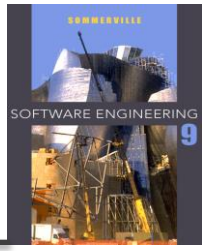
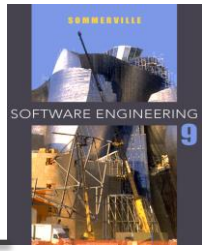


Chapter 7 – Tasarım ve Gerçekleştirim

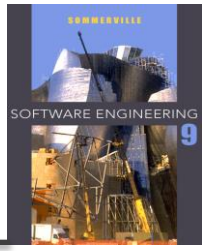
Lecture 1

Konular



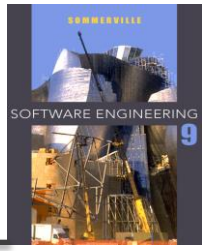
- ✧ UML ile nesneye dayalı tasarım
- ✧ Tasarım Desenleri
- ✧ Gerçekleştirim konuları
- ✧ Açık kaynak gerçekleştirim

Tasarım ve Gerçekleştirim



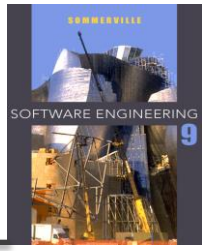
- Yazılım tasarımı ve geliştirme, yazılım mühendisliğinin çalıştırılabilir bir yazılım sistemi geliştirme aşamasıdır.
- Tasarım ve gerçekleştirim aşamaları iç içe geçmiş aşamalardır
  - Tasarım aşaması, müşterinin gereksinimleri doğrultusunda yazılım bileşenlerinin belirlendiği ve bu bileşenlerin ilişkilerinin tanımlandığı **yaratıcı** bir aşamadır.
  - Gerçekleştirim aşaması, tasarımın bir program olarak gerçekleştirilmesi aşamasıdır.

Geliştirmek mi? Satın almak mı?



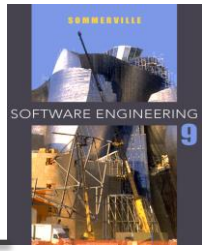
- Bir çok alan için, raf ürünü haline gelmiş hazır yazılım sistemlerini satın alıp kullanıcının ihtiyacına uygun hale getirmek mümkün.
  - Örneğin, bir hastane bilgi sistemi geliştirmek istiyorsanız, hali hazırda hastanelerde kullanılan bir sistemi satın almak, bir programlama dili ile bu sistemi baştan geliştirmekten daha ucuz ve hızlı olacaktır.
- Eğer uygulamanızı bu yaklaşımla geliştiriyorsanız tasarım aşaması, kullanıcı gereksinimlerini karşılayacak biçimde sistemi nasıl “ayarlarım” sorusuna indirgenebilir.

Nesneye Dayalı Bir Tasarım Süreci



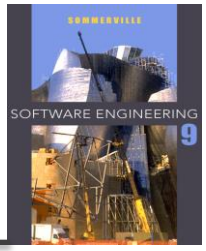
- Yapısal nesneye dayalı tasarım süreçleri birkaç farklı sistem modeli geliştirilmesini içerir
- Bu süreçler, modellerin geliştirilmesi ve bakımı için oldukça fazla efor gerektirirler. Bu durum, küçük sistemler için maliyet-etkin olmaz.
- Ancak, farklı gruplar tarafından geliştirilen büyük sistemler için bu modeller iyi birer haberleşme mekanizması görevi görürler.

Sürecin aşamaları



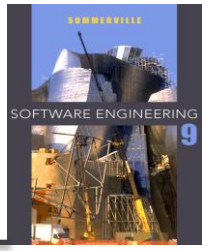
- Sürecin nasıl organize edildiğine bağlı olarak birçok tipte farklı nesneye dayalı tasarım süreçleri vardır.
- Bu süreçlerin ortak olarak bulundurdukları aşamalar:
  - Sistemin bağlamının ve kullanım biçimlerinin tanımlanması
  - Sistemin mimarisinin tanımlanması
  - Sistemin temel nesnelerinin belirlenmesi
  - Tasarım modellerinin geliştirilmesi
  - Nesne arayüzlerinin tanımlanması
- Bu sunumum devamında hava durumu tahmin istasyonu örneğini kullanacağız.

Sistemin bağlamı ve etkileşimleri



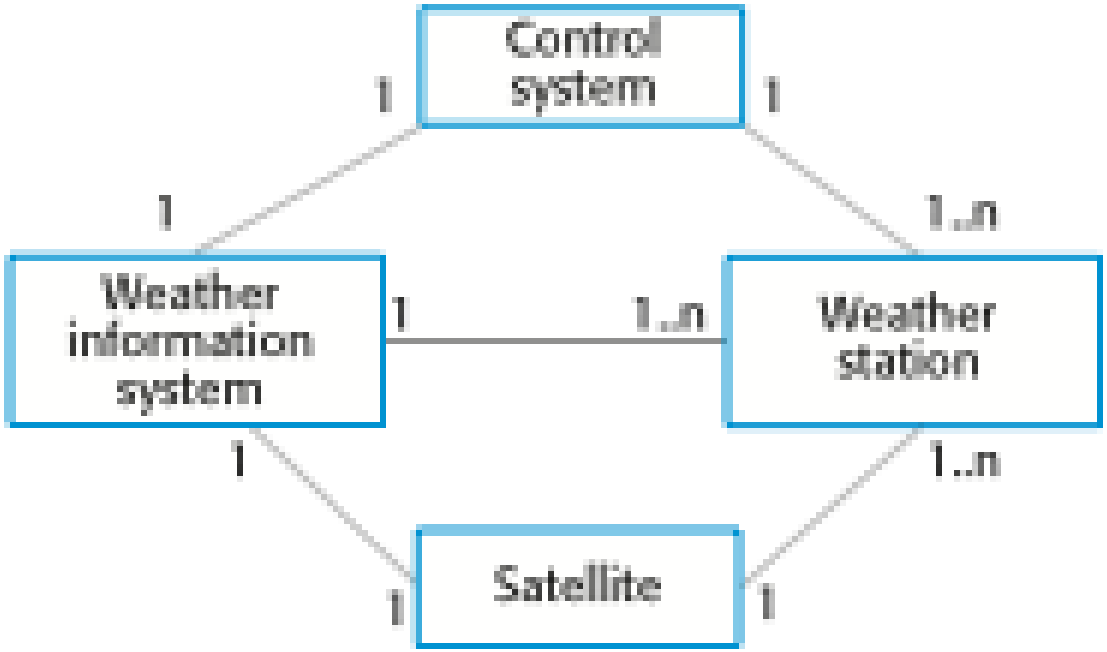
- Geliştirilmekte olan sistemin ve bu sistemin kullanılacağı çevrenin arasındaki ilişkilerin anlaşılması, gereksinim duyulan sistem fonksiyonelliklerinin nasıl sağlanacağı ve sistemin çevresi ile nasıl iletişimde olacağı konusuna temel teşkil eder.
- Sistemin bağlamının anlaşılması aynı zamanda sistemin sınırlarının belirlenmesinde kullanılır. Bu da, hangi fonksiyonelliklerin sistemde yer alacağı ve hangilerinin almayacağı sorusuna cevap bulmakta yardımcı olur.

Bağlam ve Etkileşim Modelleri

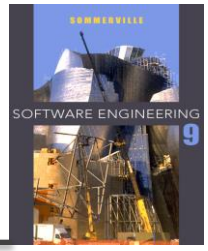


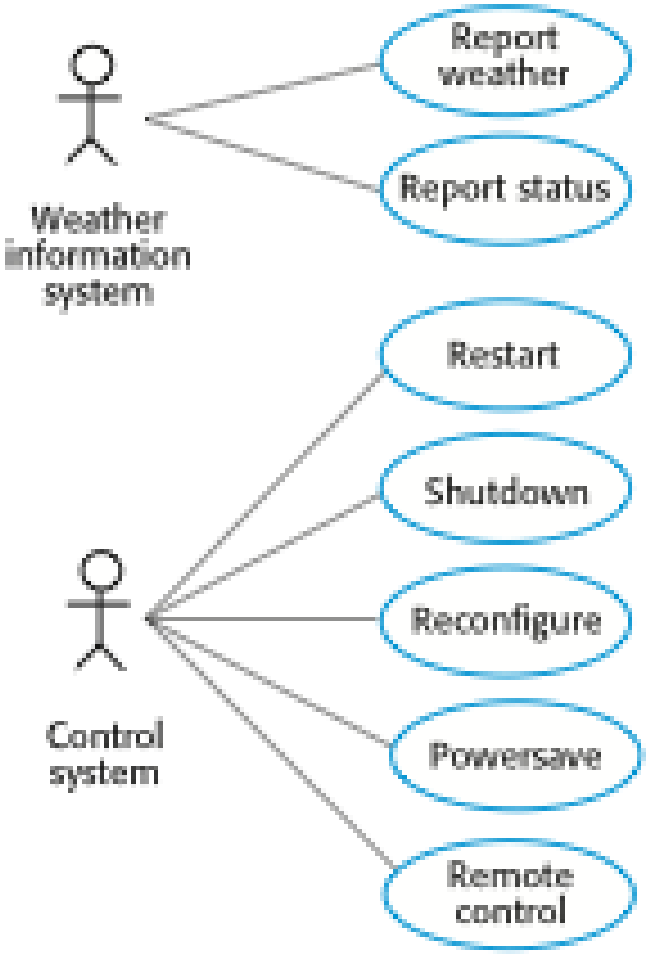
- Bir içerik modeli, geliştirilmekte olan sistemin çevresinde bulunan diğer sistemleri gösteren yapısal bir modeldir.
- Bir etkileşim modeli, geliştirilmekte olan sistemin çevresindeki sistemlerle nasıl etkileşeceğini gösteren dinamik bir modeldir.

Hava tahmin istasyonu için sistem bağlamı

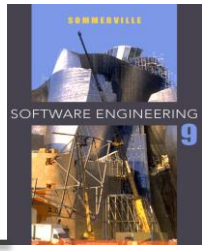


Hava tahmin istasyonu kullanım durumları



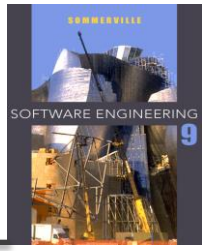


Kullanım durumu tanımı—Hava tahminini raporla



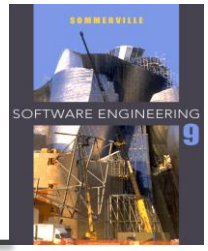
Sistem	Hava tahmin istasyonu
Kullanım Durumu	Hava tahminini raporla
Aktörler	Hava durumu bilgi sistemi, hava tahmin istasyonu
Tanım	Hava tahmin istasyonu, veri toplama periyodunda cihazlardan topladığı bilgilerin bir özetini Hava durumu bilgi sistemine gönderir. Gönderilen veri, toprak ve havanın maksimum, minimum ve ortalama sıcaklıkları; maksimum minimum ve ortalama hava basıncı değerleri; maksimum minimum ve ortalama rüzgar hızı değerleri; toplam yağış miktarı ve 5 dakika aralıklarla ölçülen rüzgar yönü bilgileridir.
Uyarıcılar	Hava durumu bilgi sistemi bir uydu bağlantısı üzerinden hava tahmin istasyonu ile bir bağlantı kurar ve ilgili bilgileri ister.
Yanıt	Özetlenen bilgiler Hava durumu bilgi sistemine gönderilir.
Yorumlar	Hava tahmin istasyonlarından genellikle saatte bir veri göndermesi istenir ancak bu sıklık bir istasyondan diğerine değişebilir ve gelecekte bu veri isteme periyodu değiştirilebilir.

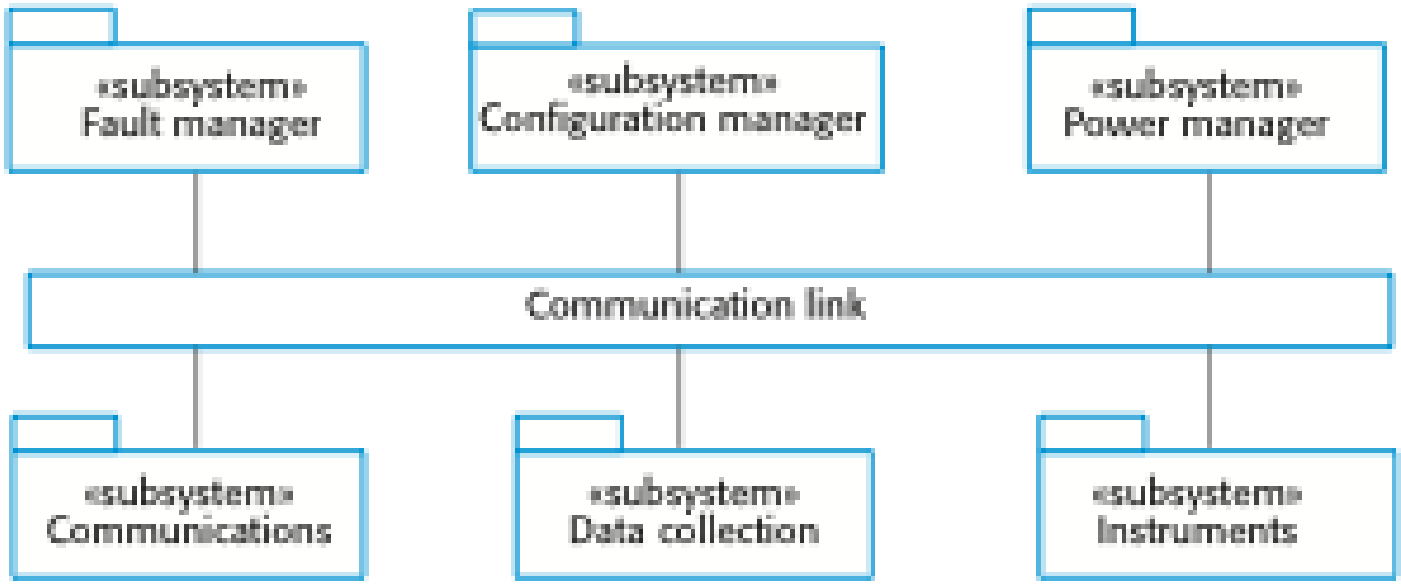
Mimari tasarım



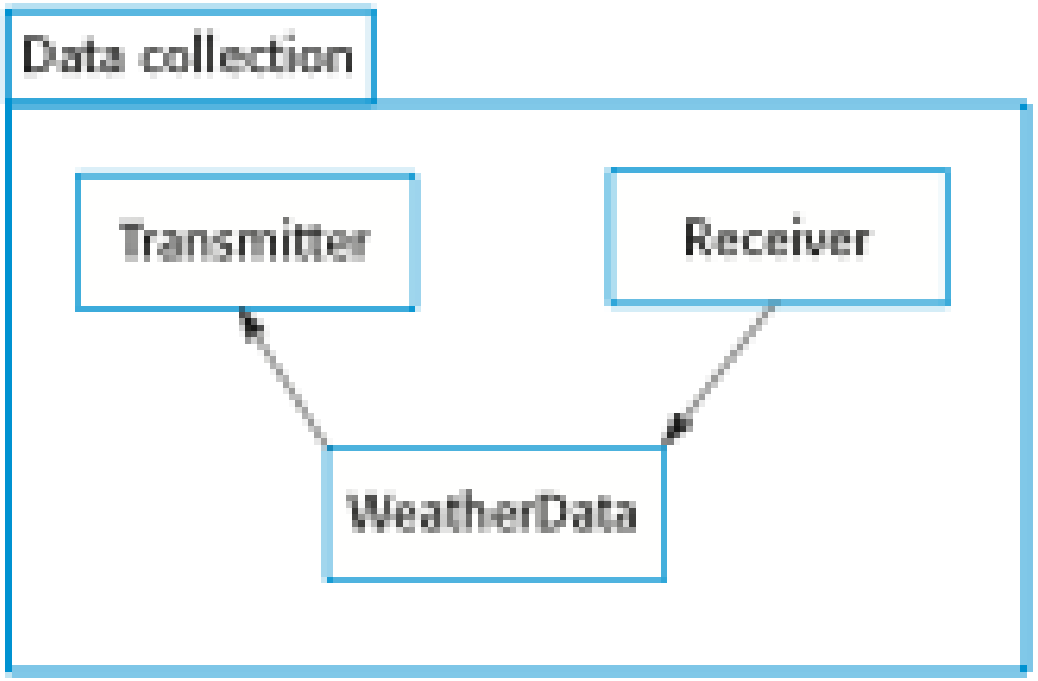
- Geliştirilmekte olan sistemin ve çevresinin etkileşimi anlaşıldıktan sonra bu bilgi sistemin mimarisini geliştirirken kullanılabilir.
- Sistemi oluşturan büyük bileşenler ve bunlar arasındaki ilişkiler belirlenir. Katmanlı mimari, istemci-sunucu modeli gibi mimari desenlerden biriyle bu bileşenler organize edilebilir.
- Hava durumu istasyonu, ortak bir altyapı üzerinden yayın yaparak birbirleri ile haberleşen bağımsız bileşenlerden oluşmaktadır.

Hava tahmin istasyonu için yüksek seviyeli bir mimari model

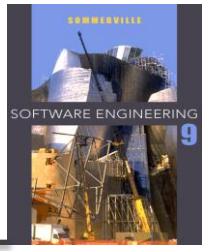




Veri toplama sisteminin mimarisi

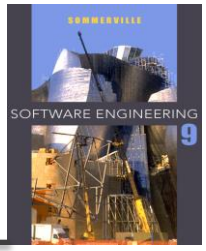


Nesne sınıflarını belirleme



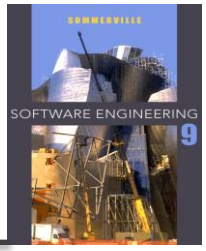
- Nesne sınıflarının belirlenmesi aşaması, nesneye dayalı tasarımın zor bir bölümüdür.
- Nesne sınıflarını belirlemek için “sihirli bir formül” yoktur. Bu iş biraz yetenek, tecrübe ve saha bilgisi gerektirir.
- Nesne sınıfı belirleme süreci tekrarlı bir süreçtir. İlk seferinde bütün sınıflar doğru olarak belirlenemeyebilir.

Nesne sınıflarını belirleme yaklaşımları



- Sistemin tanımı için doğal bir dili temel alan kurallı bir yaklaşım kullanılabilir.
- Uygulama alanındaki somut nesneler temel alınarak sınıflar belirlenebilir.
- Nesnelerin hangi davranışla nelere müdahil olduklarını belirlemek için davranışsal bir model kullanılabilir.
- Senaryo tabanlı analizler kullanılabilir. Her bir senaryodaki nesneler, özellikler ve metodlar belirlenebilir.

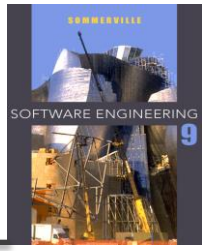
Hava tahmin istasyonu için bir tanım



Hava tahmin istasyonu, cihazlardan bazı bilgileri toplayan; bu bilgiler üzerinde bazı başlangıç veri işleme süreçlerini gerçekleştiren ve verileri veri yönetim sistemine ileten bir yazılım paketidir. Cihazlar şunlardır: toprak ve hava termometresi, rüzgar hızı ölçer, rüzgar yönü ölçer, barometre ve yağış ölçer. Veriler periyodik olarak toplanır.

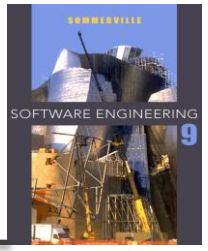
Hava durumu bilgisini iletmesi için bir komut aldığı zaman hava tahmin istasyonu, topladığı verileri özetler ve işler. Bir istek alındığı zaman özetlenen veri, daha önceden belirlenen bir bilgisayara iletilir.

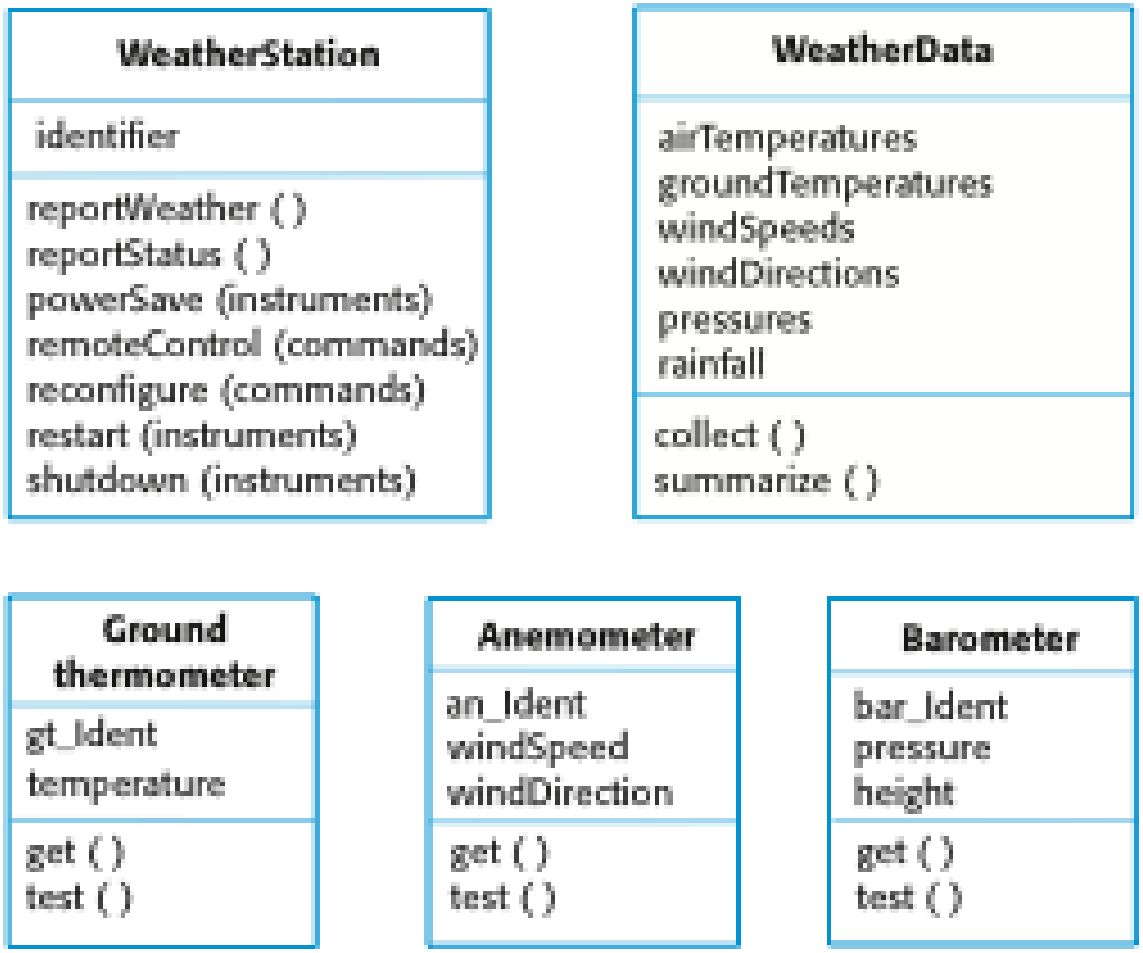
Hava tahmin istasyonu nesne sınıfları



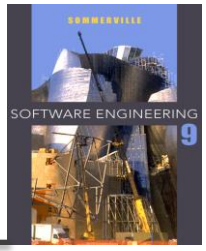
- Hava tahmin istasyonu için nesne sınıflarını belirleme işi somut nesneler ve veri üzerinden gerçekleştirilebilir.
  - Toprak termometresi, Rüzgar Hızı Ölçer, Barometre
    - Application domain objects that are ‘hardware’ objects related to the instruments in the system.
  - Hava tahmin istasyonu
    - Hava tahmin istasyonunun temel “arayüzü”. Bu arayüz, kullanım durumu modellerindeki etkileşimleri yansıtır.
  - Hava durumu bilgisi
    - Cihazlardan alınan bilgileri kapsar.

Hava tahmin istasyonu nesne sınıfları



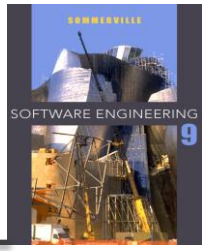


Tasarım modelleri



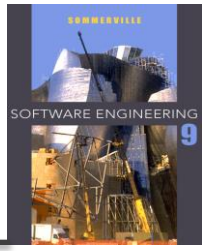
- Tasarım modelleri nesneler ve nesne sınıfları ve bunlar arasındaki ilişkileri gösterir.
- Statik modeller, sistemi nesne sınıfları ve bunların ilişkileri cinsinden statik olarak gösterir.
- Dinamik modeller, nesneler arasındaki dinamik etkileşimleri gösterir.

Tasarım modelleri örnekleri



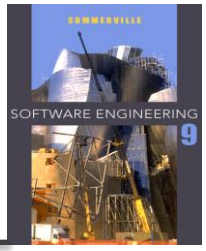
- Alt sistem modelleri, nesnelerin mantıksal olarak ilgili alt sistemler altında gruplanmasını gösterir.
- Sequence modelleri, nesne etkileşimlerinin sırasını gösterir.
- Durum makinesi modelleri, bireysel nesnelerin olaylara karşın durumlarını nasıl değiştirdiklerini gösterir.
- Diğer bazı modeller şöyledir: kullanım durumu modelleri, toplanım modelleri, genelleme modelleri vb.

Alt sistem modelleri



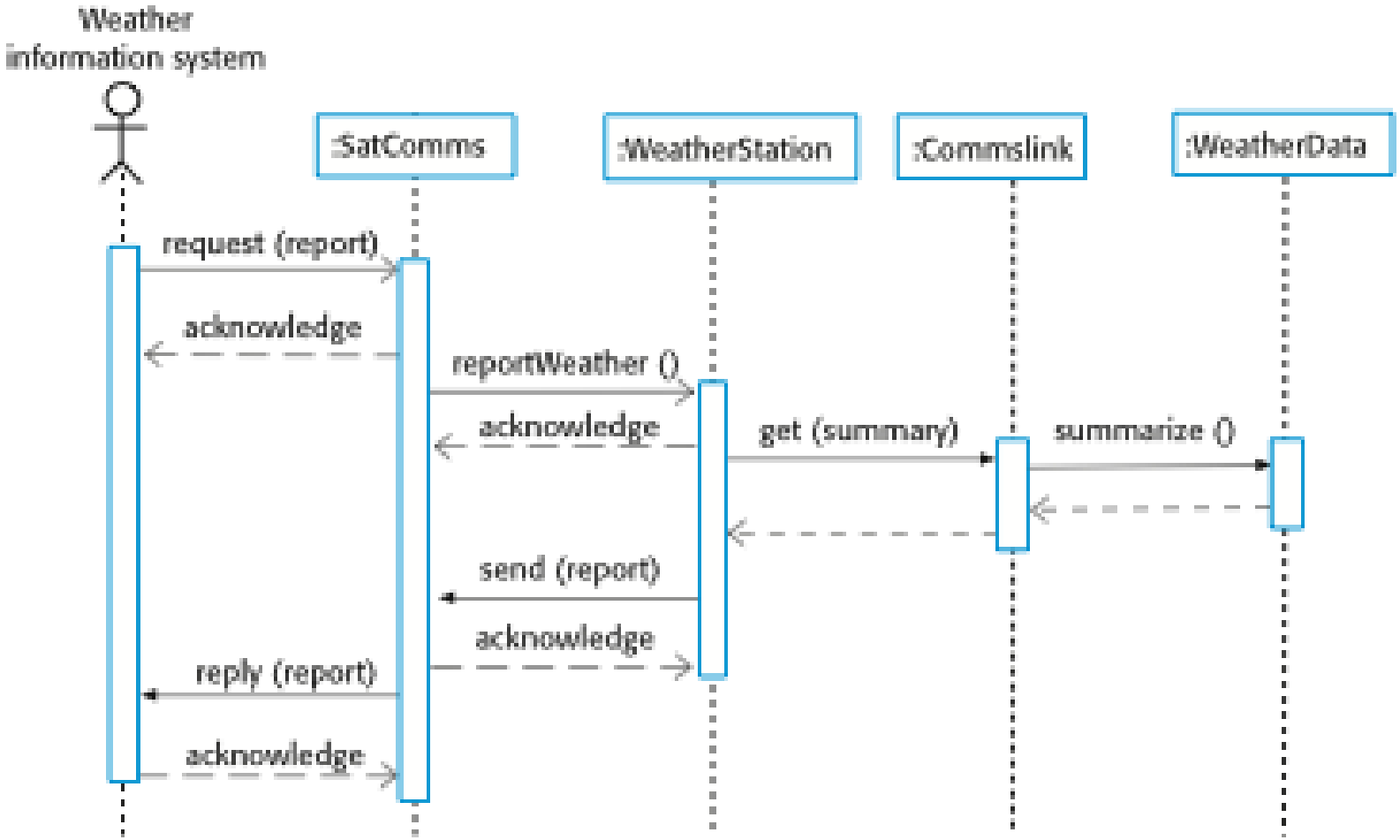
- Tasarımın mantıksal olarak birbirleri ile ilgili nesnelerin grupları olarak nasıl organize edildiklerini gösterir.
- UML'de bunlar paketler olarak gösterilir ve bu bir mantıksal modeldir.

Sequence modelleri

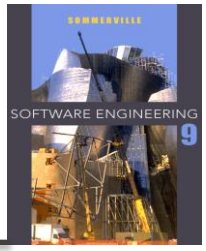


- Sequence modelleri, nesne etkileşimlerinin sırasını gösterir.
  - Nesneler en üstte yatay olarak sıralanır.
  - Zaman dikey olarak gösterilir.
  - Etkileşimler etiketlenmiş oklar ile gösterilir. Farklı tip okların farklı anlamları vardır.
  - Bir nesnenin yaşam sürecindeki kalın dikdörtgen çizgi ile gösterilen kısmı, nesnenin “kullanıldığı” zamanı gösterir.

Veri toplama işini tanımlayan sequence diyagramı

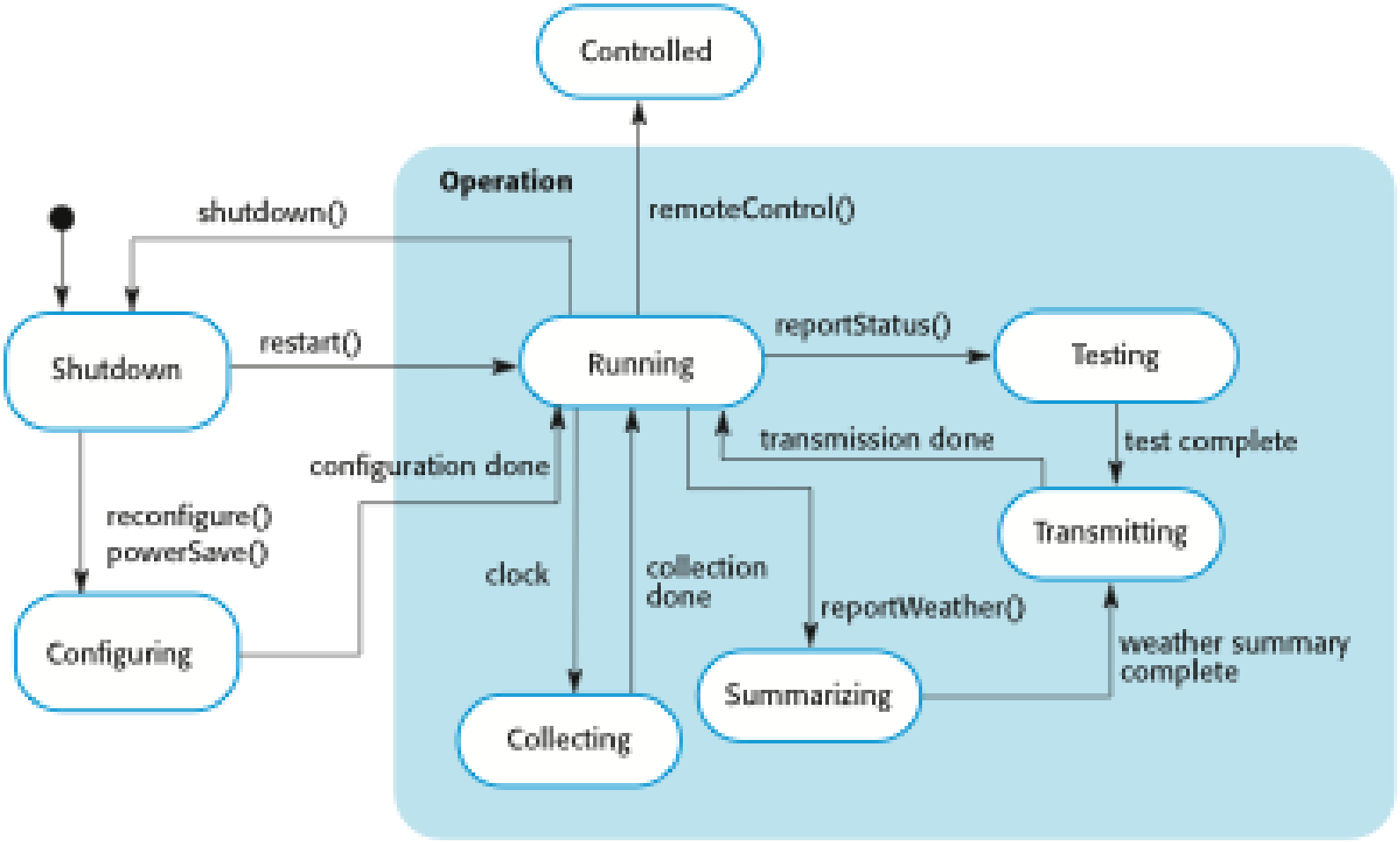


Durum diyagramları

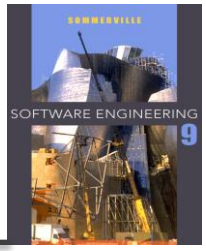


- Durum diyagramları, bireysel nesnelerin olaylara karşın durumlarını nasıl değiştirdiklerini gösterir.
- Bir sistemde her zaman ihtiyaç duyulmaz. Bir sistemdeki çoğu nesne basittir ve durum diyagramı gereksiz karmaşıklık ekler.

Hava tahmin istasyonu durum diyagramı

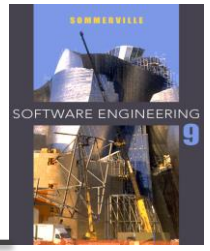


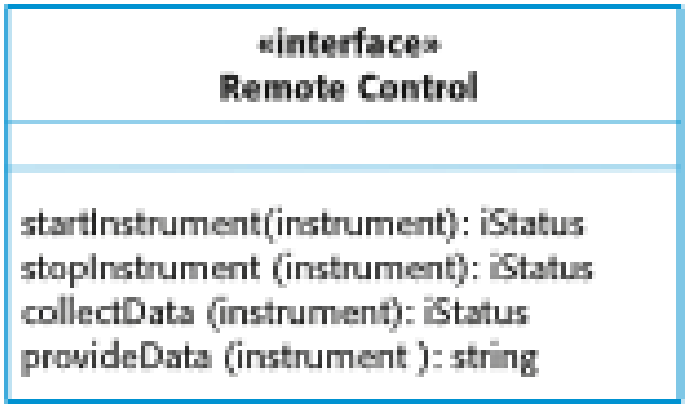
Arayüzün belirlenmesi

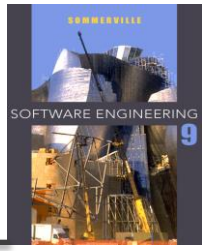


- Farklı nesnelerin ve bileşenlerin paralel olarak tasarlanmaları için nesne arayüzlerinin belirlenmesi zorunludur.
- Arayüz nesnenin içerisine gizlenmiş olarak tasarlanmalıdır.
- Nesneler farklı metodlara göre farklı arayüzlere sahip olabilirler.
- UML arayüz belirleme için sınıf diyagramlarını kullanır. Java da kullanılabilir.

Hava tahmin istasyonu arayüzleri



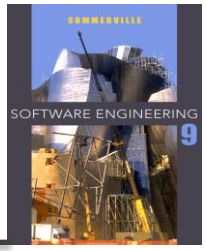




Chapter 7 – Tasarım ve Gerçekleştirim

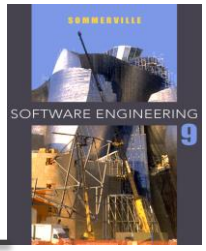
Lecture 2

Tasarım desenleri



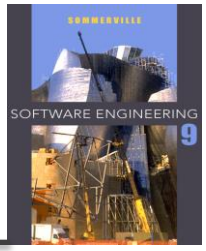
- Bir tasarım deseni, bir problem ve çözümü hakkındaki özet bilginin yeniden kullanımı için bir yoldur.
- Bir desen, bir problemin ve o problemin çözümünün esaslarının tanımıdır.
- Farklı ayarlarla kullanılabilecek kadar özet olmalıdır.
- Desen tanımları genellikle kalıtım ve çok biçimlilik gibi nesneye dayalı karakteristikler kullanır.

Desen bileşenleri



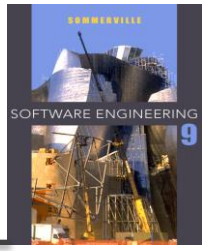
- İsim
  - Anlamlı bir tanımlayıcı
- Problemin tanımı
- Çözümün tanımı
  - Tam bir tasarımdan ziyade farklı yollarla örneklenebilecek bir çözüm taslağı
- Sonuçlar
  - Sonuçları ve desenin avantaj/dezavantaj ilişkisi

Observer (Gözlemci) deseni



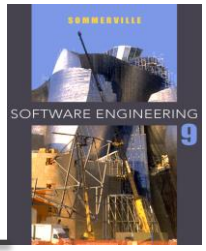
- İsim
  - Gözlemci.
- Açıklama
  - Bir nesnenin durumunun gösterimini nesnenin kendisinden ayırır.
- Problemin tanımı
  - Bir durumun birçok gösterimi gerekli olduğu zaman kullanılır.
- Çözümün tanımı
  - Takip eden sayfalara bakınız.
- Sonuçlar
  - Gösterim performansını geliştirecek olan optimizasyonlar pratik değildir.

Observer (Gözlemci) deseni (1)



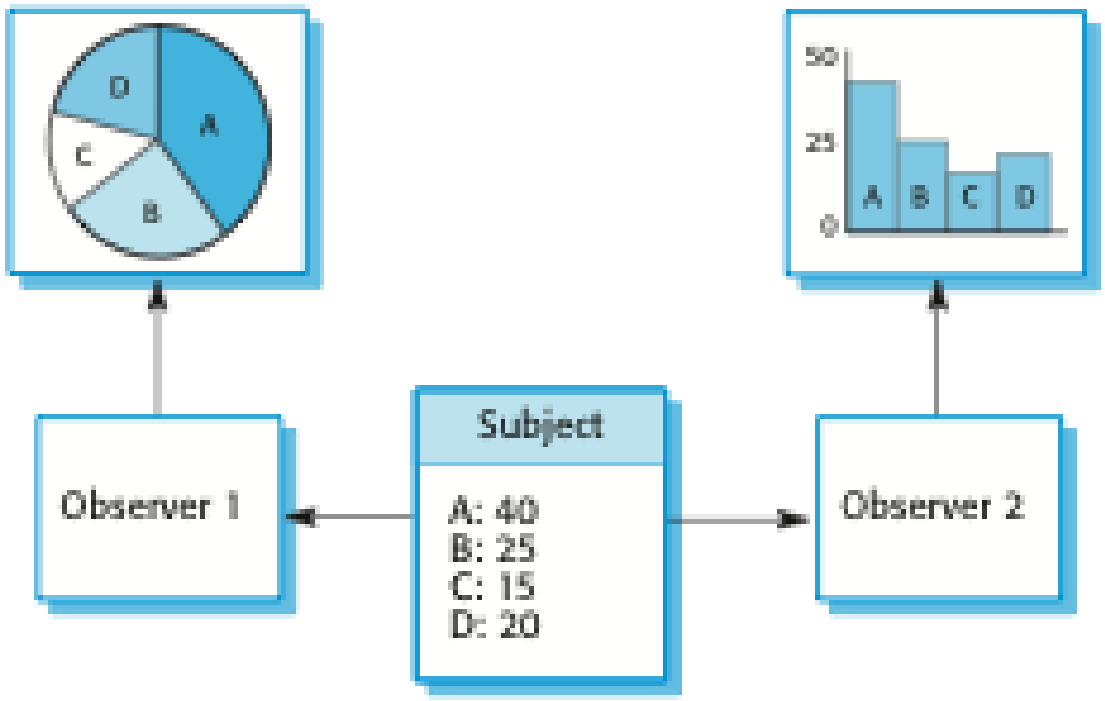
Desen adı	Observer (Gözlemci)
Açıklama	Separates the display of the state of an object from the object itself and allows alternative displays to be provided. When the object state changes, all displays are automatically notified and updated to reflect the change.
Problemin tanımı	In many situations, you have to provide multiple displays of state information, such as a graphical display and a tabular display. Not all of these may be known when the information is specified. All alternative presentations should support interaction and, when the state is changed, all displays must be updated. This pattern may be used in all situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.

Observer (Gözlemci) deseni (2)

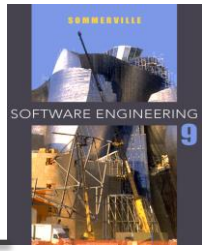


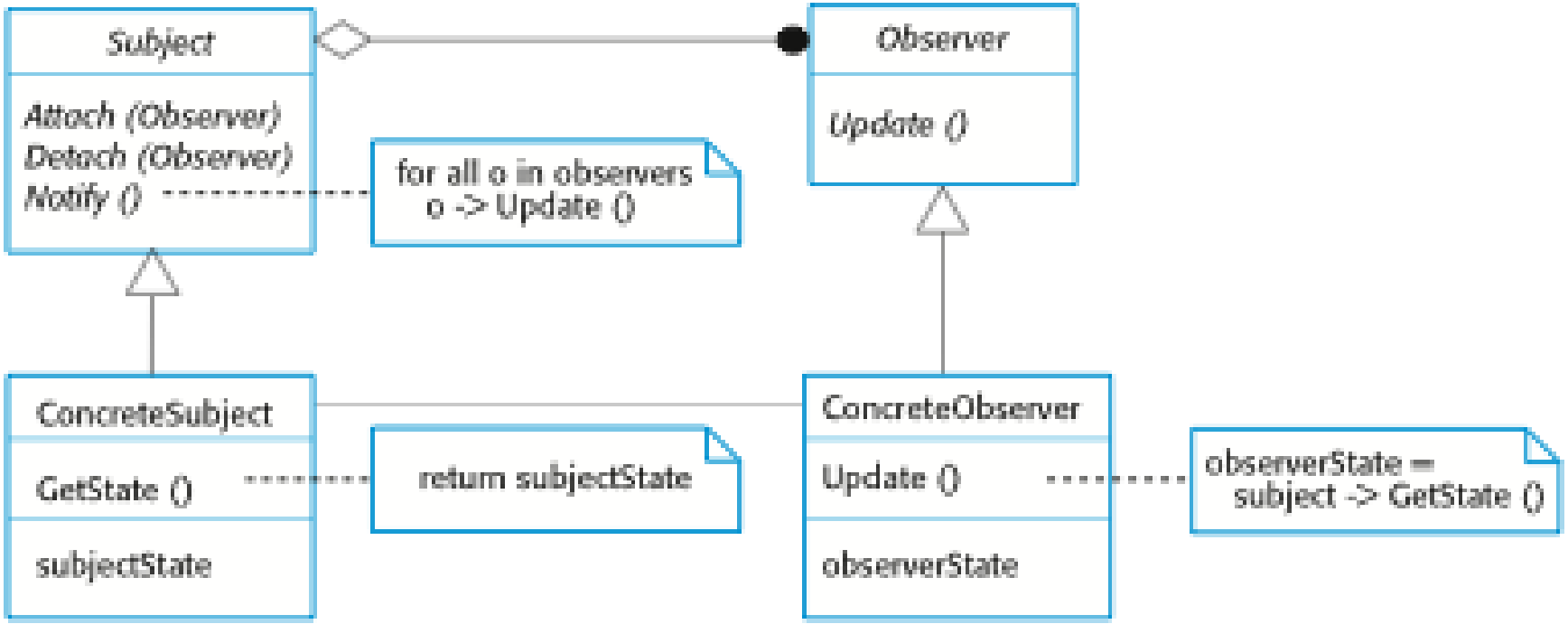
Pattern name	Observer
Çözümün tanımı	This involves two abstract objects, Subject and Observer, and two concrete objects, ConcreteSubject and ConcreteObject, which inherit the attributes of the related abstract objects. The abstract objects include general operations that are applicable in all situations. The state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject allowing it to add and remove Observers (each observer corresponds to a display) and to issue a notification when the state has changed. The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update() interface of Observer that allows these copies to be kept in step. The ConcreteObserver automatically displays the state and reflects changes whenever the state is updated.
Sonuçlar	The subject only knows the abstract Observer and does not know details of the concrete class. Therefore there is minimal coupling between these objects. Because of this lack of knowledge, optimizations that enhance display performance are impractical. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.

Observer (Gözlemci) desenini kullanan çoklu gösterimler

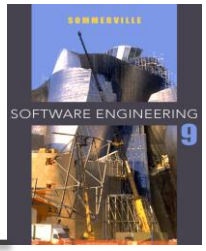


Observer (Gözlemci) deseni için bir UML modeli



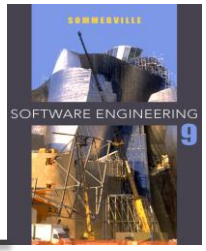


Tasarım Problemleri



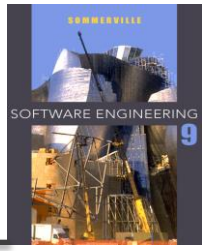
- Bir deseni kullanmak için, karşılaştığınız problemle ilgili hazır bir desenin varlığından haberdar olmanız gerekir.
  - Observer deseni
  - Façade deseni
  - Iterator deseni
  - Decorator deseni
  - …

Gerçekleştirim konuları



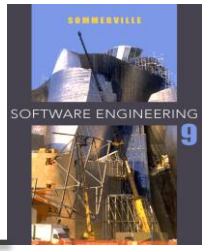
- Buradaki odak konumu programlama değil. Programlama önemli bir kondur ancak, diğer önemli gerçekleştirim konuları programlama kitaplarında pek yer almaz.
  - Yeniden Kullanım Çoğu modern yazılım, var olan sistemlerin bileşenlerinin yeniden kullanımı ile gerçekleştirilir. Bir sistemi geliştirmeye başlamadan önce var olan kodları kullanıp kullanamayacağınıza bakmalısınız.
  - Konfigürasyon Yönetimi Geliştirme sürecinde, her bir yazılım bileşeninin farklı sürümlerini bir konfigürasyon yönetim sistemi ile takip etmelisiniz.
  - Host (geliştirme ortamı)-target (çalışma ortamı) geliştirme Yazılım ürünleri genellikle geliştirildikleri bilgisayarlardan farklı bilgisayarlarda kullanılırlar.

Yeniden kullanım



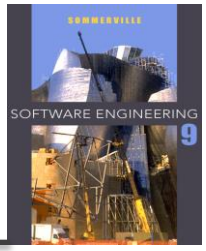
- 1960'lardan 1990'lara kadar çoğu yazılım sistemi yüksek seviyeli programlama dilleri ile yazılarak büyük zorluklarla geliştirildi.
  - Uygulanan tek yeniden kullanım yöntemi, programlama dillerinin kütüphanelerinde bulunan nesneleri veya fonksiyonları yeniden kullanmak şeklindeydi.
- Zaman ve maliyet baskıları bu yöntemi faydasız kıldı.
- Böylece var olan yazılımların yeniden kullanılması ile geliştirme yaklaşımı ortaya çıktı.

Yeniden kullanımın seviyeleri



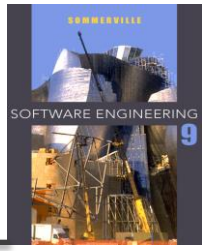
- ✧ Özetleme seviyesi
  - Bu seviyede bir yazılımın kendisini yeniden kullanmazsınız. Onu geliştirirken elde ettiğiniz tecrübeyi yeniden kullanırsınız.
- ✧ Nesne seviyesi
  - Bu seviyede yeniden yazmak yerine bir kütüphanedeki hazır nesneleri kullanırsınız.
- ✧ Bileşen seviyesi
  - Bileşenler,uygulamalarınızda kullanabileceğiniz nesneler ve nesne sınıfları topluluğudur.
- ✧ Sistem seviyesi
  - Bu seviyede bütün sistemi yeniden kullanırsınız.

Yeniden kullanımın maliyetleri



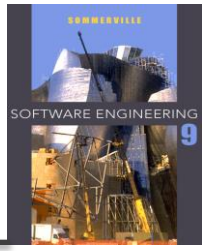
- Yeniden kullanılabilecek bir yazılımın var olup olmadığına ve uygunluğuna bakmak için harcanan zaman.
- Eğer mevcutsa, yeniden kullanılabilir yazılımın satın alma maliyeti.Büyük satılmaya hazır sistemler için bu maliyet yüksek olabilir.
- Geliştirmekte olduğunuz sistemin gereksinimlerini karşılayabilmeleri için var olan yeniden kullanılabilir bileşenlerin veya sistemlerin ayarlanması (konfigürasyonu) ve adapte edilmesi.
- Eğer farklı firmaların bileşenlerini kullanıyorsanız veya kodların bir kısmını kendiniz geliştirdiyseniz, bu bileşenlerin birbirleri ile uyumlu çalışması için bütünleştirme maliyeti.

Konfigürasyon Yönetimi



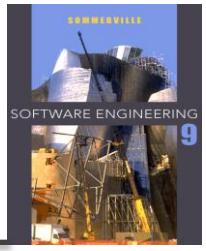
- Konfigürasyon yönetimi, yazılım sistemindeki değişiklikleri yönetme sürecine verilen isimdir.
- Konfigürasyon yönetiminin amacı sistemin entegre edilme sürecini desteklemektir. Bu amaçla, bütün geliştiriciler proje kodlarına ve dokümanlarına kontrollü olarak erişim sağlarlar, ne gibi değişikliklerin yapıldığını görürler ve bileşenleri derleyerek sistemi oluştururlar.

Konfigürasyon Yönetimi Aktiviteleri



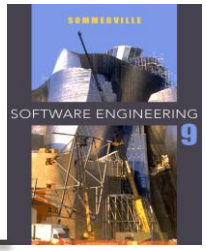
- Sürüm yönetimi, sistem bileşenlerinin farklı sürümlerinin izlerinin tutulması için destek sağlar. Sürüm yönetim sistemleri, farklı programcıları koordine edebilecek şeyler barındırmalıdır.
- Sistem entegrasyonu, sistemin hangi sürümü için bileşenlerin hangi sürümlerinin kullanıldığını tanımlamak için destek sağlar. Bu bilgi daha sonra bir sistemin otomatik olarak derlenmesi ve bağlanması için kullanılır.
- Ör: Microsoft Team Foundation Server
- Problem izleme, kullancıların program hatalarını ve problemlerini rapor edebilmeleri ve geliştiricilerin bu problemler üzerinde kimlerin çalıştıklarını ve ne zaman düzeltildiklerini görebilmeleri için destek sağlar.
- Ör: Mozilla Foundation Bugzilla.

Host (geliştirme ortamı)-target (çalışma ortamı) geliştirme



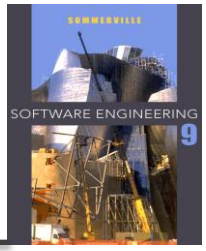
- Çoğu yazılım bir bilgisayarda geliştirilir ancak başka bir bilgisayarda çalışır.
- Daha genel bir ifadeyle, bir geliştirme platrofmu ve çalışma platformundan bahsedebiliriz.
  - Bir platform donanımdan fazlasıdır.
  - Yüklü olan işletim sistemini, veritabanı yönetim sistemi, geliştirme platformları gibi diğer yardımcı yazılımları içerir.

Geliştirme platform araçları



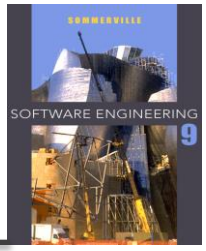
- Kod oluşturmanızı ve derlemenizi sağlayan bütünleşik bir derleyici barındıran ve söz dizimi kontrolü yapan bir editör.
- Bir dil hata ayıklama sistemi.
- UML modelleri gibi modelleri düzenlemeye yarayan bir editör.
- Programın yeni bir sürümü için otomatik olarak test oluşturulabilecek ve çalıştırılabilecek bir test aracı.
- Geliştirilen kodları farklı projeler için organize eden proje destek araçları.

Integrated development environments (IDEs)
Bütünleşik Geliştirme Ortamları



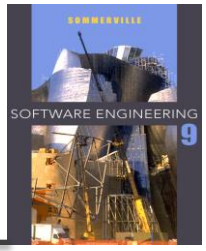
- ✧ Geliştirme araçları genellikle IDE'ler içerisinde toplanır.
- ✧ IDE'ler belirli bir programlama dilini desteklemek için yaratılabileceği gibi genel amaçlı da olabilir.

Açık Kaynak Geliştirme



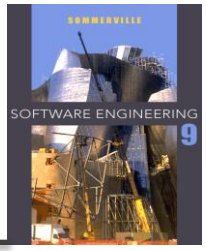
- Açık kaynaklı geliştirme, geliştirilen sistemin kaynak kodlarının yayınlandığı ve gönüllülerin geliştirme sürecine dahil olmaları için davet edildiği bir yaklaşımdır.
- Kökleri Free Software Foundation'a (www.fsf.org) dayanır. Bu vakıf, kaynak kodun sahipliğinin olmaması gerektiğini ve kullanıcılar için açık olması gerektiğini savunur.

Açık kaynaklı sistemler



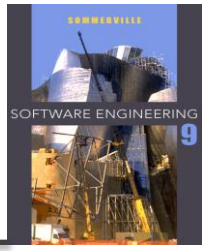
- En iyi bilinen açık kaynaklı ürün Linux işletim sistemidir.
- Diğer önemli açık kaynaklı ürünler Java, Apache web server ve mySQL veritabanı yönetim sistemidir.

Açık kaynaktaki problemler



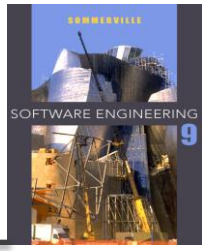
- Geliştirilmekte olan sistemde açık kaynaklı bileşenler kullanılmalı mı?
- Geliştirme süreci için açık kaynak yaklaşımı kullanılmalı mı?

Açık kaynaklı ürün ticareti



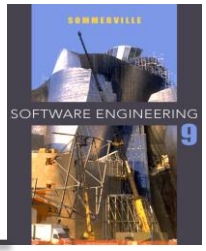
- Daha fazla sayıda firma açık kaynaklı yaklaşımı benimsemekte.
- Bunların iş modeli yazılım satma üzerine kurulmamış, yazılıma destek verme üzerine kurulmuştur.
- Bunlar, açık kaynaklı geliştirme ile yazılımın daha hızlı ve daha ucuza geliştirilebileceğini ve bunun da bir kullanıcı topluluğu yaratacağına inanırlar.

Açık kaynağın lisanslanması



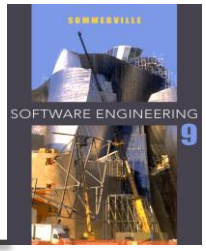
- Açık kaynağın temel prensiplerinden biri herkesin kodlara serbestçe erişebilmesidir ancak bu herkesin her istediğini yapabileceği anlamına gelmez.
  - Hukuken, kodun geliştiricisi kodun sahibidir ve kodun kullanımı ile ilgili kısıtlamalar belirleyebilir.
  - Bir kısım geliştiriciler, eğer bir sistemde açık kaynaklı bir bileşen kullanılmışsa sistemin tamamının açık kaynaklı olması gerektiğine inanırlar.
  - Diğer bir kısım geliştiriciler ise, bu kısıtlama olmadan kodlarının kullanılabilmesine izin verirler. Geliştirilen sistem tamamen kapalı kaynak kodlu olarak satılabilir.

Lisanslama modelleri



- GNU General Public License (GPL). Aynı zamanda “iki taraflı” lisanslama olarak da bilinir. Eğer GPL lisansı altındaki bir açık kaynaklı yazılımı kullanırsanız, geliştirdiğiniz yazılım da açık kaynak kodlu olmak zorundadır.
- GNU Lesser General Public License (LGPL), GPL lisansının bir türüdür. GPL lisansı kullanan açık kaynaklı bileşenlere bağlantı yapan ancak kodlarını yayınlamadığınız bileşenler geliştirebilirsiniz.
- Berkley Standard Distribution (BSD) License. Bu, iki taraflı olmayan bir lisans türüdür. Açık kaynak üzerinde yaptığınız hiçbir değişikliği yayınlamak zorunda değilsinizdir. Bu lisans altındaki kodları, tescillenen ve satılan sistemlerde kullanabilirsiniz.

Lisans yönetimi



- İndirilen ve kullanılan açık kaynaklı bileşenlerin bilgilerini tutmak için bir yöntem kullan.
- Değişik lisanslama tiplerinin farkında ol ve bir bileşeni kullanmadan önce lisansı hakkında bilgi edin.
- Çalışanları açık kaynak hakkında eğit
- Yerinde denetleme sistemine sahip ol.
- Açık kaynak topluluklarına katıl.