IT522 – Yazılım Mühendisliği 2021

PhD Furkan Gözükara, Toros University

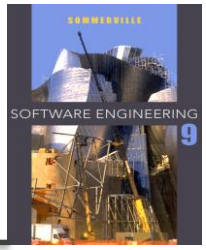
https://github.com/FurkanGozukara/Yazilim-Muhendisligi-IT522-2021

Ders 7

Tasarım ve Uygulama

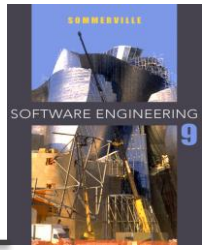
Kaynak : https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html

Ders 7'de İşlenen Konular



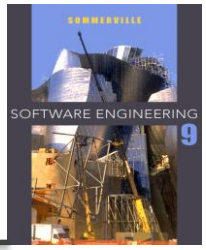
- UML kullanarak nesneye yönelik tasarım
- Tasarım desenleri
- Uygulama sorunları
- Açık kaynak geliştirme

Tasarım ve Uygulama



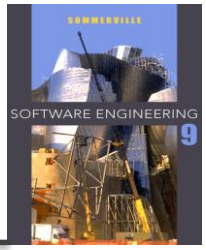
- Yazılım tasarımı ve uygulaması, yazılım mühendisliği sürecinde yürütülebilir bir yazılım sisteminin geliştirildiği aşamadır.
- Yazılım tasarım ve uygulama faaliyetleri her zaman birbiriyle ilişkilidir.
  - Yazılım tasarımı, bir müşterinin gereksinimlerine göre yazılım bileşenlerini ve bunların ilişkilerini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

Yap veya Satın Al



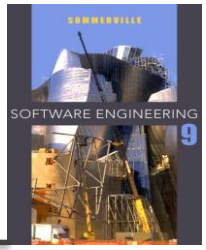
- Çok çeşitli alanlarda, kullanıcıların gereksinimlerine göre uyarlanabilen ve özelleştirilebilen hazır sistemler (commercial of the shelf - COTS) satın almak artık mümkün.
  - Örneğin, bir tıbbi kayıt sistemi uygulamak istiyorsanız, hastanelerde halihazırda kullanılan bir paketi satın alabilirsiniz. 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.

Nesne Odaklı Bir Tasarım Süreci



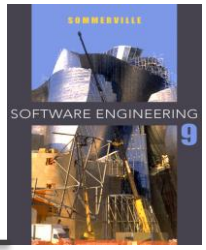
- Yapılandırılmış nesneye yönelik tasarım süreçleri, bir dizi farklı sistem modelinin geliştirilmesini içerir.
- Bu modellerin geliştirilmesi ve bakımı için çok çaba gerekir ve küçük sistemler için bu, uygun maliyetli olmayabilir.
- Ancak, farklı gruplar tarafından geliştirilen büyük sistemler için tasarım modelleri önemli bir iletişim mekanizmasıdır.

Süreç Aşamaları



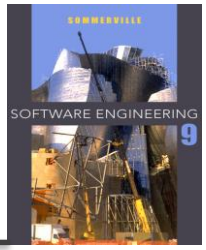
- Süreci kullanan kuruluşa bağlı olan çeşitli farklı nesneye yönelik tasarım süreçleri vardır.
- Bu süreçlerdeki ortak faaliyetler şunları içerir:
  - Sistemin bağlamını ve kullanım biçimlerini tanımlayın;
  - Sistem mimarisini tasarlayın;
  - Ana sistem nesnelerini tanımlayın;
  - Tasarım modelleri geliştirin;
  - Nesne ara yüzlerini belirtin.
- Burada, doğa hava durumu istasyonu için bir tasarım kullanılarak gösterilen süreç.

Sistem Bağlamı ve Etkileşimleri



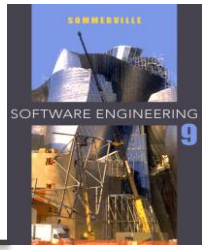
- Tasarlanmakta olan yazılım ile dış ortamı arasındaki ilişkilerin anlaşılması, gerekli sistem işlevselliğinin nasıl sağlanacağına ve sistemin çevresiyle iletişim kuracak şekilde nasıl yapılandırılacağına karar vermek için çok önemlidir.
- Bağlamın anlaşılması, sistemin sınırlarını da belirlemenizi sağlar. Sistem sınırlarının belirlenmesi, tasarlanan sistemde hangi özelliklerin uygulanacağına ve diğer ilişkili sistemlerde hangi özelliklerin olduğuna karar vermenize yardımcı olur.

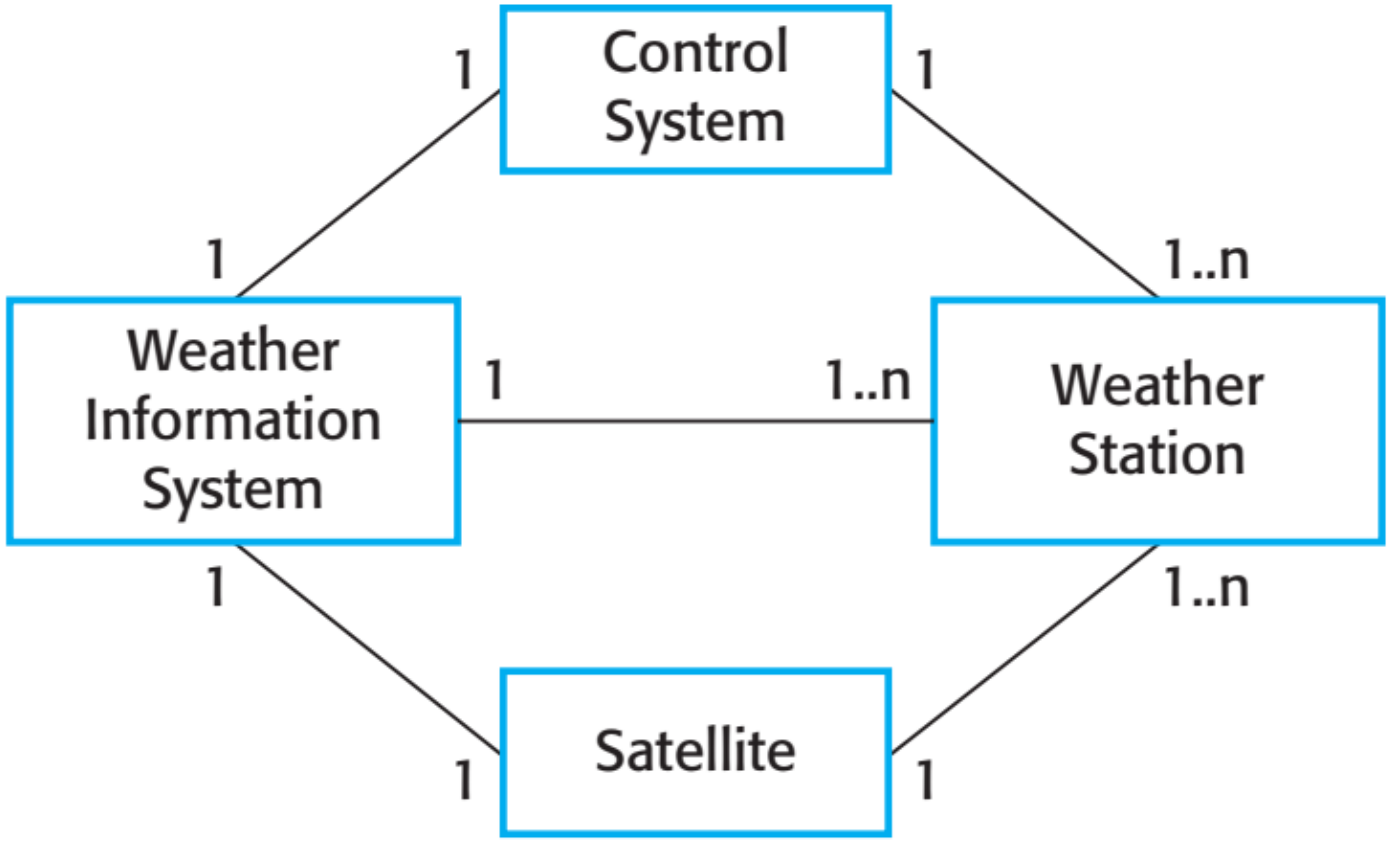
Bağlam ve Etkileşim Modelleri



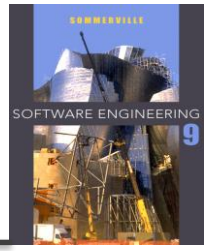
- Sistem bağlam modeli, geliştirilmekte olan sistemin ortamındaki diğer sistemleri gösteren yapısal bir modeldir.
- Etkileşim modeli, sistemin kullanıldıkça çevresi ile nasıl etkileşime girdiğini gösteren dinamik bir modeldir.

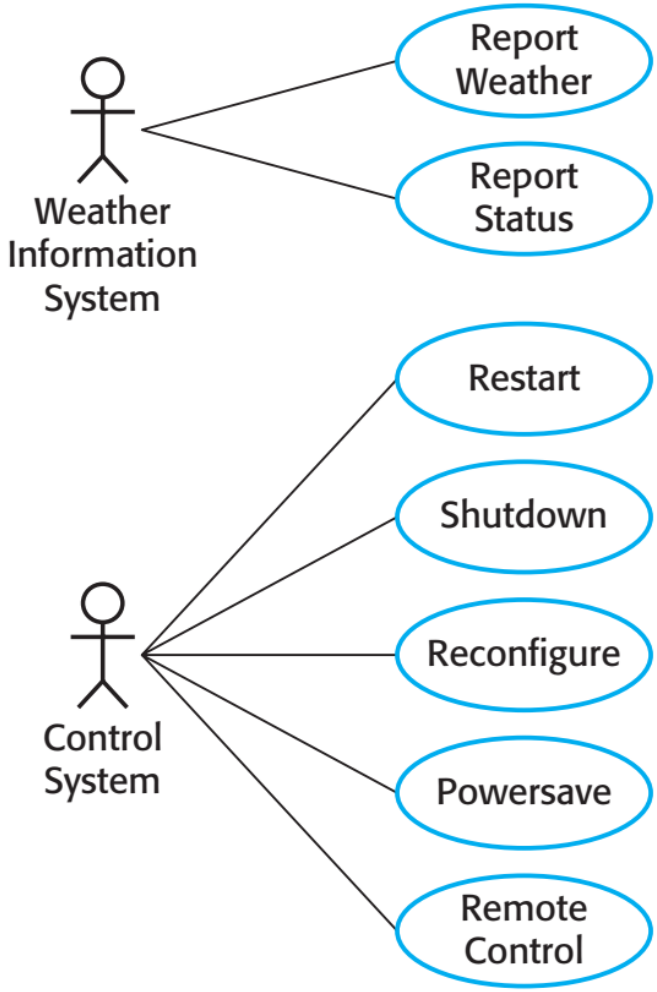
Meteoroloji İstasyonu İçin Sistem Bağlamsal İçeriği



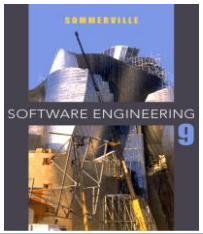


Hava Durumu İstasyonu Kullanım Durumları



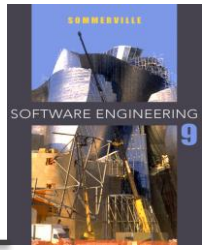


Kullanım Durumları Açıklamaları - Hava Durumunu Bildir



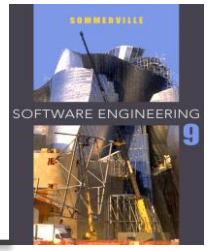
Sistem	Meteoroloji istasyonu
Kullanım alanı	Hava durumunu bildir
Aktörler	Hava durumu bilgi sistemi, Hava durumu istasyonu
Açıklama	Hava durumu istasyonu, toplama periyodunda cihazlardan toplanan hava durumu verilerinin bir özetini hava durumu bilgi sistemine gönderir. Gönderilen veriler maksimum, minimum 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 yönü.
Uyaran	Hava durumu bilgi sistemi, hava durumu istasyonu ile bir uydu iletişim bağlantısı kurar ve verilerin iletimini talep eder.
Tepki	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.

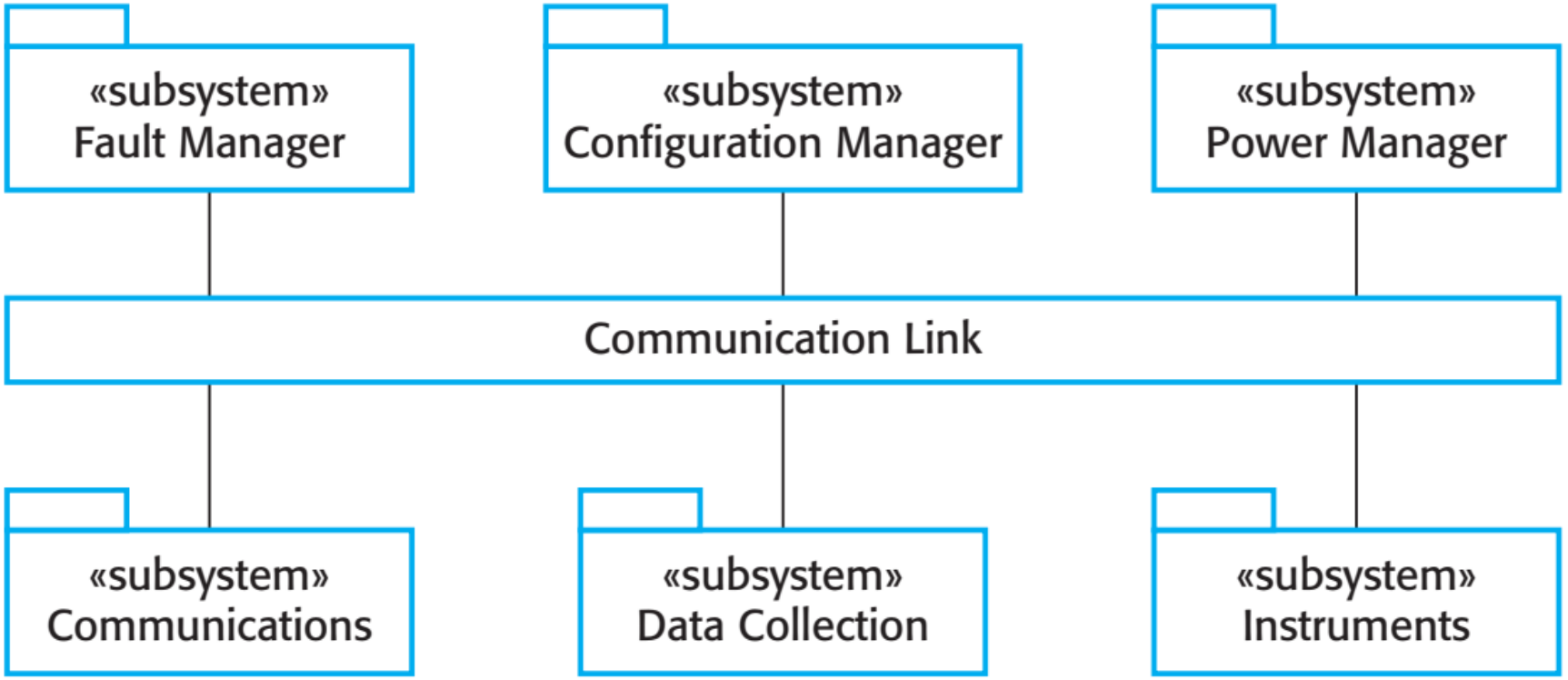
Mimari Tasarım



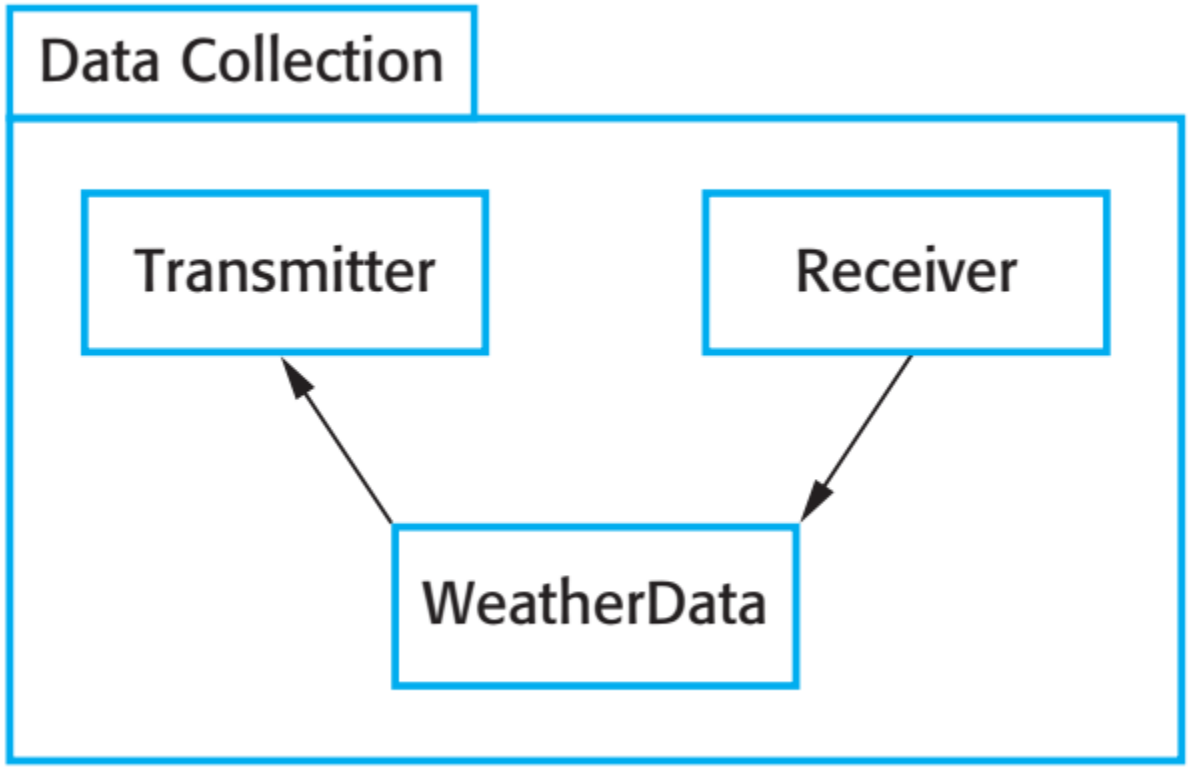
- Sistem ve çevresi arasındaki etkileşimler anlaşıldıktan sonra, bu bilgiyi sistem mimarisini tasarlamak için kullanırsınız.
- Sistemi oluşturan ana bileşenleri ve bunların etkileşimlerini belirlersiniz ve ardından bileşenleri katmanlı veya istemci-sunucu modeli gibi mimari bir model kullanarak düzenleyebilirsiniz.
- Hava durumu istasyonu, ortak bir altyapı üzerinde mesajlar yayınlayarak iletişim kuran bağımsız alt sistemlerden oluşur.

Meteoroloji İstasyonunun Üst Düzey Mimarisi

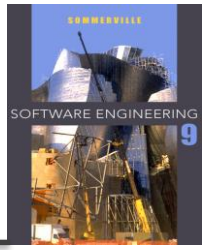




Veri Toplama Sistemi Mimarisi

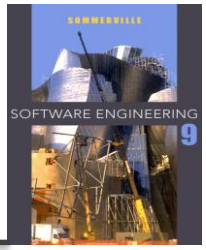


Nesne Sınıfı Tanımlama



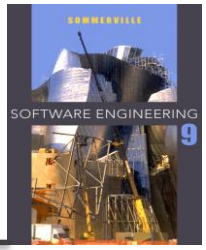
- Nesne sınıflarını belirlemek, genellikle nesne yönelimli tasarımın zor bir parçasıdır.
- Nesne tanımlaması için 'sihirli formül' yoktur. Sistem tasarımcılarının becerisine, deneyimine ve alan bilgisine dayanır.
- Nesne tanımlama, yinelemeli bir süreçtir. İlk seferinde doğru yapmak pek olası değil.

Tanımlamaya Yönelik Yaklaşımlar



- Sistemin doğal dil tanımına dayalı bir gramer yaklaşımı kullanın (Hood OOD yönteminde kullanılır).
- 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, öznitelikler ve yöntemler tanımlanır.

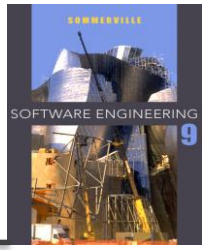
Hava Durumu İstasyonu Açıklaması



Bir hava durumu istasyonu, verileri toplayan, bazı veri işlemlerini gerçekleştiren ve bu verileri daha fazla işlem için ileten yazılım kontrollü araçlar paketidir. Aletler, hava ve yer termometreleri, bir anemometre, bir rüzgar gülü, bir barometre ve bir yağmur ölçeri içerir. Veriler periyodik olarak toplanır.

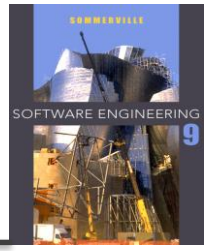
Hava durumu verilerini iletmek için bir komut verildiğinde, hava durumu istasyonu toplanan verileri işler ve özetler. Özetlenen veriler, bir talep alındığında eşleştirme bilgisayarına iletilir.

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 içerir.

Hava Durumu İstasyonu Nesne Sınıfları



WeatherStation
identifier
reportWeather ( ) reportStatus ( ) powerSave (instruments) remoteControl (commands) reconfigure (commands) restart (instruments) shutdown (instruments)

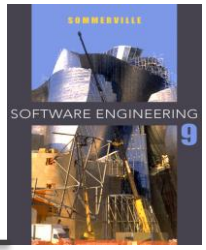
WeatherData
airTemperatures groundTemperatures windSpeeds windDirections pressures rainfall
collect ( ) summarize ( )

Ground Thermometer
gt_Ident temperature
get ( ) test ( )

Anemometer
an_Ident windSpeed windDirection
get ( ) test ( )

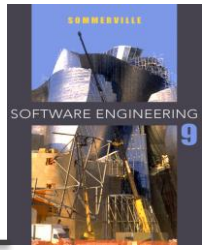
Barometer
bar_Ident pressure height
get ( ) test ( )

Tasarım Modelleri



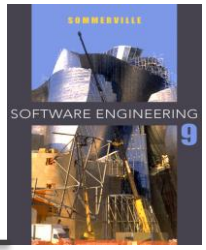
- Tasarım modelleri, nesneleri ve nesne sınıflarını ve bu varlıklar arasındaki ilişkileri gösterir.
- Statik modeller, sistemin statik yapısını nesne sınıfları ve ilişkiler açısından tanımlar.
- Dinamik modeller, nesneler arasındaki dinamik etkileşimleri tanımlar.

Tasarım Modellerine Örnekler



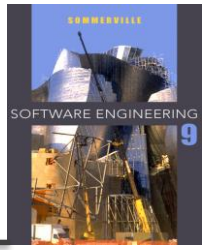
- Nesnelerin mantıksal gruplamalarını tutarlı alt sistemler halinde gösteren alt sistem modelleri.
- Nesne etkileşimlerinin sırasını gösteren dizi modelleri.
- Olaylara yanıt olarak nesnelerin durumlarını nasıl değiştirdiğini gösteren durum makinesi modelleri.
- Diğer modeller, kullanım durumu modellerini,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

Alt Sistem Modelleri



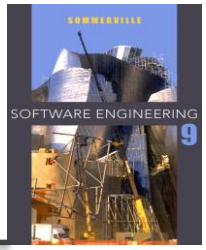
- Tasarımın mantıksal olarak ilişkili nesne grupları halinde nasıl düzenlendiğini gösterir.
- UML'de bunlar, bir kapsülleme yapısı olan paketler kullanılarak gösterilir. Bu mantıklı bir modeldir. Sistemdeki nesnelerin gerçek organizasyonu farklı olabilir.

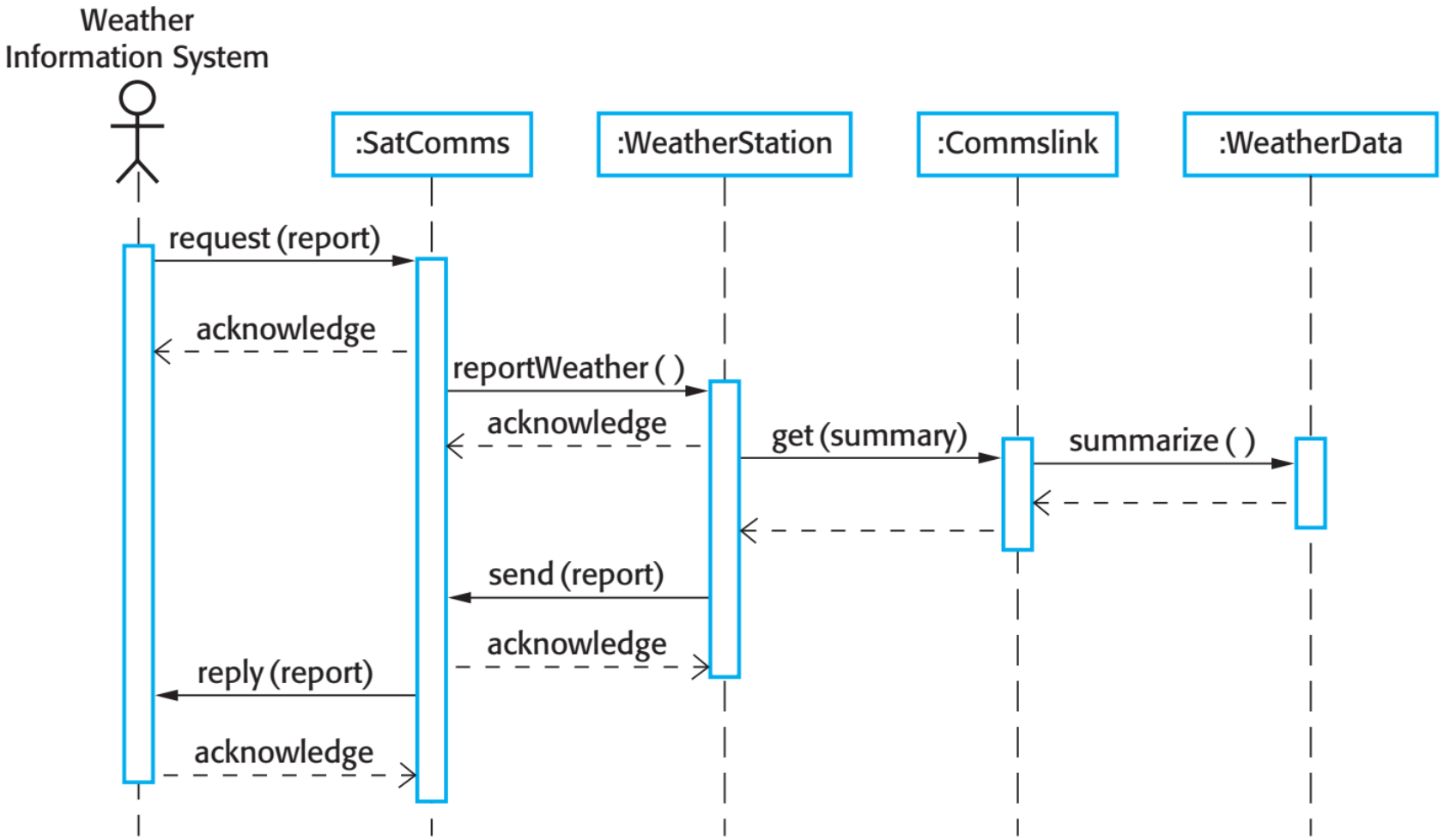
Sıra Modelleri



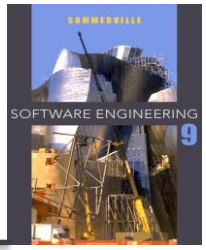
- Sıra modelleri, meydana gelen nesne etkileşimlerinin sırasını gösterir
  - Nesneler, üstte yatay olarak düzenlenmiştir;
  - Zaman dikey olarak temsil edilir, böylece modeller yukarıdan aşağıya okunur;
  - Etkileşimler etiketli oklarla temsil edilir. Farklı ok stilleri, farklı etkileşim türlerini temsil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

Veri Toplamayı Açıklayan Sıra Diyagramı



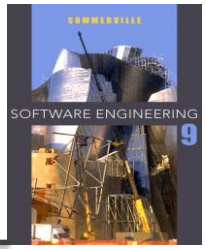


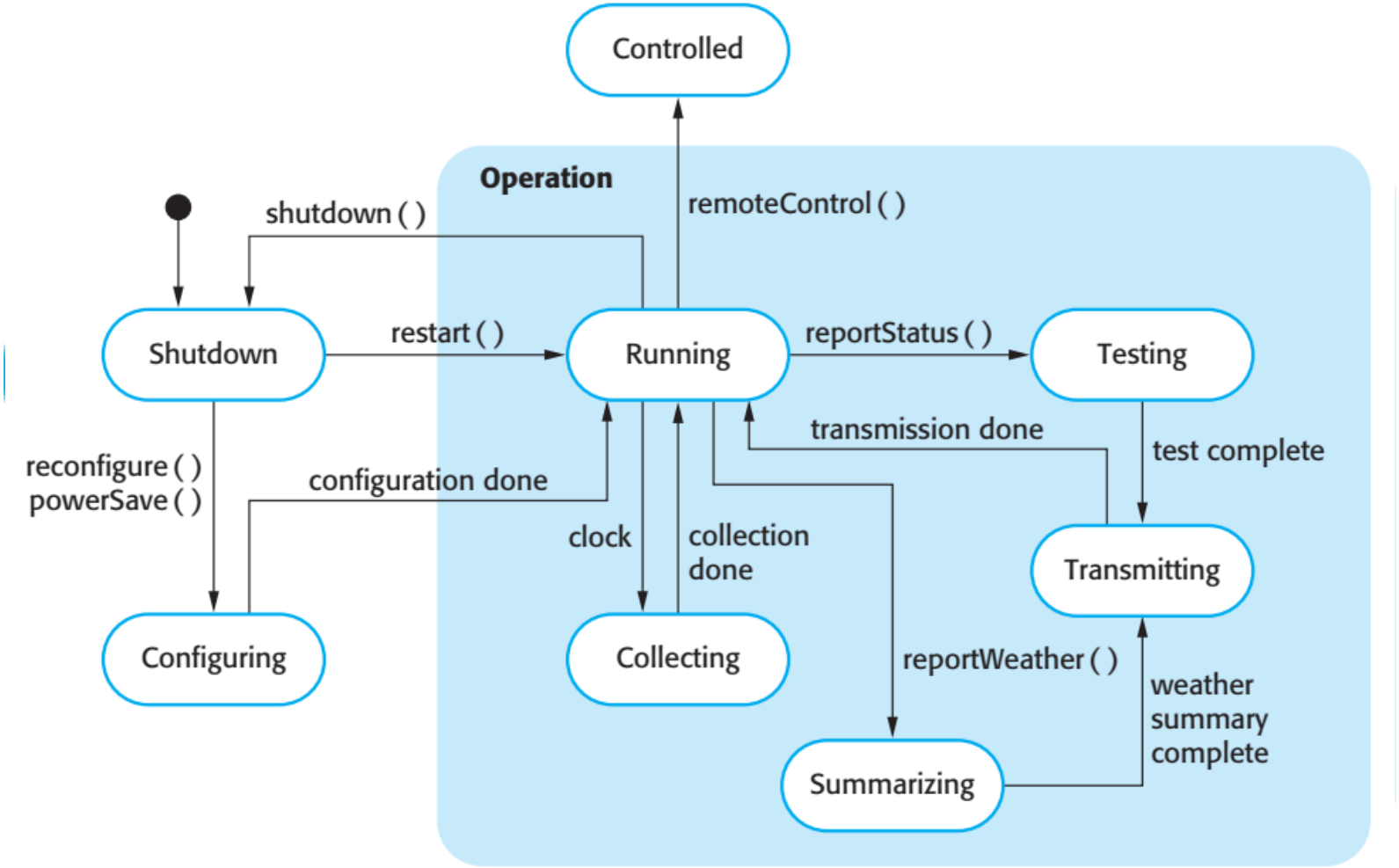
Durum Diyagramları



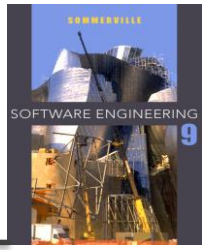
- Durum diyagramları, nesnelerin farklı hizmet isteklerine nasıl yanıt verdiğini ve bu isteklerle tetiklenen durum geçişlerini göstermek için kullanılır.
- Durum diyagramları, bir sistemin veya bir nesnenin çalışma zamanı davranışının yararlı üst düzey modelleridir.
- Sistemdeki tüm nesneler için genellikle bir durum diyagramına ihtiyacınız yoktur. Bir sistemdeki nesnelerin çoğu nispeten basittir ve bir durum modeli tasarıma gereksiz ayrıntılar ekler.

Hava İstasyonu Durum Diyagramı



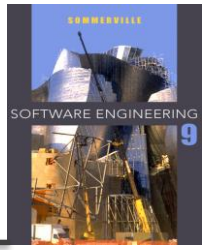


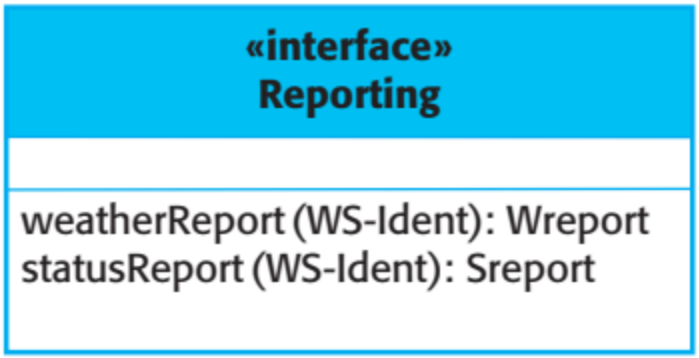
Arayüz Özellikleri

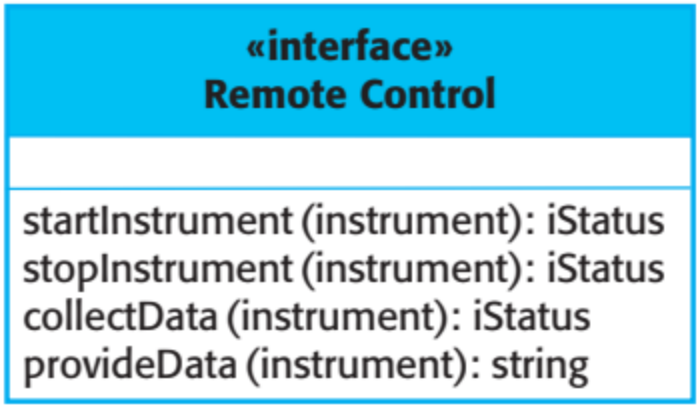


- Nesne arayüzleri, nesnelerin ve diğer bileşenlerin paralel olarak tasarlanabilmesi için belirlenmelidir.
- Tasarımcılar arayüz temsilini tasarlamaktan kaçınmalı, ancak bunu nesnenin kendisinde saklamalıdır.
- Nesneler, sağlanan yöntemlere ilişkin bakış açıları olan birkaç arayüze sahip olabilir.
-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

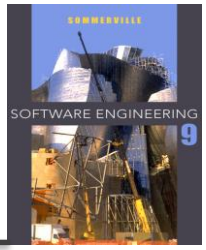
Hava Durumu İstasyonu Arayüzleri



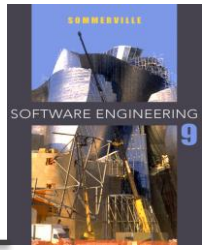




Bölüm 1'in Anahtar Noktaları



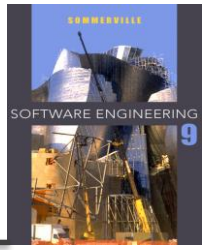
- Yazılım tasarımı ve uygulaması, birbiriyle ilişkili faaliyetlerdir. Tasarımdaki ayrıntı seviyesi, sistemin türüne ve plan odaklı veya çevik bir yaklaşım kullanıp kullanmadığınıza bağlıdır.
- Nesne yönelimli tasarım süreci, sistem mimarisini tasarlama, sistemdeki nesneleri tanımlama, farklı nesne modellerini kullanarak tasarımı tanımlama ve bileşen arayüzlerini belgeleme faaliyetlerini içerir.
- Nesneye yönelik bir 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 arayüzleri, diğer nesnelerin kullanabilmesi için tam olarak tanımlanmalıdır. Arayüzleri tanımlamak için bir UML arayüz klişesi kullanılabilir.



Ders 7 - Tasarım ve Uygulama

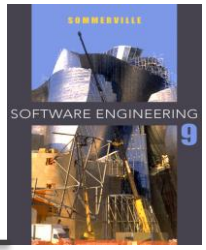
Bölüm 2

Tasarım Desenleri



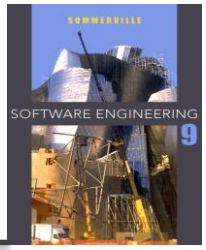
- Bir tasarım modeli, bir problem ve çözümü hakkındaki soyut bilgileri yeniden kullanmanın bir yoludur.
- Bir kalıp, sorunun tanımı ve çözümünün özüdür.
- Farklı ortamlarda yeniden kullanılabilmesi için yeterince soyut olmalıdır.
- Desen açıklamaları genellikle kalıtım ve çok biçimlilik gibi nesne yönelimli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

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 ödünleşimleri.

Gözlemci Deseni



- İsim
  - Gözlemci.
- Açıklama
  - Nesne durumunun görüntüsünü nesnenin kendisinden ayırır.
- Sorun Açıklaması
  - Birden fazla durum göstergesi gerektiğinde kullanılır.
- Çözüm açıklaması
  - UML açıklamalı slayta bakın.
- Sonuçlar
  - Ekran performansını artırmaya yönelik optimizasyonlar pratik değildir.

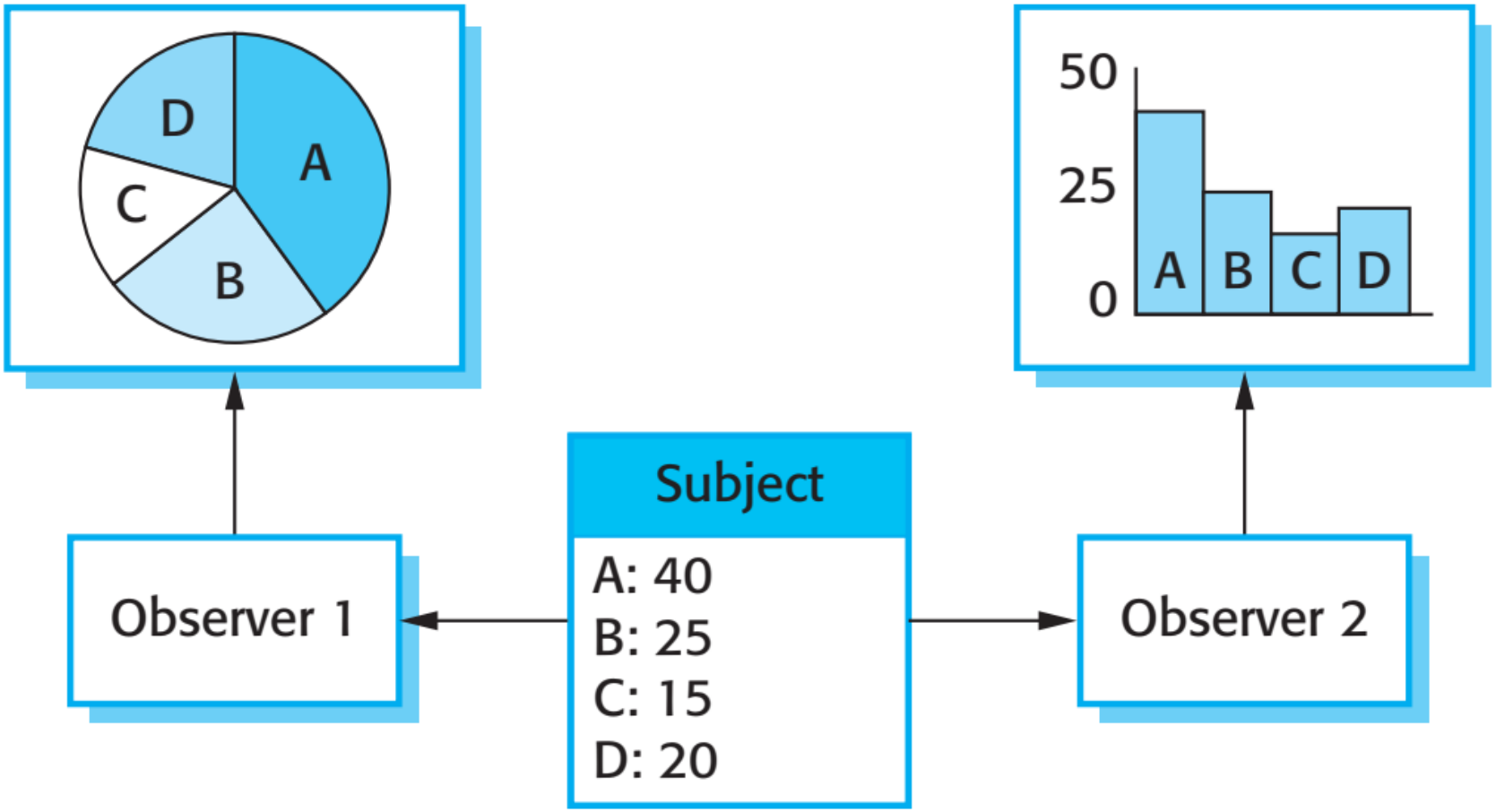
Gözlemci Deseni (1)

Desen adı	Gözlemci
Açıklama	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.
Sorun Açıklaması	Çoğu durumda, grafik ekran ve tablo görünümü gibi birden çok durum bilgisi sunmanız 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.

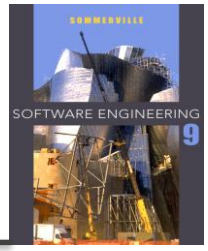
Gözlemci Deseni (2)

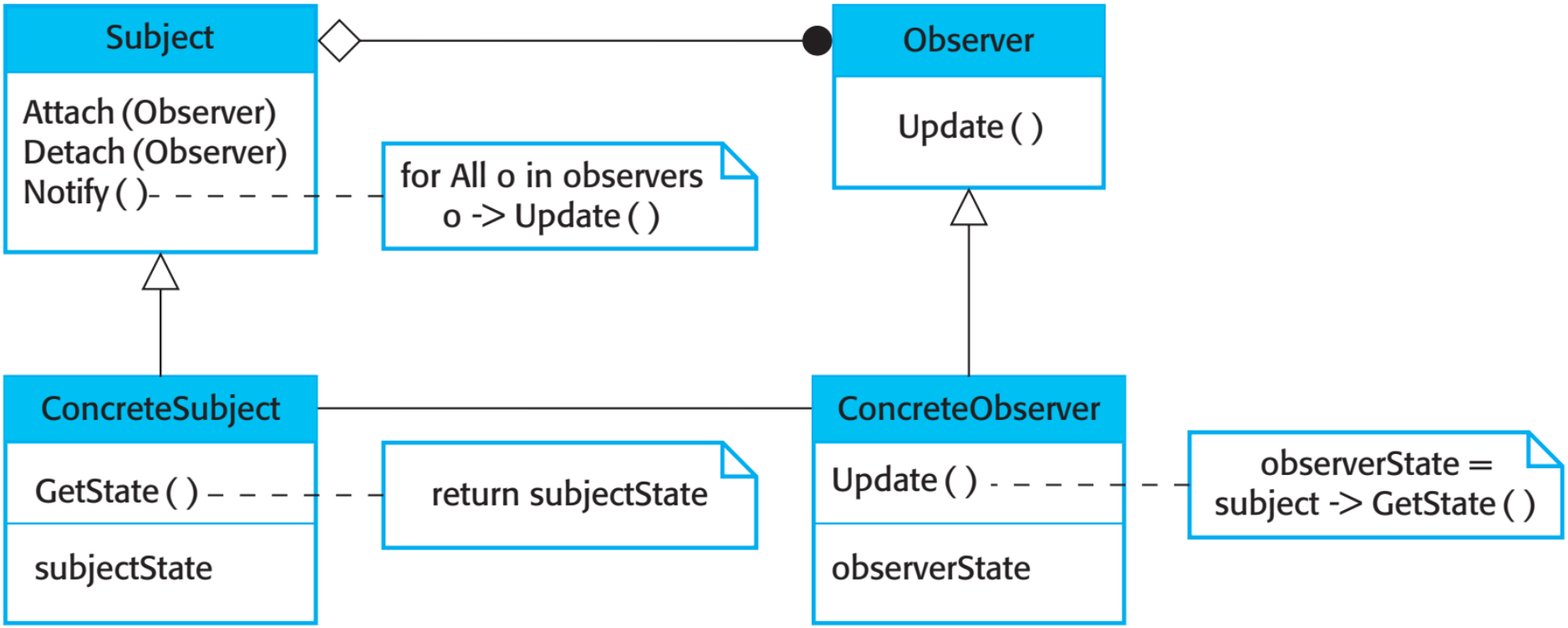
Desen adı	Gözlemci
Çözüm açıklaması	Bu, konu ve Gözlemci olmak üzere iki soyut nesneyi ve ilgili soyut nesnelerin niteliklerini miras alan iki somut nesneyi, ConcreteSubject ve ConcreteObject'i içerir. Soyut nesneler, her durumda uygulanabilen genel işlemleri içerir. Görüntülenecek durum, Gözlemcileri eklemesine ve kaldırmasına (her bir gözlemci bir ekrana karşılık gelir) ve durum değiştiğinde bir bildirim göndermesine izin veren özneden işlemleri devralan ConcreteSubject'te tutulur. ConcreteObserver, ConcreteSubject durumunun bir kopyasını saklar ve bu kopyaların adım adım tutulmasına izin veren Observer'ın Update () arayüzünü uygular. ConcreteObserver, durumu otomatik olarak görüntüler ve durum her güncellendiğinde değişiklikleri yansıtır.
Sonuçlar	Denek yalnızca soyut Gözlemci'yi bilir ve somut sınıfın ayrıntılarını bilmez. Bu nedenle, bu nesneler arasında minimum bağlantı vardır. Bu bilgi eksikliği nedeniyle, ekran performansını artıran optimizasyonlar pratik değildir. Konudaki değişiklikler, gözlemciler için bir dizi bağlantılı güncellemenin üretilmesine neden olabilir, bunlardan bazıları gerekli olmayabilir.

Gözlemci Desenini Kullanan Birden Çok Ekran

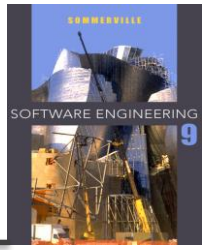


Gözlemci Deseninin Bir UML Modeli



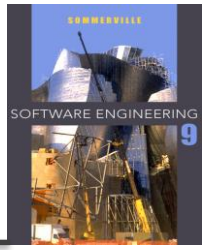


Tasarım Sorunları



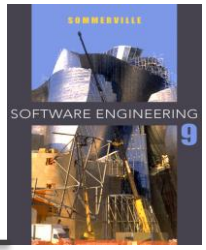
- Tasarımınızda desenleri kullanmak için, karşılaştığınız herhangi bir tasarım probleminin uygulanabilecek ilişkili bir desene sahip olabileceğini bilmeniz gerekir.
  - Birkaç nesneye başka bir nesnenin durumunun değiştiğini söyleyin (Gözlemci deseni).
  - Arayüzleri, genellikle aşamalı olarak geliştirilen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

Entegrasyon Sorunları



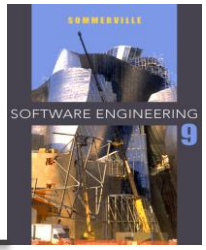
- Burada odaklanma, programlamaya değil, açıkça önemli olmasına rağmen, genellikle programlama metinlerinde ele alınmayan diğer uygulama sorunlarına odaklanmaktadır:
  - Yeniden Kullanım Çoğu modern yazılım, mevcut bileşenlerin veya sistemlerin yeniden kullanılmasıyla oluşturulur. Yazılım geliştirirken, mevcut koddan olabildiğince fazla yararlanmalısınız.
  - Konfigürasyon yönetimi Geliştirme süreci sırasında, bir konfigürasyon yönetim sistemindeki her bir yazılım bileşeninin birçok farklı sürümünü izlemeniz gerekir.
  - Ana-hedef geliştirme Üretim yazılımı genellikle yazılım geliştirme ortamıyla aynı bilgisayarda çalıştırılmaz. Bunun yerine, onu bir bilgisayarda (ana sistem) geliştirir ve ayrı bir bilgisayarda (hedef sistem) yürütürsünüz.

Yeniden Kullanım



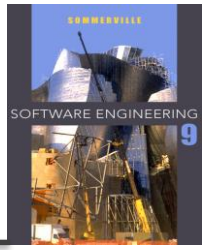
- 1960'lardan 1990'lara kadar, çoğu yeni yazılım, tüm kodların üst düzey bir programlama dilinde yazılmasıyla sıfırdan geliştirildi.
  - Tek önemli yeniden kullanım veya yazılım, programlama dili kitaplıklarında işlevlerin ve nesnelerin yeniden kullanılmasıydı.
- Maliyetler ve program baskısı, bu yaklaşımın 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 Seviyeleri



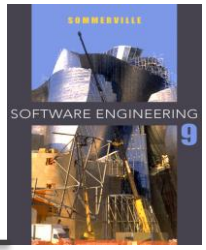
- Soyutlama seviyesi
  - Bu düzeyde, yazılımı doğrudan yeniden kullanmazsınız, ancak yazılımınızın tasarımında başarılı soyutlama bilgilerini kullanırsınız.
- Nesne seviyesi
  - Bu düzeyde, kodu kendiniz yazmak yerine doğrudan bir kitaplıktaki nesneleri yeniden kullanırsınız.
- Bileşen seviyesi
  - Bileşenler,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

Yeniden Kullanım Maliyetleri



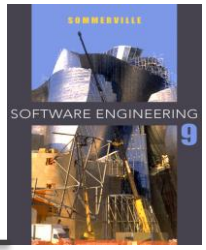
- Yeniden kullanmak ve ihtiyaçlarınızı karşılayıp karşılamadığını değerlendirmek için yazılım aramak için harcanan zamanın maliyeti.
- Uygulanabildiği yerde, yeniden kullanılabilir yazılımı satın almanın maliyetleri. Kullanıma hazır büyük sistemler için bu maliyetler çok yüksek olabilir.
- Yeniden kullanılabilir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

Konfigürasyon Yönetimi



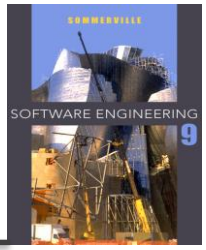
- Konfigürasyon yönetimi, değişen bir yazılım sistemini yönetmenin genel sürecine verilen addır.
- Konfigürasyon yönetiminin amacı, tüm geliştiricilerin proje koduna ve belgelere kontrollü bir şekilde erişebilmesi, hangi değişikliklerin yapıldığını bulabilmesi ve bir sistem oluşturmak için bileşenleri derleyip bağlayabilmesi için sistem entegrasyon sürecini desteklemektir.
- Ayrıca Ders 25'e bakınız.

Yapılandırma Yönetimi Faaliyetleri



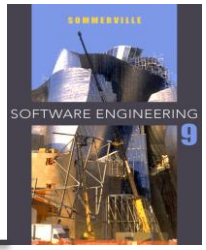
- Yazılım bileşenlerinin farklı sürümlerini takip etmek için desteğin sağlandığı sürüm yönetimi. Sürüm yönetimi sistemleri, birkaç programcı tarafından geliştirmeyi koordine edecek tesisleri içerir.
- Geliştiricilerin, bir sistemin her sürümünü oluşturmak için hangi bileşen sürümlerinin kullanıldığını tanımlamasına yardımcı olmak için desteğin sağlandığı sistem entegrasyonu. Bu açıklama daha sonra gerekli bileşenleri derleyip 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.

Ana-hedef Geliştirme



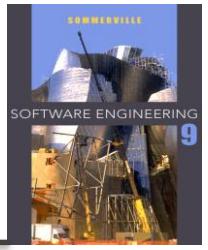
- Çoğu yazılım bir bilgisayarda (ana bilgisayar) geliştirilir, ancak ayrı bir makinede (hedef) çalışır.
- Daha genel olarak, bir geliştirme platformu ve bir yürütme platformundan bahsedebiliriz.
  - Bir platform, donanımdan daha fazlasıdır.
  - Kurulu işletim sistemini ve bir veritabanı yönetim sistemi 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.

Geliştirme Platformu Araçları



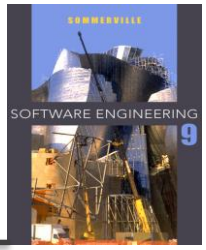
- Kod oluşturmanıza, düzenlemenize ve derlemenize olanak tanıyan entegre bir derleyici ve sözdizimine yönelik düzenleme sistemi.
- Bir dil hata ayıklama sistemi.
- UML modellerini düzenlemek için araçlar gibi grafik düzenleme araçları.
- Bir programın yeni sürümünde otomatik olarak bir dizi test çalıştırabilen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

Entegre Geliştirme Ortamları (IDE'ler)



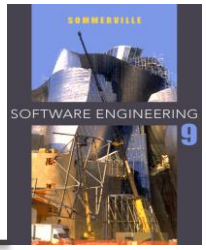
- Yazılım geliştirme araçları genellikle bir entegre geliştirme ortamı (IDE) oluşturmak için gruplanır.
- Bir IDE, bazı ortak çerçeve ve kullanıcı arabirimi dahilinde yazılım geliştirmenin farklı yönlerini destekleyen bir dizi yazılım aracıdır.
- IDE'ler, C# gibi belirli bir programlama dilinde geliştirmeyi desteklemek için oluşturulur. Dil IDE'si özel olarak geliştirilebilir veya belirli dil destek araçlarıyla genel amaçlı bir IDE'nin somutlaşmış hali olabilir.

Bileşen / Sistem Dağıtım Faktörleri



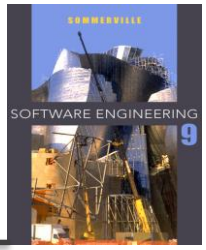
- Bir bileşen belirli bir donanım mimarisi için tasarlanmışsa veya başka bir yazılım sistemine dayanıyorsa, gerekli donanım ve yazılım desteğini sağlayan bir platformda kullanılması gerektiği açıktır.
- Yüksek kullanılabilirlikli sistemler, bileşenlerin birden fazla platformda konuşlandırılmasını gerektirebilir. 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



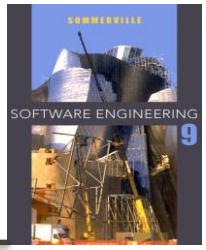
- Açık kaynak geliştirme, bir yazılım sisteminin kaynak kodunun yayınlandığı ve gönüllülerin geliştirme sürecine katılmaya davet edildiği bir yazılım geliştirme yaklaşımıdır.
- Kökleri, kaynak kodunun tescilli olmaması gerektiğini savunan Özgür Yazılım Vakfı'na (www.fsf.org) dayanmaktadır, bunun yerine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

Açık Kaynak Sistemler



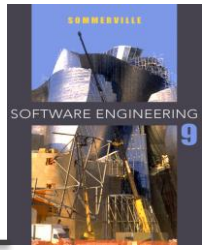
- En çok bilinen açık kaynaklı ürün, elbette, yaygın olarak bir sunucu sistemi ve giderek artan bir şekilde masaüstü ortamı olarak kullanılan Linux işletim sistemidir.
- Diğer önemli açık kaynaklı ürünler Java, .NET, Apache web sunucusu ve mySQL veritabanı yönetim sistemidir.

Açık Kaynak Sorunları



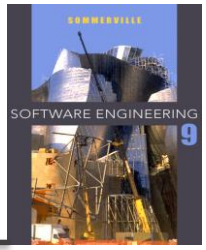
- Geliştirilmekte olan ürün açık kaynaklı bileşenlerden yararlanmalı mı?
- Yazılımın geliştirilmesi için açık kaynaklı bir yaklaşım kullanılmalı mı?

Açık Kaynak İşletmesi



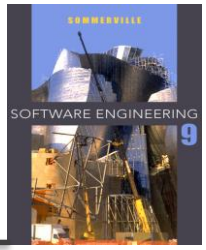
- Giderek daha fazla ürün şirketi, geliştirme için açık kaynaklı bir yaklaşım kullanıyor.
- İş modelleri, bir yazılım ürünü satmaya değil, o ürün için destek satmaya dayalıdır.
- Açık kaynak topluluğunun dahil edilmesinin, yazılımın daha ucuza, daha hızlı geliştirilmesine olanak sağlayacağına ve yazılım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

Açık Kaynak Lisanslama



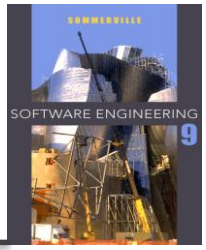
- Açık kaynak geliştirmenin temel bir ilkesi, kaynak kodunun ücretsiz olarak erişilebilir olmasıdır, bu, herhangi birinin bu kodla dilediğini yapabileceği anlamına gelmez.
  - Yasal olarak, kodun geliştiricisi (bir şirket veya bir birey) hala koda sahiptir. Açık kaynaklı bir yazılım lisansına yasal olarak bağlayıcı koşullar ekleyerek nasıl kullanıldığına ilişkin kısıtlamalar getirebilirler.
  - Bazı açık kaynak geliştiricileri, yeni bir sistem geliştirmek için açık kaynaklı bir bileşen kullanılıyorsa, bu 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



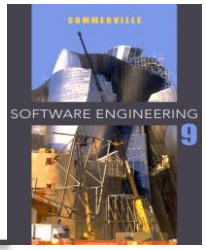
- GNU Genel Kamu Lisansı (GPL). Bu sözde "karşılıklı" bir lisanstır, yani GPL lisansı altında lisanslanan açık kaynaklı bir yazılım kullanıyorsanız, o zaman bu yazılımı açık kaynak yapmanız gerekir.
- GNU Kısıtlı Genel Kamu Lisansı (LGPL), bu bileşenlerin kaynağını yayınlamak zorunda kalmadan açık kaynak koduna bağlanan bileşenleri yazabileceğiniz bir GPL lisansı çeşididir.
- Berkley Standart Dağıtım (BSD) Lisansı. Bu karşılıklı olmayan bir lisanstır, yani 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 kaynaklı bileşenlerle ilgili bilgileri korumak için bir sistem kurun.
- Farklı lisans türlerinin farkında olun ve kullanılmadan önce bir bileşenin nasıl lisanslandığını anlayın.
- Bileşenler için gelişim yollarının farkında olun.
- İnsanları açık kaynak konusunda eğitin.
- Yerinde denetim sistemleri bulundurun.
- Açık kaynak topluluğuna katılın.

Bölüm 2'nin Anahtar Noktaları



- Yazılım geliştirirken, mevcut yazılımı bileşenler, hizmetler veya eksiksiz 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, ana bilgisayar <-> hedef geliştirmedir. Yürütülmek 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.