygulama sahasına aktarılma işlemlerini içerir.



# Kurulum Süreci

## Girdi

Bu sürecin temel girdisi, tümüyle sıyanmış ve kabul edilmiş yeni yazılım sürümüdür.

## İşlem/Süreç

Bu sürecin işlemleri:

- a. Fiziksel ortam denetiminin yapılması,
  - b. Kullanıcıların bilgilendirilmesi,
  - c. Varolan sistemin yedeklerinin alınması,
  - d. Kullanıcı tarafından kurulum ve eğitimlerin yapılması
- biçimindedir.

# Kurulum Süreci

## Denetim

Denetim işlemleri:

- a. Fiziksel ortam denetiminin yapılması,
- b. Sistem ile ilgili bilgi ve belgelerin kullanıcıya ulaştırılması,
- c. Sürüm Tanımlama raporunun tamamlanması ve
- d. Yazılım konfigürasyon ortamına aktarımın sağlanması alanlarını içermektedir.

# Kurulum Süreci

## Çıktı

Bu sürecin temel çıktıları, fiziksel ortam denetim raporu ve Sürüm tanımlama raporudur.

## Ölçüt

Bu süreçte kullanılabilecek ölçüt, belgeleme değişiklikleridir.

# Alınan Dersler

1. Kurulum planlamasında mevsimsel koşulların dikkate alınması önemlidir. Örneğin, kış mevsiminde yoğun kar alan doğu bölgelerindeki sahalara kurulum zaman zaman olanaksızdır. Bu nedenle, kurulum planlamasında, bu tür bölgeler bahar ya da yaz ayları içerisinde planlanmalıdır.

# Alınan Dersler

2. Kurulum, yalnızca uygulama yazılımı sürümlerinin yüklenmesini içermemekte, zaman zaman teknolojik alt yapı değişimlerinde ve yeni sistem yazılımı sürümlerinin yüklenmesi gerekebilmektedir. Bu nedenle, kurulum elemanlarının, gerekli teknik bilgilerle donatılması ve teknolojik değişimleri izlemelerinin sağlanması gerekmektedir.

# Alınan Dersler

3. Uç kullanıcılar zaman zaman kendi bilgisayarlarına dışarıdan getirdikleri ya da internet ortamından sağladıkları yazılımları yüklerler. Virüs riski taşıyan bu tür yazılımlara karşı kullanıcıları sürekli uyarmak ve anti-virus yazılımlarının kullanımını özendirmek gerekmektedir.

# Alınan Dersler

4. Ülkemizde kullanılan yazılım üretim yöntemleri düşünüldüğünde, bilinen anlamda bakım yapılmadığı, yazılımları hazırlayan kişilerin yıllarca kendi yazdıkları yazılıma yamalar yaptıkları, yazılımı usta-çırak ilişkisi içerisinde başkalarına devretmeye çalıştıkları ve çalışan yazılıma ilişkin elde düzgün teknik belgelerin olmadığı bir ortamla karşılaşılmaktadır. Bu durum zaten pahalı olan bilişim işgütünü verimsiz kullanmamıza neden olmaktadır.

# Alınan Dersler

5. Kullanıcı ve yerinde destek elemanları arasındaki tüm iletişimin kayıt altına alınmasının sağlanması ve izlenmesi gerekmektedir.