BİL 407 Yazılım Mühendisliği
Hafta 10

Doç. Dr. Sedat Akleylek

sedat.akleylek@bil.omu.edu.tr

İçerik

- UML kullanarak nesneye yönelik tasarım
- Tasarım desenleri
- Uygulama sorunları
- Açık kaynak geliştirme

Kaynaklar

- *Ian Sommerville, Software Engineering, 10th Edition, Addison-Wesley, 2015.*
- *SWEBOK, Guide to the Software Engineering Body of Knowledge: 2004 Version, IEEE.*
- https://www.youtube.com/user/SoftwareEngBook
- IEEE-Software Requirements Specification Template

Tasarım ve Uygulama

- Yazılım tasarımı ve uygulaması, yazılım mühendisliği sürecinde yürütülebilir bir yazılım sisteminin geliştirildiği aşamadır.
- Yazılım tasarım ve uygulama faaliyetleri her zaman birbiriyle ilişkilidir.
  - Yazılım tasarımı, bir müşterinin gereksinimlerine göre yazılım bileşenlerini ve bunların ilişkilerini tanımladığınız yaratıcı bir etkinliktir.
  - Uygulama, tasarımın bir program olarak gerçekleştirilmesi sürecidir.

İnşa Et veya Satın Al

- Çok çeşitli alanlarda, kullanıcıların gereksinimlerine göre uyarlanabilen ve özelleştirilebilen hazır sistemler (buy off-the-shelf systems - COTS) satın almak artık mümkün.
- Örneğin bir tıbbi kayıt sistemi uygulamak istiyorsanız, hastanelerde halihazırda kullanılan bir paketi satın alabilirsiniz. Geleneksel bir programlama dilinde bir sistem geliştirmek yerine bu yaklaşımı kullanmak daha ucuz ve daha hızlı olabilir.
- Bu şekilde bir uygulama geliştirdiğinizde, tasarım süreci, sistem gereksinimlerini sağlamak için o sistemin konfigürasyon özelliklerinin nasıl kullanılacağıyla ilgilenir.

UML Kullanarak Nesneye Yönelik Tasarım

Nesne Yönelimli Tasarım Süreci

- Yapılandırılmış nesneye yönelik tasarım süreçleri, bir dizi farklı sistem modelinin geliştirilmesini içerir.

- Bu modellerin geliştirilmesi ve bakımı için çok çaba gerektirirler ve küçük sistemler için bu, uygun maliyetli olmayabilir.

- Ancak, farklı gruplar tarafından geliştirilen büyük sistemler için tasarım modelleri önemli bir iletişim mekanizmasıdır.

Süreç Aşamaları

- Süreci kullanan kuruluşa bağlı olan çeşitli farklı nesneye yönelik tasarım süreçleri vardır.

- Bu süreçlerdeki ortak faaliyetler şunları içerir:
  - Sistemin bağlamını ve kullanım biçimlerini tanımlayın;
  - Sistem mimarisini tasarlayın;
  - Ana sistem nesnelerini tanımlayın;
  - Tasarım modelleri geliştirin;
  - Nesne ara yüzlerini belirtin.

- Örnek olarak «kötü hava durumu istasyonu» için bir tasarım kullanılarak süreç aşamaları gösterilecektir.

Sistem Bağlamı ve Etkileşimleri

- Tasarlanmakta olan yazılım ile dış ortamı arasındaki ilişkilerin anlaşılması, gerekli sistem işlevselliğinin nasıl sağlanacağına ve sistemin çevresiyle iletişim kuracak şekilde nasıl yapılandırılacağına karar vermek için çok önemlidir.

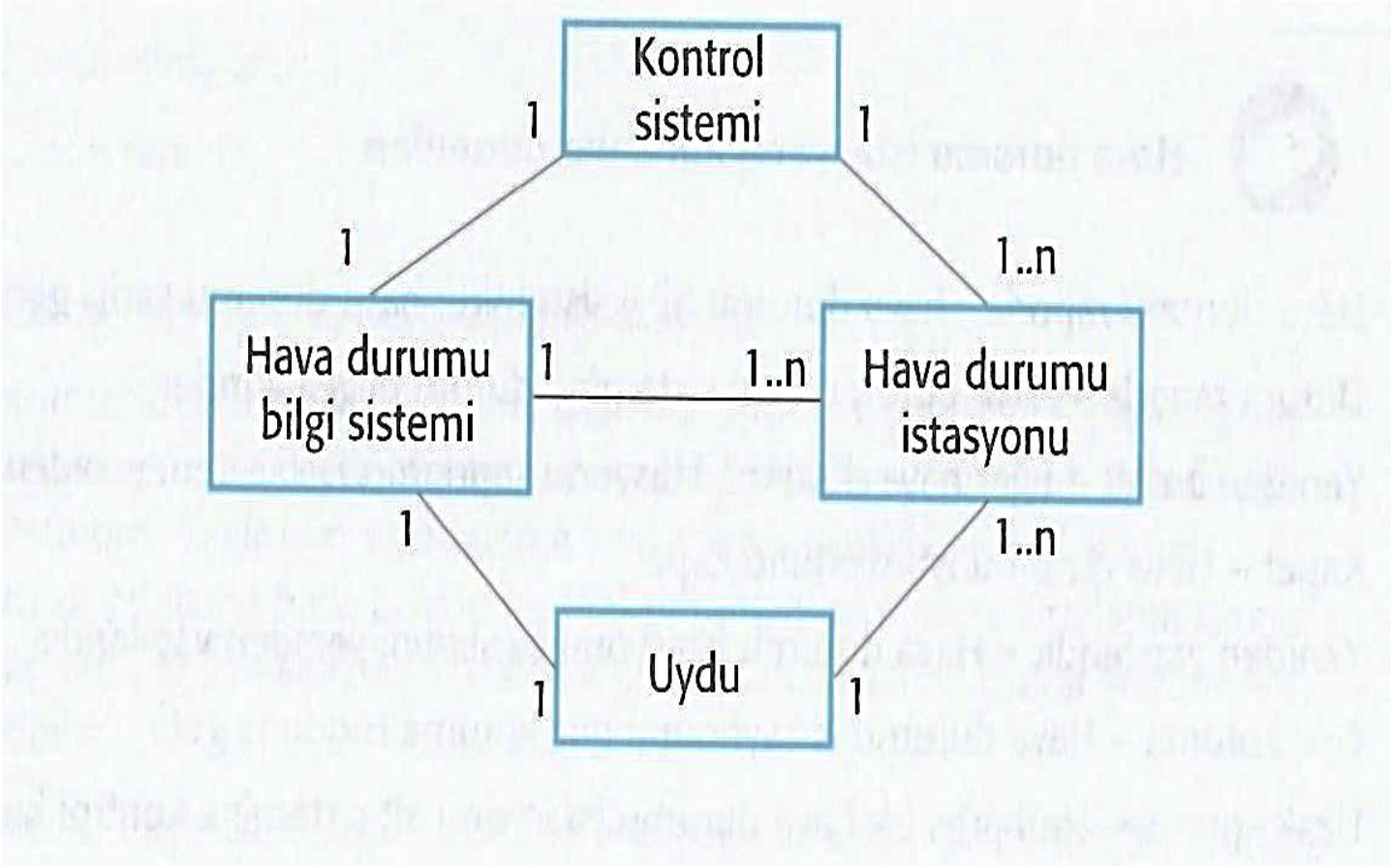
- Bağlamın anlaşılması, sistemin sınırlarını da belirlemenizi sağlar. Sistem sınırlarının belirlenmesi, tasarlanan sistemde hangi özelliklerin uygulanacağına ve diğer ilişkili sistemlerde hangi özelliklerin olduğuna karar vermenize yardımcı olur.

Bağlam ve Etkileşim Modelleri

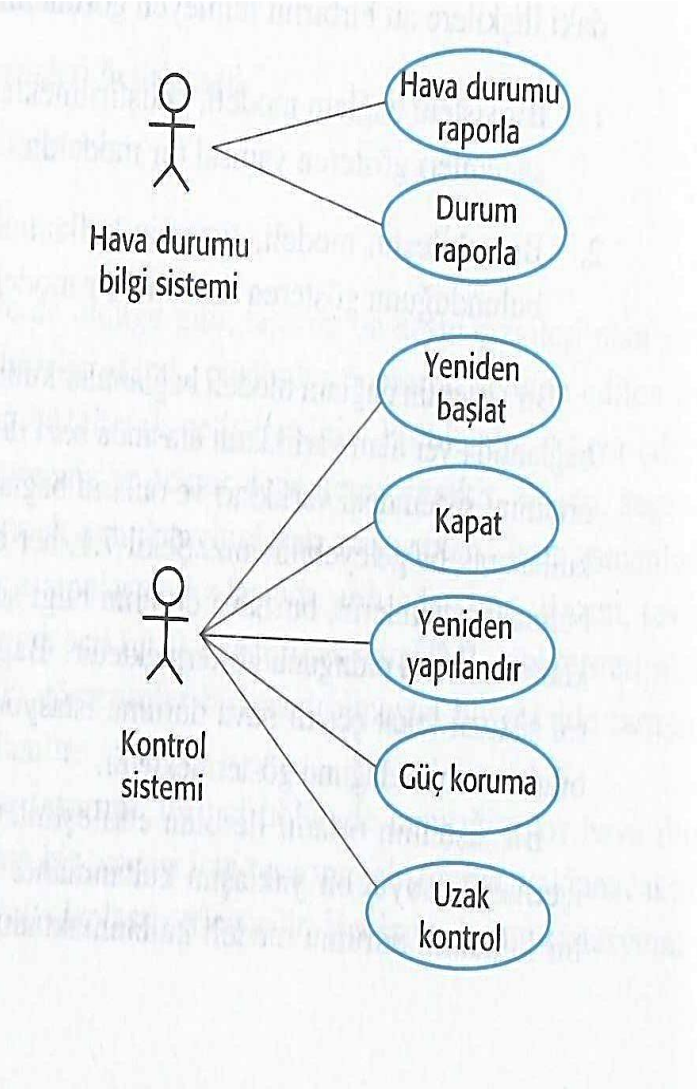
- Sistem bağlam modeli, geliştirilmekte olan sistemin ortamındaki diğer sistemleri gösteren yapısal bir modeldir.

- Etkileşim modeli, sistemin kullanıldıkça çevresi ile nasıl etkileşime girdiğini gösteren dinamik bir modeldir.

Meteoroloji İstasyonu İçin Sistem İçeriği



Hava Durumu İstasyonu Kullanım Durumları



Hava Durumunu Raporla Örneği İçin Tanımlar

Sistem	Hava Durumu İstasyonu
Kullanım Durumu	Hava durumunu raporla
Aktörler	Hava durumu bilgi sistemi, Hava durumu istasyonu
Veri	Hava durumu istasyonu, toplama periyodunda cihazlardan toplanan hava durumu verilerinin bir özetini hava durumu bilgi sistemine gönderir. Gönderilen veriler maksimum, minimum ve ortalama yer ve hava sıcaklıklarıdır; maksimum, minimum ve ortalama hava basınçları; maksimum, minimum ve ortalama rüzgar hızları; toplam yağış; ve beş dakikalık aralıklarla örneklenen rüzgar yönünden oluşmaktadır.
Uyaranlar	Hava durumu bilgi sistemi, hava istasyonu ile bir uydu iletişim bağlantısı kurar ve verilerin iletimini talep eder.
Cevaplar	Özetlenen veriler hava durumu bilgi sistemine gönderilir.
Yorumlar	Hava durumu istasyonlarından genellikle saatte bir rapor vermeleri istenir, ancak bu sıklık bir istasyondan diğerine farklılık gösterebilir ve gelecekte değiştirilebilir.

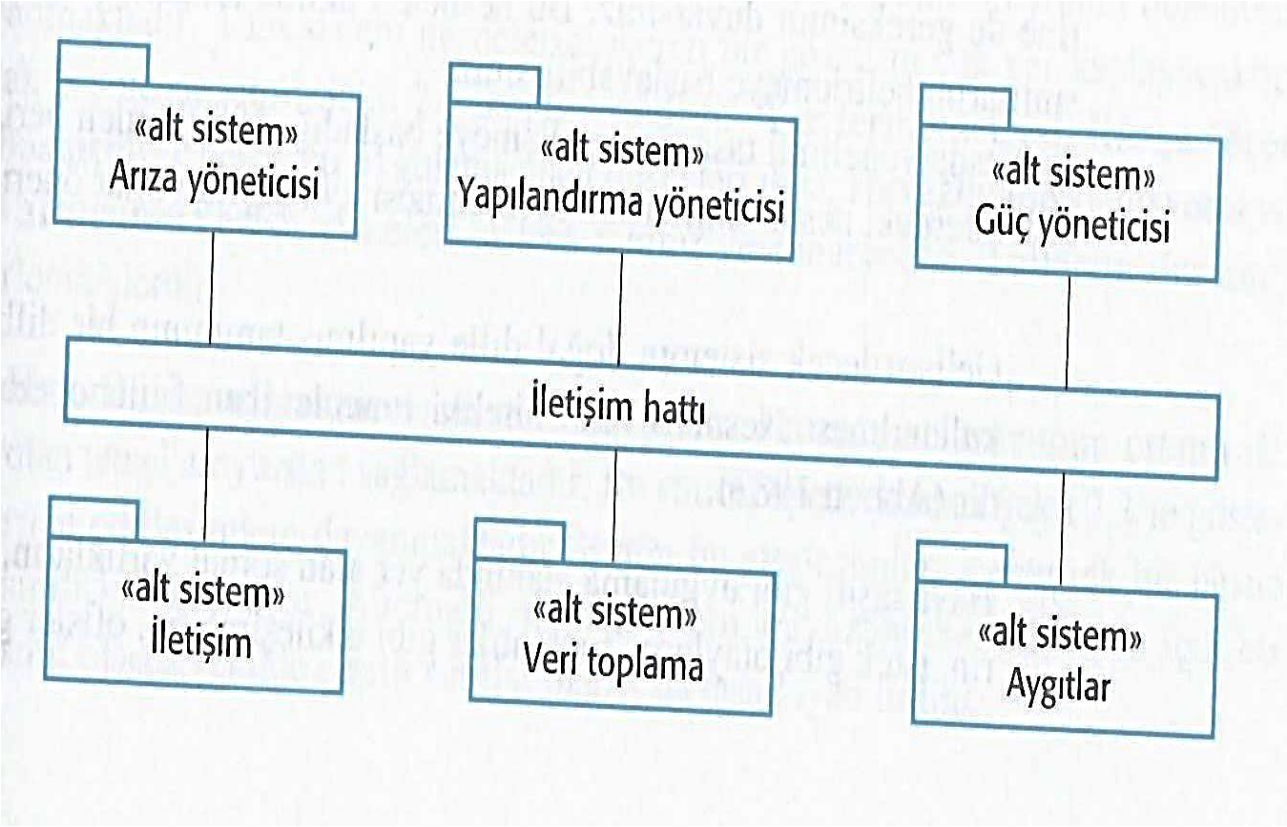
Mimari Tasarım

- Sistem ve çevresi arasındaki etkileşimler anlaşıldıktan sonra, bu bilgi sistem mimarisini tasarlamak için kullanılır.

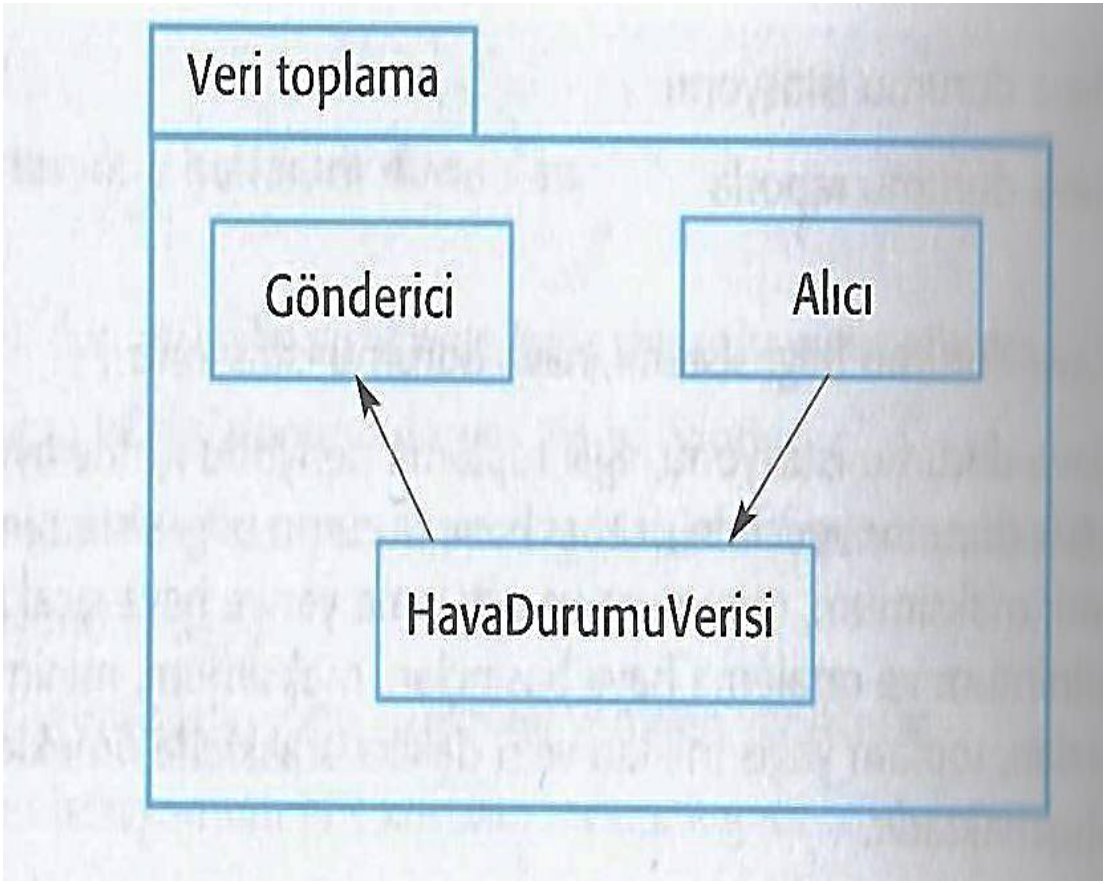
- Sistemi oluşturan ana bileşenleri ve bunların etkileşimlerini belirlersiniz ve ardından bileşenleri katmanlı veya istemci-sunucu modeli gibi mimari bir model kullanarak düzenleyebilirsiniz.

- Hava durumu istasyonu, ortak bir altyapı üzerinde mesajlar yayınlayarak iletişim kuran bağımsız alt sistemlerden oluşur.

Meteoroloji İstasyonunun Üst Düzey Mimarisi



Veri Toplama Sistemi Mimarisi



Nesne Sınıfı Tanımlama

- Nesne sınıflarını belirlemek, genellikle nesne yönelimli tasarımın zor bir parçasıdır.
- Nesne tanımlaması için 'sihirli formül' yoktur. Sistem tasarımcılarının becerisine, deneyimine ve alan bilgisine dayanır.
- Nesne tanımlama yinelemeli bir süreçtir. İlk seferde doğru yapılması beklenmez.

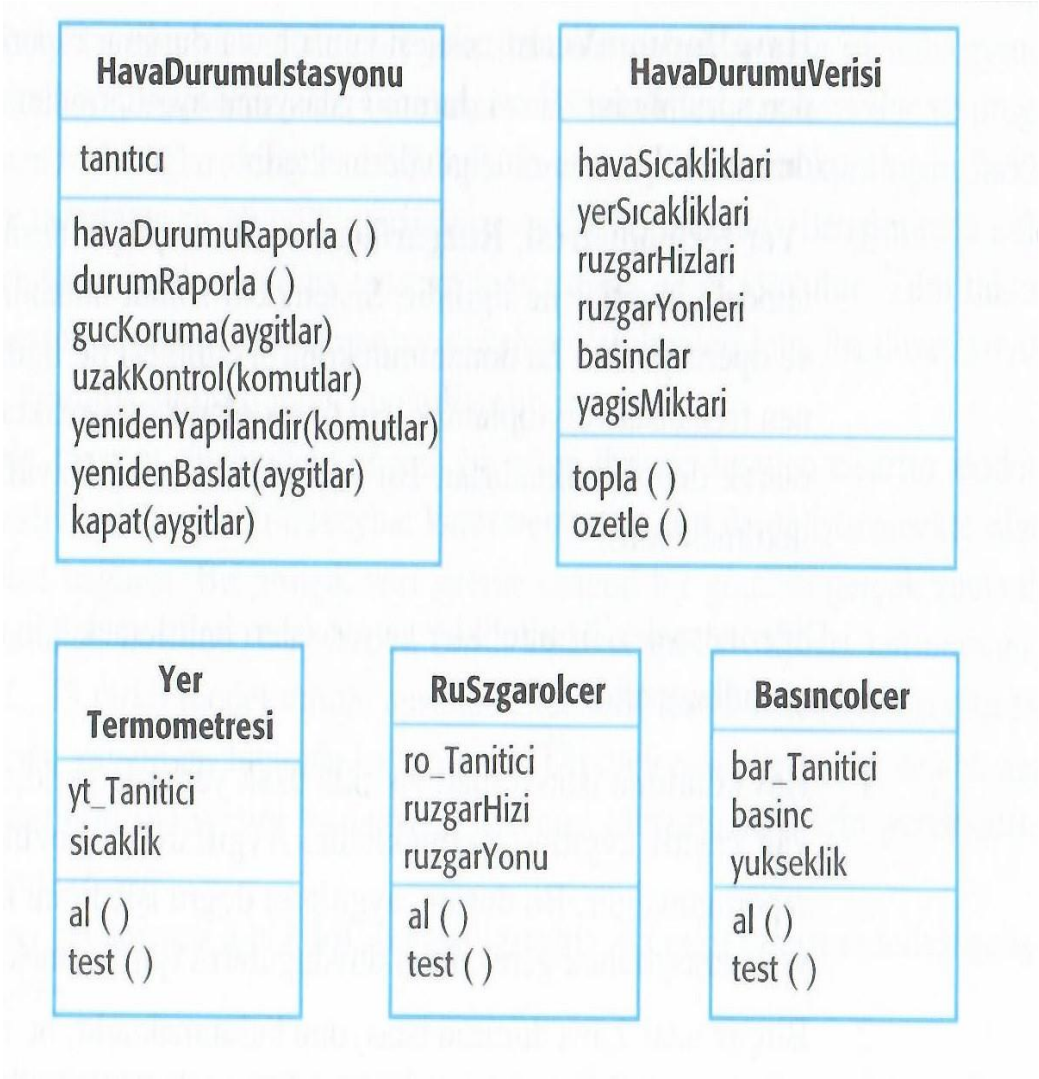
Tanımlamaya Yönelik Yaklaşımlar

- Sistemin doğal dil tanımına dayalı bir dilbilgisi yaklaşımı kullanın.
- Tanımlamayı, uygulama alanındaki somut şeylere dayandırın.
- Davranışsal bir yaklaşım kullanın ve hangi davranışa neyin katıldığına bağlı olarak nesneleri tanımlayın.
- Senaryoya dayalı bir analiz kullanın. Her senaryodaki nesneler, nitelikler ve yöntemler tanımlanır.

«Hava Durumu İstasyonu» Nesne Sınıfları

- Meteoroloji istasyonu sistemindeki nesne sınıfı tanımlaması, sistemdeki somut donanım ve verilere dayanabilir:
- Yer termometresi, Anemometre, Barometre
  - Sistemdeki araçlarla ilgili "donanım" nesneleri olan uygulama etki alanı nesneleri.
- Meteoroloji istasyonu
  - Meteoroloji istasyonunun çevresiyle olan temel ara yüzü. Bu nedenle, kullanım durumu modelinde tanımlanan etkileşimleri yansıtır.
- Hava durumu verileri
  - Enstrümanlardaki özetlenmiş verileri kapsüller.

Hava Durumu İstasyonu Nesneleri



Tasarım Modelleri

- Tasarım modelleri, nesneleri ve nesne sınıflarını ve bu varlıklar arasındaki ilişkileri gösterir.
- İki tür tasarım modeli vardır:
  - Yapısal modeller, sistemin statik yapısını nesne sınıfları ve ilişkiler açısından tanımlar.
  - Dinamik modeller, nesneler arasındaki dinamik etkileşimleri tanımlar.

Tasarım Modellerine Örnekler

- Nesnelerin mantıksal gruplamalarını tutarlı alt sistemler halinde gösteren alt sistem modelleridir.
- Nesne etkileşimlerinin sırasını gösteren dizi modelleridir.
- Olaylara tepki olarak nesnelerin durumlarını nasıl değiştirdiğini gösteren durum makinesi modelleridir.
- Diğer modeller, kullanım durumu modelleri, toplama modelleri, genelleme modelleri vb. içerir.

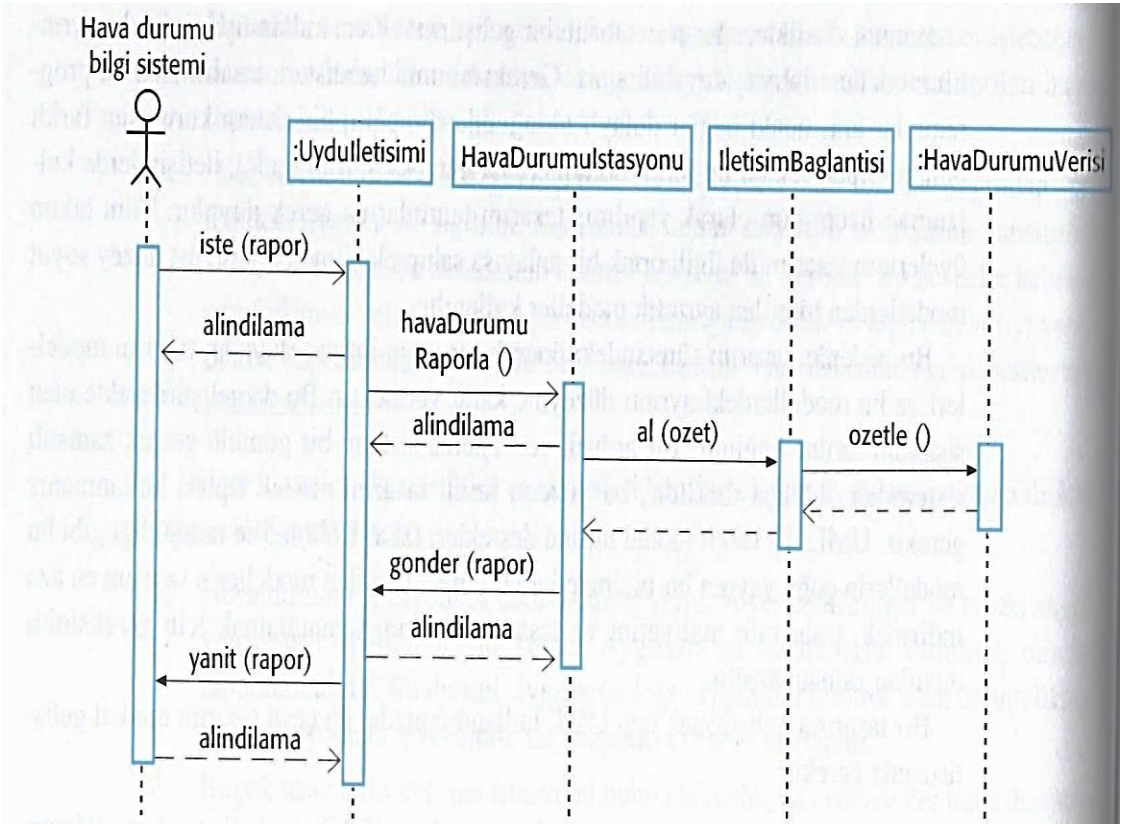
Alt Sistem Modelleri

- Tasarımın mantıksal olarak ilişkili nesne grupları halinde nasıl düzenlendiğini gösterir.
- UML'de bunlar, bir paketleme/kapsülleme yapısı olan paketler kullanılarak gösterilir. Bu mantıksal bir modeldir.
- Sistemdeki nesnelerin gerçek organizasyonu farklı olabilir.

Sıra Modelleri

- Sıra modelleri, gerçekleşen nesne etkileşimlerinin sırasını gösterir.
  - Nesneler üstte yatay olarak düzenlenmiştir;
  - Zaman dikey olarak temsil edildiğinden modeller yukarıdan aşağıya okunur;
  - Etkileşimler etiketli oklarla temsil edilir. Farklı ok stilleri, farklı etkileşim türlerini temsil eder;
  - Bir nesne yaşam çizgisindeki ince bir dikdörtgen, nesnenin sistemde kontrol eden nesne olduğu zamanı temsil eder.

Veri Toplamayı Tanımlayan Sıra Diyagramı



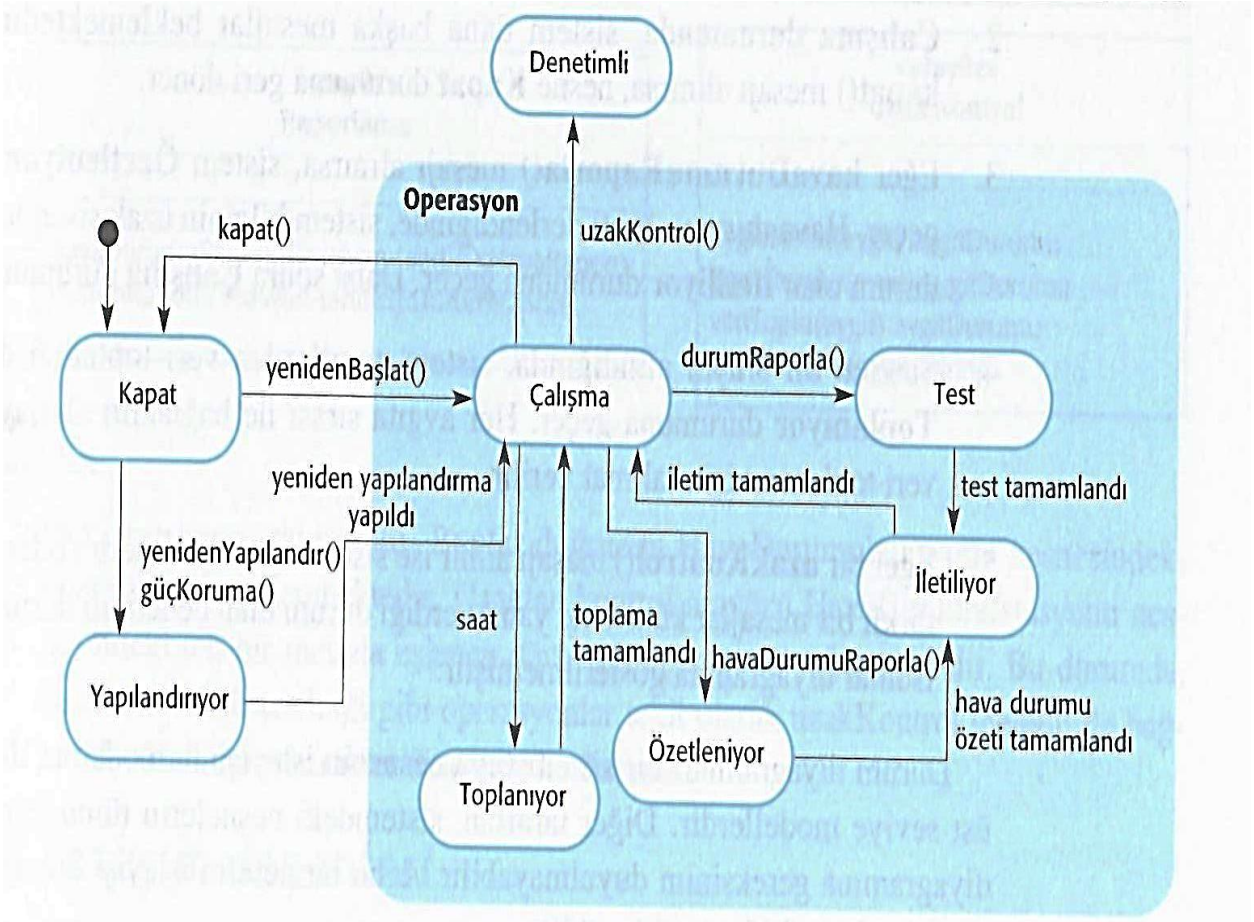
Durum Diyagramları

- Durum diyagramları, nesnelerin farklı hizmet taleplerine nasıl yanıt verdiğini ve bu talepler tarafından tetiklenen durum geçişlerini göstermek için kullanılır.

- Durum diyagramları, bir sistemin veya bir nesnenin çalışma zamanı davranışının yararlı üst düzey modelleridir.

- Sistemdeki tüm nesneler için genellikle bir durum diyagramına ihtiyacınız yoktur. Bir sistemdeki nesnelerin çoğu nispeten basittir ve bir durum modeli tasarıma gereksiz ayrıntılar ekler.

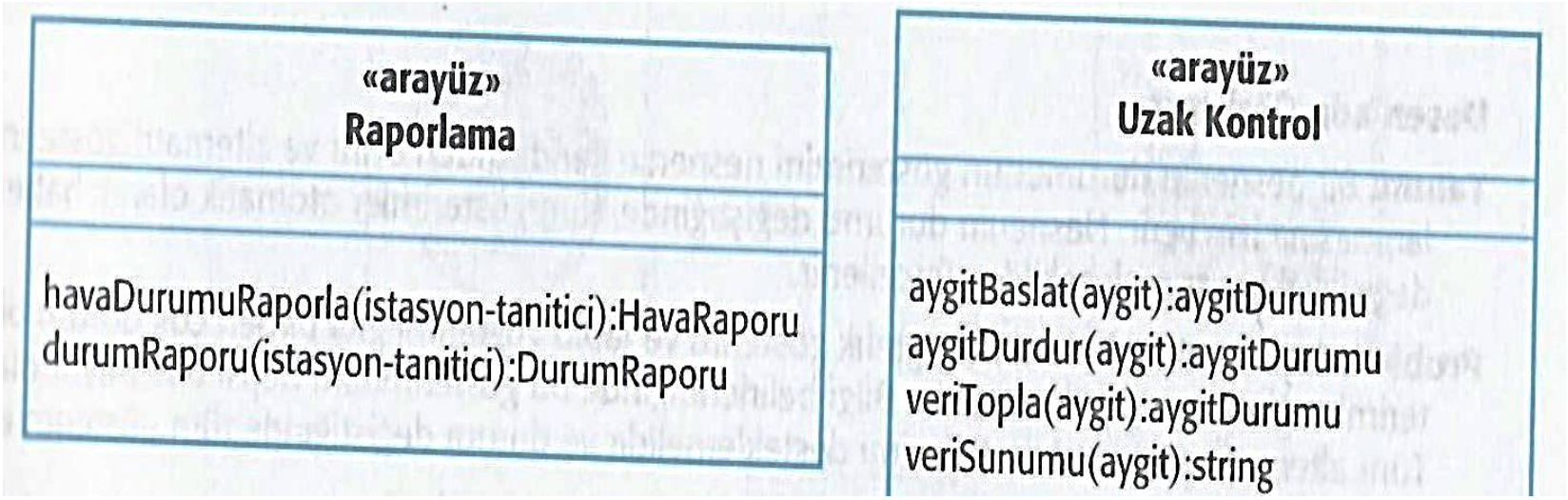
Hava İstasyonu Durum Diyagramı



Arayüz Özellikleri

- Nesne arayüzleri, nesnelerin ve diğer bileşenlerin paralel olarak tasarlanabilmesi için belirlenmelidir.
- Tasarımcılar arayüz temsilini tasarlamaktan kaçınmalı, ancak bunu nesnenin kendisinde saklamalıdır.
- Nesnelerin, sağlanan yöntemlere bakış açıları olan birkaç arabirimi olabilir.
- UML, arayüz belirtimi için sınıf diyagramları kullanır, ancak Java da kullanılabilir.

Hava Durumu İstasyonu Arayüzleri



Tasarım Desenleri

Tasarım Desenleri

- Bir tasarım deseni, bir problem ve çözümü hakkındaki soyut bilgileri yeniden kullanmanın bir yoludur.
- Bir desen, sorunun tanımı ve çözümünün özüdür.
- Farklı ortamlarda yeniden kullanılmak için yeterince soyut olmalıdır.
- Desen açıklamaları genellikle kalıtım ve çok biçimlilik gibi nesne yönelimli özelliklerden yararlanır.

Desenler

Desenler ve Desen Dilleri, en iyi uygulamaları, iyi tasarımları tanımlamanın ve başkalarının bu deneyimi yeniden kullanabileceği şekilde deneyimi yakalamanın yollarıdır.

Desen Öğeleri

- **İsim**
  - Anlamlı bir model tanımlayıcı.
- **Sorun Açıklaması**
- **Çözüm açıklaması**
  - Somut bir tasarım değil, farklı şekillerde somutlaştırılabilen bir tasarım çözümü için bir şablon.
- **Sonuçlar**
  - Deseni uygulamanın sonuçları ve ikilemler.

Gözlemci Desenleri

- **İsim**
  - Gözlemci.
- **Açıklama**
  - Nesne durumunun görüntüsünü nesnenin kendisinden ayırır.
- **Problem Açıklaması**
  - Birden fazla durum göstergesi gerektiğinde kullanılır.
- **Çözüm Açıklaması**
  - UML açıklamalı slaytta yer almaktadır.
- **Sonuçlar**
  - Görüntü performansını artırmaya yönelik optimizasyonlar pratik değildir.

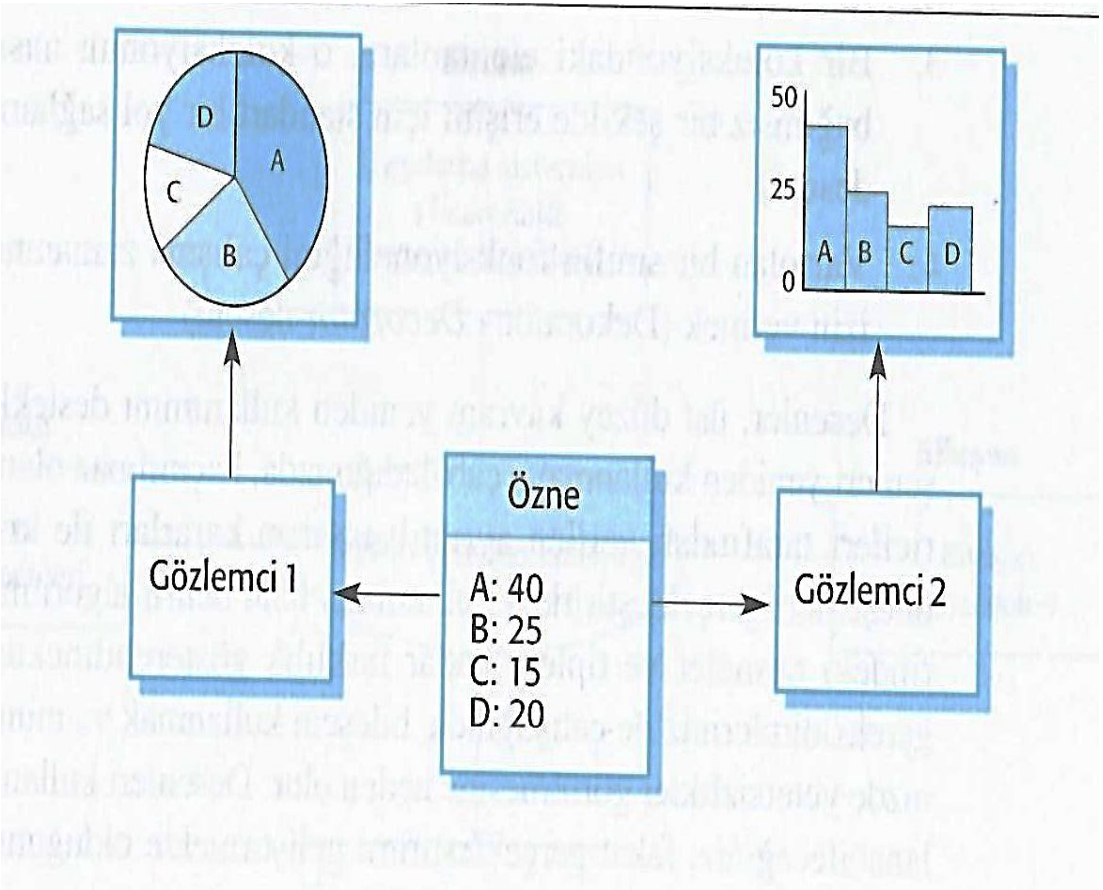
Gözlemci Deseni

Desen Adı	Gözlemci
Tanım	Bir nesnenin durumunun görüntüsünü nesnenin kendisinden ayırır ve alternatif görüntülerin sağlanmasına izin verir. Nesne durumu değiştiğinde, tüm ekranlar otomatik olarak bilgilendirilir ve değişikliği yansıtacak şekilde güncellenir.
Problem Tanımı	Çoğu durumda, grafik ekran ve tablo görünümü gibi birden çok durum bilgisi gösterimi sağlamanız gerekir. Bilgi belirtildiğinde bunların hepsi bilinmeyebilir. Tüm alternatif sunumlar etkileşimi desteklemeli ve durum değiştiğinde tüm ekranlar güncellenmelidir. Bu model, durum bilgisi için birden fazla gösterim formatının gerekli olduğu ve durum bilgisini muhafaza eden nesnenin kullanılan özel gösterim formatları hakkında bilgi sahibi olmasının gerekli olmadığı tüm durumlarda kullanılabilir.

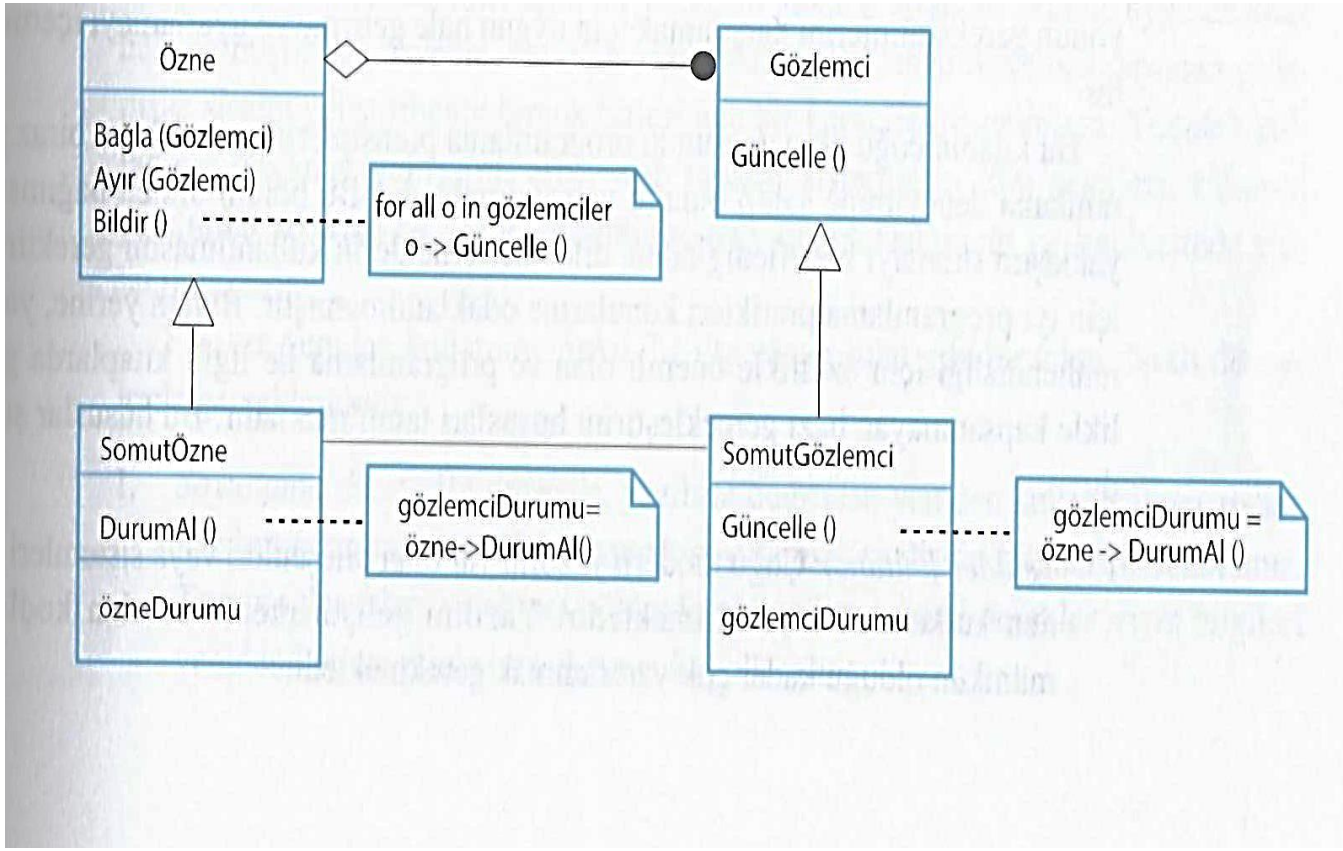
Gözlemci Deseni

Desen Adı	Gözlemci
Çözüm Tanımı	Bu, konu ve Gözlemci olmak üzere iki soyut nesneyi ve ilgili soyut nesnelerin niteliklerini devralan iki somut nesneyi, SomutÖzne ve SomutNesne içerir. Soyut nesneler, her durumda geçerli olan genel işlemleri içerir. Görüntülenecek durum, İşlemleri Özneden devralan (her bir gözlemci bir ekrana karşılık gelir) ve durum değiştiğinde bir bildirim yayınlamasına olanak tanıyan işlemleri devralan SomutÖzne'de tutulur. SomutGözlemci, SomutÖzne'nin durumunun bir kopyasını tutar ve bu kopyaların adım adım tutulmasına izin veren Gözlemci'nin Güncelle() ara yüzünü uygular. SomutGözlemci, durumu otomatik olarak görüntüler ve durum her güncellendiğinde değişiklikleri yansıtır.
Sonuçlar	Özne sadece soyut Gözlemci'yi bilir ve somut sınıfın ayrıntılarını bilmez. Bu nedenle, bu nesneler arasında minimum bağlantı vardır. Bu bilgi eksikliği nedeniyle, görüntü performansını artıran optimizasyonlar pratik değildir. Konudaki değişiklikler, gözlemciler için bir dizi bağlantılı güncellemenin üretilmesine neden olabilir, bunlardan bazıları gerekli olmayabilir.

Gözlemci Modelini Kullanan Birden Fazla Gösterim



Gözlemci Desenine İlişkin UML Modeli



Tasarım Problemleri

- Tasarımınızda desenleri kullanmak için, karşılaştığınız herhangi bir tasarım probleminin uygulanabilecek ilişkili bir desene sahip olabileceğini bilmeniz gerekir.
  - Birkaç nesneye başka bir nesnenin durumunun değiştiğini söyleyin (Gözlemci deseni).
  - Ara yüzleri, genellikle aşamalı olarak geliştirilen bir dizi ilgili nesneyle düzenleyin (Fasat deseni).
  - Koleksiyonun nasıl uygulandığına bakılmaksızın, bir koleksiyondaki öğelere erişmenin standart bir yolunu sağlayın (Yineleyici deseni).
  - Mevcut bir sınıfın işlevselliğini çalışma zamanında genişletme olasılığına izin verin (Dekoratör deseni).

Gerçekleştirim Konuları

Gerçekleştirim Konuları

- Burada odaklanmak, programlamaya değil, açıkça önemli olmasına rağmen, genellikle programlama metinlerinde ele alınmayan diğer uygulama sorunlarına odaklanmaktadır:
  - Yeniden Kullanım: Modern yazılımların çoğu, mevcut bileşenler veya sistemler yeniden kullanılarak oluşturulmuştur. Yazılım geliştirirken, mevcut koddan olabildiğince fazla yararlanmalısınız.
  - Yapılandırma Yönetimi: Geliştirme süreci sırasında, bir konfigürasyon yönetim sistemindeki her bir yazılım bileşeninin birçok farklı sürümünü takip etmeniz gerekir.
  - Konakçı/Evsahibi-Hedef (Host-target) Geliştirme: Üretim yazılımı genellikle yazılım geliştirme ortamıyla aynı bilgisayarda çalıştırılmaz. Bunun yerine, bir bilgisayarda (ana sistem) geliştirir ve ayrı bir bilgisayarda (hedef sistem) yürütürsünüz.

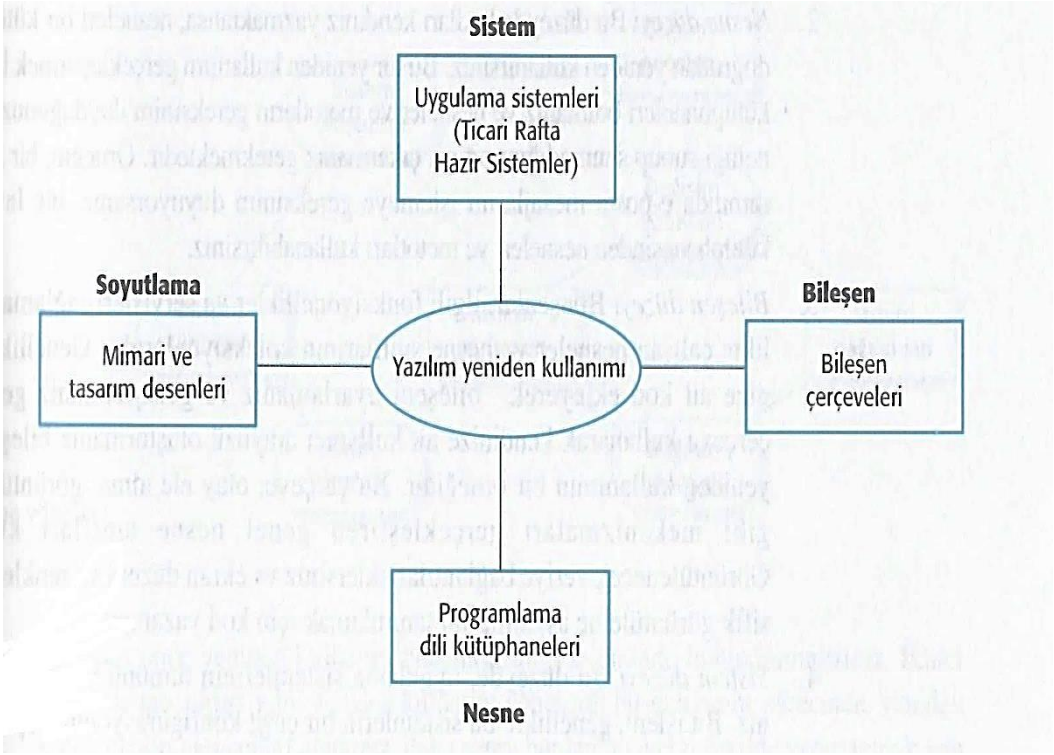
Yeniden Kullanım

- 1960'lardan 1990'lara kadar, çoğu yeni yazılım, tüm kodların üst düzey bir programlama dilinde yazılmasıyla sıfırdan geliştirildi.
  - Tek önemli yeniden kullanım veya yazılım, programlama dili kitaplıklarında işlevlerin ve nesnelerin yeniden kullanılmasıydı.
- Maliyetler ve program baskısı, bu yaklaşımın özellikle ticari ve İnternet tabanlı sistemler için giderek daha dayanılmaz hale geldiği anlamına geliyor.
- Mevcut yazılımın yeniden kullanımına dayalı bir geliştirme yaklaşımı ortaya çıktı ve şu anda genellikle ticari ve bilimsel yazılımlar için kullanılıyor.

Yeniden Kullanım Düzeyleri

- **Soyutlama düzeyi**
  - Bu düzeyde, yazılımı doğrudan yeniden kullanmazsınız, ancak yazılımınızın tasarımında başarılı soyutlama bilgilerini kullanırsınız.
- **Nesne düzeyi**
  - Bu düzeyde, kodu kendiniz yazmak yerine doğrudan bir kitaplıktaki nesneleri yeniden kullanırsınız.
- **Bileşen düzeyi**
  - Bileşenler, uygulama sistemlerinde yeniden kullandığınız nesne ve nesne sınıfları koleksiyonlarıdır.
- **Sistem düzeyi**
  - Bu düzeyde, tüm uygulama sistemlerini yeniden kullanırsınız.

Yazılım Yeniden Kullanımı



Yeniden Kullanım Maliyeti

- Yazılım aramak için harcanan zamanın maliyeti, yazılımı yeniden kullanmak ve ihtiyaçlarınızı karşılayıp karşılamadığını değerlendirmek için harcanan zaman.
- Uygulanabildiği durumlarda, yeniden kullanılabilir yazılımı satın almanın maliyetleri.
- Kullanıma hazır büyük sistemler için bu maliyetler çok yüksek olabilir. Yeniden kullanılabilir yazılım bileşenlerini veya sistemlerini, geliştirmekte olduğunuz sistemin gereksinimlerini yansıtacak şekilde uyarlama ve yapılandırma maliyetleri vardır.
- Yeniden kullanılabilir yazılım öğelerini birbirleriyle (farklı kaynaklardan yazılım kullanıyorsanız) ve geliştirdiğiniz yeni kodla entegre etmenin maliyetleri de ele alınmalıdır.

Konfigürasyon (Yapılandırma) Yönetimi

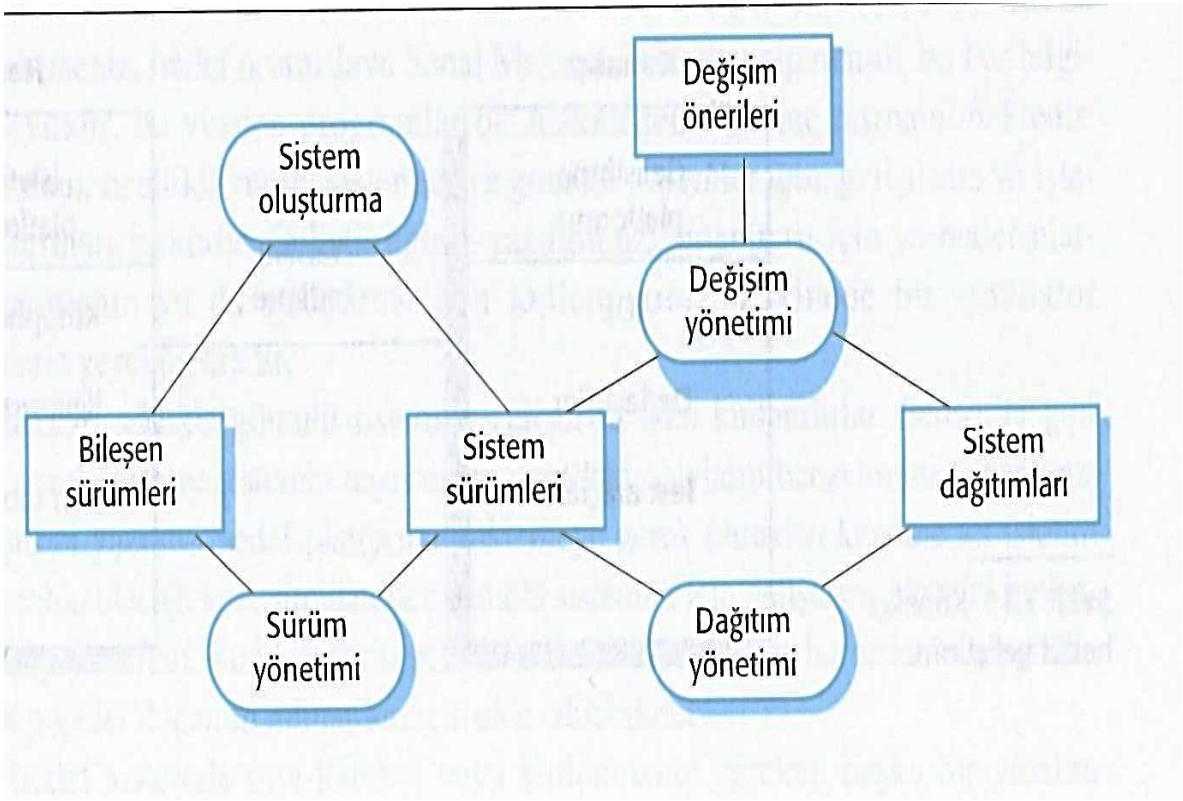
- Konfigürasyon yönetimi, değişen bir yazılım sistemini yönetmenin genel sürecine verilen addır.

- Konfigürasyon yönetiminin amacı, tüm geliştiricilerin proje koduna ve belgelere kontrollü bir şekilde erişebilmesi, hangi değişikliklerin yapıldığını bulabilmesi ve bir sistem oluşturmak için bileşenleri derleyip bağlayabilmesi için sistem entegrasyon sürecini desteklemektir.

Konfigürasyon Yönetimi faaliyetleri

- Yazılım bileşenlerinin farklı sürümlerini takip etmek için destek sürüm yönetimi tarafından sağlanır.
- Sürüm yönetimi sistemleri, birkaç programcı tarafından geliştirmeyi koordine eden tesisleri içerir. Geliştiricilerin bir sistemin her sürümünü oluşturmak için hangi bileşen sürümlerinin kullanıldığını tanımlamasına yardımcı olmak için desteğin sağlandığı sistem entegrasyonu. Bu açıklama daha sonra gerekli bileşenleri derleyerek ve bağlayarak otomatik olarak bir sistem oluşturmak için kullanılır.
- Kullanıcıların hataları ve diğer sorunları bildirmesine ve tüm geliştiricilerin bu sorunlar üzerinde kimin çalıştığını ve ne zaman düzeltildiğini görmesine olanak sağlamak için desteğin sağlandığı sorun izleme işlemleri yer alır.

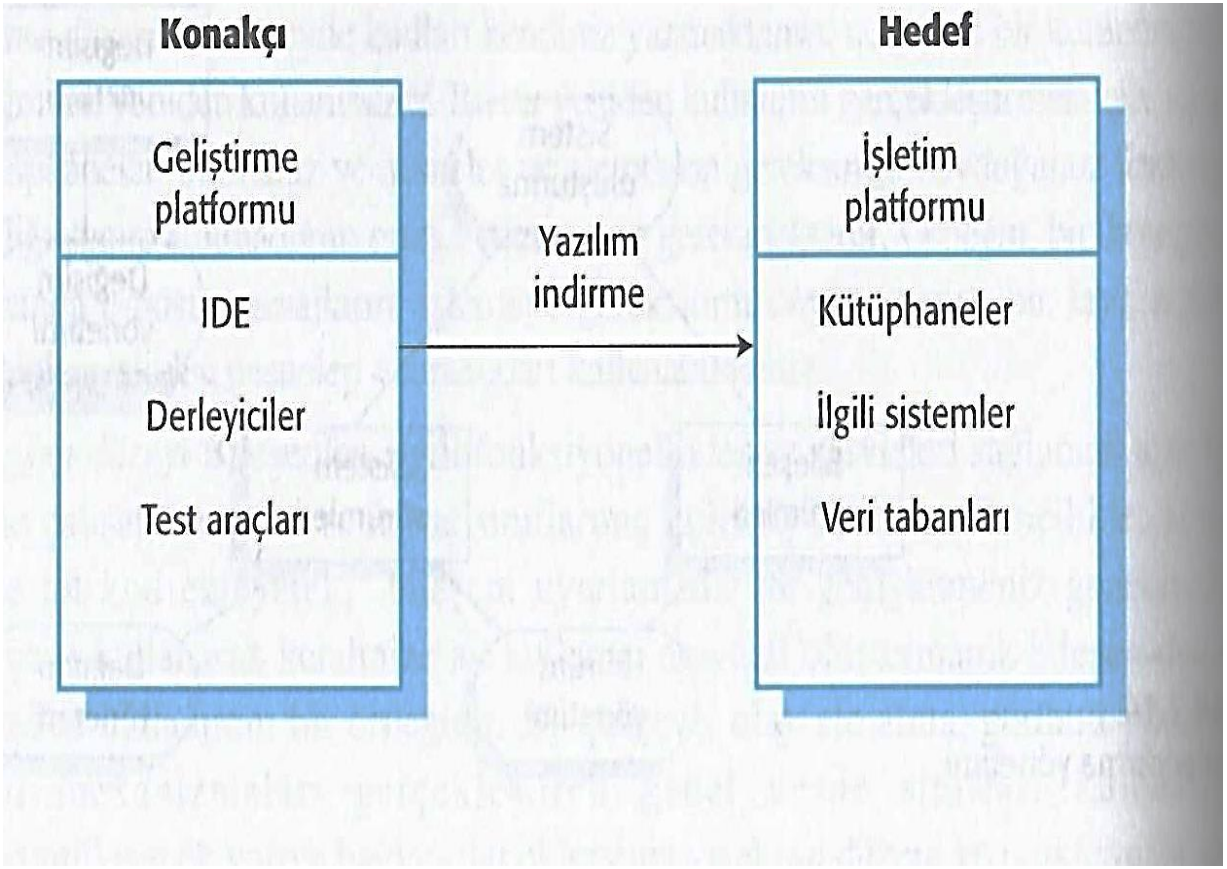
Konfigürasyon Yönetimi



Konakçı-Hedef Geliştirme

- Çoğu yazılım bir bilgisayarda (ana bilgisayar) geliştirilir, ancak ayrı bir makinede (hedef) çalışır.
- Daha genel olarak, bir geliştirme platformu ve bir yürütme platformundan bahsedebiliriz.
  - Platform, donanımdan daha fazlasıdır.
  - Kurulu işletim sistemini ve bir veri tabanı yönetim sistemi veya geliştirme platformları için etkileşimli bir geliştirme ortamı gibi diğer destekleyici yazılımları içerir.
- Geliştirme platformunun genellikle yürütme platformundan farklı bir yüklü yazılımı vardır; bu platformların farklı mimarileri olabilir.

Konakçı-Hedef Geliştirme



Geliştirme Platformu Araçları

- Kod oluşturmanıza, düzenlemenize ve derlemenize olanak tanıyan entegre bir derleyici ve sözdizimine yönelik düzenleme sistemi.
- Bir dil hata ayıklama sistemi.
- UML modellerini düzenlemek için araçlar gibi grafik düzenleme araçları.
- Bir programın yeni sürümünde otomatik olarak bir dizi test çalıştırabilen Junit gibi test araçları.
- Farklı geliştirme projeleri için kodu düzenlemenize yardımcı olan proje destek araçları.

Bütünleşik Geliştirme Ortamı - EGO (Integrated Development Environments - IDEs)

- Yazılım geliştirme araçları genellikle entegre bir geliştirme ortamı (IDE) oluşturmak için gruplanır.

- Bir IDE, bazı ortak çerçeve ve kullanıcı arabirimi dahilinde yazılım geliştirmenin farklı yönlerini destekleyen bir dizi yazılım aracıdır.

- IDE'ler, Java gibi belirli bir programlama dilinde geliştirmeyi desteklemek için oluşturulur. Dil IDE'si özel olarak geliştirilebilir veya belirli dil destek araçlarıyla birlikte genel amaçlı bir IDE'nin somutlaşmış hali olabilir.

Bileşen / Sistem Dağıtım Faktörleri

- Bir bileşen belirli bir donanım mimarisi için tasarlandıysa veya başka bir yazılım sistemine dayanıyorsa, gerekli donanım ve yazılım desteğini sağlayan bir platformda kullanılması gerektiği açıktır.

- Yüksek kullanılabilirliği olan sistemler, bileşenlerin birden fazla platformda konuşlandırılmasını gerektirebilir. Bu, platform arızası durumunda bileşenin alternatif bir uygulamasının mevcut olduğu anlamına gelir.

- Bileşenler arasında yüksek düzeyde iletişim trafiği varsa, bunları aynı platformda veya fiziksel olarak birbirine yakın platformlarda konuşlandırmak genellikle mantıklıdır. Bu, bir mesajın bir bileşen tarafından gönderildiği ve bir başkası tarafından alındığı zaman arasındaki gecikmeyi azaltır.

Açık Kaynak Geliştirme

Açık Kaynak Geliştirme

- Açık kaynak geliştirme, bir yazılım sisteminin kaynak kodunun yayınlandığı ve gönüllülerin geliştirme sürecine katılmaya davet edildiği bir yazılım geliştirme yaklaşımıdır.
- Kökleri, kaynak kodunun tescilli olmaması gerektiğini savunan Özgür Yazılım Vakfı'na (Free Software Foundation - www.fsf.org) dayanmaktadır. Kullanıcıların istedikleri gibi inceleyip değiştirebilmeleri için kodun her zaman erişilebilir olması gerektiğini savunmaktadır.
- Açık kaynaklı yazılım, çok daha büyük bir gönüllü geliştirici popülasyonunu işe almak için İnternet'i kullanarak bu fikri genişletmiştir. Birçoğu aynı zamanda kodun kullanıcılarıdır.

Açık Kaynak Sistemler

- En çok bilinen açık kaynaklı ürün, elbette, yaygın olarak bir sunucu sistemi ve giderek artan bir şekilde masaüstü ortamı olarak kullanılan Linux işletim sistemidir.

- Diğer önemli açık kaynaklı ürünler Java, Apache web sunucusu ve mySQL veritabanı yönetim sistemidir.

Açık Kaynak Konuları

- Geliştirilmekte olan ürün açık kaynaklı bileşenlerden yararlanmalı mı?
- Yazılımın geliştirilmesi için açık kaynaklı bir yaklaşım kullanılmalı mı?

Açık Kaynak İşletmesi

- Giderek daha fazla ürün şirketi, geliştirme için açık kaynaklı bir yaklaşım kullanıyor.
- İş modelleri, bir yazılım ürünü satmaya değil, o ürün için destek satmaya dayanır.
- Açık kaynak topluluğunun dahil edilmesinin, yazılımın daha ucuza, daha hızlı geliştirilmesine olanak sağlayacağına ve yazılım için bir kullanıcı topluluğu oluşturacağına inanıyorlar.

Açık Kaynak Lisanslama

- Açık kaynak geliştirmenin temel bir ilkesi, kaynak kodunun ücretsiz olarak erişilebilir olmasıdır, bu, herhangi birinin bu kodla dilediğini yapabileceği anlamına gelmez.
  - Yasal olarak, kodun geliştiricisi (bir şirket veya bir birey) hala koda sahiptir. Açık kaynaklı bir yazılım lisansına yasal olarak bağlayıcı koşullar ekleyerek nasıl kullanılacağına ilişkin kısıtlamalar getirebilirler.
  - Bazı açık kaynak geliştiricileri, yeni bir sistem geliştirmek için açık kaynaklı bir bileşen kullanılıyorsa, o sistemin de açık kaynak olması gerektiğine inanıyor.
  - Diğerleri kodlarının bu kısıtlama olmadan kullanılmasına izin vermeye isteklidir. Geliştirilen sistemler tescilli olabilir ve kapalı kaynak sistemleri olarak satılabilir.

Lisans Modelleri

- GNU Genel Kamu Lisansı (GPL).
  - Bu lisans, "karşılıklı" denilen bir lisanstır, daha açık bir ifadeyle GPL lisansı altında lisanslanan açık kaynaklı bir yazılım kullanıyorsanız, o zaman bu yazılımı açık kaynaklı yapmanız gerekir.
- GNU Kısıtlı Genel Kamu Lisansı (LGPL).
  - Bu lisans, bileşenlerin kaynağını yayınlamak zorunda kalmadan açık kaynak koduna bağlanan bileşenleri yazabileceğiniz GPL lisansının bir çeşitidir.
- Berkley Standart Dağıtım (BSD) Lisansı.
  - Bu lisans karşılıklı olmayan bir lisanstır, daha açık bir ifadeyle açık kaynak kodunda yapılan herhangi bir değişikliği veya değişikliği yeniden yayınlamak zorunda değilsiniz. Kodu, satılan tescilli sistemlere dahil edebilirsiniz.

Lisans Yönetimi

- İndirilen ve kullanılan açık kaynak bileşenlerle ilgili bilgileri korumak için bir sistem kurun.
- Farklı lisans türlerinin farkında olun ve kullanılmadan önce bir bileşenin nasıl lisanslandığını anlayın.
- Bileşenler için evrim yollarının farkında olun. İnsanları açık kaynak konusunda eğitin.
- Yerinde denetim sistemleri bulundurun.
- Açık kaynak topluluğuna katılın.

Önemli Noktalar

- Yazılım tasarımı ve uygulaması, birbiriyle ilişkili faaliyetlerdir. Tasarımdaki ayrıntı seviyesi, sistemin türüne ve plan odaklı veya çevik bir yaklaşım kullanıp kullanmadığınıza bağlıdır.
- Nesne yönelimli tasarım süreci, sistem mimarisini tasarlama, sistemdeki nesneleri tanımlama, farklı nesne modellerini kullanarak tasarımı tanımlama ve bileşen ara yüzlerini belgeleme faaliyetleri içerir.
- Nesneye yönelik tasarım sürecinde bir dizi farklı model üretilebilir. Bunlar, statik modelleri (sınıf modelleri, genelleme modelleri, ilişkilendirme modelleri) ve dinamik modelleri (sıra modelleri, durum makinesi modelleri) içerir.
- Bileşen ara yüzleri, diğer nesnelerin kullanabilmesi için tam olarak tanımlanmalıdır. Ara yüzleri tanımlamak için bir UML ara yüz **stereotipi** kullanılabilir.

Önemli Noktalar

- Yazılım geliştirirken, mevcut yazılımı bileşenler, hizmetler veya komple sistemler olarak yeniden kullanma olasılığını her zaman göz önünde bulundurmalısınız.
- Yapılandırma yönetimi, gelişen bir yazılım sistemindeki değişiklikleri yönetme sürecidir. Yazılım geliştirmek için bir ekip işbirliği yaptığında bu çok önemlidir.
- Çoğu yazılım geliştirme, konakçı-hedef geliştirmedir. Yürütmek üzere bir hedef makineye aktarılan yazılımı geliştirmek için bir ana makinede bir IDE kullanırsınız.
- Açık kaynak geliştirme, bir sistemin kaynak kodunu halka açık hale getirmeyi içerir. Bu, birçok kişinin yazılımda değişiklikler ve iyileştirmeler önerebileceği anlamına gelir.

Sonraki Ders

- Yazılım test mühendisliği
- Geliştirme testi
- Test güdümlü geliştirme
- Dağıtım testi
- Kullanıcı testi

BİL 407 Yazılım Mühendisliği
Hafta 10

Doç. Dr. Sedat Akleylek

sedat.akleylek@bil.omu.edu.tr