



@



Yazılım Mühendisliği

ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI



HEDEFLER

- ✓ Yazılım, program ve algoritma kavramları anlar.
- ✓ Yazılım ve donanım maliyetlerinin zamansal değişimlerini ve nedenleri hakkında yorum yapar.
- ✓ Yazılım mühendisliği ile Bilgisayar mühendisliği kavramlarını karşılaştırabilir.
- ✓ Yazılım yaşam döngüsünün çekirdek süreçlerini bilir.
- ✓ Yazılım geliştirme sürecindeki maliyet ve hataları süreçlere dağıtabilir.



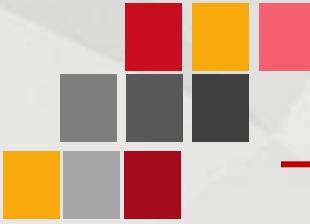
YAZILIM

Yazılım, elektronik aygıtların belirli bir işi yapmasını sağlayan **programların** tümüne verilen isimdir. Bir başka deyişle, var olan bir problemi çözmek amacıyla bilgisayar (programlama) dili kullanılarak oluşturulmuş **anlamlı anlatımlar** bütünüdür.

Yazılım için çeşitli programlama dilleri mevcuttur. Tarihsel gelişimlerine göre;

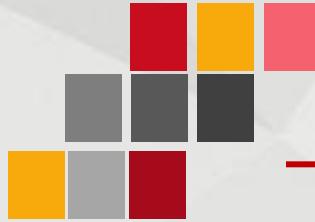
- Makine dili (0, 1)
- Assembly (put, move, save)
- Fortran (goto, if, do)
- C (printf, scanf)
- Java (metod, class, package)

Şeklinde özetlenebilirse de günümüzde yüzden fazla programlama dili olduğu söylenebilir. Bu kadar çok programlama dili olmasının nedeni, ilgilenilen problemin içерdiği kavramların problemin çözümünde kullanılacak programlama dilinde kolay tanımlanabilmesinin istenmesidir.



Programlama Dili

- ✓ **Programlama dili**, yazılımcının bir **algoritmayı** ifade etmek amacıyla, bir bilgisayara ne yapmasını istediğini anlatmasının tektipleştirilmiş yoludur.
- ✓ Programlama dilleri, yazılımcının bilgisayara hangi veri üzerinde işlem yapacağını, verinin nasıl depolanıp iletileceğini, hangi koşullarda hangi işlemlerin yapılacağını tam olarak anlatmasını sağlar.

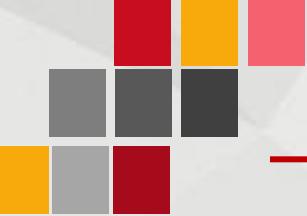


Algoritma

Algoritma, belli bir problemi çözmek veya belirli bir amaca ulaşmak için tasarlanan yol. Matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler kümesidir.

Örneğin klavyeden girilen iki sayının toplamını bulan ve sonucu ekrana yazdıran programın algoritması ve akış diyagramı,

Değişkenler: x, y, toplam

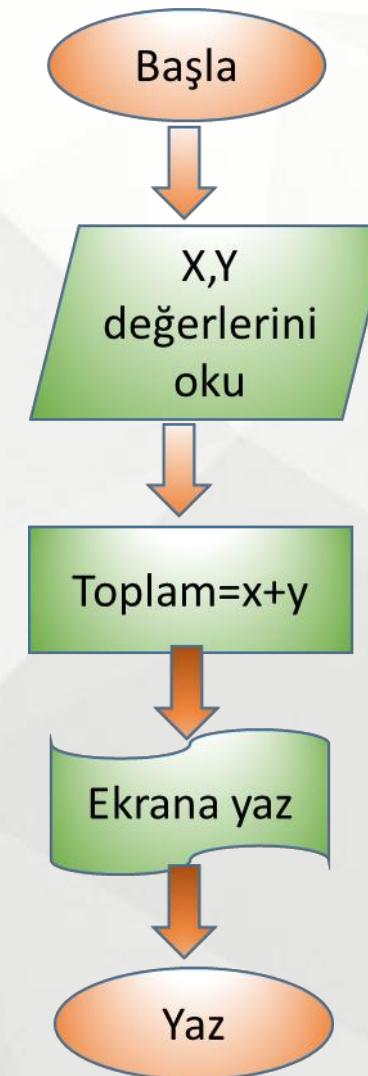


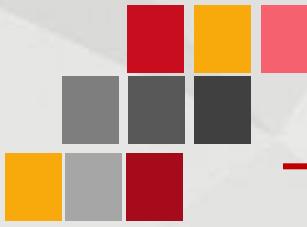
Algoritma --> Gösterim(AKİŞ diyagramı)

Algoritma

1. Başla
2. x değişkeninin değerini oku.
3. y değişkeninin değerini oku.
4. x ve y sayılarını topla sonucu toplam değişkenine aktar.
5. Toplam değerini ekranaya yazdır.
6. Dur

AKİŞ diyagramı

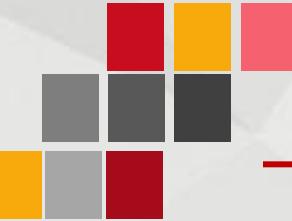




Yazılım Nedir ?

Yazılım, elektronik aygıtların belirli bir işi yapmasını sağlayan programların tümüne verilen isimdir. Bu tanımı içерdiği bileşenler cinsinden,

Yazılım= programlama dili + algoritma+ belge+ insan +....
yazılabilir.

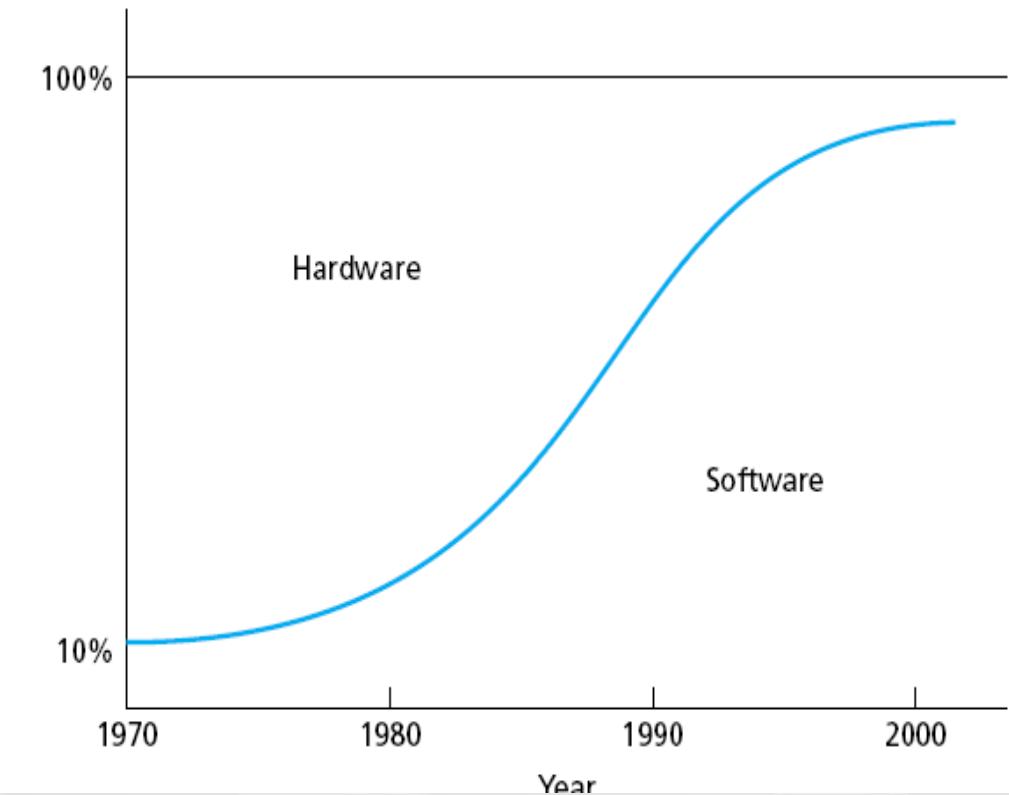


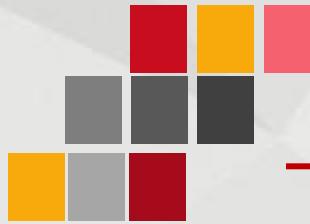
Yazılım Donanım Karşılaştırılması

Donanım, programlanabilir elektronik birim.
Donanım bir kez geliştirildikten fabrikalarda
çoğaltırlar.

Yazılım her bir müşteri problemi için
geliştirilmek zorundadır. ([Paket Programlar
nereye konacak!!!!!!](#))

Donanım – Yazılım fiyatlarının değişimi yan
tarafta verilmiştir. Günümüzde yazılım
geliştirme fiyatları toplam sistem fiyatının
%95'lik bölümünü oluşturmaktadır.





Yazılım Türleri

Bilgisayar yazılımları genel olarak 2 ana grupta incelenebilir.

1- Sistem Yazılımları:

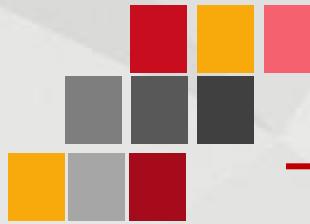
Bilgisayarın kendisinin işletilmesini sağlayan, işletim sistemi, derleyiciler (yazılan programı makine diline çeviren program) ve çeşitli yardımcı yazılımlardır.

Bütün sistem programları içinde en temel yazılım işletim sistemidir ki, bilgisayarın bütün donanım ve yazılım kaynaklarını kontrol ettiği gibi, kullanıcılarla ait uygulama yazılımlarının da çalıştırılmalarını ve denetlenmelerini sağlar.

2- Uygulama Yazılımları:

Kullanıcıların işlerine çözüm sağlayan örneğin çek, senet, stok kontrol, bordro, kütüphane kayıtlarını tutan programlar, bankalardaki müşterilerin para hesaplarını tutan programlar vs. gibi yazılımlardır.

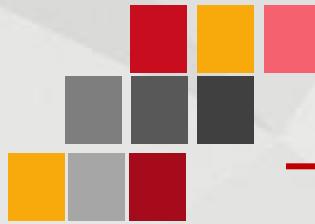
✓ Bazen bunlara ek olarak uygulama yazılımı geliştirmede kolaylık sağlamak amacıyla geliştirilmiş kütüphane yazılımları ek olarak verilebilmektedir.



Yazılım Mühendisliği Nedir?

Yazılım Mühendisliği; sistemli ve ölçülebilir bir yaklaşımın yazılımın geliştirilmesinde, işletilmesinde ve bakımında uygulanmasıdır. Özette, mühendisliğin yazılıma uygulanmasıdır. Yazılım geliştirmek, dışarıdan bakıldığından “bilene kolay” gibi görünse de karmaşık yazılımların geliştirilmesi ve var olan sistemlere entegrasyonu mühendislik eğitimini gereklilik kılmuştur.

Mühendislik, herhangi bir bilim alanındaki teoriyi sistematik olarak pratiğe geçirmeyi hedefler ve bilim ile matematiği kullanır. Yazılım da uygulanan mühendislik: Yapılacak işi planlamayla başlar ve geliştirilen sistemi kullanma esnasındaki bakıma kadar uzanan tüm etkinlikleri kapsar. Sadece teknik etkinlikle de değil, yönetim etkinliklerini de içerir.



Bilgisayar Mühendisliği – Yazılım Mühendisliği

Bilgisayar Mühendisliği

Algoritmalar

Hesaplama Kuramları

Derleyiciler

İşletim sistemleri

Yazılım Geliştirme

Yazılım Mühendisliği

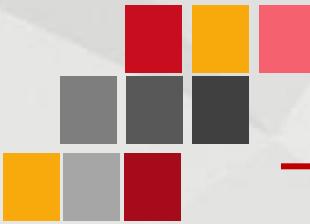
Yazılım mimarisi

Proje Yönetimi

Teknik Planlama

Risk Yönetimi

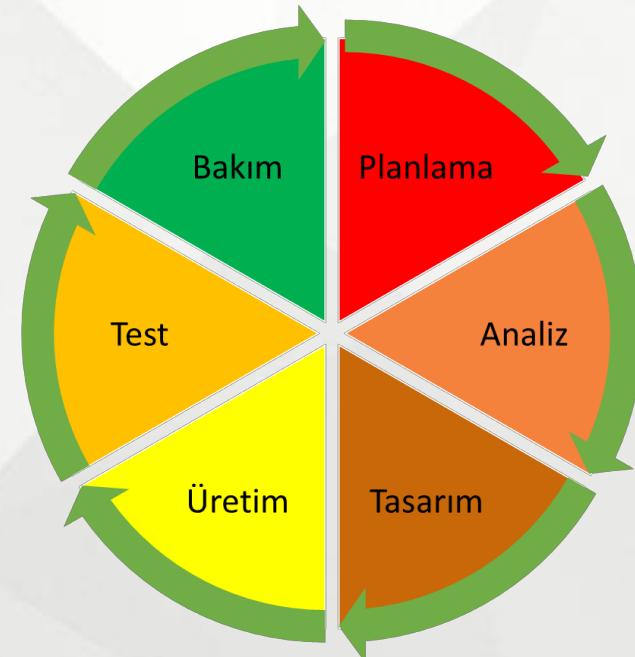
Yazılım Kalitesi ve Güvenliği



Yazılım Yaşam Döngüsü

Yazılım Mühendisliği, bilgisayar mühendisliğinin ilgi alanlarından biri olan yazılım geliştirme alanı üzerinde gelişmektedir. Yazılım Mühendisliğinde yazılım doğrusal değil döngüsel bir süreç olarak kabul edilir ve yazılım yaşam döngüsü kavramıyla ifade edilir.

Yazılım Yaşam döngüsü çoğunlukla 5 veya 6 çekirdek süreçten oluşturulur. Yazılım karmaşıklığı arttıkça üretim ile test süreçlerini birbirinden ayırmaya eğilimi artmaktadır.

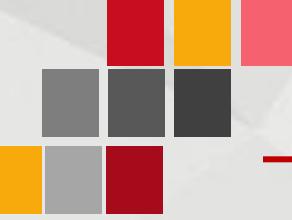




Çekirdek Süreçler

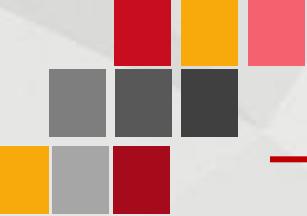
Planlama: Temel ihtiyaçlar belirlenir, proje için **fizibilite** çalışmaları yapılır (maliyetlerin ve sistemin yararlarının tanımlanması) ve proje planlaması gerçekleştirilir.

Analiz: Sistemin işlevlerini ve kesin gereksinimleri açıklığa kavuşturarak dokümante etmektir. Bu çalışma müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi vb. rollerin bir araya geldiği gruplar tarafından yapılabilir. İhtiyaçların net olmadığı durumlarda yazılım mühendisi ve müşteri arasında iletişim ve birlikte çalışmanın çok daha fazla olması gereklidir. Çeşitli yazılım geliştirme metodolojilerinde bu aşamada kullanım dokümanları ve test plan dokümanları da oluşturulabilir.



Çekirdek Süreçler

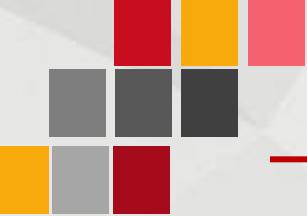
Tasarım: Yazılım ürün tasarımı, müşterinin gereksinim ve isteklerini karşılamak üzere yazılım ürününün özellikleri, yetenekleri, ve arayüzlerinin belirlenmesi etkinliğidir. İki tür tasarımdan bahsetmek mümkündür (Yüksek düzeyde tasarım – Mimari tasarım ve Detay tasarım). Mimari tasarım, yazılım modüllerinin genel yapıları ve organizasyon içerisindeki etkileşimleri ile ilgilenir. Detay tasarım aşamasında Mimari tasarım dokümanları genelde revize edilirler. Tasarım ve analiz aşamalarının ayırmı “Problem **Ne?**/Problem **Nasıl** Çözülür?” sorularının kullanımı ile ilgilidir. Gereksinimlerin belirlendiği analiz aşaması problemin ne olduğu ile ilgilidir.



Çekirdek Süreçler

Gerçekleştirim (Kodlama ve Test)Tasarım aşamasının belirli bir olgunluğa ulaşmasıyla birlikte **Kodlama** aşaması başlar. Müşteriye teslim edilecek ürünü programlama aşamasıdır.

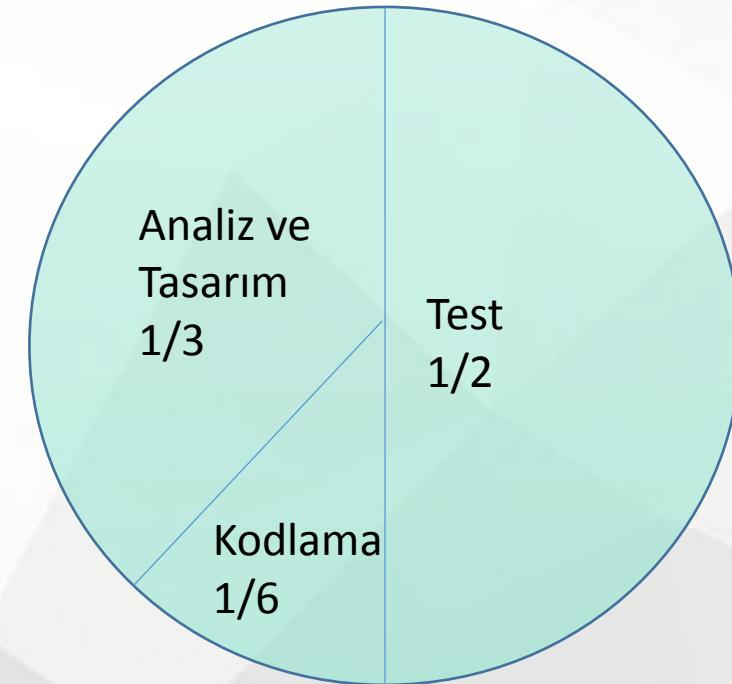
Bakım Teslim ile birlikte bakım aşaması da başlar. Hata giderici, önleyici, altyapıyı iyileştirici, ürüne yeni özellikler ekletici gibi farklı bakım faaliyetleri mevcuttur.

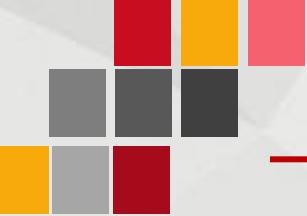


Yazılım Geliştirme Süreçlerinin Maliyet Oranları

Yazılımın yaşam döngüsünün beş yıllık toplam maliyetinin %70'e yakınını bakım süreci oluşturmaktadır.

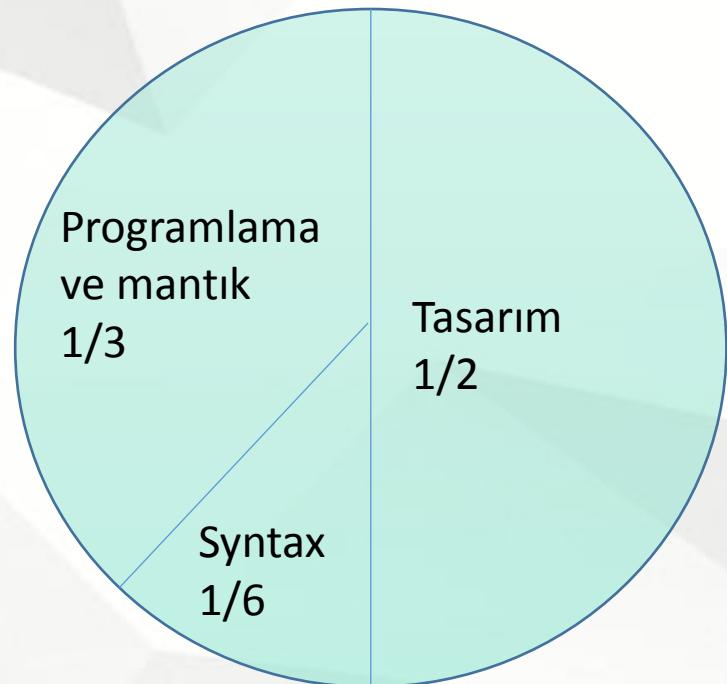
Yazılım geliştirme sürecine ait alt süreçlerin maliyet oranları yan tarafta verilmiştir.



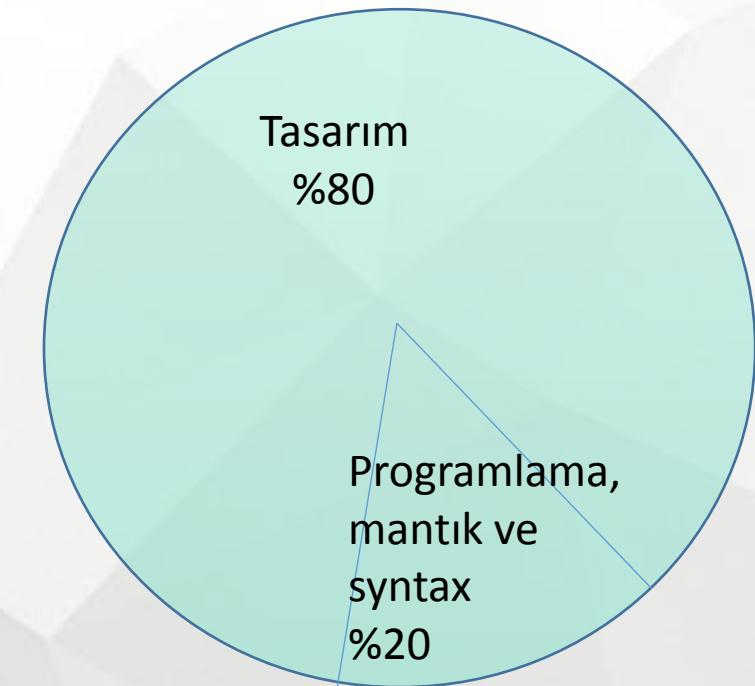


Yazılım Geliştirmede Hataların Süreçlere Dağılımı

- ✓ Geliştirmede yapılan hataların süreçlere dağılımı



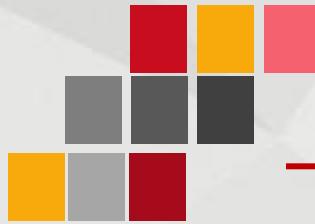
- ✓ Hata düzeltme maliyet oranları





Sorular

1. Yazılım ile programlama kavramlarını tanımlayınız.
2. Algoritma nedir? Bir algoritmanın gerçeklemesinde kullanılacak dilin seçimi neden önemlidir?
3. Yazılım Mühendisi ile bilgisayar mühendislerinin temel çalışma alanlarını yazınız.
4. Büyük çaplı bir yazılım projesinin geliştirilmesinde yalnızca yazılım mühendislerinin bulunması yeterli olur mu?
5. Yazılı yaşam döngüsünün çekirdek süreçleri nelerdir?
6. Analiz ve tasarım süreçleri arasındaki fark nedir?
7. Yazılım + donanım sistemi ediniminin yıllara göre maliyet değişimi nasıldır? Grafiği açıklayınız.
8. Analiz ve tasarım süreçleri yazılım geliştirmenin ne kadarlık zamanında yapılır.



Önümüzdeki Hafta

Süreç modelleri

1. Barok Model
2. Şelale modeli
3. V-süreç modeli
4. Helezonik Model
5. Artımsal Geliştirme modeli
6. Artımsal geliştirme modeli

Metodoloji Nedir?

Metodolojiler ile süreç modellerinin karşılaştırılması



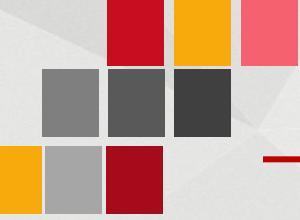
@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği Süreç Modelleri

Doç. Dr. Recep ERYİĞİT
₁



HEDEFLER

- ✓ Yazılım Yaşam Döngüsü (Çekirdek Süreçler)
- ✓ Belirtim Yöntemleri –Süreç Modelleri
- ✓ Yazılım Süreç Modelleri
- ✓ Gelişgüzell Model
- ✓ Barok Model
- ✓ Çağlayan Modeli
- ✓ V Modeli
- ✓ Spiral Model
- ✓ Evrimsel Geliştirme Süreç Modeli
- ✓ Artırımsal Geliştirme Süreç Modeli
- ✓ Araştırma Tabanlı Süreç Modeli
- ✓ Metodolojiler



Yazılım Yaşam Döngüsü

1. Planlama

Personel ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşamadır.

2. Analiz

Sistem gereksinimlerinin ve işlevlerinin ayrıntılı olarak çıkarıldığı aşama. Var olan işler incelenir, temel sorunlar ortaya çıkarılır.

mantıksal; önerilen sistemin yapısı anlatılır,

3. Tasarım

Belirlenen gereksinimlere yanıt verecek yazılım sisteminin temel yapısının oluşturulduğu aşamadır.

mantıksal; önerilen sistemin yapısı anlatılır,

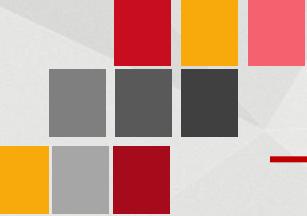
fiziksel; yazılımı içeren bileşenler ve bunların ayrıntıları.

4. Gerçekleştirim

Kodlama, test etme ve kurulum çalışmalarının yapıldığı aşamadır.

5. Bakım

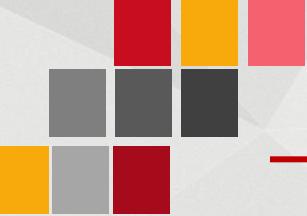
Hata giderme ve yeni eklentiler yapma aşaması.



Belirtim Yöntemleri

Bir çekirdek süreçce ilişkin fonksiyonları yerine getirmek veya çekirdek süreçler arası geçişlerin belirtilmesinde kullanılan yöntemler, **Belirtim Yöntemleri** olarak adlandırılır.

yazılım yaşam döngüsündeki çekirdek süreçlerin geliştirme aşamasında hangi sırada uygulanacağını tanımlayan modellere **Süreç Modelleri** denir.



Belirtim Yöntemleri

Süreç Akışı İçin Kullanılan Belirtim Yöntemleri

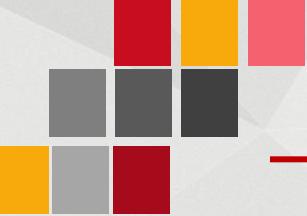
Süreçler arası ilişkilerin ve iletişimini gösterdiği yöntemler (**Veri Akış Şemaları, Yapısal Şemalar, Nesne/Sınıf Şemaları**).

Süreç Tanımlama Yöntemleri

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

Veri Tanımlama Yöntemleri

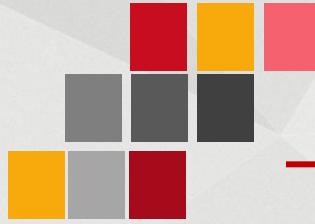
Süreçler tarafından kullanılan verilerin tanımlanması için kullanılan yöntemler (**Nesne İlişki Modeli, Veri Tabanı Tabloları, Veri Sözlüğü**).



Yazılım Süreç Modelleri

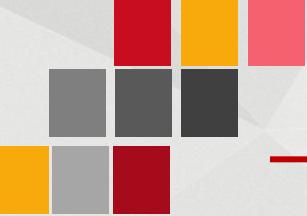
- ✓ Yazılım üretim işinin genel yapılma düzenleme ilişkin rehberlerdir.
Süreçlere ilişkin ayrıntılarla ya da süreçler arası ilişkilerle ilgilenmezler.

1. Gelişigüzel Model
2. Barok Modeli
3. Çağlayan (Şelale) Modeli
4. V Modeli
5. Spiral Model
6. Evrimsel Model
7. Artırımsal Model
8. Araştırma Tabanlı Model



Gelişigüzel Model

- ✓ Herhangi bir model ya da yöntem yok.
- ✓ Geliştiren kişiye bağlı.
- ✓ İzlenebilirliği ve bakımı oldukça zor.
- ✓ Genellikle tek kişilik üretim ortamı.
- ✓ Karmaşık olmayan yazılım sistemleri.



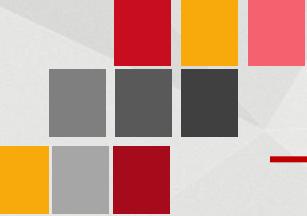
Barok Model

- ✓ Analiz
- ✓ Tasarım
- ✓ Kodlama
- ✓ Modül Testleri
- ✓ Altsistem Testleri
- ✓ Sistem Testi
- ✓ Belgeleme
- ✓ Kurulum

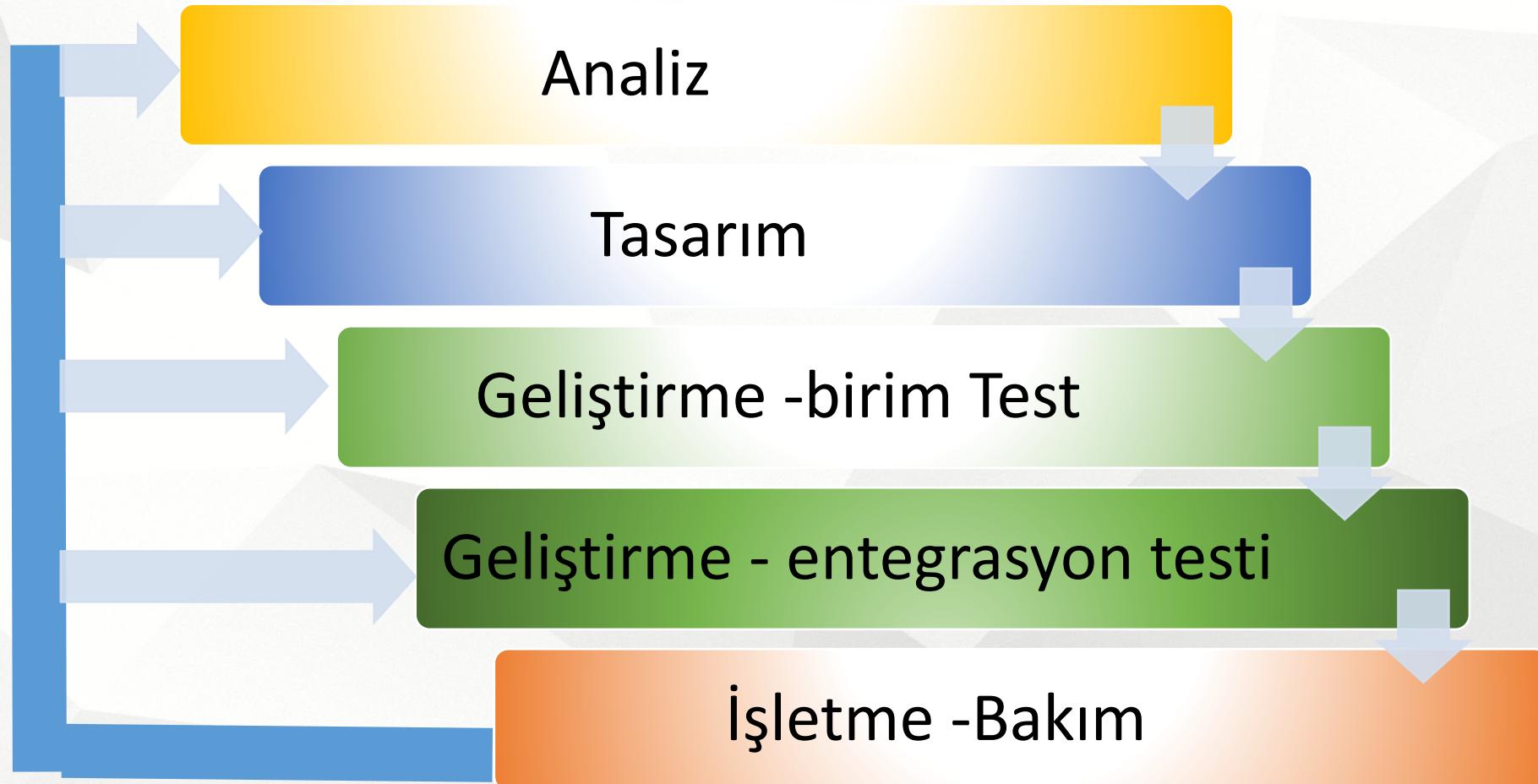
Yaşam döngüsü temel adımlarının doğrusal bir şekilde geliştirildiği model.

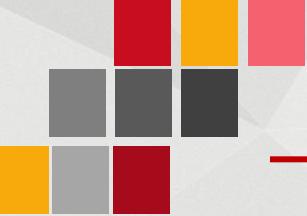
Belgelemeyi ayrı bir süreç olarak ele alır, ve yazılımın geliştirilmesi ve testinden hemen sonra yapılmasının öngörür.

Aşamalar arası geri dönüşlerin nasıl yapılacağı tanımlı değil.



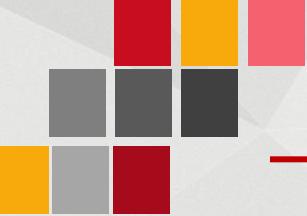
Çağlayan Modeli





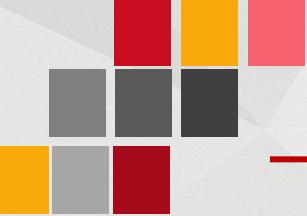
Çağlayan Modeli

- ✓ 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.

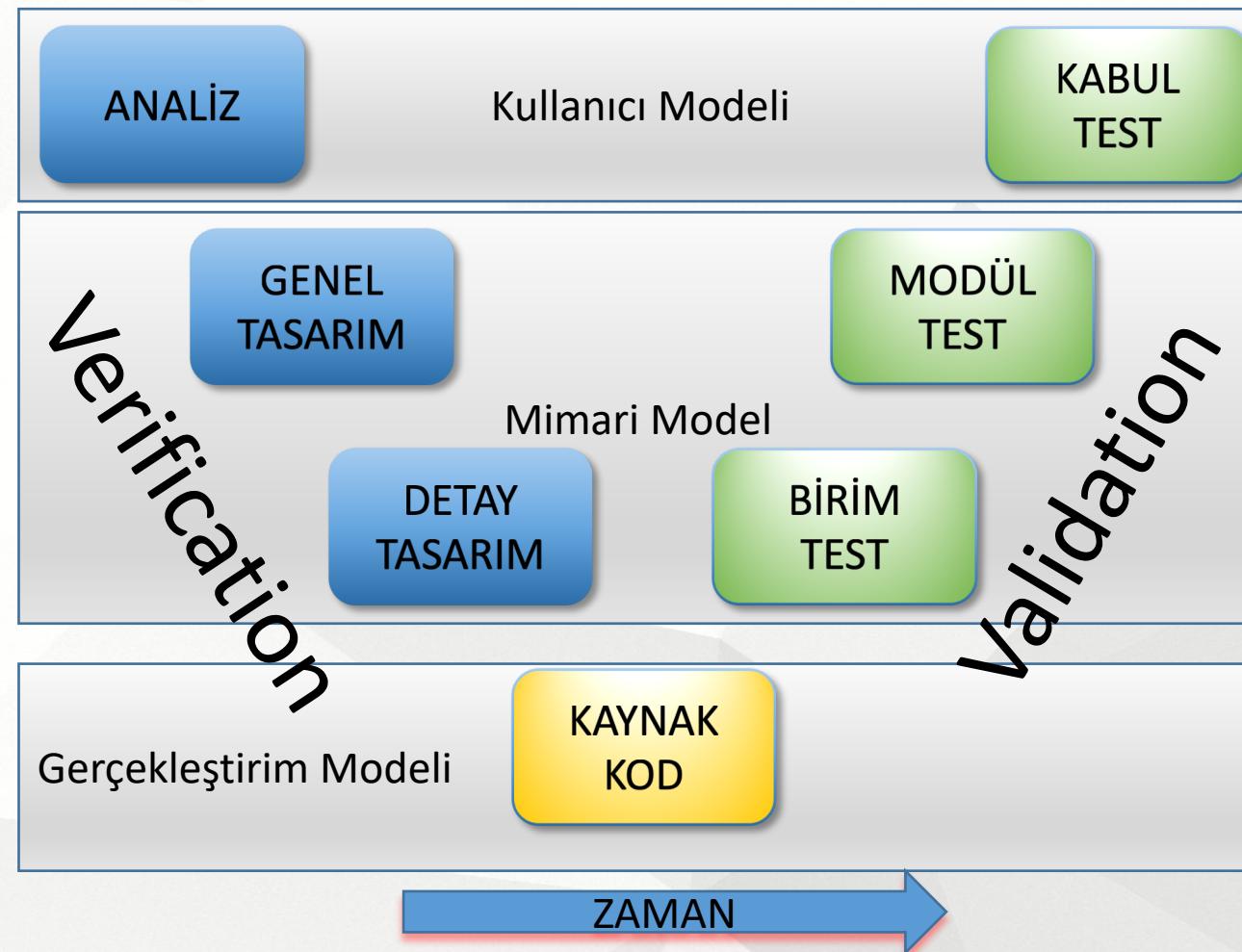


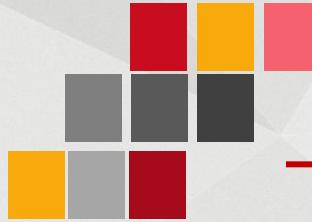
Çağlayan Modeli

- ✓ Gerçek yaşamdaki projeler genelde yineleme gerektirir.
- ✓ Genelde yazılımın kullanıcıya ulaşma zamanı uzundur.
- ✓ Gereksinim tanımlamaları çoğu kez net bir şekilde yapılamadığından dolayı, yanlışların düzeltilme ve eksiklerin giderilme maliyetleri yüksektir.
- ✓ Yazılım üretim ekipleri bir an önce program yazma, çalışma ve sonucu görme eğiliminde olduklarıdan, bu model ile yapılan üretimlerde ekip mutsuzlaşmakta ve kod yazma dışında kalan (ve iş yükünün %80'ini içeren) kesime önem vermemektedirler.
- ✓ Üst düzey yönetimlerin ürünü görme süresinin uzun oluşu, projenin bitmeyeceği ve sürekli gider merkezi haline geldiği düşüncesini yaygınlaştırmaktadır



V Süreç Modeli





V Süreç Modeli

Sol taraf üretim, sağ taraf sınama işlemleridir.

V süreç modelinin temel çıktıları;

Kullanıcı Modeli

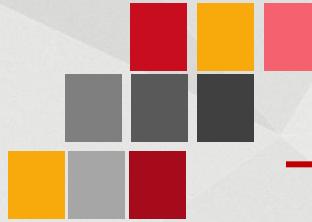
Geliştirme sürecinin kullanıcı ile olan ilişkileri tanımlanmakta ve sistemin nasıl kabul edileceğine ilişkin sınama belirtimleri ve planları ortaya çıkarılmaktadır.

Mimari Model

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

Gerçekleştirme Modeli

Yazılım modüllerinin kodlanması ve sınanmasına ilişkin fonksiyonlar.

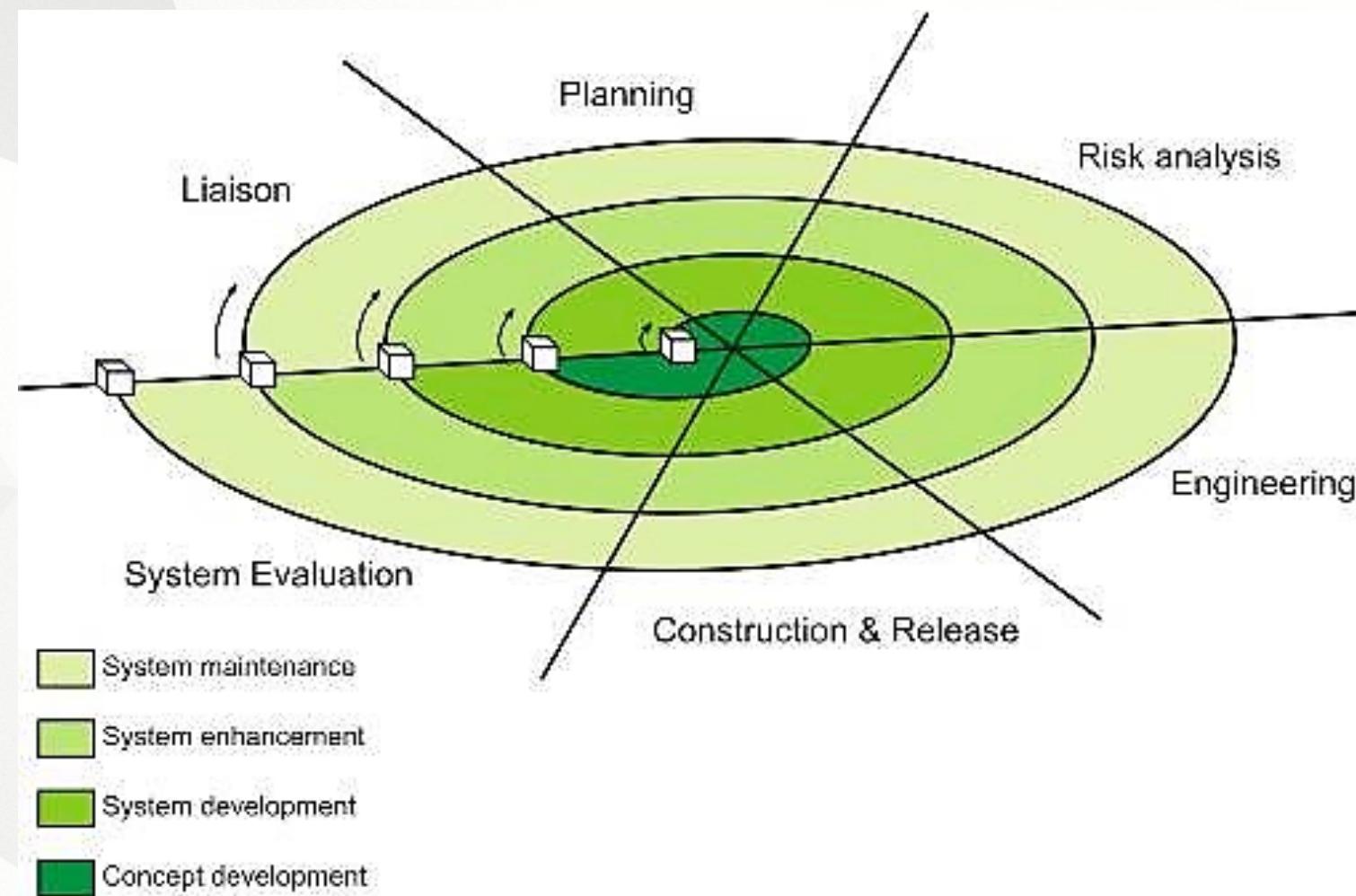


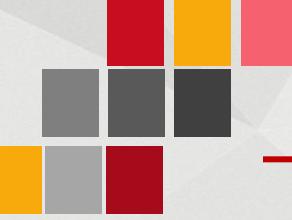
V Süreç Modeli

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



Spiral Model





Spiral Model

Tasarımı doğrusal bir süreç olarak gören diğer modellerin aksine, bu model spiral bir süreç olarak看起来. Bu, yineleyici tasarım döngülerini genişleyen bir spiral olarak temsil ederek yapılır.

Genellikle iç çevrimler, gereksinim tanımının rafine edilmesi için prototipleme ile birlikte ihtiyaç analizinin erken evresini ve dış spiraller yazılım tasarımını aşamalı olarak temsil eder.

Her helezonda, tasarım çabalarını ve bu yineleme için ilgili riski değerlendirmek için bir risk değerlendirme aşaması vardır. Her spiralin sonunda, mevcut spiralin gözden geçirilebilmesi ve bir sonraki aşamanın planlanabilmesi için gözden geçirme aşaması vardır.



Spiral Model

Her tasarım sarmalının altı ana faaliyeti altı temel görevle temsil edilmektedir:

1. Müşteri İletişimi
2. Planlama
3. Risk Analizi
4. Yazılım Tasarımı
5. Üretim-dağıtım
6. Müşteri onayı

Avantajları

1. Risk analizi yapmaktadır.
2. Bu yazılım tasarım modeli, büyük yazılım projelerini tasarlamak ve yönetmek için daha uygundur.

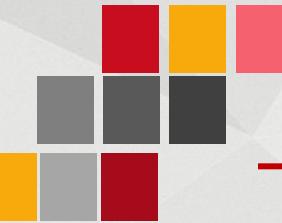
Dezavantajları

1. Risk analizi yüksek uzmanlık gerektirir.
2. Kullanması pahalı model
3. Küçük projeler için uygun değildir.

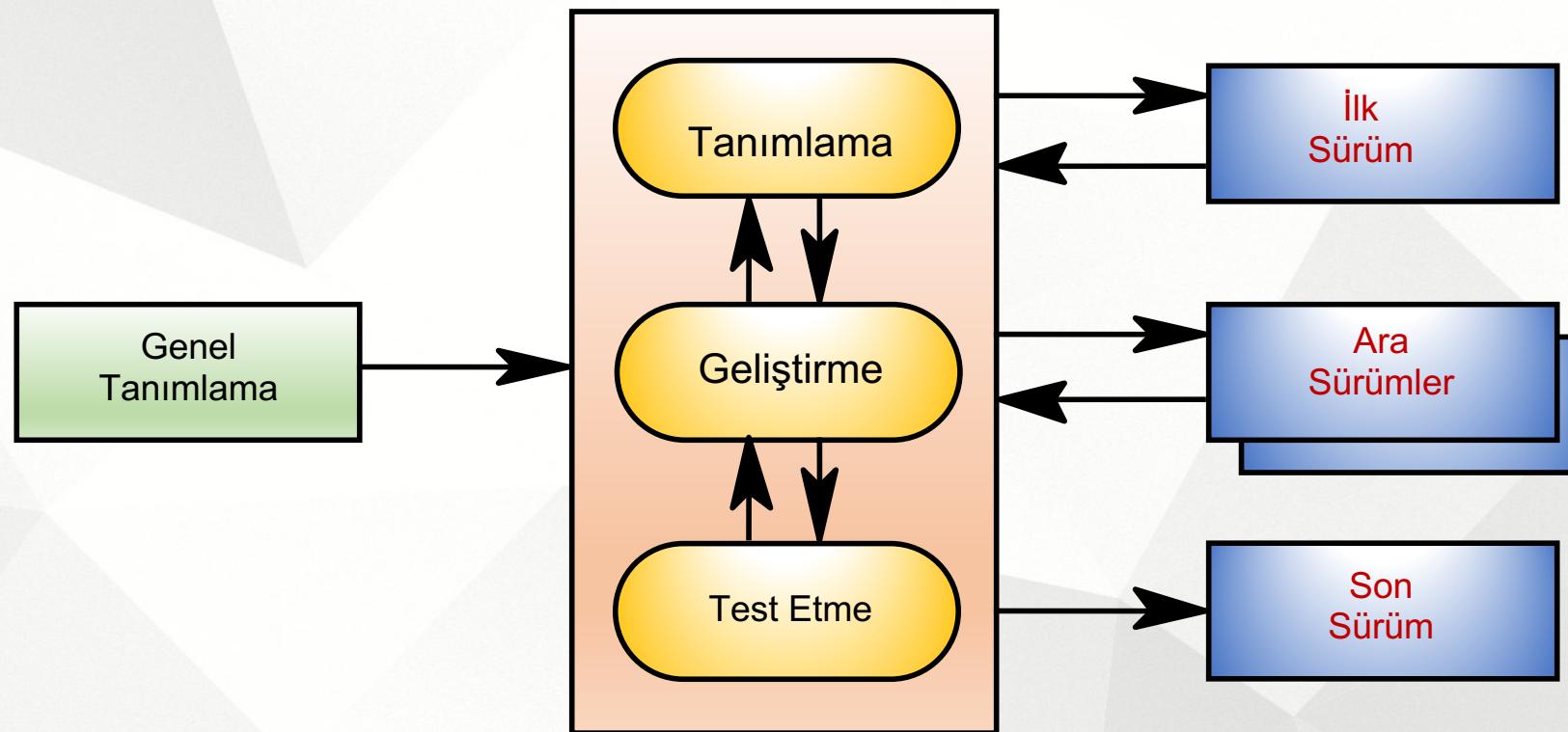


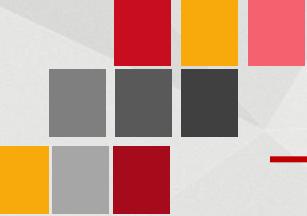
Evrimsel Geliştirme Süreç Modeli

- İlk tam ölçekli modeldir.
- Coğrafik olarak geniş alana yayılmış, çok birimli organizasyonlar için önerilmektedir.
- Her aşamada üretilen ürünler, üretildikleri alan için tam işlevselliği içermektedirler.
- Pilot uygulama kullan, test et, güncelle diğer birimlere taşı.
- Modelin başarısı ilk evrimin başarısına bağlıdır



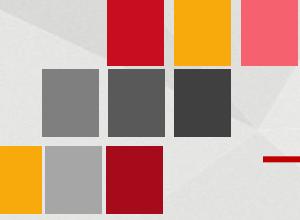
Evrimsel Geliştirme Süreç Modeli





Evrimsel Geliştirme Süreç Modeli

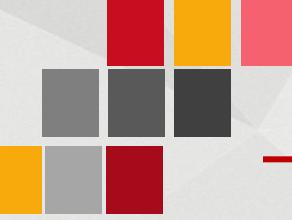
- ✓ Çok birimli banka uygulamaları.
- ✓ Önce sistem geliştirilir ve Şube-1'e yüklenir.
- ✓ Daha sonra aksaklılıklar giderilerek geliştirilen sistem Şube-2'ye yüklenir.
- ✓ Daha sonra geliştirilen sistem Şube-3'e,.... yüklenir.
- ✓ Belirli aralıklarla eski şubelerdeki güncellemeler yapılır.



EKSİKLERİ

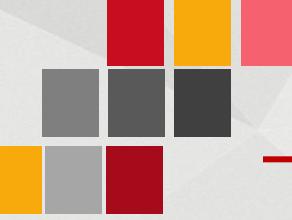
- ✓ Değişiklik denetimi

- ✓ Konfigürasyon Yönetimidir
 - Sürüm Yönetimi
 - Değişiklik Yönetimi
 - Kalite Yönetimi

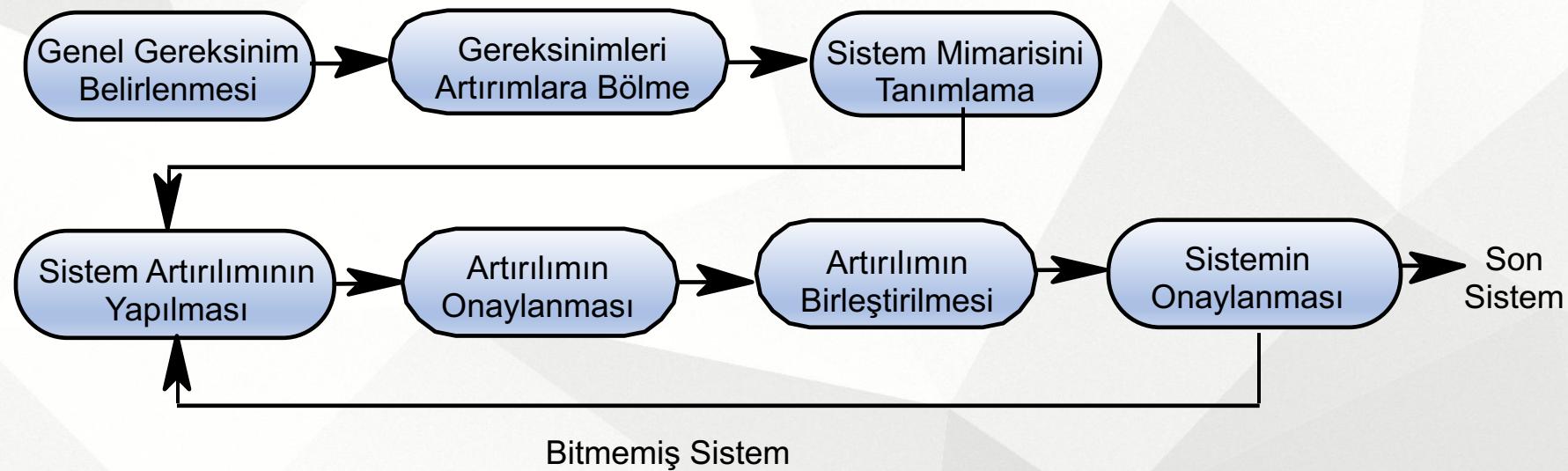


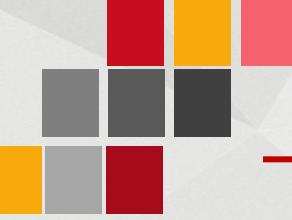
Artırımsal Geliştirme Süreç Modeli

- ✓ 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



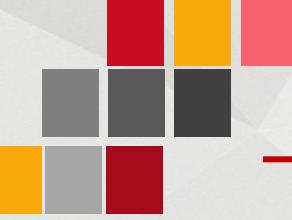
Artırımsal Geliştirme Süreç Modeli





Araştırma Tabanlı Süreç Modeli

- ✓ Yap-at prototipi olarak ta 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.



Metodolojiler

Metodoloji: Bir BT projesi ya da yazılım yaşam döngüsü aşamaları boyunca kullanılacak birbirleriyle uyumlu yöntemler bütünü.

Bir metodoloji,

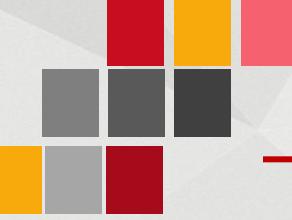
- bir süreç modelini ve
- belirli sayıda belirtim yöntemini içerir

Günümüzdeki metodolojiler; **çağlayan, V ya da spiral süreç modellerini** temel almaktadır.



Bir Metodolojide Bulunması Gereken Temel Bileşenler

- ✓ Ayrıntılandırılmış bir süreç modeli
 - ✓ Ayrıntılı süreç tanımları
 - ✓ İyi tanımlı üretim yöntemleri
 - ✓ Süreçler arası ara yüz 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 Metodolojide Bulunması Gereken Temel Bileşenler

- ✓ Metodoloji bileşenleri ile ilgili olarak bağımsız kuruluş (IEEE, ISO, vs.) ve kişiler tarafından geliştirilmiş çeşitli standartlar ve rehberler mevcuttur.
- ✓ Kullanılan süreç modelleri ve belirtim yöntemleri zaman içinde değiştiği için standart ve rehberler de sürekli güncellenmektedir.



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 tasarımı	Başlangıç Tasarım Raporu
Tasarım	Yapısal Şemalar, Program Tasarım Dili, Veri Tabanı Tabloları	Genel Tasarım Ayrıntılı Tasarım Veri Tasarımı	Sistem Tasarım Raporu



@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

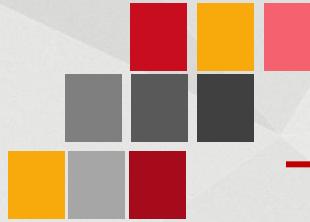
Yazılım Mühendisliği Temel Süreçler - PLANLAMA

Doç. Dr. Recep ERYİĞİT
₁



HEDEFLER

- ✓ Proje kaynakları
- İnsan, donanım ve yazılım kaynakları



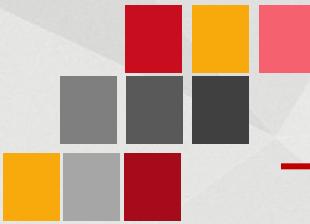
Planlama

Proje planlama aşamasında yapılan işlemler

- ❖ Kaynakların Belirlenmesi
- ❖ Maliyetlerin Kestirilmesi
- ❖ Proje Ekip Yapısının Oluşturulması
- ❖ Ayrıntılı Proje Planının Yapılması
- ❖ Projenin İzlenme Yönteminin Belirlenmesi

Çıktı: **Proje Planı**

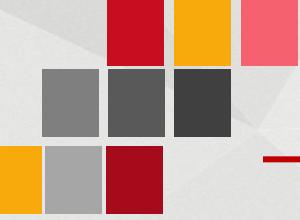
Proje planı, tüm proje süresince güncellenenerek, kaynak kullanım planlarının doğruluğu gözlemlenir.



Proje Kaynakları

- ✓ İnsan Kaynakları
- ✓ Donanım Kaynakları
- ✓ Yazılım Kaynakları

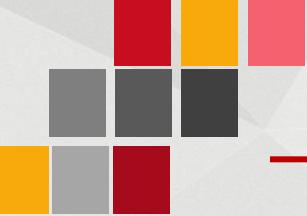
Planlama; bu kaynakların tanımını yapar ve zaman kullanımı, görev süreleri, edinilme zamanlarını planlar



İnsan Kaynakları

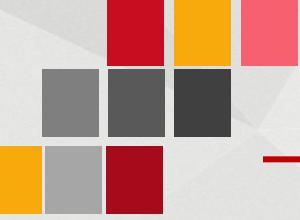
- ✓ Planlama; hangi tür elemanların, hangi süre ile ve projenin hangi aşamalarında yer alacağını belirler

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

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



✓ Test araçları

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

✓ Prototipleme ve simülasyon araçları

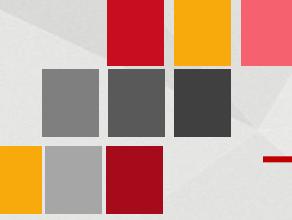
- Geliştirmenin erken aşamalarında kullanıcıya, sonuç ürünün çalışması ile ilgili fikir veren ve yönlendiren araçlar.

✓ 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

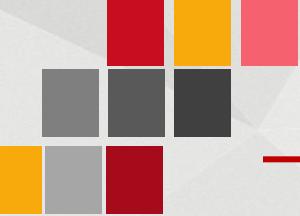
- ✓ **Maliyet kestirimi;** bir bilgi sistemi ya da yazılım için gerekebilecek iş gücü ve zaman maliyetlerinin üretimden önce belirlenebilmesi için yapılan işlemlerdir.

- ✓ **Kullanılan Unsurlar**
 - Geçmiş projelere ilişkin bilgiler
 - Proje ekibinin deneyimleri
 - İzlenen geliştirme modelibirden çok kez uygulanabilir



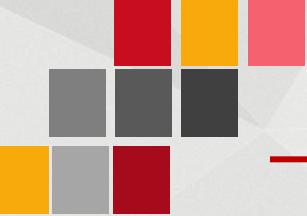
Maliyet yönetimi sayesinde;

- ✓ 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

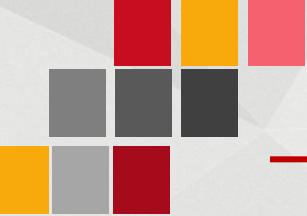
- Proje büyüklüğünü kestiren yöntemler
- Proje zaman ve iş gücünü kestiren yöntemler

2. Projelerin büyüklüğüne göre

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

3. Uygulanış biçimlerine göre

- Yalın düzeyde
- Orta düzeyde
- Ayrıntılı düzeyde



Maliyet Kestirim Yöntemleri

. Değişik aşamalarda kullanılabilirlik

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

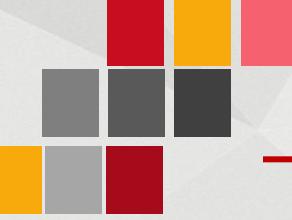
5. Yöntemlerin yapılarına göre

- Uzman deneyimine gereksinim duyan
- Önceki projelerdeki bilgileri kullanan yöntemler



İşlev Noktaları Yöntemi

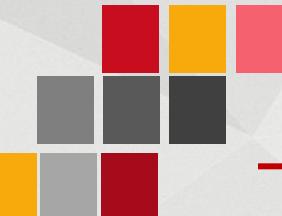
- ✓ İşlev noktaları geliştirmenin erken aşamalarında (analiz aşamasında) saptanan bir değerdir.
- ✓ Sistemin oluşturulduğu ortamdan bağımsız elde edilir.
- ✓ Problem tanımı girdi olarak alınarak üç temel adım izlenir:
 - Problemin bilgi ortamının incelenmesi
 - Problemin teknik karmaşıklığının incelenmesi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



Problemin bilgi ortamının incelenmesi

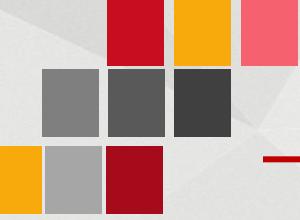
- ✓ **Kullanıcı Girdileri:** personel sicil bilgileri, personel izin bilgileri gibi
- ✓ **Kullanıcı Çıktıları:** her türlü mantıksal çıktı; raporlar, ekran çıktıları, hata iletleri,...
- ✓ **Kullanıcı Sorguları:** personel sicil bilgilerinin sorgulaması, personel maaş bilgilerinin sorgulaması
- ✓ **Dosyalar:** Her türlü mantıksal bilgi yığını, tablolar, veri tabanları
- ✓ **Dışsal arayüzler:** Başka programlarla veri传递. import/export

Yukarıda verilen durumlara ait sayılar ağırlık faktörleriyle çarpılarak toplam işlemi yapılır. Çıkan değer **Ayarlanmamış İşlev Noktası (AİN)** olarak adlandırılır.



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 Soru Sayısı	?	3	4	6	=
Kütük Sayısı	?	7	10	15	=
Dışsal Araryüz Sayısı	?	5	7	10	=
Toplam Sayı					=

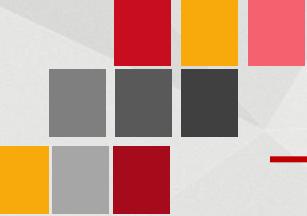


Problemin teknik karmaşıklığının incelenmesi

1. Uygulama, güvenilir yedekleme ve kurtarma gerektiriyor mu?
2. Veri iletişimini gerektiriyor mu?
3. Dağıtılmış İşlemler 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.

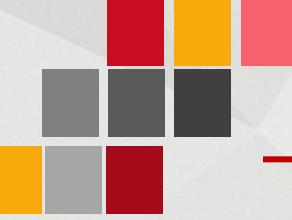


İşlev noktası sayısı hesaplama

✓ $iN = A_iN * (0,65 * 0,01 * TKF)$

Değişik amaçlarla kullanılabilir

- **Üretkenlik** = $iN / \text{Kişi-Ay}$
- **Kalite** = $\text{Hatalar} / iN$
- **Maliyet** = $\$ / iN$



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

Örneğin,
 $IN=300$ ise ve Nesne Tabanlı bir dilde
geliştirme yapılrsa
 $\text{Satır Sayısı}=300*30$, kod üreticileri ile
geliştirme yapılrsa
 $\text{Satır Sayısı}=300*15$ satır kadarlık kod
yazılması gereklidir.
Buradaki kod satır sayısını açıklamalar
hariç olarak düşünmek gereklidir.



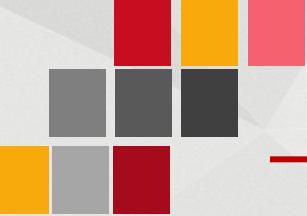
@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

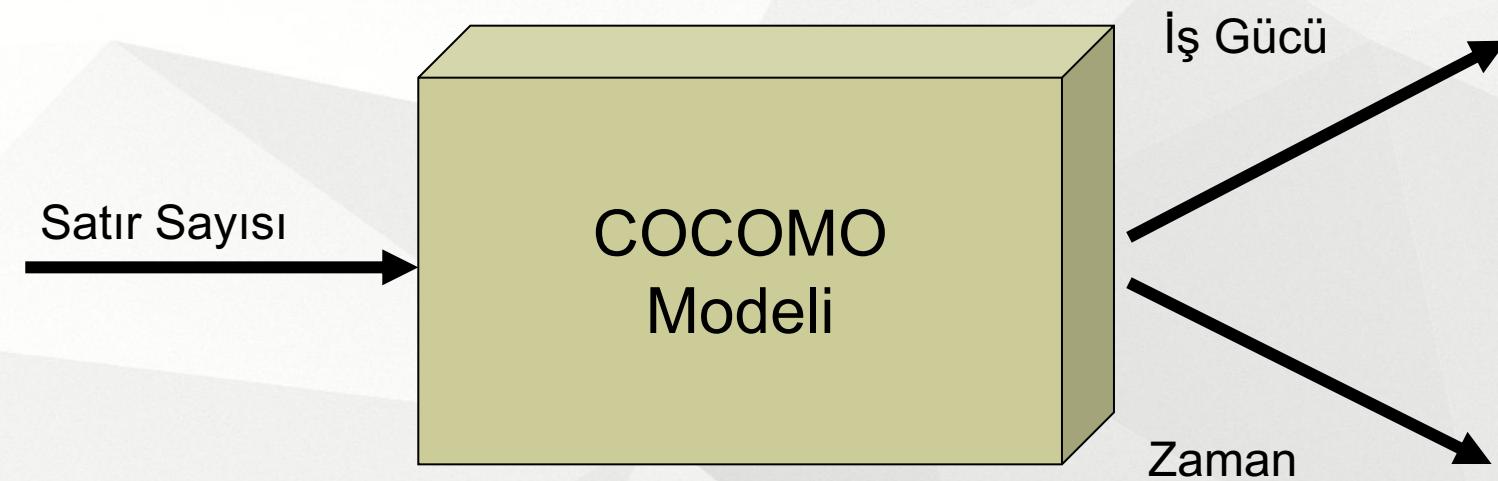
Yazılım Mühendisliği Temel Süreçler – PLANLAMA II

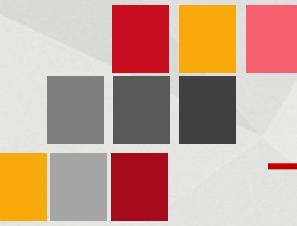
Doç. Dr. Recep ERYİĞİT
₁



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



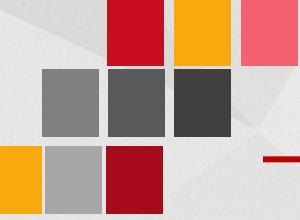


COCOMO formülleri

- ✓ 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:**

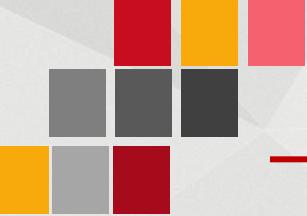
- Boyutları küçük,
- Deneyimli personel tarafından gerçekleştirilmiş
- LAN üzerinde çalışan insan kaynakları yönetim sistemi gibi

✓ **Yarı Gömülü:**

Hem bilgi boyutu hem donanım sürme boyutu olan projeler

✓ **Gömülü Projeler:**

Donanım sürmeye 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



Ayrık Projeler

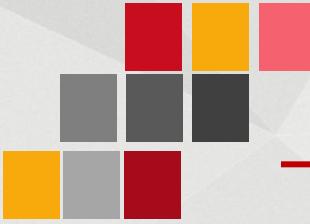
- İş Gücü $K=2.4*S^{1,05}$
- Zaman $T=2.5*K^{0,38}$

Yarı Gömülü Projeler

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

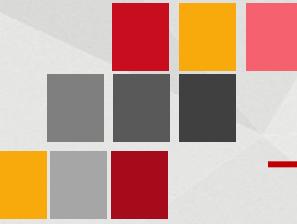
Gömülü Projeler

- 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

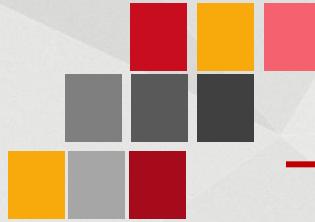


İş Gücü Hesaplama

✓ Ayrık Projeler $K=3.2*S^{1,05}$

✓ Yarı Gömülü Projeler $K=3,0*S^{1,12}$

✓ Gömülü Projeler $K=2.8*S^{1,20}$



Maliyet Çarpanı Hesaplama

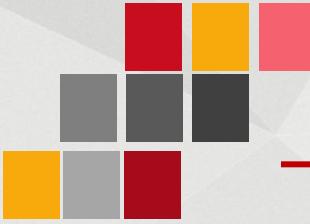
- ✓ Maliyet Çarpanı 15 maliyet etmeninin çarpımı sonucudur.

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



Maliyet Etmenleri

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



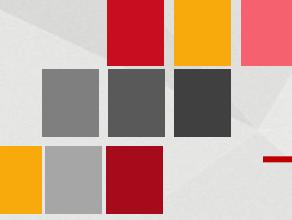
Ürün Özellikleri

- ✓ Rely: Yazılımın güvenirliği
- ✓ Data: Veri Tabanının Büyüklüğü.
Burada program büyülüğüne 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.

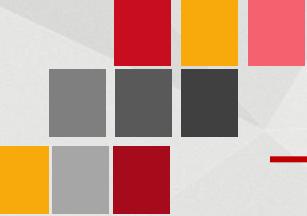
Proje ekibinin ortalama tecrübesi.

✓ **Pcap:** Programcı Yeteneği.

✓ **Vexp:** Bilgisayar Platformu Deneyimi.

Proje ekibinin geliştirilecek platformu tanıma oranı.

✓ **Lexp:** Programlama dili deneyimi.



Proje Özellikleri

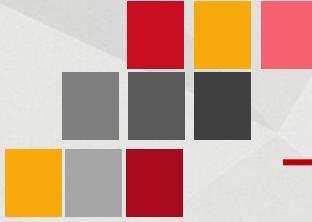
✓ **Modp:** Modern Programlama Teknikleri.

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

✓ **Tool:** Yazılım Geliştirme araçları kullanımı.

- CASE araçları
- Metin düzenleyiciler
- Ortam yönetim araçları

✓ **Sced:** Zaman Kısıtı.

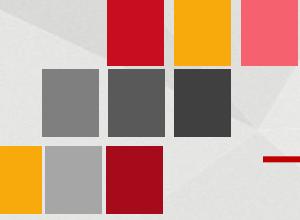


İlk İşgücü değerini Düzeltme

✓ $K_d = K * C$

Kd= Düzeltilmiş
İşgücü

* Temel Formüldeki Zamanla formülü kullanılarak zaman maliyeti hesaplanır.

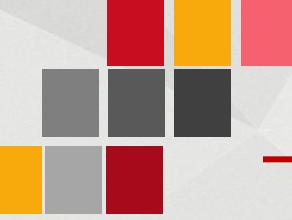


Ayrıntı modeli

Temel ve ara modele ek olarak iki özellik taşıır.

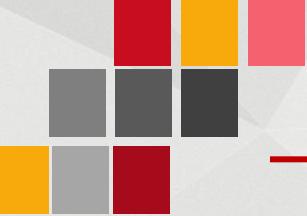
- ✓ Aşama ile ilgili işgücü katsayıları: her aşama için (planlama, analiz, tasarım, geliştirme, test etme) farklı katsayılar, karmaşıklık belirler
- ✓ Üç düzey ürün sıra düzeni: yazılım maliyet kestiriminde
 - Modül
 - Altsistem
 - Sistem

Sıra düzenini dikkate alır



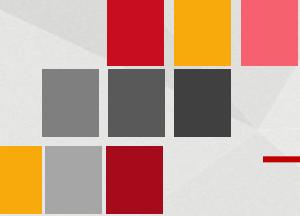
Proje Ekip Yapısı Oluşturma

- ✓ 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 Eşgündüm Birimi
 - Eğitim Birimi
 - Uygulama Destek Birimi



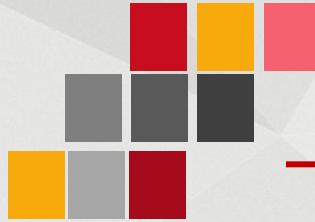
Yüklenici Proje Ekip Yapısı

- ✓ 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



@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği Temel Süreçler - *Sistem Analizi*

Doç. Dr. Recep ERYİĞİT
₁



HEDEFLER

- ✓ Planlama raporu içeriği
- ✓ Yazılım Yaşam Döngüsü
- ✓ Analiz- Greksininim nedir? Gereksinim türleri
- ✓ Gereksinim Verisi Toplama Yöntemleri
- ✓ Kullanıcı Arayüz Prototipleme (KAP)
- ✓ Sistem Analiz Raporu



Proje Planı(Faaliyet-Zaman-Maliyet Çizelgesi)

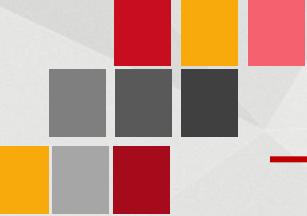
Proje Kaynakları-

1. İnsan kaynakları : Proje şamalarında görev alacak personelin nitelikleri ve çalışma zamanları
 2. Sistemin geliştirilmesinde ve nihai sistemde kullanılacak donanım kaynaklarının edinilme zaman çizelgesi
 3. Sistem geliştirme sunumunda kullanılacak yazılım kaynaklarının edinilme tarihleri
- Bu aşamanın en önemli görünürlük çıktıları projenin çıktılarına ait zaman çizelgesidir.
- İlk maliyet hesaplama bu aşamada olmasına karşın proje planı raporunda genellikle yer almaz.

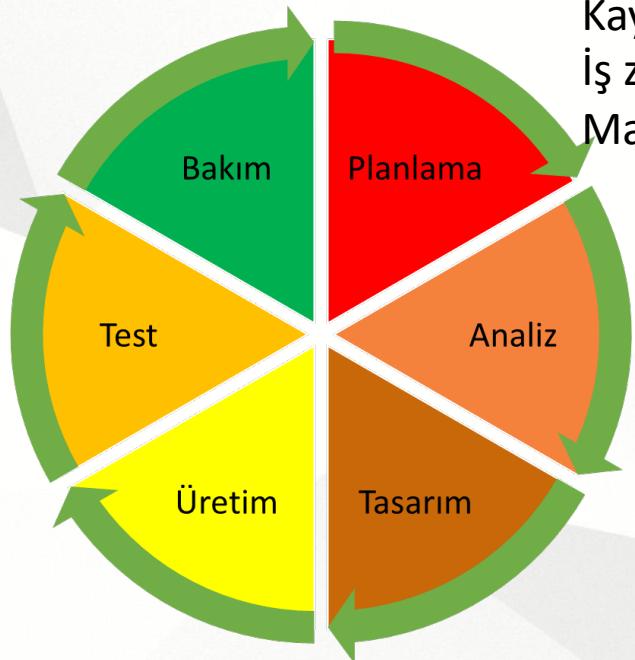
Projenin başlangıç tarihi : .../.../20...

20... Yılı Fiyatlarıyla

Faaliyet	I. Yıl				II. Yıl				III. Yıl	Maliyet (Bin TL)
	1-3. ay	4-6. ay	7-9. ay	10-12. ay	13-15. ay	16-18. ay	19-21. ay	22-24. ay		
1.										
1.1.										
1.2.										
2.										
2.1.										
2.2.										
2.3										
3.										
4.										
5.										
Toplam Tutar										

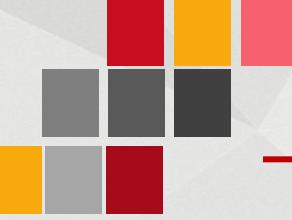


Yazılım Yaşam Döngüsü



Kaynak: –insan, donanım-yazılım
İş zaman çizelgesi
Maliyet hesabı

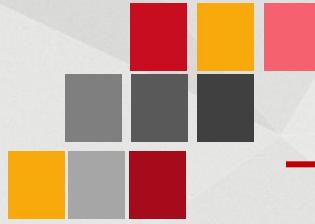
Halihazırda işin yapılmakta olan işlerin
algoritmalarının belirlenmesi



Analiz (Çözümleme)

- ✓ Amaç: Sistemin işlevlerini ve kesin gereksinimleri açılığa kavuşturmak ve sonucunda bunları belirli bir formatta **doküman** etmektir.

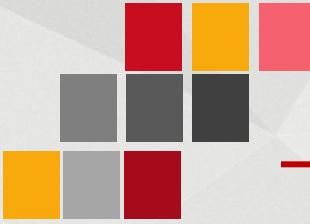
Analiz çalışması; müşteri, yazılım mühendisi, sistem analisti, iş analisti, ürün yöneticisi vb. rollerin bir araya geldiği gruplar tarafından yapılabilir. İhtiyaçların net olmadığı durumlarda yazılım mühendisi ve müşteri arasında iletişim ve birlikte çalışmanın çok daha fazla olması gereklidir. Çeşitli yazılım geliştirme metodolojilerinde bu aşamada kullanıcı dokümanlarının taslakları ile ve test plan dokümanları da oluşturulabilir.



Gereksinim Nedir?

Gereksinim, sistemin amaçlarını yerine getirme yeteneği olan bir özellik ya da belirtim olarak tanımlanmaktadır.

- ✓ Gereksinim işlevlerinin nasıl yerine getirileceği ile ilgili değildir. Ne olduğu ile ilgilidir.



İşlevsel Gereksinim

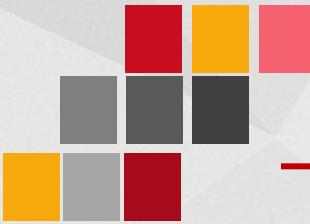
- ✓ İşlevsel gereksinim (Kullanıcı gereksinimi); sistem ile çevresi arasındaki iletişimini belirleyen gereksinimlerdir. Geliştirilecek olan sistemi kullanacak aktörlerin ihtiyaçlarını karşılayacak gereksinimlerdir.
 - bordronun ne zaman alınacağı
 - hangi verilerin alınacağı
 - çıktı formatı



İşlevsel Olmayan Gereksinimler

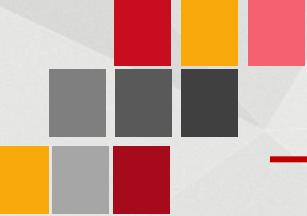
- ✓ İşlevsel olmayan gereksinimler, kullanıcının sorunundan bağımsız olarak çözülmesi gereken işlemlerdir.

- ✓ Sistem kısıtları olarak ta adlandırılabilir
 - kullanılacak bilgisayarın türü
 - yazılım geliştirme ortamı
 - kullanılacak veri tabanı yönetim sistemi



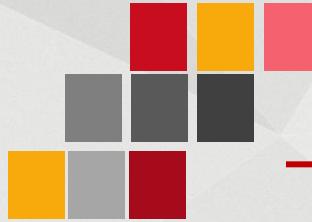
Gereksinim Türleri

- ✓ Fiziksel Çevre
- ✓ Arayüzler
- ✓ Kullanıcı ve İnsan etmeni
- ✓ İşlevsellik
- ✓ Belgeleme
- ✓ Veri
- ✓ Kaynaklar
- ✓ Güvenlik
- ✓ Kalite Güvencesi



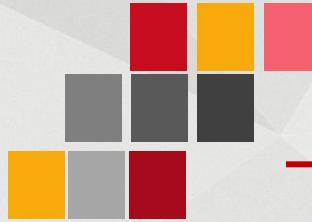
Fiziksel Çevre

- ✓ İşlevlerin geliştirileceği, işletileceği aygıtlar nerededir?
- ✓ Sistem tek bir yerde mi olacak? Fiziksel olarak ayrı yerler söz konusu mu?
- ✓ Sıcaklık nem oranı veya manyetik etkileşim gibi çevresel kısıtlamalar var mı?



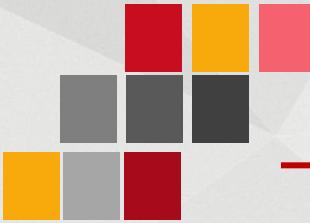
Arayüzler

- ✓ 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?



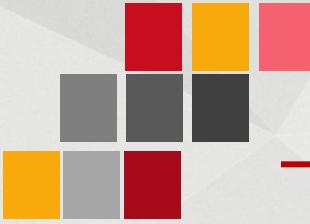
Kullanıcı ve İnsan etmeni

- ✓ 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?



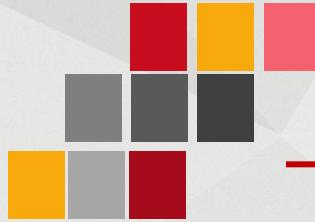
İşlevsellilik

- ✓ 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?

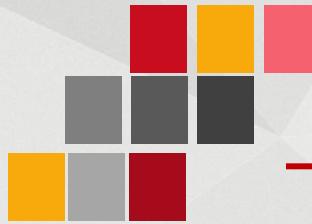


Belgeleme

- ✓ Ne kadar belgeleme gereklidir?
- ✓ Belgeleme hangi kullanıcı kitlesini hedeflemektedir?
- ✓ Geliştirilecek sistemin bakım sürecine ait belgeleme gerekli mi?



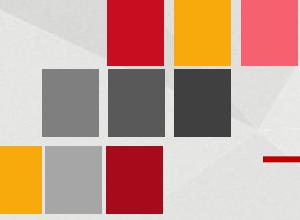
- ✓ Hem giriş hem çıkış için verinin biçimi ne olmalıdır?
- ✓ Bu veri ne sıklıkla alınacak veya gönderilecektir?
- ✓ Bu verinin doğruluk ölçüsü ne olmalıdır?
- ✓ Hesaplamaların sonuçları hangi hassasiyette olmalıdır?
- ✓ Sistemde ne kadar veri akışı olacaktır?
- ✓ Verinin depolanma ve işleme süresi ne kadar olacak?



Kaynaklar

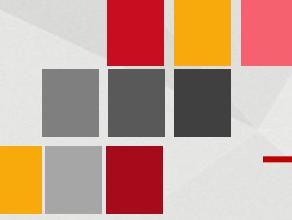
- ✓ Sistemi kurmak, kullanmak ve bakımını yapmak için ne kadar malzeme, personel ve diğer kaynaklara ihtiyaç var?
- ✓ Geliştiriciler hangi yeteneklere sahip olmalı?
- ✓ Sistem ne kadar fiziksel yer kaplayacak?
- ✓ Güç, ısıtma ve soğutma için kısıtlar nelerdir?
- ✓ Geliştirim için tavsiye edilen bir zaman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

- ✓ 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

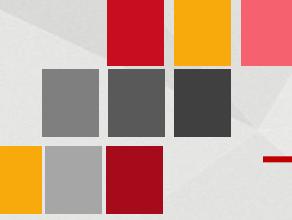
- ✓ Güvenirlilik için gereksinimler nelerdir?
- ✓ Sistemin özellikleri insanlara nasıl aktarılmalıdır?
- ✓ Sistem çökmeleri arasında öngörülen zaman aralığı nedir?
- ✓ Kaynak kullanımı ve yanıt süresine ilişkin verimlilik ölçütleri nelerdir?



Gereksinim Özellikleri

Gereksinimler üç amaca hizmet eder

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



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?



@

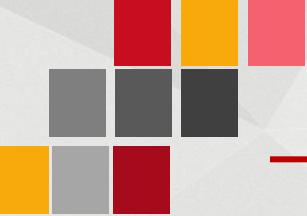


ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği

Temel Süreçler - *Sistem Analizi II*

Doç. Dr. Recep ERYİĞİT
₁

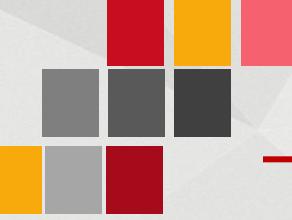


Sistem Çözümleme Çalışması

✓ Geliştirilecek bilgi sistemi yada yazılımla ilgili olarak;

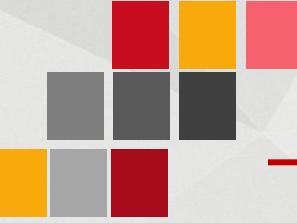
- tüm gereksinimlerin araştırılması,
- tanımlanması,
- ortaya çıkarılması ve
- bir gösterim biçimi ile açıklanması

çalışmasıdır.



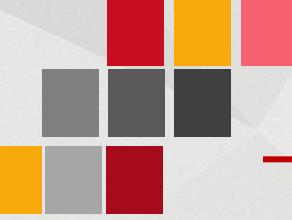
Mevcut sistemin incelemesi

- ✓ 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.



Önerilen Sistemin Modellenmesi

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



Gereksinim Verisi Toplama Yöntemleri

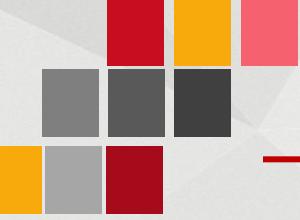
✓ Gereksinim Verisi Toplama Yöntemleri

- Sorma
- Karşılıklı görüşme (Anket)
- Psikolojik türetme
- İstatistiksel teknikler

✓ Veri Modelleme Yöntemleri

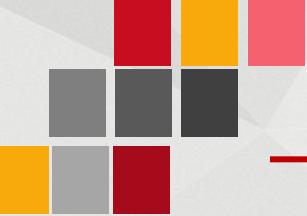
- Nesne İlişki şemaları (1-1, 1-N, M-N)
- Veri Sözlüğü

✓ Süreç/İşlem Modelleme yöntemleri



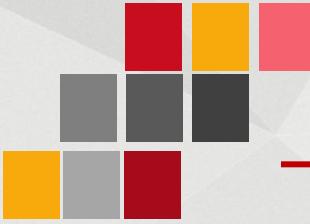
Sorma Yöntemi

- ✓ Amaçlar, resmi olmayan yöntemler, duygular ve düşünceler araştırılır.
- ✓ Yönlendirici sorular (**bence....**) ve iki nesneli sorulardan kaçınılmalıdır (**ne zaman ve nasıl...?**).



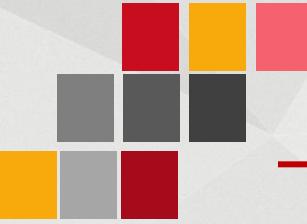
Anket Yöntemi

- ✓ Kullanıcı sayısının fazla olduğu durumlarda eğilimleri ve davranış biçimlerini saptamak için kullanılır.
- ✓ Anket değerlendirilirken gerçekçi olmayan değerlendirmeler çıkarılmalıdır.



Psikolojik Türetme Teknikleri

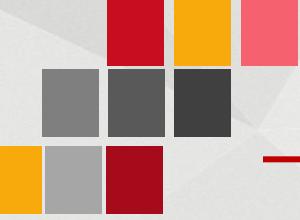
- ✓ Özellikle belirsizliğin fazla olduğu ve zayıf yapılı ortamlarda, bilgi edinebilmek amacıyla insan psikolojisine dayalı teknikler kullanılır.



İstatistiksel Teknikler

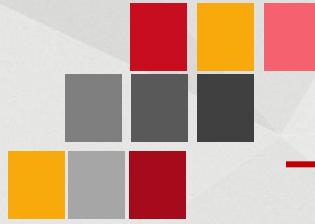
✓ Veri yoğun ve veri hacmi yüksek ortamlarda verinin özelliklerini belirlemek amacıyla kullanılır.

Örnekleme yöntemi ve PIRA yöntemi.



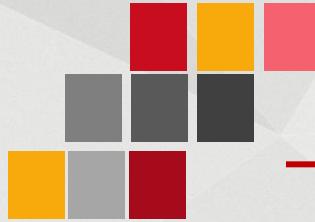
Kullanıcı Arayüz Prototipleme (KAP)

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



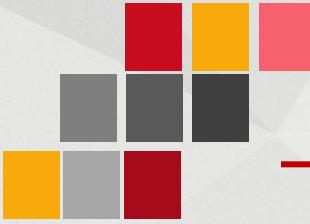
KAP Özellikleri

- ✓ 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. Söz Konusu rapor çalışmanın tüm ayrıntılarını içerir.

- Giriş
- Mevcut sistemin incelenmesi
- İstenen sistem mantıksal modeli
- Arayüz gerekleri
- Belgeleme gerekleri



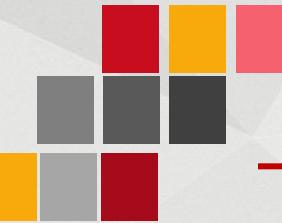
@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği Temel Süreçler - *Tasarım*

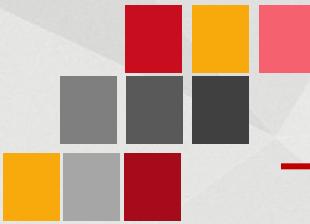
Doç. Dr. Recep ERYİĞİT
₁



HEDEFLER

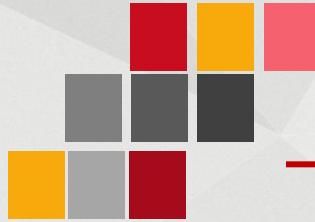
Tasarım, Sistem Analizi çalışması sonucunda üretilen Mantıksal Modelin Fiziksel Modele dönüştürülme çalışmasıdır.

- ✓ 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



HEDEFLER

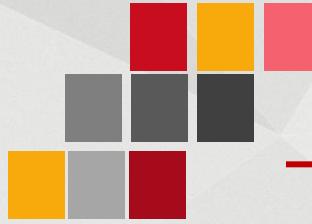
- ✓ Tasarım Kavramları
- ✓ MODÜL-işlevsel Bağımsızlık
- ✓ Veri Tasarımı
- ✓ Tasarlanması Gereken ortak alt sistemler
- ✓ Tasarım Kalitesi
- ✓ Bağlaşıklık -Yapışıklık



TASARIM

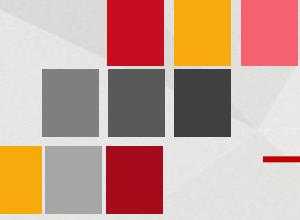
- ✓ Tasarım, Sistem Analizi çalışması sonucunda üretilen **Mantıksal Modelin Fiziksel Modele dönüştürülme** çalışmasıdır.

- ✓ 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



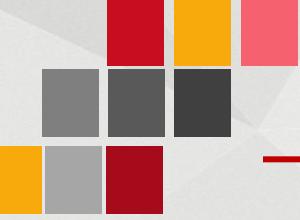
TASARIM KAVRAMLARI

- ✓ **Soyutlama (abstraction)**: Detayları gizleyerek yukarıdan bakabilme imkanı sağlanır.
- ✓ **İyileştirme (enhancement)**: Soyutlama düzeyinde irdeleme bittikten sonra, daha alt seviyelere inilerek tanımlamalarda ayrıntı, bazen de düzeltme yapılarak tasarımın daha kesinlik kazanması sağlanır.
- ✓ **Modülerlik (modularity)**: Sistemi istenen kalite faktörleri ışığında parçalara ayırtırma sonucu elde edilir.

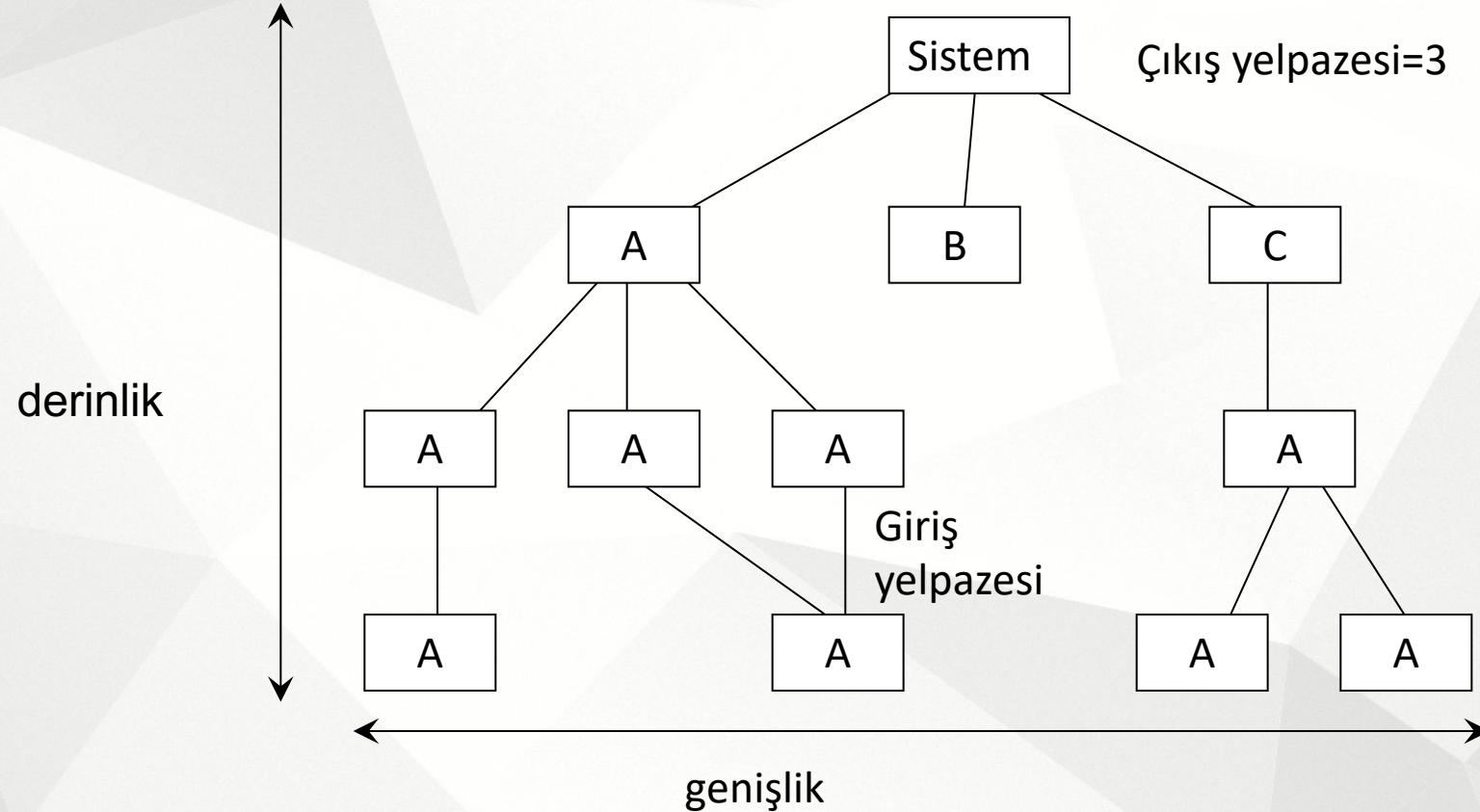


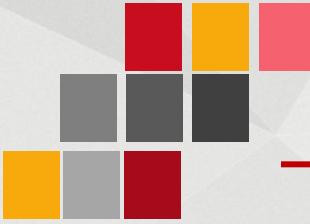
MODÜL

- ✓ 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 tümlestirilen birimlerdir.



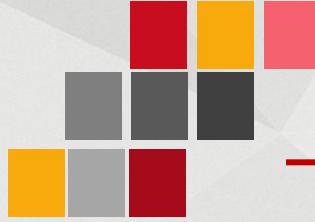
MODÜL





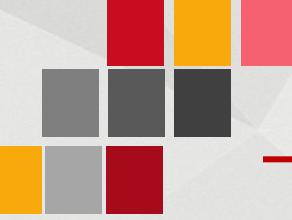
İşlevsel Bağımsızlık

- ✓ 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.



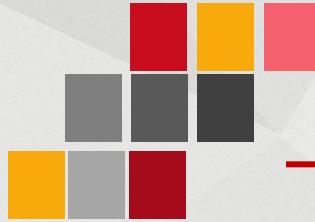
Veri Tasarımı

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



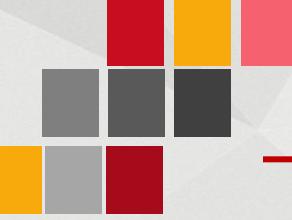
Veri tasarımindan dikkat edilecek konular

- ✓ 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.



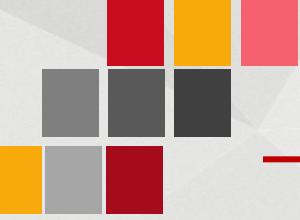
Yapısal Tasarım

- ✓ 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 parçada incelenebilir
 - Girdi Akışı
 - Çıktı Akışı
 - İşlem Akışı



AYRINTI TASARIM- Süreç Tasarımı

- ✓ Süreç tasarımları; veri, yapı ve arayüz 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)



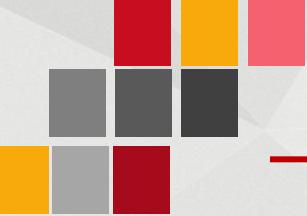
Yapısal Program Yapıları

✓ Yapısal programlamanın temel amacı;

- program karmaşıklığını en aza indirmek,
- program anlaşılabilirliğini artırmaktır.

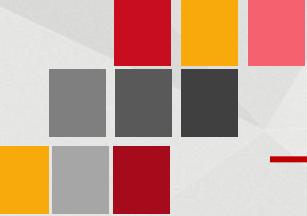
✓ Bu amaçla şu yapılar kullanılır;

- Ardışıl İşlem yapısı
- Koşullu işlem yapısı
- Döngü yapısı



Karar Tabloları

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



Program Tanımlama Dili

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

DO

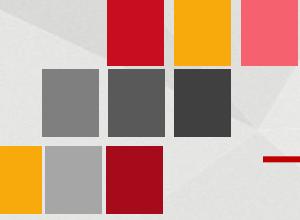
Hesap Numarasını Oku

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

IF (para yatırma işlemi) { para_yatır(); Başlangıça dön}

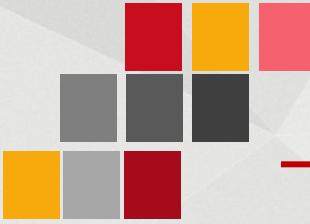
IF (yeterli bakiye yok) başlangıça dön

WHILE



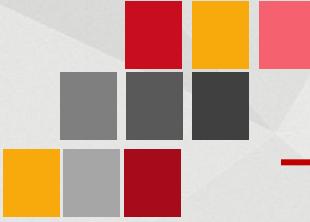
Tasarlanması Gereken Ortak Alt Sistemler

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



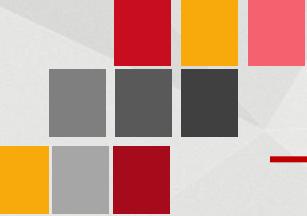
Yetkilendirme Alt Sistemi

- ✓ Özellikle kurumsal uygulamalarda farklı kullanıcıların kullanabilecekleri ve kullanamayacakları özellikleri ifade eder.
 - İşlev bazında yetkilendirme
 - Ekran bazında yetkilendirme
 - Ekran alanları bazında yetkilendirme



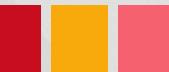
Güvenlik Alt Sistemi

- ✓ Yapılan bir işlemde, işlemi yapan kullanıcının izlerinin saklanması işlemleri.
- ✓ LOG files (Sistem günlüğü)

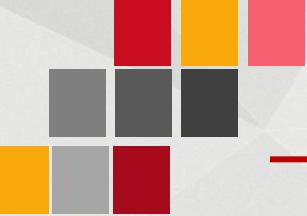


Yedekleme Alt Sistemi

- ✓ Her bilgi sisteminin olağanüstü durumlara hazırlıklı olmak amacıyla kullandıkları veri tabanı (sistem) yedekleme ve yedekten geri alma işlemlerinin olması gerekmektedir.

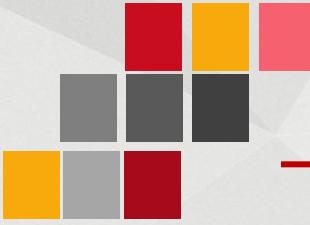


-
- ✓ Coğrafi olarak dağıtılmış hizmet birimlerinde çalışan makineler arasında veri akışının sağlanması işlemleri
 - ✓ Çevirim içi veri iletimi (real-time)
 - ✓ Çevirim dışı veri iletimi (disketler, teypler)



Arşiv Alt Sistemi

- ✓ Belirli bir süre sonrasında sık olarak kullanılmayacak olan bilgilerin ayrılması ve gerektiğinde bu bilgilere erişimi sağlayan alt sistemlerdir.
- ✓ Aktif veri tabanı.



Dönüştürme Alt Sistemi

- ✓ Geliştirilen bilgi sisteminin uygulamaya alınmadan önce veri dönüştürme (mevcut sistemdeki verilerin yeni bilgi sistemine aktarılması) işlemlerine ihtiyaç vardır.



Dönüştürme Alt Sistemi

- ✓ Geliştirilen bilgi sisteminin uygulamaya alınmadan önce veri dönüştürme (mevcut sistemdeki verilerin yeni bilgi sistemine aktarılması) işlemlerine ihtiyaç vardır.



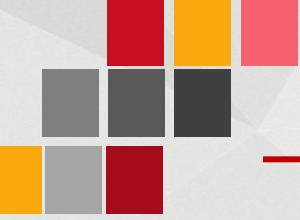
@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği Temel Süreçler – *Tasarım II*

Doç. Dr. Recep ERYİĞİT₁



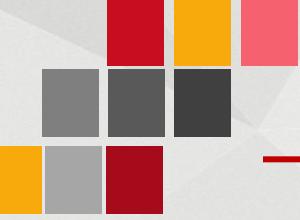
Kullanıcı Ara yüz Tasarımı

✓ Kullanıcı ile ilişkisi olmayan ara yüzler

- Modüller arası ara yüz
- Sistem ile dış nesneler arası ara yüz

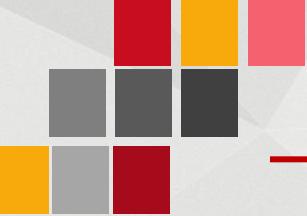
✓ Kullanıcı arayüzleri

- Kullanım kolaylığı ve işlevsel gereksinimlerin kısa sürede ve kısa adımlarla yapılması önemlidir.
- Kullanıcıda çoğunlukla yazılım = arayüz yaklaşımı vardır.



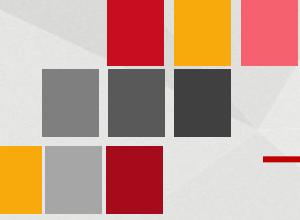
Genel Prensipler

- ✓ Veri giriş formlarının tutarlı olması
- ✓ Önemli silmelerde teyit alınmalı
- ✓ Yapılan çoğu işlem geri alınabilmeli
- ✓ Hataların affedilmesi, yanlış girişte kırılmama
- ✓ Komut isimlerinin kısa ve basit olması
- ✓ Menülerin ve diğer etkileşimli araçların standart yapıda kullanımı



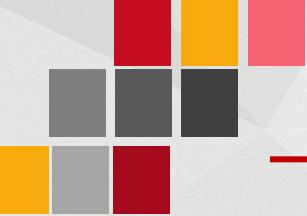
Bilgi Gösterimi

- ✓ 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 indirilmeli
- ✓ Gösterim ve girdi sayfaları birbirinden ayrılmalı
- ✓ Kullanıcının tema oluşturmasına izin verilmeli
- ✓ Her bir işlev için minimum koşullar ve detay bilgiler ayrı sayfalarda tutulmalı
- ✓ Bütün girdiler için yardım kolaylıklarını olmalı



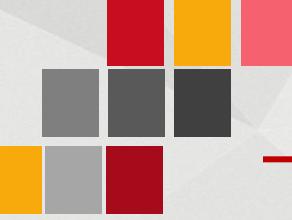
Kullanıcı Arayüz Prototipi

✓ 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 çıktığının ifade edilebilmesi açısından tüm ekranların aynı şablon üzerine oturtulması önerilmektedir.

- **Menü Çubuğu**
- **Araç Çubuğu**
- **Gövde (Değişebilir)**
- **Durum Çubuğu**

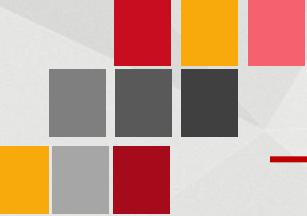


Başlangıç Tasarım Gözden Geçirme

✓ Yapılan tasarım çalışmasının bir önceki geliştirme aşaması olan analiz aşamasında belirlenen gereksinimleri karşılayıp karşılamadığının belirlenmesidir.

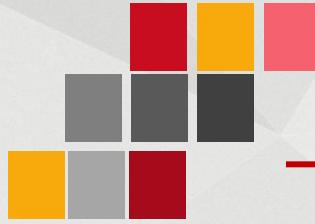
- Sistem gereksinimlerine yardımcı olan kullanıcılar
- Sistem analizini yapan çözümleyiciler
- Sistemin kullanıcıları
- Tasarımcılar
- Yönlendirici
- Sekreter
- Sistemi geliştirecek programcılar

dan oluşan bir grup tarafından yapılır.



Ayrıntılı Tasarım Gözden Geçirme

- ✓ Başlangıç tasarıımı 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.



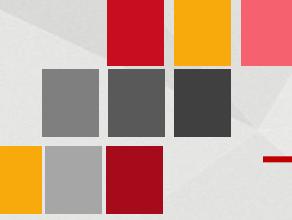
Tasarım Kalite Ölçütleri

✓ Bağlaşım (Coupling)

Tasarımı oluşturan modüller arası ilişki ile ilgilidir.

✓ Yapışıklık (Cohesion)

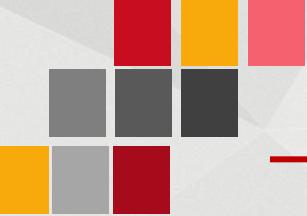
Modüllerin iç yapısı ile ilgilidir.



Bağlaşım

- ✓ Modüller arası bağlılığın 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 karmaşıklığının azaltılması

nedenleri ile istenmektedir



Yalın Veri Bağlaşımı

- ✓ Herhangi iki modül arası iletişim yalın veriler ([tamsayı](#), [karakter](#), [boolean](#), [vs](#)) aracılığı ile gerçekleştiriliyorsa bu iki modül yalın veri bağlaşımılıdır şeklinde tanımlanır.



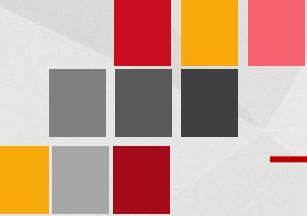
Karmaşık Veri Bağlaşımı

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



Denetim Bağlaşımı

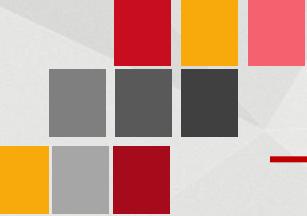
- ✓ İki Modül arasında iletişim parametresi olarak **denetim verisi** kullanılıyorsa bu iki modül denetim bağlantılı olarak tanımlanır.



Ortak Veri Bağlaşımı

- ✓ Eğer iki modül ortak bir alanda tanımlanmış verilere ulaşabiliyorsa bu iki modül **ortak veri bağıntılı** olarak tanımlanır.

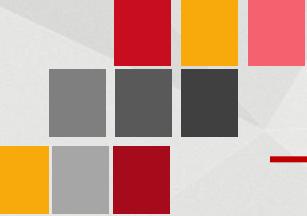
- ✓ Verilerin ortak veri bağıntılı olmaları şu nedenlerden dolayı fazla istenmez;
 - Ortak veri alanını izlemek zordur.
 - Ortak veri kullanan modüllerde yapılan değişiklikler diğer modüller etkiler.
 - Ortak veri üzerinde yapılacak değişikliklerde bu veriyi kullanacak bütün modüller gözüne alınmalıdır.



Ortak Veri Bağlaşımı

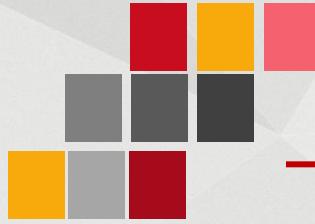
- ✓ Eğer iki modül ortak bir alanda tanımlanmış verilere ulaşabiliyorsa bu iki modül **ortak veri bağıntılı** olarak tanımlanır.

- ✓ Verilerin ortak veri bağıntılı 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 gözüne alınmalıdır.



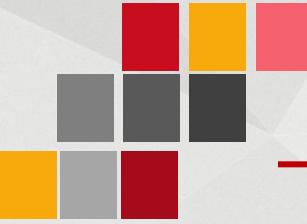
İçerik Bağlaşımı

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



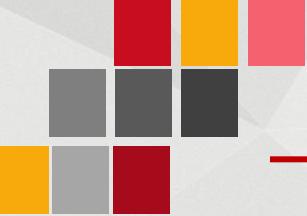
Yapışıklık

- ✓ 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



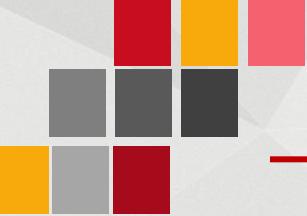
Sırasal Yapışıklık

- ✓ 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

Duzeltilmis_Ham_Veri_Kaydini_Dogrula

Dogrulanmis_Kaydi_Gonder



İletişimsel Yapışıklık

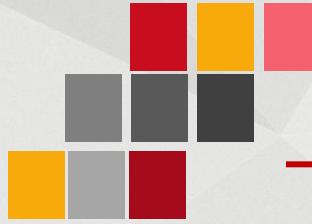
- ✓ Bir modülün içindeki farklı işlemler aynı girdi ya da çıktıyı kullanıyorlarsa bu modül **İletişimsel yapışık** bir modül olarak adlandırılır.

Sicil_No_yu_Al

Adres_Bilgisini_Bul

Telefon_Bilgisini_Bul

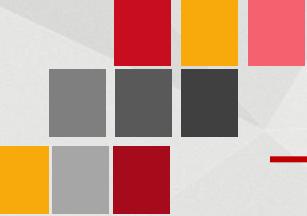
Maas_Bilgisini_Bul



Yordamsal Yapışıklık

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

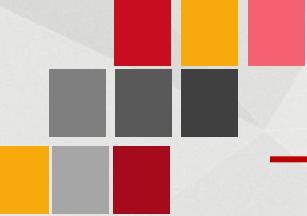
Ekran_Goruntusunu_Yaz
Giris_Kaydini_Oku



Mantıksal Yapışıklık

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

Dizilere değer atama işlemleri



Gelişigüzel Yapışıklık

- ✓ İşlemler arasında herhangi bir ilişki bulunmaz.

Ara_Kayit_Oku

B_dizisine_baslangic_deger_ata

Stok_kutugu_oku

Hata_iletisi_yaz



@



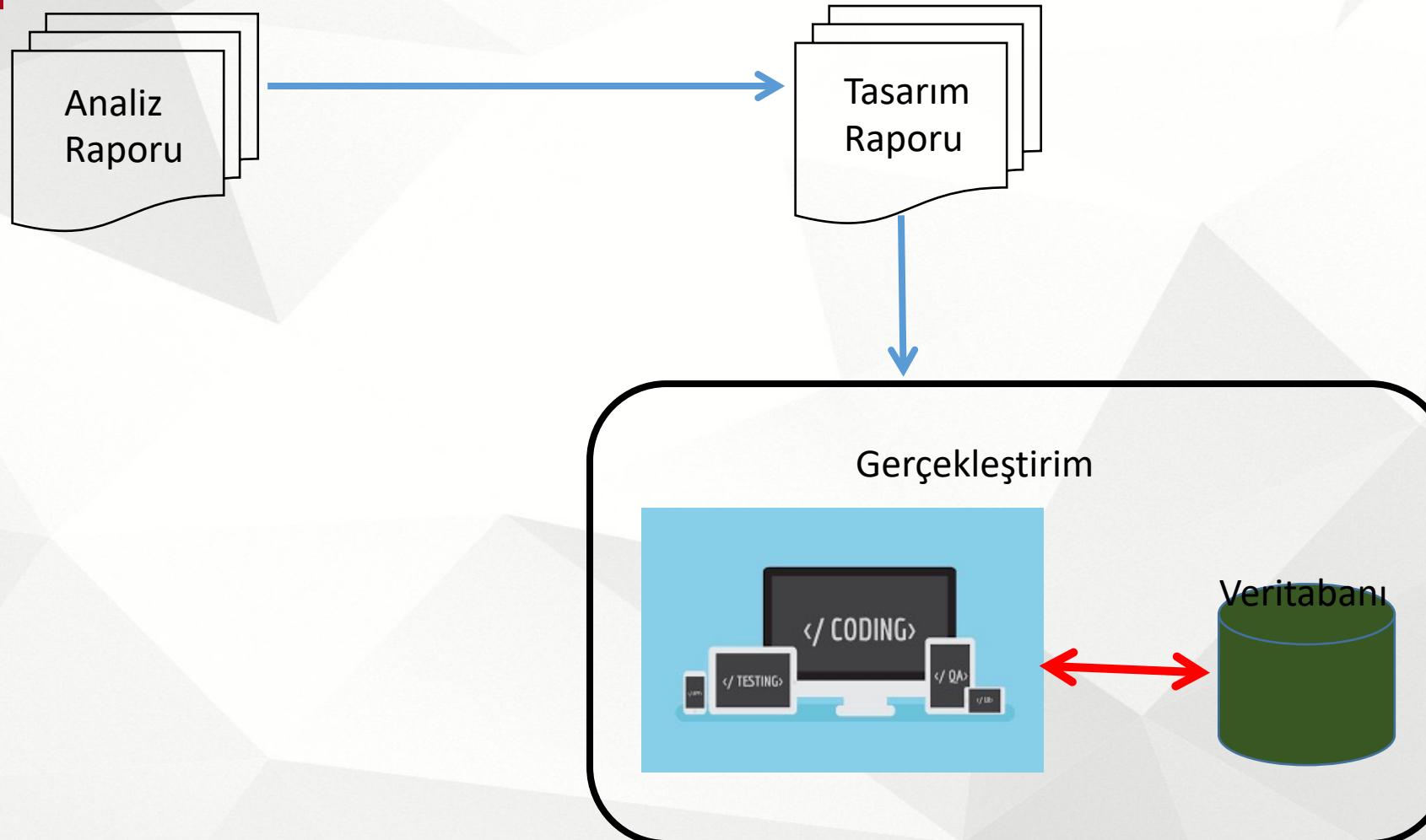
ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

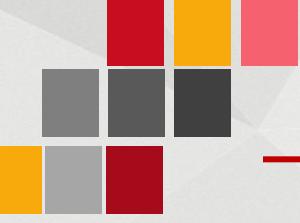
Yazılım Mühendisliği Temel Süreçler - *Geçekleştirim*

Doç. Dr. Recep ERYİĞİT₁



HEDEFLER



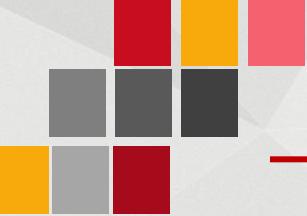


Giriş

- ✓ Tasarım sonucu üretilen süreç ve veri tabanının fiziksel yapısını içeren fiziksel modelin bilgisayar ortamında çalışan yazılım biçimine dönüştürülmesi çalışmasıdır.
- ✓ Birinci yapılması gereken;

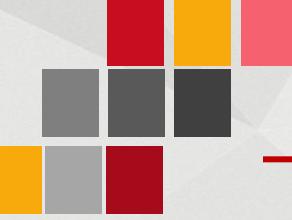
Yazılım geliştirme ortamı seçimi

(programlama dili, veri tabanı yönetim sistemi, yazılım geliştirme araçları (CASE)).



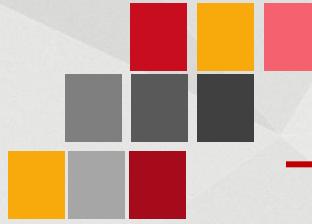
Programlama Dilleri

- ✓ Geliştirilecek Uygulamaya Göre Programlama dili seçilmelidir.
 - **Veri İşleme Yoğunluklu Uygulamalar**
 - Cobol,
 - [Görsel Programlama Dilleri ve Veri Tabanları \(java,C#\)](#)
 - **Hesaplama Yoğun Uygulamalar**
 - Fortran
 - C
 - Paralel Fortran ve C
 - **Süreç ağırlıklı uygulamalar**
 - Assembly
 - C
 - **Sistem programlamaya yönelik uygulamalar**
 - C
 - **Yapay Zeka Uygulamaları**
 - Lisp
 - Prolog



Programlama Dilleri

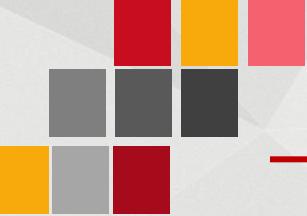
- ✓ Katalog Bilgisi, Veri Sözlüğü Varlığı
- ✓ Veri Soyutlama
 - Kullanıcıdan verinin saklanması konusundaki detayları gizleyerek veri soyutlamasının sağlanması
- ✓ Program-veri, program-İşlem bağımsızlığı
 - Geleneksel Dosya İşleme yöntemlerinde veri dosyalarının yapısı bu dosyalara erişen programların içine gömülüşlerdir. Bu sebeple değiştirmek zor.
- ✓ Birden çok kullanıcı desteği
- ✓ Birden fazla işlem arası paylaşım



Veri Modelleri

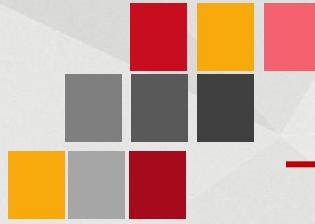
- ✓ Fiziksel veri modelleri verinin bilgisayarda nasıl saklandığının detayları ile ilgili kavramları sağlar.

- ✓ 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).



Şemalar

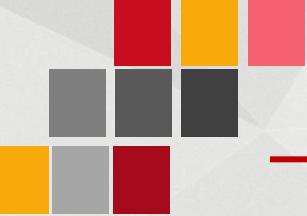
- ✓ Herhangi bir veri modelinde veri tabanının tanımlaması ile kendisini ayırmak önemlidir.
- ✓ Veri tabanının tanımlanması, veri tabanı şeması veya meta-veri olarak adlandırılır.
- ✓ Veri tabanı şeması tasarım sırasında belirlenir ve fazla değişmesi beklenmez.



VTYS Mimarisi

✓ VTYS mimarisi üç seviyede tanımlanabilir.

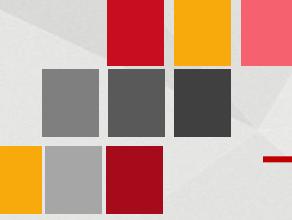
- **İçsel Düzey:** Veri tabanının fiziksel saklama yapısını açıklar.
- **Kavramsal Düzey:** Kavramsal şema içerir ve kullanıcılar için veritabanının yapısını açıklar.
- **Dışsal Düzey:** Dış şemalar ve kullanıcı görüşlerini içerir.



Veritabanı Dilleri ve Arabirimleri

- ✓ Veri tabanı tasarıımı tamamlandıktan sonra bir VTYS seçilir.

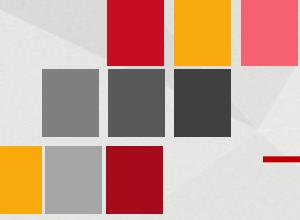
Veri Tanımlama Dili	Kavramsal şemaları tanımlamak üzere kullanılır.
Saklama Tanımlama Dili	İşsel şemayı tanımlamak üzere kullanılır.
Görüş Tanımlama Dili	Dışsal şemayı tanımlamak için kullanılır.
Veri İşleme Dili	Veri tabanı oluşturulduktan sonra veri eklemek, değiştirmek ve silmek için kullanılır.



VTYS nin Sınıflandırılması

Sınıflandırmalar kullanılan Veri Modeline göre yapılır.

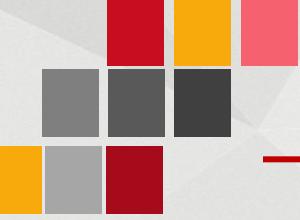
- ✓ **İlişkisel Model:** Veri tabanı tablo yığınından oluşmuştur. Her bir tablo bir dosya olarak saklanabilir.
- ✓ **Ağ Modeli:** Veriyi kayıt ve küme tipleri olarak gösterir.
- ✓ **Hiyerarşik Model:** Veri ağaç yapısında gösterilir.
- ✓ **Nesne Yönelimli Model:** Veri tabanı nesneler, özellikleri ve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



Hazır Program Kütüphaneleri

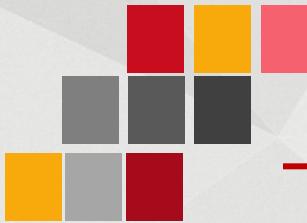
- ✓ Hemen hemen tüm programlama platformlarının kendilerine özgü hazır kütüphaneleri bulunmaktadır.
 - Pascal *.tpu
 - C *.h
 - Java *.jar

- ✓ Günümüzde bu kütüphanelerin temin edilmesi internet üzerinden oldukça kolaydır.



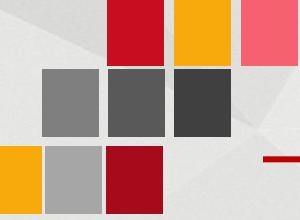
CASE Araç ve Ortamları

- ✓ Günümüzde bilgisayar destekli yazılım geliştirme ortamları oldukça gelişmiştir.
- ✓ CASE araçları yazılım geliştirme sürecinin her aşamasında üretilen bilgi ya da belgelerin bilgisayar ortamında 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



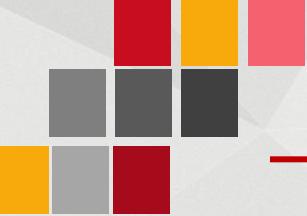
Açıklama Satırları

✓ Modülün başlangıcına

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

gibi genel bilgilendirici açıklamaları yapılır

✓ Aynı zamanda modülün kritik bölümleri vurgulanarak açıklanır.

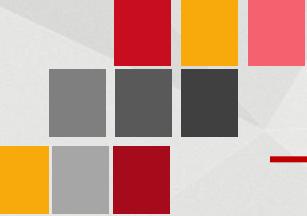


Kod Biçimlemesi

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

```
for i:=1 to 50 do begin i:=i+1; end; // Kötü kodlanmış
```

```
for i:=1 to 50 do          // iyi kodlanmış
begin
  i:=i+1;
end;
```



Anlamlı İsimlendirme

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

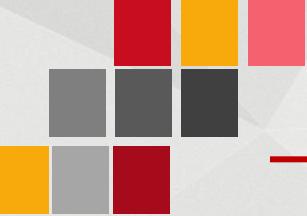


-
- ✓ Yapısal programlama yapıları temelde içinde **goto** komutlarının bulunmadığı, tek giriş ve tek çıkışlı öbeklerden oluşan yapılardır. Bunlar temelde;
 - Ardisil işlem yapıları,
 - Koşullu işlem yapıları,
 - Döngü yapıları.
 - ✓ Teorik olarak herhangi bir bilgisayar programının sadece bu yapılar kullanılarak yazılabileceği kanıtlanmıştır.



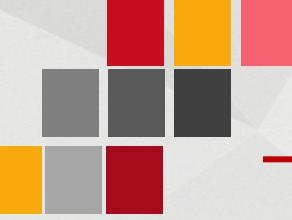
Yapısal programlama Yapıları

- ✓ 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.



Program Karmaşıklığı

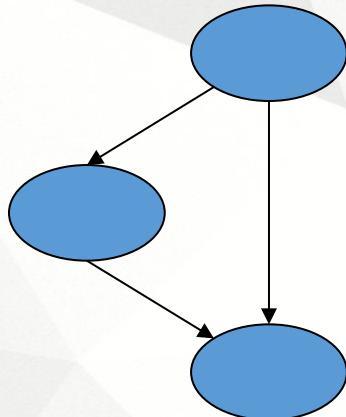
- ✓ Program karmaşıklığı konusunda çeşitli modeller geliştirilmiştir.
- ✓ Bunların en eskisi ve yol göstericisi McCabe tarafından 1976 yılında geliştirilen modeldir.
- ✓ Bunun için önce programın akış diyagramına dönüştürülmesi, sonra da **karmaşıklık ölçütünün** hesaplanması gerekmektedir.



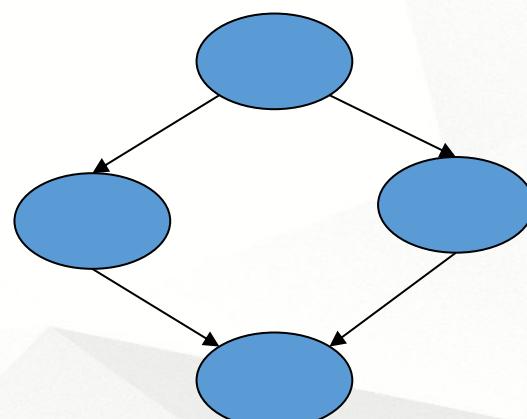
Akış Diyagramına Dönüşürme



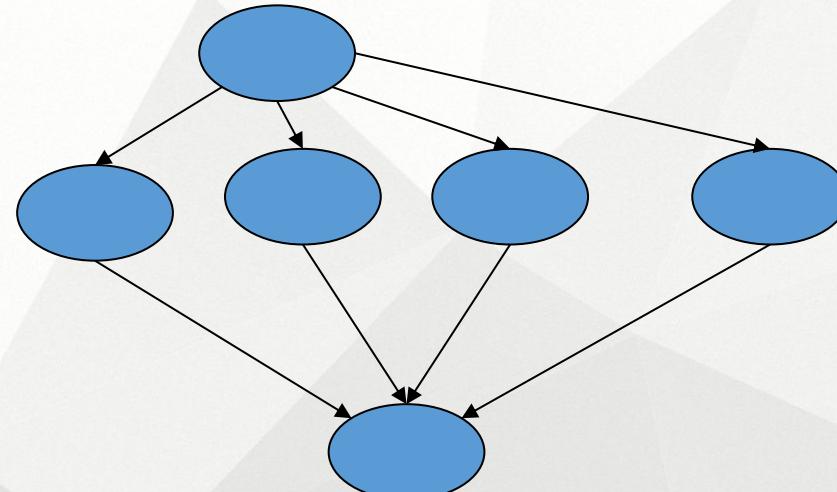
Bir ya da birden fazla ardışık işlem



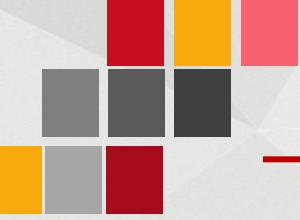
If-Then işlemi



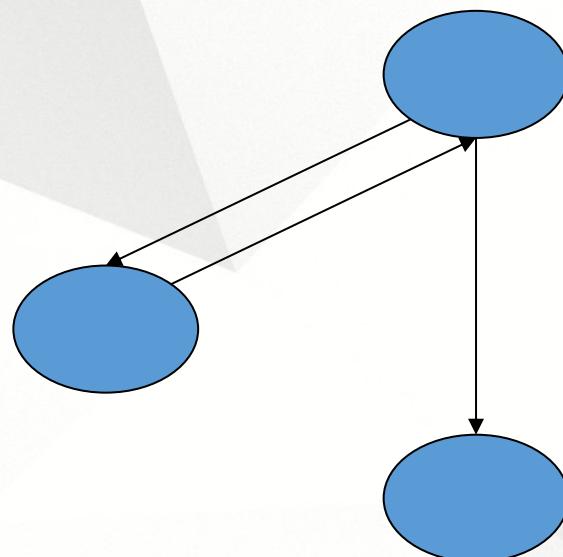
If-Then-Else işlemi



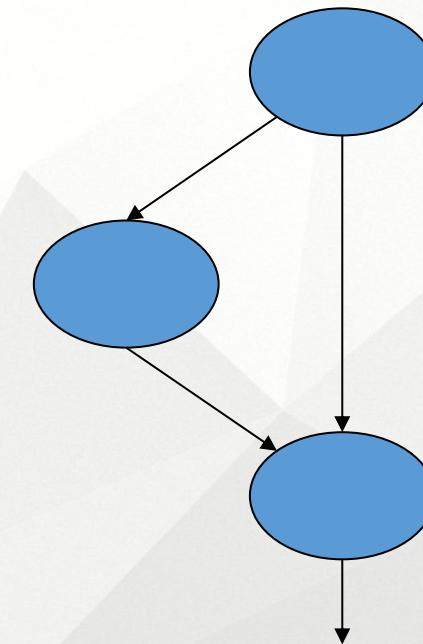
Case işlemi



Akış Diyagramına Dönüşürme



While Döngüsü



Repeat Döngüsü



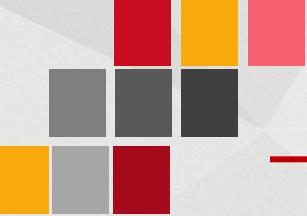
McCabe Karmaşıklık Ölçütü

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

K: Diyagramdaki kenar çizgi sayısı

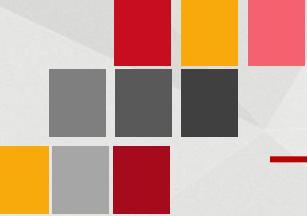
D: Diyagramdaki düğüm sayısı

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 olmasıdır.



Kod Gözden Geçirme

- ✓ **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 olmasıdır.



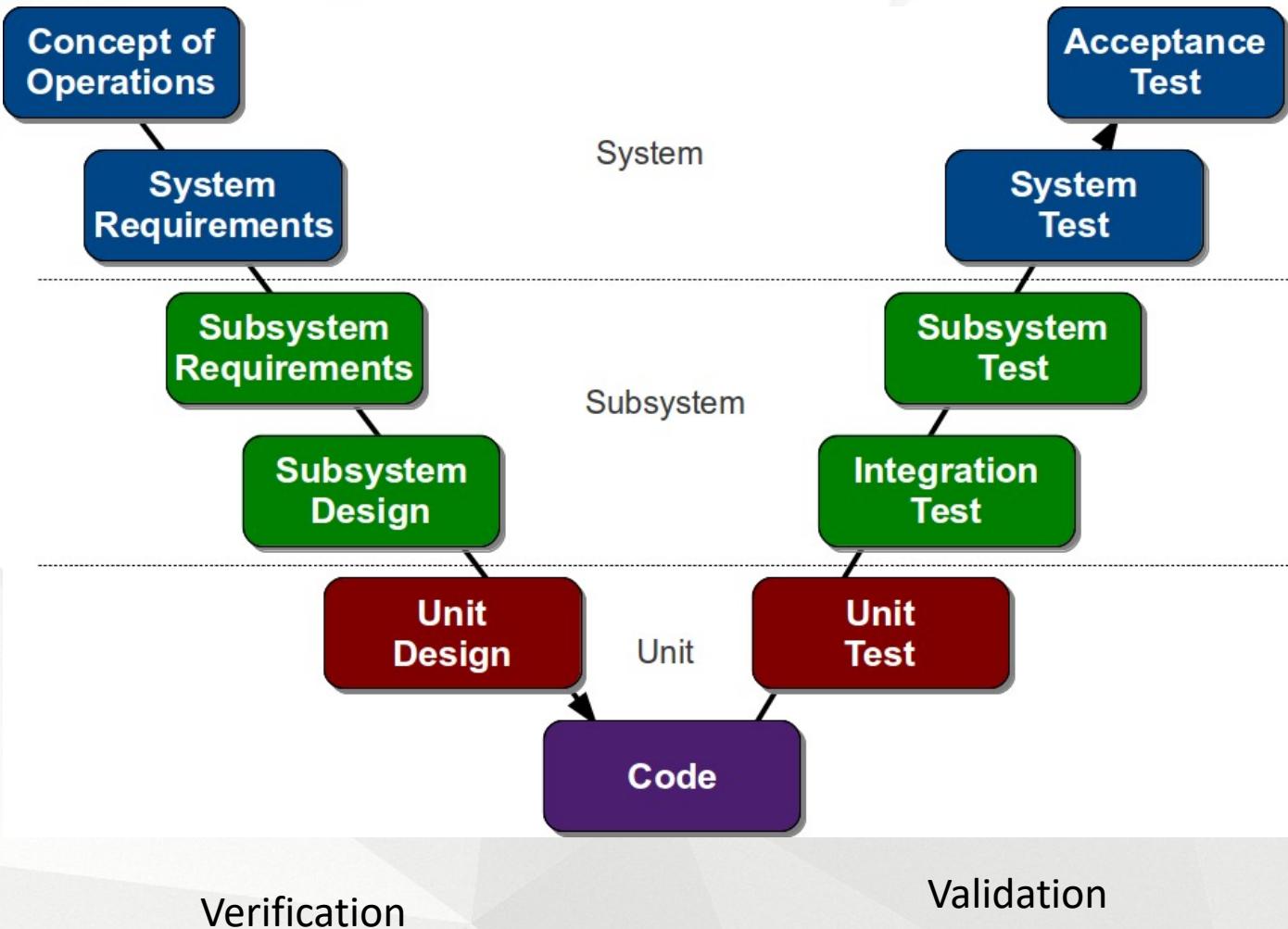
@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği Temel Süreçler - Yazılım *Doğrulama ve Geçerleme*

Doç. Dr. Recep ERYİĞİT
₁





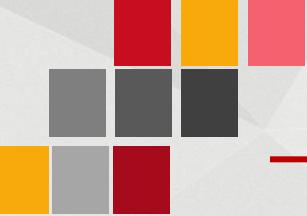
Test nedir?

- ✓ Test, bir sistemin veya bileşenlerinin, belirli şartları yerine getirip getirmedğini belirlemek amacıyla potansiyel kullanıcıları tarafından denenmesidir.
- ✓ ANSI / IEEE 1059 standardına göre: Mevcut ile gereksinimler arasındaki farkları tespit etmek ve yazılım ögesinin özelliklerini değerlendirmek için yazılım ögesini analiz etme işlemi olarak tanımlanır.



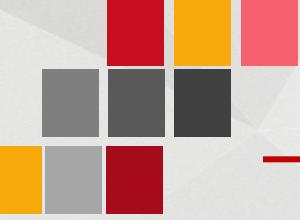
Testleri kim Yapar?

- ✓ Proje sürecine ve projenin ilgili paydaşlarına bağlıdır. BT endüstrisinde, büyük şirketlerin, verilen gereksinimler bağlamında geliştirilen yazılımları değerlendirmekle yükümlü bir ekibi vardır. Geliştiriciler, Birim Test olarak adlandırılan testi yürütürler. Çoğu durumda, aşağıdaki uzmanlar, sistemin test edilmesine katılırlar:
- ✓ Yazılım Test Görevlileri
- ✓ Yazılım Geliştiriciler
- ✓ Proje Lideri / Müdürü
- ✓ Son kullanıcı



Test Türleri

- ✓ **Siyah kutu testi** - İç sistem tasarıımı bu tür testlerde dikkate alınmaz. Testler, gereksinimler ve işlevsellik temel alınarak yapılır.
- ✓ **Beyaz kutu testi** - Bu test, bir uygulamanın kodunun mantığına dayalıdır. Cam Kutusu Testi olarak da bilinir. Bu tür bir test için yazılım ve kod çalışması bilinmelidir. Testler, kod tablolarının, yolların ve koşulların kapsammasına dayanır.
- ✓ **Birim testi** - Yazılım bileşenleri veya modüllerinin test edilmesi. Program tasarıımı ve kodunun ayrıntılı bilgi gerektirmesi nedeniyle tipik olarak programcılar tarafından değil de test görevlileri tarafından yapılır.
- ✓ **Artımlı entegrasyon testi** - Test için alt yaklaşımı, yani bir uygulamanın yeni işlevler eklendiğinde test edilmesi; Uygulama işlevleri ve modülleri ayrı ayrı test edilebilecek kadar bağımsız olmalıdır. Programcılar veya test edenler tarafından yapılır.

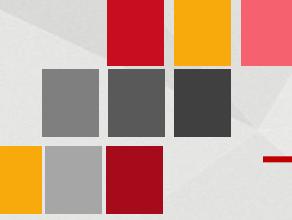


Test Türleri

- ✓ **Regresyon testi** - Herhangi bir modülde veya işlevde modifikasyon için uygulamanın bir bütün olarak test edilmesi. Regresyon testinde tüm sistemi kapsamak zordur, dolayısıyla tipik olarak bu test türleri için otomasyon araçları kullanılır.
- ✓ **Kabul testi** - Normalde bu test türü, sistemin müşterinin belirlediği şartları karşıladığılığını doğrulamak için yapılır. Kullanıcı veya müşteri, başvuruyu kabul edip etmeyeceğini belirlemek için bu sınava katılır.
- ✓ **Yük testi** - Yük altında sistem davranışını kontrol etmek için bir performans testi. Bir uygulamanın ağır yükler altında test edilmesi,örneğin, bir web sitesinin çeşitli yükler altında test edilmesi ve sistemin yanıt verme süresinin hangi noktada bozulduğunu veya başarısız olduğunu belirlemek.
- ✓ **Alfa testi** - Sanal kullanıcı ortamı bu tür testler için oluşturulabilir. Testler geliştirme bitiminde yapılır. Bu testin bir sonucu olarak küçük tasarım değişiklikleri yapılabilir.
- ✓ **Beta test** - Genellikle son kullanıcılar veya başkaları tarafından test edilir. Ticari amaçla başvuruyu yapmadan önce nihai test.

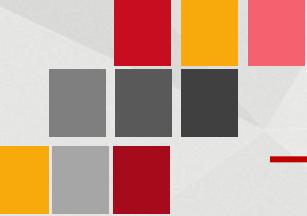


-
- ✓ **Entegrasyon testi** - Entegrasyon sonrasında kombine işlevselligi doğrulamak için entegre modüllerin test edilmesi. Modüller genellikle kod modülleri, bireysel uygulamalar, bir ağdaki istemci ve sunucu uygulamaları, vs. Bu tür testler özellikle istemci / sunucu ve dağıtılan sistemlerle ilgilidir.
 - ✓ **Fonksiyonel test** – Fonksiyonların gereksinimleri karşılamasının test edildiği bu test klasik bir kara kutu testidir.
 - ✓ **Sistem testi** - Sistemin tamamı gereksinimlerine göre test edilir. Genel gereksinim özelliklerine dayanan kara kutu tipi test, bir sistemin tümleşik parçalarını kapsar.
 - ✓ **Uçtan uca sınama** - Sistem sınamasına benzer şekilde, bir veritabanı ile etkileşim kurma, ağ iletişimini kullanma veya diğer donanım, uygulamalar veya sistemler ile etkileşim kurma gibi gerçek dünya kullanımını taklit eden bir durumda eksiksiz bir uygulama ortamının test edilmesini içerir. Eğer uygunsa.



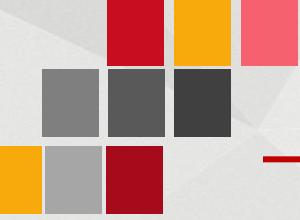
Doğrulama ve Geçerleme

- ✓ Geliştirilecek bilgi sistemi yazılımın doğrulanması ve geçerlenmesi işlemi üretim süreci boyunca süren etkinliklerden oluşur. Bu etkinlikler;
 - Her bir etkinlik sonunda alınan çıktıların tamam, doğru, açık ve tutarlı olduğunun doğrulanması.
 - Her etkinlikte ürünün teknik yeterliliğinin değerlendirilmesi ve uygun çözüm elde edilene kadar aktivitelerin tekrarlanması.
 - Geliştirilen belirtimlerin önceki belirtimlerle karşılaştırılması.
 - Yazılım ürünlerinin tüm uygulanabilir gereklerinin sağlandığının gerçekleşmesi için sınavaların hazırlanıp yürütülmesi.



Doğrulama ve Geçerleme

- ✓ **Doğrulama:** Doğru ürünü mü üretiyoruz?
 - ✓ **Geçerleme:** Ürünü doğru mu üretiyoruz?
-
- ✓ **Doğrulama** ürünü kullanacak kişilerin isteklerinin karşılanıp karşılanmasılığını test eden etkinliklerden,
 - ✓ **Geçerleme** ise ürünün içsel niteliğine ilişkin izleme ve denetim etkinliklerinden oluşur.



Sınamalar

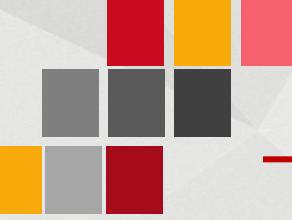
- ✓ Sınamalar ve Bütünleştirme işlemlerinin bir strateji içinde gerçekleştirilmesi, planlanması ve tekniklerinin seçilmesi gerekmektedir.

- ✓ Sınamalar işlemleri dört ana sınıfta incelenebilir:
 - Birim sınavma
 - Alt-sistem sınavma
 - Sistem sınavma
 - Kabul sınavması



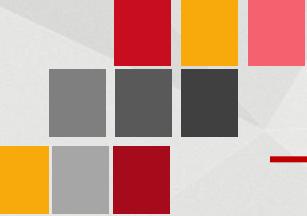
Birim Sınama

- ✓ Bağlı oldukları diğer sistem unsurlarından tümüyle soyutlanmış olarak birimlerin doğru çalışmalarının belirlenmesi amacıyla yapılır.



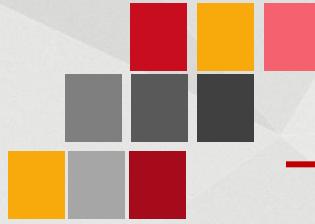
Alt-sistem Sınama

- ✓ Alt-sistemler modüllerin bütünləşirilmeleri ile ortaya çıkarlar.
- ✓ Yine bağımsız olarak sınavaları yapılmalıdır.
- ✓ Bu aşamada en çok hata arayüzlerde bulunmaktadır. Bu yüzden arayüz hatalarına doğru yoğunlaşılmalıdır



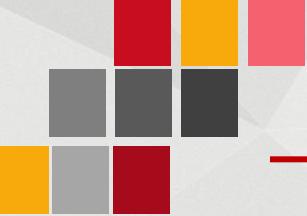
Sistem Sınaması

- ✓ Üst düzeyde, bileşenlerin sistem ile olan etkileşiminde çıkacak hatalar aranmaktadır.
- ✓ Ayrıca, belirtilen ihtiyaçların doğru yorumlandıkları da sınanmalıdır.



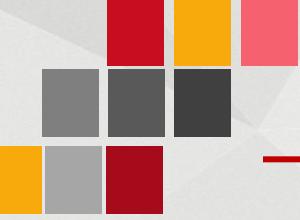
Kabul Sınaması

- ✓ Çalıştırılmadan önce sistemin son sınavasıdır.
- ✓ Artık, yapay veriler yerine gerçek veriler kullanılır.
- ✓ Bu sınavma türü alfa sınavası veya beta sınavası olarak da bilinir.



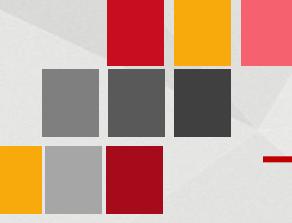
Alfa vs Beta Sınaması

- ✓ **Alfa Sınamada;** sistemin geliştirildiği yerde kullanıcıların gelerek katkıda bulunması sistemi test etmesi amaçlanmaktadır.
- ✓ **Beta Sınamasında;** kullanıcı, geliştirilen sistemi kendi yerleşkesinde, bir gözetmen eşliğinde yapar.



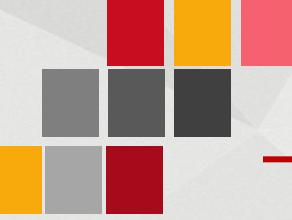
Sınama

- ✓ Sınamalar, hatalardan kurtulmanın bir güvencesi değildir. Hatalardan bütünüyle arınıldığı gibi bir kanı elde edilmemelidir.
- ✓ Ne kadar hata sıklığına erişildiğinde sınama işlemlerinin durdurulacağına, maliyet ve kalite arasında yapılacak bir en iyileme çalışması ile ulaşılır.
- ✓ Yazılımın kritiklik düzeyine göre sınamaya ayrılan süre ve çaba artar.



Doğrulama ve Geçerleme Yaşam Döngüsü

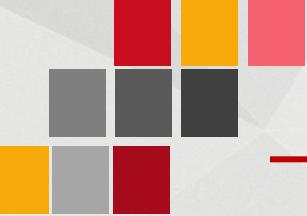
- ✓ Gerçekleştirim aşamasına kadar olan süreçlerde doğrulama ve geçerleme işlemlerinin planlaması yapılır.
- ✓ Planlama genellikle;
 - alt-sistem,
 - bütünlştirme,
 - sistem ve
 - kabul sınavlarının tasarımlarını içerir.
- ✓ Gerçekleştirim aşamasının sonunda ise söz konusu plan uygulanır.



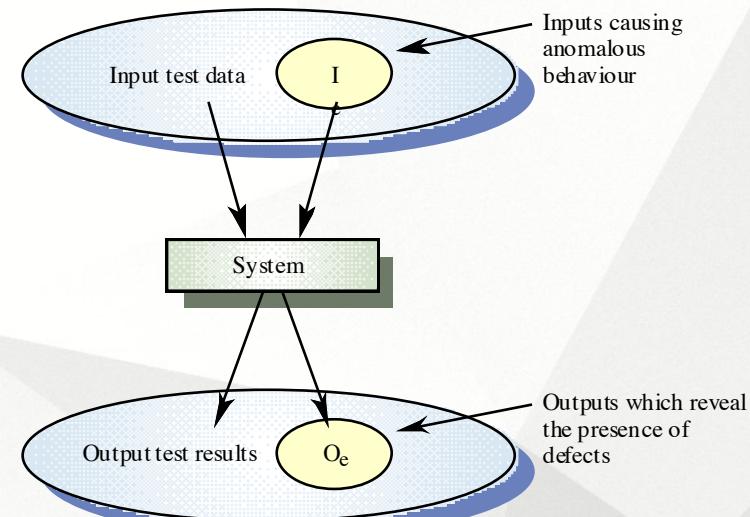
Sınama Yöntemleri

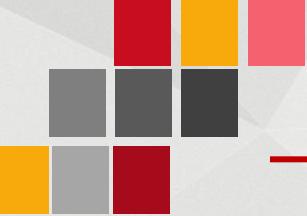
✓ Her yazılım Mühendisliği ürünü iki yoldan sınanır:

- **Kara kutu testi (Black-Box testing):** Sistemin tümüne yönelik işlevlerin doğru yürütüldüğünün testidir. Sistem şartnamesinin gerekleri incelenir.
- **Beyaz Kutu Testi (White Box testing):** İç işlemlerin belirtimlere uygun olarak yürütüldüğünün bileşenler tabanında sınanmasıdır.



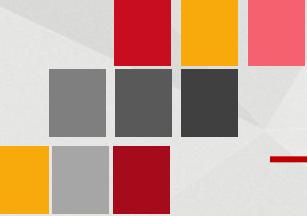
Kara Kutu Testi



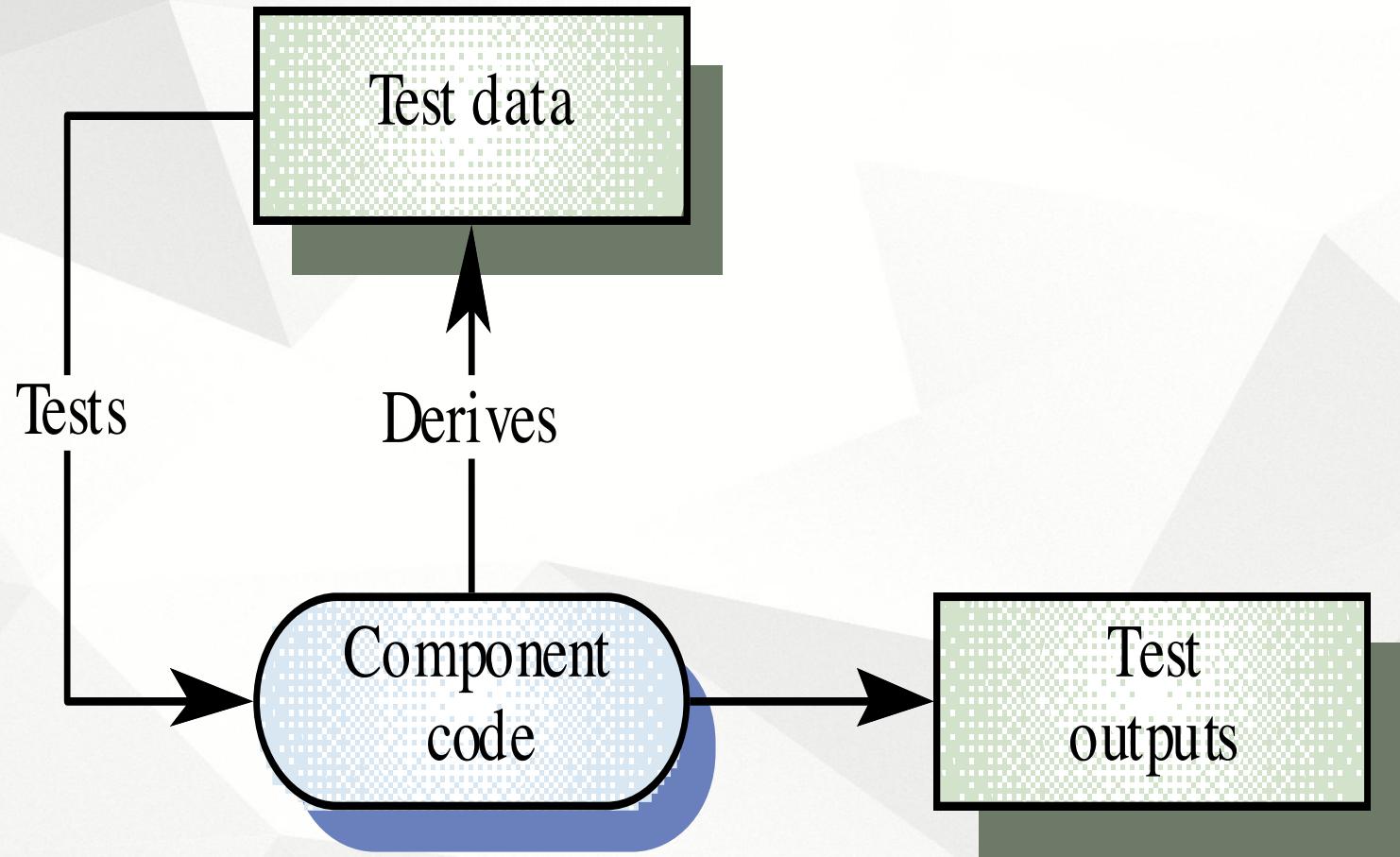


Beyaz Kutu Testi

- ✓ Bütün bağımsız yolların en az bir kez sınanması gereklidir.
- ✓ Bütün mantıksal karar noktalarında iki değişik karar için sınamalar yapılmalıdır.
- ✓ Bütün döngülerin sınır değerlerinde sınanması
- ✓ İç veri yapılarının denenmesi



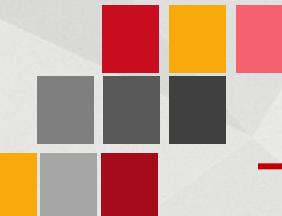
Beyaz Kutu Testi



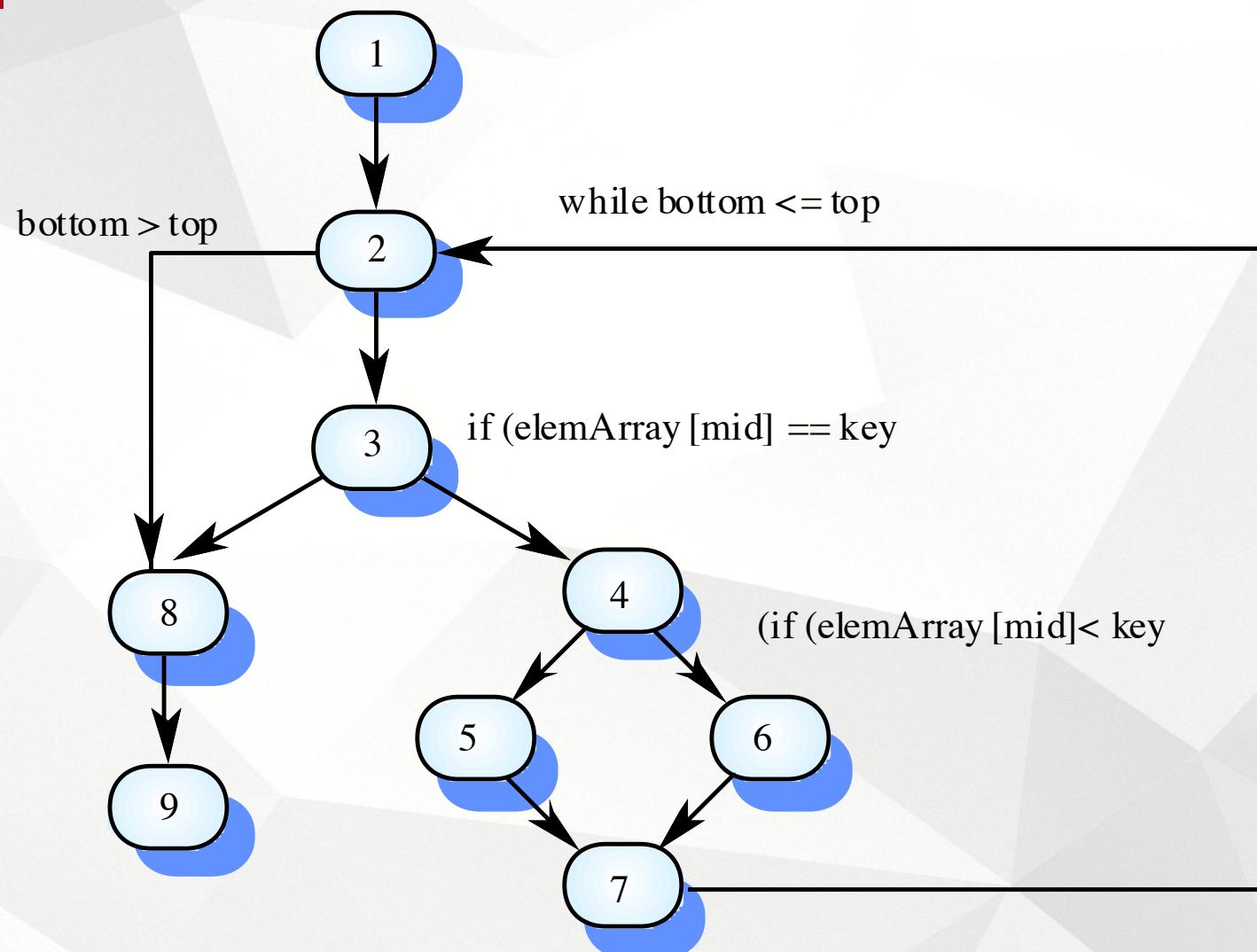


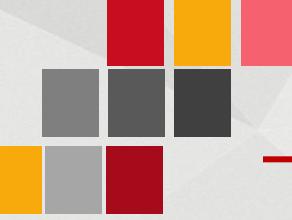
Beyaz Kutu Testi

```
class BinSearch {  
    public static void search ( int key, int [] elemArray, Result r )  
    {  
        int bottom = 0 ;  
        int top = elemArray.length - 1 ;  
        int mid ;  
        r.found = false ;  
        r.index = -1 ;  
        while ( bottom <= top )  
        {  
            mid = (top + bottom) / 2 ;  
            if (elemArray [mid] == key)  
            {  
                r.index = mid ;  
                r.found = true ;  
                return ;  
            } // if part  
            else  
            { if (elemArray [mid] < key)    bottom = mid + 1 ;  
              else      top = mid - 1 ;      }  
        } //while loop      } // search  } //BinSearch
```



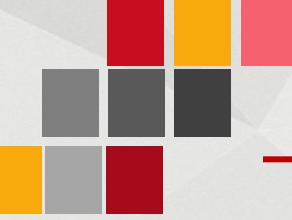
Beyaz Kutu Testi





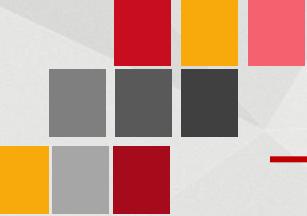
Sınamalar ve Bütünleştirme Stratejileri

- ✓ Genellikle Sınamalar Stratejisi, bütünlendirme stratejisi ile birlikte değerlendirilir.
- ✓ Ancak bazı sınav stratejileri bütünlendirme dışındaki hataları hedefleyebilir.
- ✓ Örneğin, yukarıdan-aşağı ve aşağıdan-yukarıya stratejileri bütünlendirme yöntemine bağlıdır.



Yukarıdan Aşağıya Bütünleştirme

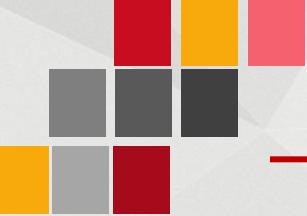
- Yukarıdan-aşağıya bütünləştirməde önce sistemin üst düzeylerinin sıyanması ve sonra aşağıya doğru olan düzeyləre ilgili modülləri takılarak sıyanması söz konusudur.
- En üst noktadaki bileşen sıandıktan sonra alt düzeye geçilmelidir.
- Alt bileşenler henüz hazırlanmamışlardır. Bu sebeple Koçanlar kullanılır. **Koçan:** Bir alt bileşenin, üst bileşen ile arayüzünü temin eden, fakat işlevsel olarak hiçbir şey yapmayan çerçeve programıdır.



Yukarıdan Aşağıya Bütünleştirme

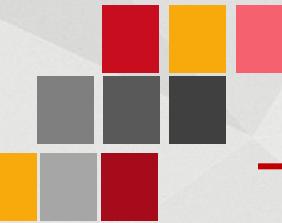
✓ İki temel Yaklaşım vardır:

- **Düzen Öncelikli Bütünleştirme:** En üst düzeyden başlanır ve aynı düzeydeki birimler bütünleştirilir.
- **Derinlik Öncelikli Bütünleştirme:** En üst düzeyden başlanır ve her dal soldan sağa olmak üzere ele alınır.

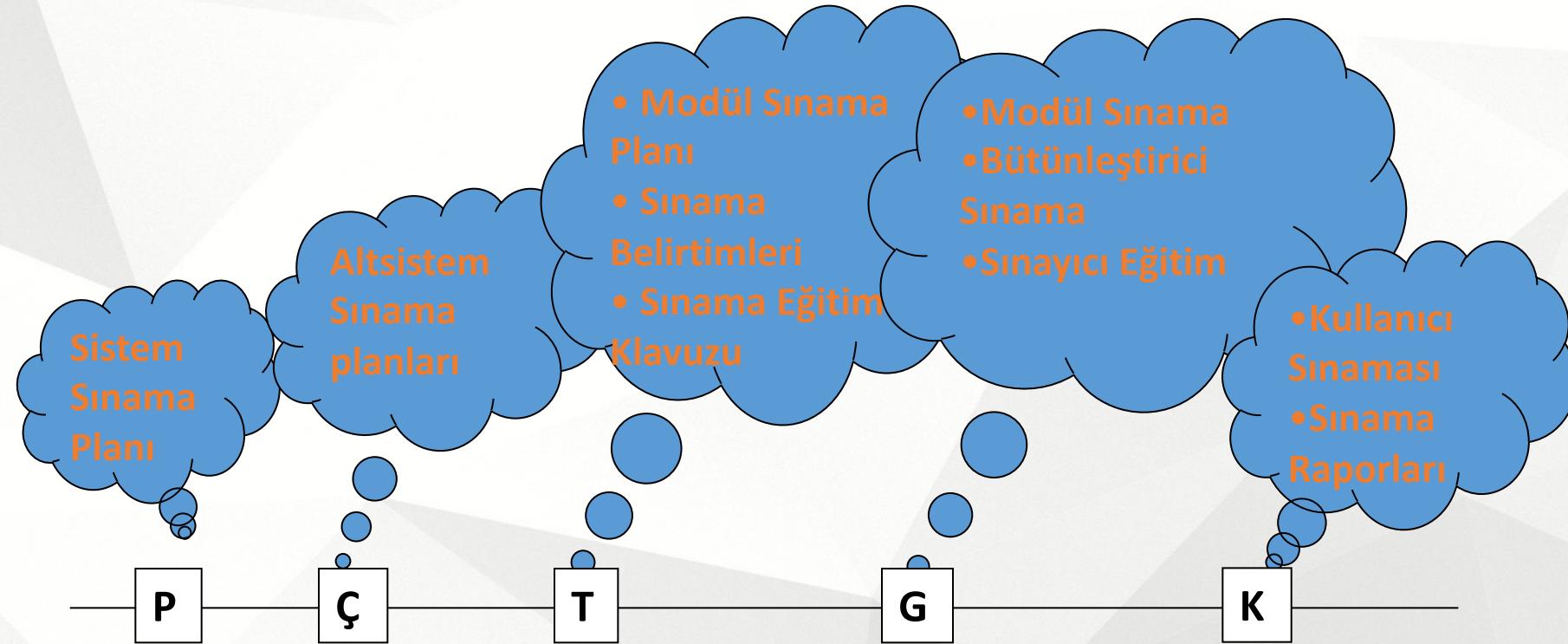


Aşağıdan Yukarıya Bütünleştirme

- ✓ Önceki yöntemin tersine uygulama yapılır.
- ✓ Önce en alt düzeydeki işçi birimler sınanır ve bir üst düzey ile sınanması gerektiğinde bu düzey bir sürücü ile temsil edilir.
- ✓ Bu kez kodlama, bütünlendirme ve sınama, aşağı düzeylerden yukarı düzeylere doğru gelişir.



Yaşam Döngüsü Boyunca Sınama



P: Planlama Ç: Çözümleme T: Tasarım G: Gerçekleştirme K: Kurulum



@



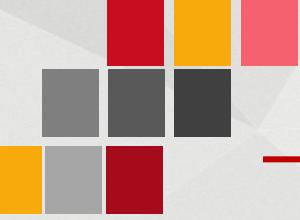
ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği Temel Süreçler – *YAZILIM BAKIMI*

Doç. Dr. Recep ERYİĞİT₁



-
- ✓ Bakım Süreci
 - ✓ Sistem Dokümantasyonu
 - ✓ Program Geliştirme Dinamikleri
 - ✓ Bakım Masrafları
 - ✓ Bakım Ölçümü

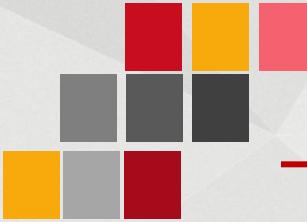


Bakım

- ✓ Kullanıma alınmış programın değiştirilmesi
- ✓ Bakım yönetimi, değişim sürecinin planlanması ve gerçekleştirilmesi ile ilgilidir.



-
- ✓ Sistemin gereksinimleri, zaman geçtiği için sistem geliştirilirken muhtemelen değişecektir. Dolayısıyla, teslim edilen bir sistem gereksinimleri karşılamayacak!
 - ✓ Sistemler çevrelerine sıkı sıkıya bağlıdırlar. Bir sistem bir ortamda kurulduğunda o ortamı değiştirir ve bu nedenle sistem gereksinimlerini değiştirir.
 - ✓ Sistemin kullanıcı gereksinimlerini karşılaması isteniyorsa bakım yapılması zorunludur.



Bakım Türleri

İyileştirme (perfective) Bakımı

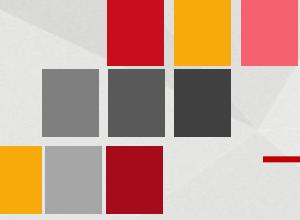
- ✓ Bir sistemi, gereksinimlerini daha etkili bir şekilde karşılamak için değiştirmeye

Uyarlama (Adaptive) Bakımı

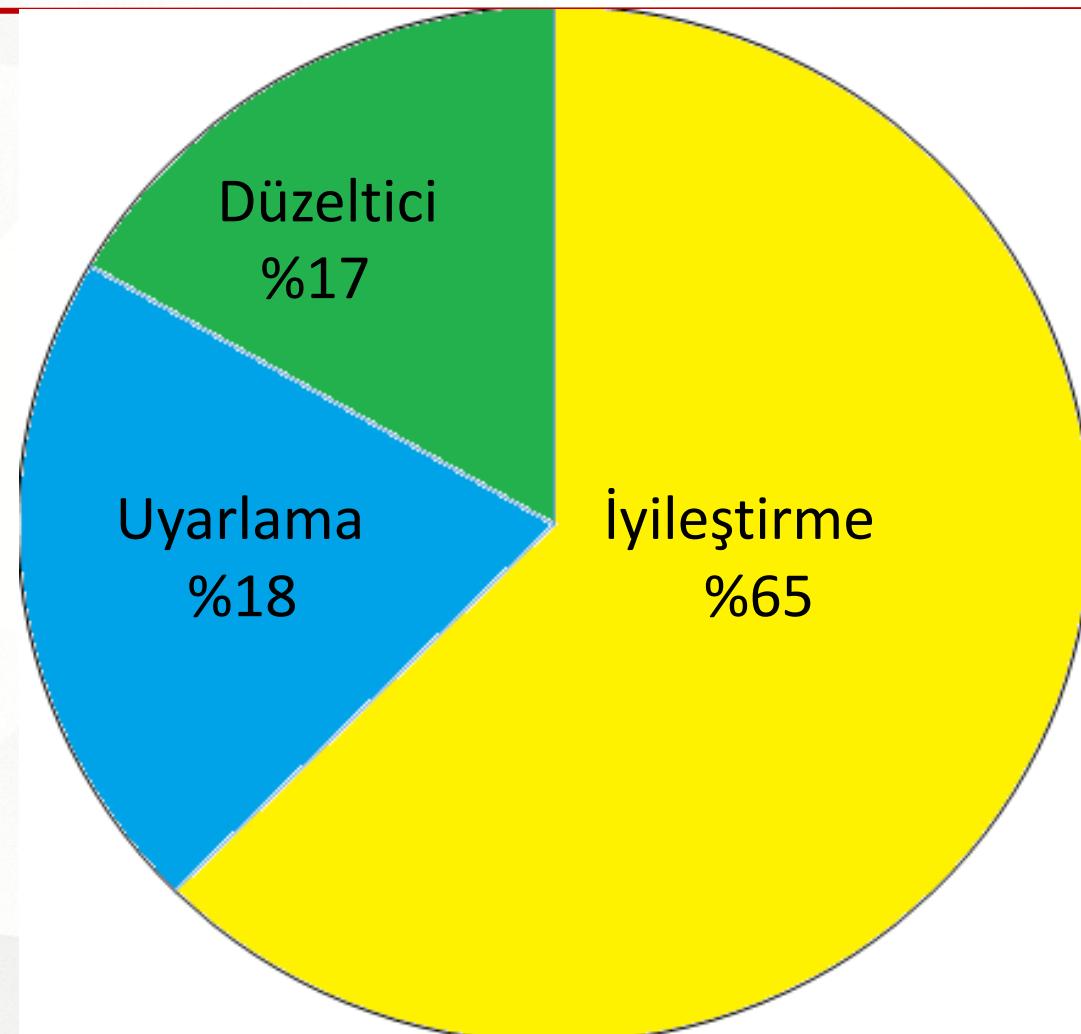
- ✓ Bir sistemi yeni gereksinimleri karşılamak üzere değiştirmeye

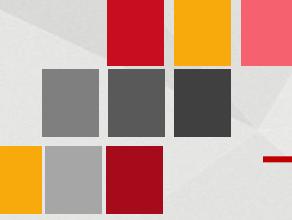
Düzelteci (corrective) Bakım

- ✓ Bir sistemi, eksikliklerin gereksinimlerine uygun şekilde düzeltmek için değiştirmeye



Bakım Oranları





Sistemin Geliştirilmesi

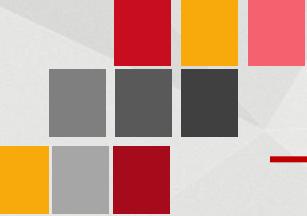
Sisteme geliştirildikten sonra sisteme yeni işlevsellik ekleme genellikle sistem geliştirilme aşamasına göre fazla maliyet gerektirir.

- ✓ Bakım personeli çoğu zaman deneyimsizdir ve uygulama alanına yabancıdır.
- ✓ Programlar kötü yapılandırılmış ve anlaşılması zor olabilir
- ✓ Sistemin karmaşıklığı, etki değerlendirmesini zorlaştırdığı için değişiklikler yeni hatalar getirebilir
- ✓ Sürekli değişimden dolayı yapı bozulabilir
- ✓ Programı açıklayan belgeler mevcut olmayabilir

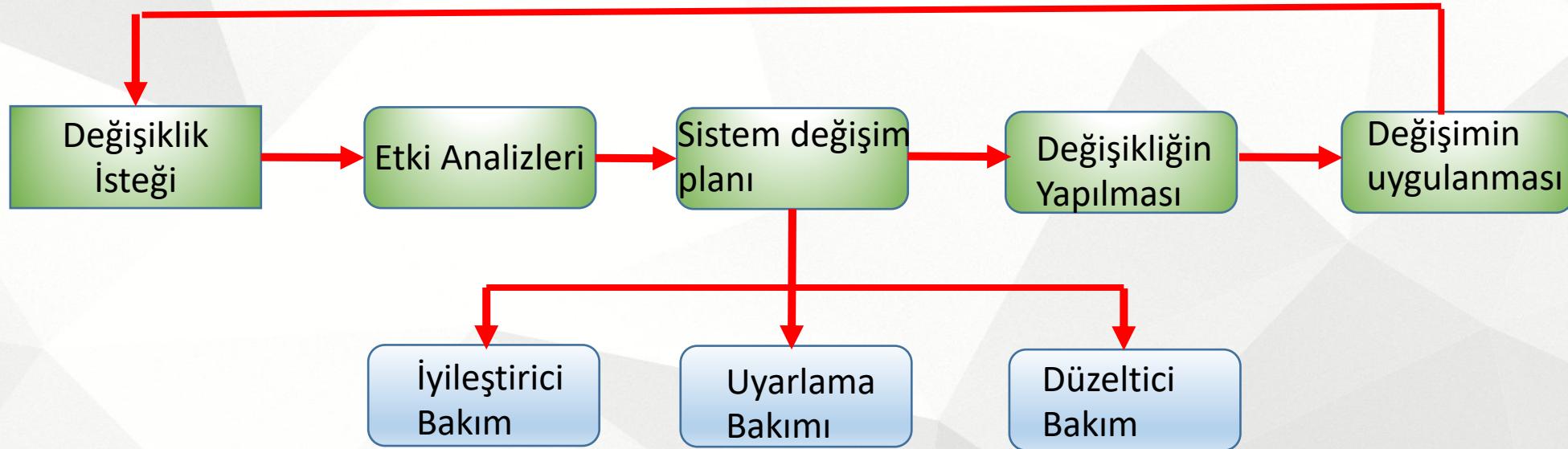
- ✓ Zorlayıcı ve yaratıcı olarak görülmeyen bakım, geliştirme kadrosunda zayıf bir imaja sahiptir.
- ✓ Yazılım muhafaza edildiğinde bakım maliyetleri artar mı?
- ✓ Sürdürülecek yazılım miktarı zamanla artar
- ✓ Yetersiz yapılandırma yönetimi sıklıkla bir sistemin farklı gösterimlerinin adım dışı olduğu anlamına gelir



- ✓ Bakım, müşterilerin değişim talepleri veya pazarlama gereksinimleri tarafından tetiklenir
- ✓ Değişiklikler normalde sistemin yeni bir sürümünde toplanır ve uygulanır
- ✓ Programlar bazen eksiksiz bir süreç yinelemesi yapılmaksızın onarılmaya ihtiyaç duyarlar, ancak dokümantasyon ve programların adım adım uygulanmasına yol açtığı için bu tehlikelidir (düzeltici ve uyarlama bakımları)



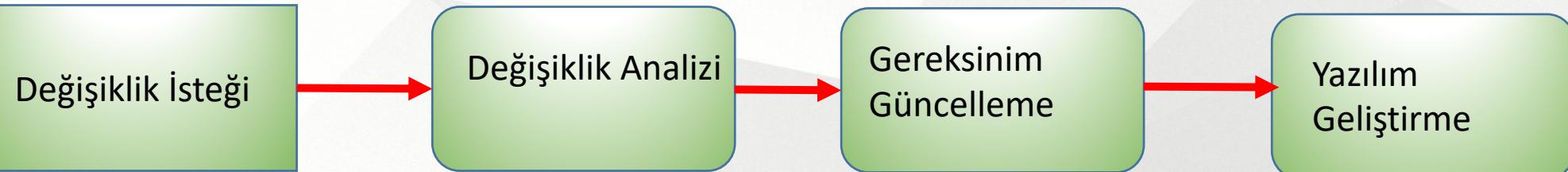
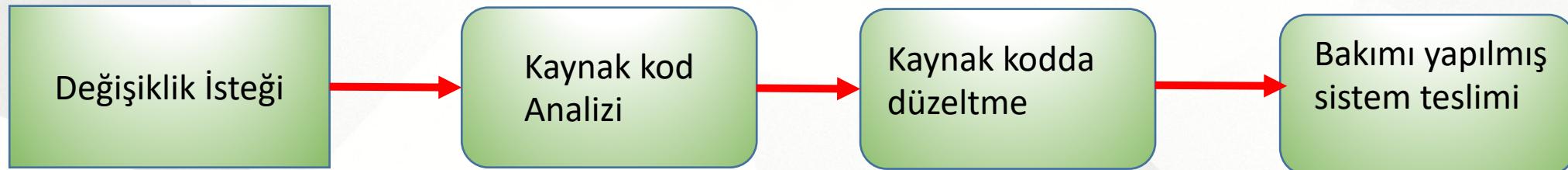
BAKIM SÜRECİ



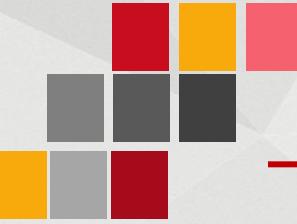


DEĞİŞİM SÜREÇLERİ

Arıza onarım süreci

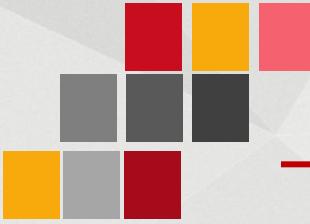


Yinelemeli geliştirme süreci



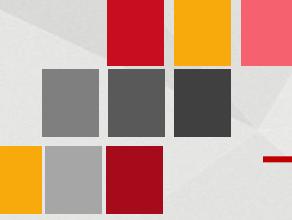
SİSTEM BELGELERİ

- ✓ Gereksinim Belgesi
- ✓ Program Tasarım Belgeleri
- ✓ Kaynak Kodu Listeleri
- ✓ Test Planları ve Geçerlilik Raporları
- ✓ Sistem Bakım Kılavuzu



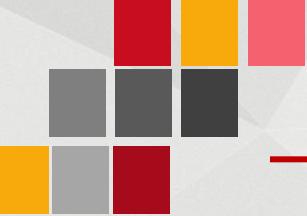
BELGE ÜRETİMİ

- ✓ Okuyucuya daha ayrıntılı teknik tanımlamalara yönlendiren genel bakışları içeren dokümanlar dökümanlar hazırlayın
- ✓ Kaliteli, okunabilir el kitapları hazırlayın - 20 yıl sürmelidirler
- ✓ Mümkin olduğunda araç tarafından üretilen belgeleri kullanın.

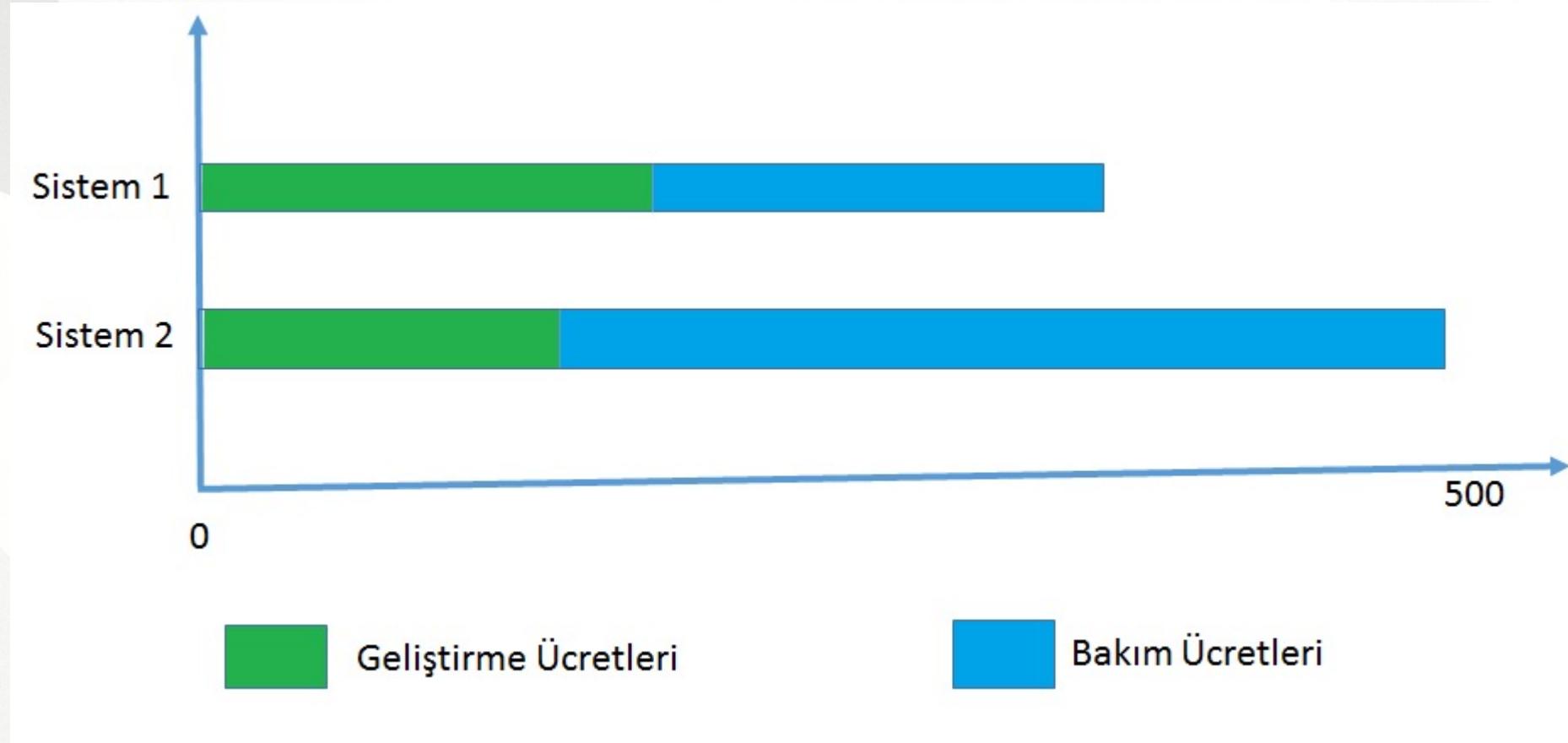


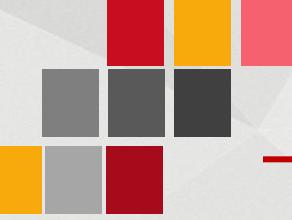
BAKIM ÜCRETLERİ

- ✓ Genellikle geliştirme maliyetlerinden daha büyütür (uygulamaya bağlı olarak 2 * ila 100 *)
- ✓ Hem teknik hem de teknik olmayan faktörlerden etkilenir
- ✓ Yazılım devam ettikçe artar. Bakım, yazılım yapısını bozar, bu nedenle daha fazla bakım yapılmasını zorlaştırır.
- ✓ Yaşlanma yazılım destek ücretini arttırır mı? (Örneğin, eski diller, derleyiciler vb.)



Geliştirme/Bakım Ücretleri





Bakım Ücret Faktörleri

- ✓ Modül bağımsızlığı

Bir modülü diğer modüllerin etkilemeksizin değiştirmek mümkün olmalıdır

- ✓ Programlama dili

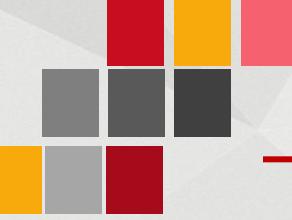
Üst düzey bir dilde geliştirilmiş programların bakımı daha kolaydır

- ✓ Programlama stili

İyi yapılandırılmış programların bakımı kolaydır

- ✓ Program doğrulama ve test

İyi test sonuçlarına sahip programlar daha az düzeltici bakıma ihtiyaç duyarlar.



Bakım Ücret Faktörleri

✓ Belgeleme

İyi belgeler, programların anlaşılmasını kolaylaştırır

✓ Konfigürasyon yönetimi

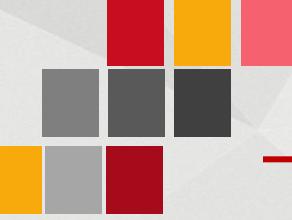
İyi konfigürasyon yönetimi, programlar ile belgeler arasındaki bağlantıların korunduğu anlamına gelir.

✓ Uygulama alanı

İyi anlaşılmış uygulama alanlarında bakım daha kolaydır

✓ Personel istikrarı

✓ Geliştirmede çalışan personelin bakıma aktarımı veya bakım süresince personel istikrarı bakım ücretlerini azaltır.



Bakım Ücret Faktörleri

✓ Program yaşı

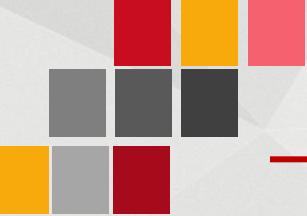
Daha eski program, daha pahalı (genellikle)

✓ Dış ortam

Bir program dış ortama bağımlıysa, çevresel değişiklikleri yansıtacak şekilde değiştirilmesi gerekebilir

✓ Donanım istikrarı

Stabil donanım için tasarlanan programlar, donanım değişiklikleri gibi değişime ihtiyaç duymazlar



Bakım Metrikleri

✓ Kontrol karmaşıklığı:

Programındaki koşullu ifadeleri inceleyerek ölçülebilir

✓ Veri karmaşıklığı:

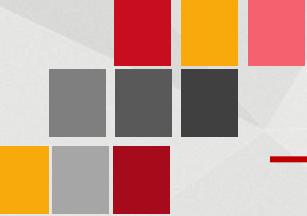
Veri yapıları ve bileşen ara yüzlerinin karmaşıklığı.

✓ Tanımlayıcı adlarının uzunluğu:

Daha uzun isimleri ima eder mi? Okunabilirlik

✓ Program yorumları:

Belki daha fazla yorum? Daha kolay bakım demek



Bakım Metrikleri

✓ Kullanıcı etkileşimi:

Programın fazla kullanıcı Girdi / Çıktı gerektirmesi değişiklik istek miktarını arttırmır.

✓ Hız ve yer gereksinimleri:

Geliştirilmesi zor, bakımı daha zor



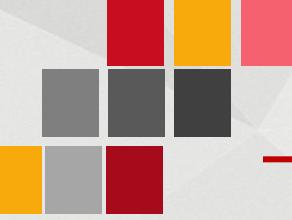
@



Nesne Yönelimli Programlama

Doç. Dr. Recep ERYİĞİT

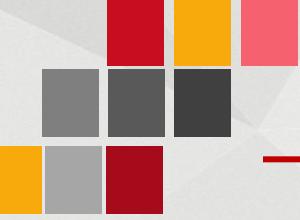
ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI



Dersin Hedefleri

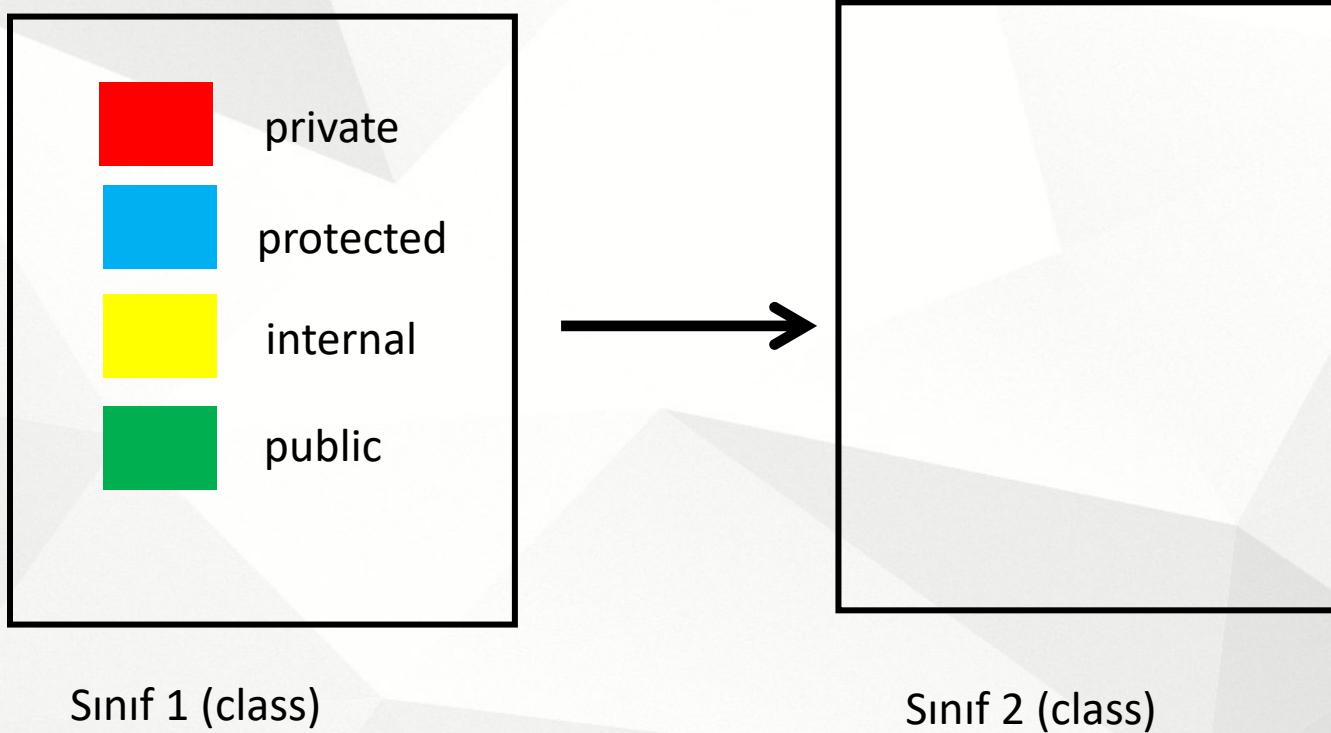
Nesne Yönelimli programlamaının karakteristik özelliklerinin C# dili syntax'ı öğrenilmesi.

- ✓ Değişken, sabitlerin kullanımı
- ✓ Döngü ve karar mekanizmaları
- ✓ Problem çözümünde metot kullanımı
- ✓ Veri soyutlama işlemininin class, struct ve enum'lar ile öğrenilmesi
- ✓ Masaüstü (Windows Form) uygulaması geliştirilmesi
- ✓ Web uygulaması geliştirilmesi
- ✓ Dosya, veritabanı ve thread işlemlerinin C# ile gerçekleştirilmesi

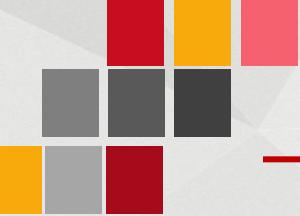


Nesne Yönelimli Programlamanın Karakteristik özellikleri

■ Kapsülleme (Encapsulation)

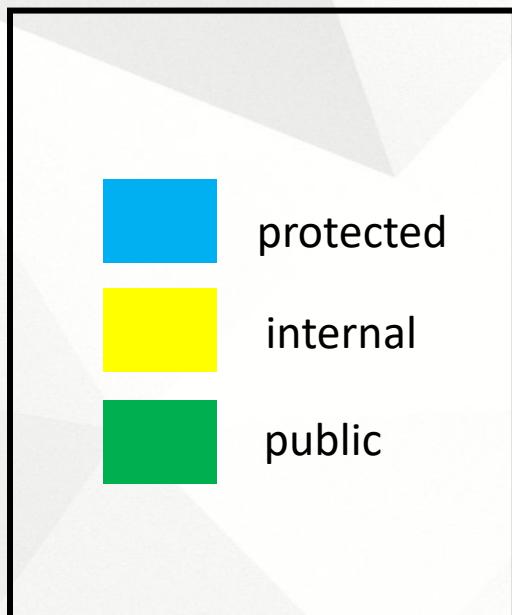


Bir sınıfın programlama birimlerinin diğer sınıflardan erişilebilirliği kapsülleme özelliğidir.



Nesne Yönelimli Programlamanın Karakteristik özellikleri

✓ Kalıtım (Inheritance)

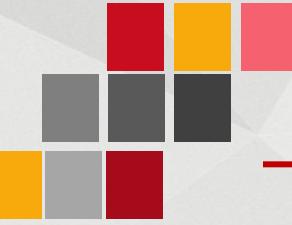


Sınıf 1 (class)



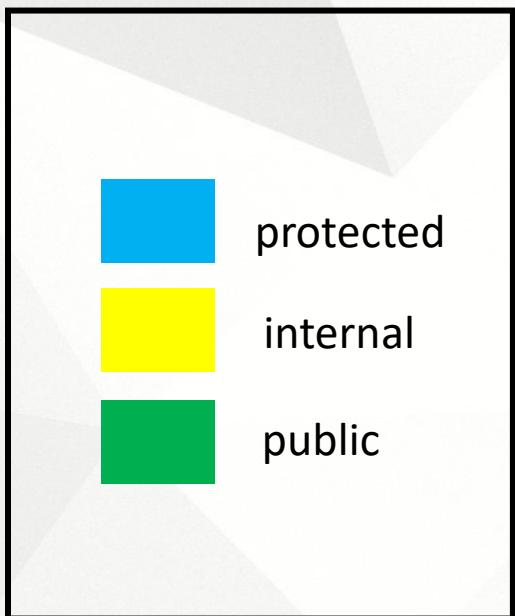
Sınıf 2 (class)

Bir sınıfın programlama birimlerinin diğer sınıf tarafından kullanılması kalıtım (miras) olarak bilinir.



Nesne Yönelimli Programlamanın Karakteristik özellikleri

✓ Çok Biçimlilik (Polymorphism)

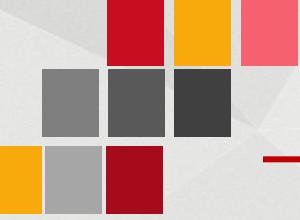


Sınıf 1 (class)



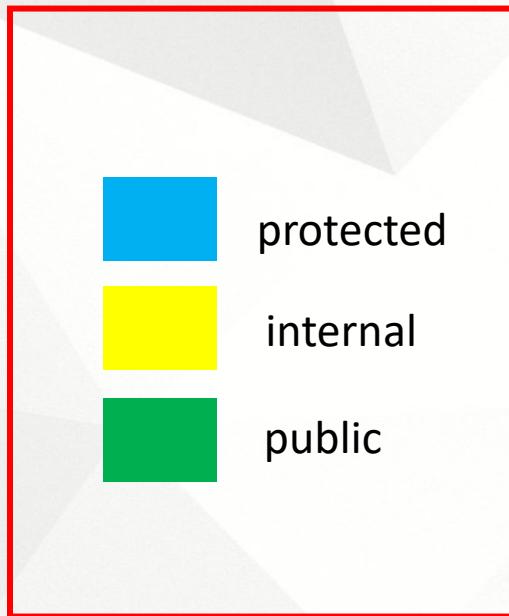
Sınıf 2 (class)

`Sınıf1 s1 = new Sınıf1();`
`Sınıf1 s2 = new Sınıf2();`
Miras veren sınıfın nesnesinin kendisi ve miras alanlarından üretilebilmesi.



Nesne Yönelimli Programlamanın Karakteristik özellikleri

✓ Soyutlama (Abstraction)



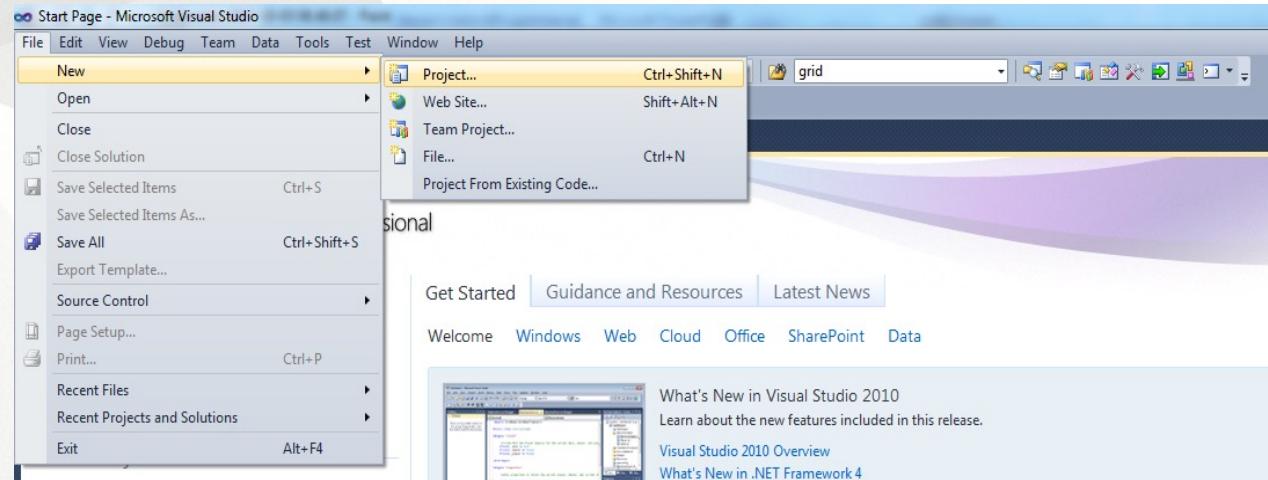
Sınıf 1 (class)



Sınıf 2 (class)

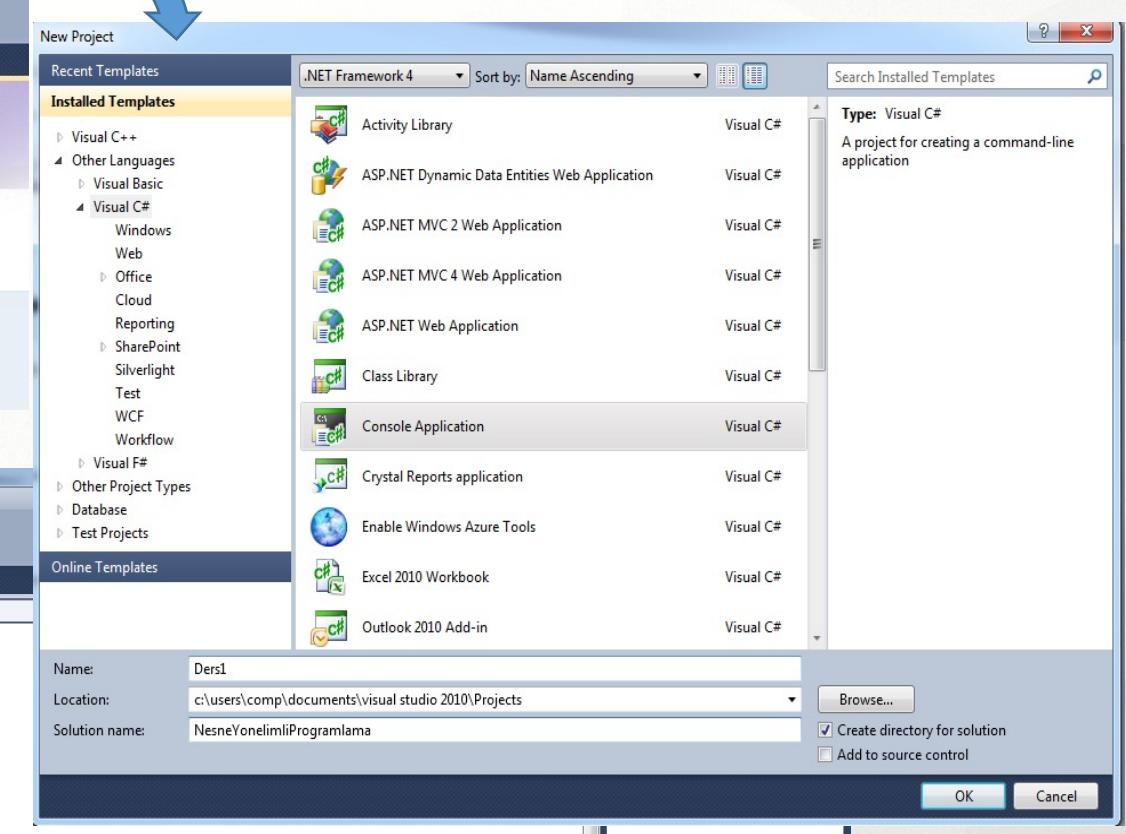
~~Sınıf1 s1 = new Sınıf1();~~
~~Sınıf1 s2 = new Sınıf2();~~
Miras veren sınıfın soyut tayımlaması ile
kendi nesnesinin oluşturulamaması.
Soyut sınıf miras vermek amacıyla
oluşturulur.

C# Başlangıç

A screenshot of the Microsoft Visual Studio IDE showing a code editor with C# code. The code defines a class named "Program" with a static void Main method. The code is as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Ders1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```





Değişken Bildirimi

- ✓ Bütün nesne yönelimli programlama dillerinde probleme ait değişkenler kullanılmadan önce derleyicilere değişkenin veri tip bildirilir. Dillerde bu işlem

veri_tipi değişken_ismi;

değişken_ismi =değer;

veya

veri_tipi değişken_ismi = değer;

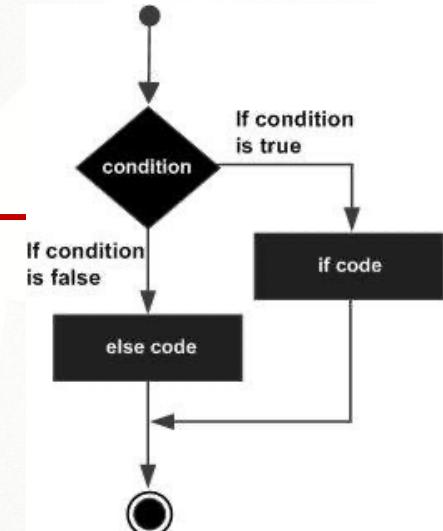
C# dilindeki tek değerli
değişkenler

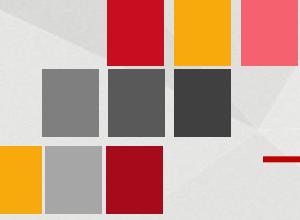
- Value types
 - Simple Types
 - Signed integral: `sbyte`, `short`, `int`, `long`
 - Unsigned integral: `byte`, `ushort`, `uint`, `ulong`
 - Unicode characters: `char`
 - IEEE floating point: `float`, `double`
 - High-precision decimal: `decimal`
 - Boolean: `bool`



Karar Yapıları

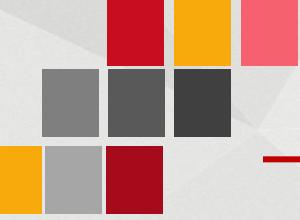
```
if(boolean_expression)
{
    /* statement(s) will execute if the boolean_expression is true */
}
else {
    /* statement(s) will execute if the boolean_expression is false */
}
```





Karar Yapıları

```
if(boolean_expression 1)
{
    /* Executes when the boolean expression 1 is true */
}
else if( boolean_expression 2)
{
    /* Executes when the boolean expression 2 is true */
}
else if( boolean_expression 3)
{
    /* Executes when the boolean expression 3 is true */
}
else
{
    /* executes when the none of the above condition is true */
}
```



Karar Yapıları

```
int caseSwitch = 1;

switch (caseSwitch)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```



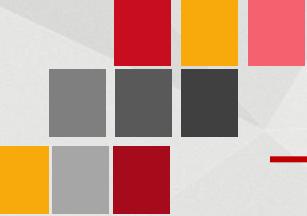

@



ANKARA ÜNİVERSİTESİ ENFORMATİK BÖLÜMÜ
TEZSİZ YÜKSEK LİSANS PROGRAMI

Yazılım Mühendisliği
Temel Süreçler – *NESNEYE DAYALI
PROGRAMLAMA*

Doç. Dr. Recep ERYİĞİT
₁



Yazılım çeşitliliği

- **Sistem Yazılımları:**

- Diğer programlara destek vermek için hazırlanan programlardır.

- **Gerçek Zamanlı (Real-Time) Yazılımlar**

- Gerçek dünyayı izleyen analiz eden yazılımlardır.

- **İş Yazılımları**

- İş bilgilerini işleyen ve analiz eden yazılımlardır.

- **Gömülü Yazılımlar**

- Salt okunur bellekte bulunurlar ve sistemin veya ürünün kontrolünde kullanılan yazılımlardır.

- **Kişisel Bilgisayar Yazılımları**

- Kişisel bilgisayarlar da geliştirilen ve kullanılan yazılımlardır.

- **Yapay Zeka Yazılımları**

- Karmaşık problemlerin çözümünde kullanılan yazılımlar.



Nesneye Yönelimli Modelleme

- ✓ Yaşadığımız dünyada herşey birer nesnedir. Örneğin kişiler, kütükler, sistemler, hepsi birer nesnedir. Her nesnenin bir çalışma tarzı ve kendine has dışa yönelik davranışları vardır. Bir nesne ile interaksiyona girmek için, nesnenin içinde ne bulunduğu bilmemize gerek yoktur. Belirli arayüzler (Interface) üzerinden mesajlar göndererek, nesne ile bağlantı kurulur. Nesne bu mesajı alır ve gerekli işlemleri yapar.



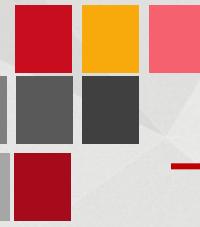
Nesneye Yönelimli Tasarım Amacı

✓ Sistem elemanlarını daha fazla **tekrar kullanılabilir (reusable)** hale getirerek sistem kalitesini iyileştirmek ve sistem analiz ve tasarımının üretkenliğini artırmaktır. Nesne yönelimliliğin ardından bir diğer anahtar fikir, **kalıtsallık (inheritance)**'tır. Nesneler, yapısal ve davranışsal karakteristikleri paylaşan nesne grupları şeklinde tanımlanan **nesne sınıfları (object classes)** şeklinde düzenlenirler. Kalıtsallık, mevcut sınıfların bazı karakteristiklerini paylaşan yeni sınıfların (class) yaratılmasına olanak tanır.



Nesne dayalı analizde temel adımlar

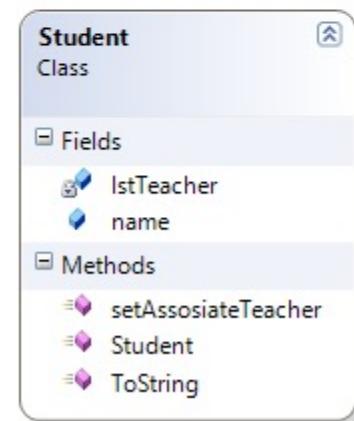
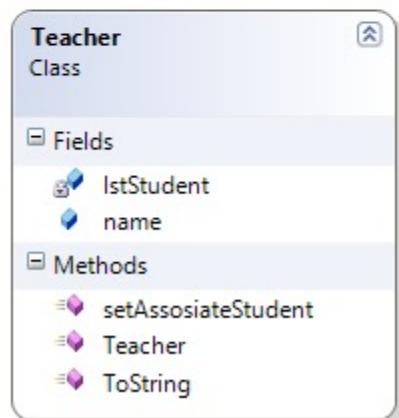
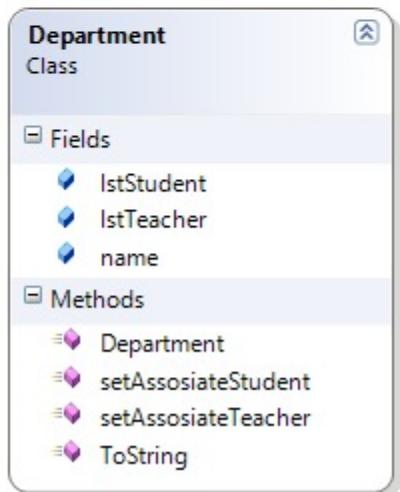
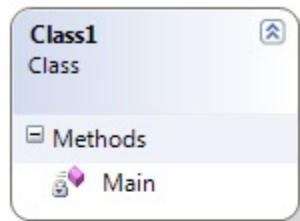
- Nesnelerin belirlenmesi
- Belirlenen nesnelerin yapı ve davranışlarının tanımlanması
- Nesnelerin birbirleri ile olan ilişkilerinin belirlenmesi
- Nesne ara yüzlerinin ortaya konması



Nesne dayalı tasarımda temel görevler

- Sistem gereksinimleri sağlanacak şekilde, nesnelerin davranışlarının ve diğer nesnele iletişimlerinin ayrıntıları ile modellenmesi
- Kalıtsallık avantajlarından daha iyi yararlanılacak şekilde nesnelerin yeniden incelenmesi ve yeniden tanımlanması gereklidir.

ÖRNEK



```
namespace Composition
{
    class University
    {
        public string name;
        public Department department;
        public University(string n)
        {
            name = n;
        }
        public void createDepartments(string n)
        {
            department = new Department(n);
            Console.WriteLine("Uni Name : " + name + "Depat name " + n);
        }
    }
}
```

```
namespace Composition
{
    class Department
    {
        public string name;
        public List<Teacher> lstTeacher;
        public List<Student> lstStudent;
        public Department(string n)
        {
            name = n;
        }
        public override string ToString()
        {
            return "Department Name: " + name;
        }
        public void setAssosiateTeacher(List<Teacher> lst)
        {
            lstTeacher = lst;
            foreach (var x in lstTeacher)
            {
                Console.WriteLine(x);
            }
        }
        public void setAssosiateStudent(List<Student> lst)
        {
            lstStudent = lst;
            foreach (var x in lstStudent)
            {
                Console.WriteLine(x);
            }
        }
    }
}
```

```
namespace Composition
{
    public class Student
    {
        public string name;
        List<Teacher> lstTeacher;
        public Student(string n)
        {
            name = n;
        }
        public override string ToString()
        {
            return "Student Name: " + name;
        }
        public void setAssosiateTeacher(List<Teacher> lst)
        {
            lstTeacher = lst;
            foreach (var x in lstTeacher)
            {
                Console.WriteLine(x);
            }
        }
    }
}
```

```
namespace Composition
{
    public class Teacher
    {
        public string name;
        List<Student> lstStudent;
        public Teacher(string n)
        {
            name = n;
        }
        public override string ToString()
        {
            return "Teaher Name: " + name;
        }
        public void setAssosiateStudent(List<Student> lst)
        {
            lstStudent = lst;
            foreach (var x in lstStudent)
            {
                Console.WriteLine(x);
            }
        }
    }
}
```

```
namespace Composition
{
    class Class1
    {
        static void Main(string[] args)
        {
            List<Student> lstStudent = new List<Student>();
            Student s1 = new Student("Fatma");
            Student s2 = new Student("Resul");
            Student s3 = new Student("Egemen");
            lstStudent.Add(s1);
            lstStudent.Add(s2);
            lstStudent.Add(s3);
            Teacher t1 = new Teacher("recep");

            Department dp = new Department("bil");
            dp.lstTeacher = new List<Teacher>();
            dp.lstTeacher.Add(t1);

            University u1 = new University("Ankara");
            u1.createDepartments("Bilgisayar");
            u1.department.setAssociateTeacher(dp.lstTeacher);
            u1.department.setAssociateStudent(lstStudent);

            Console.Read();
        }
    }
}
```