Bölüm 7

Tasarım ve Gerçekleştirim (Design and Implementation)

İçerik

- UML ile nesneye dayalı tasarım
- Tasarım kalıpları
- Gerçekleştirim konuları
- Açık kaynak geliştirim

Tasarım ve Gerçekleştirim

- Yazılım tasarımı ve gerçekleştirimi, yazılım mühendisliği sürecinin çalıştırılabilir bir yazılım sistemi geliştirilen adımıdır.
- Yazılım tasarımı ve gerçekleştiriminde yapılan işler dönüşümlü olarak çalışırlar.
- Yazılım tasarımı, öüşteri talepleri doğrultusunda, yazılım bileşenlerini ve ilişkilerini tanımladığınız yaratıcılık gerektiren bir iştir.
- Gerçekleştirim de tasarımı bir programa dönüştürmektir.

Satın mı almalı, geliştirmeli mi?

- Geniş yelpazadeki uygulama alanlarına ait hazır yazılımların sisteme adaptasyonu ve kullanıcı taleplerine göre değiştirilmesi mümkün olmaktadır.
- Örneğin, tıbbi kayıt sistemi kullanacaksanız, hastanelerde kullanılan mevcut sistemlerden birini satın alabilirsiniz. Bu yaklaşımı kullanmak, yeni bir sistem geliştirmekten daha hızlı ve ucuzdur.
- Bir uygulamayı bu şekilde geliştirirken tasarım süreci, sistem gereksinimlerini karşılamak için, yeni sürecin konfigürasyon özelliklerinin nasıl kullanılcağı problemine dönüşür.

Nesneye Dayalı Tasarım Süreci

- Yapısal nesneye dayalı tasarım süreçleri birçok farklı sistem modeli geliştirmeyi kapsar.
- Bu modellerin geliştirim ve bakımı çok fazla çaba gerektirir ve küçük sistemler için pek de karlı olmaz.
- Fakat farklı gruplar tarafından geliştirilen büyük sistemleri için tasarım modelleri önemli bir iletişim mekanizmasıdır.

Süreç Adımları

- Süreci kullanan kuruluşa bağlı olarak, birçok nesneye dayalı tasarım süreç modeli bulunmaktadır.
- Bu süreçlerde yapılan temel işler:
  - Sistem içeriği ve kullanım modlarını tanımlama;
  - Sistem mimarisini tasarlama;
  - Temel sistem nesnelerini tanımlama;
  - Tasarım modelleri geliştirme;
  - Nesne arayüzlerini belirleme.
- Bu sunumda meteoroloji istasyonu için kullanılan süreçleri inceleyeceğiz.

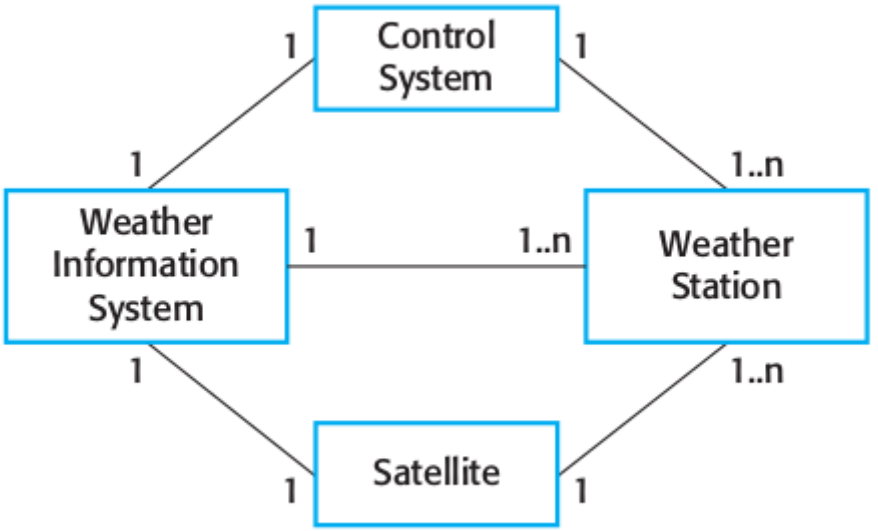
Sistem İçeriği ve Etkileşimleri

- Tasarlanan yazılımın çevresel unsurlarla olan ilişkilerini anlamak, sistemin nasıl çalışacağı ve çevresiyle nasıl iletişime geçeceğini tasarlamak için çok önemlidir.
- İçeriği anlamak aynı zamanda sistemin sınırlarını belirlemeye de yarar. Sistemin sınırlarını belirlemek, hangi özelliklerin sisteme dahil edileceğini ve hangilerinin ilişkili diğer sistemlerde gerçekleştirileceğini belirlememize yardımcı olur.

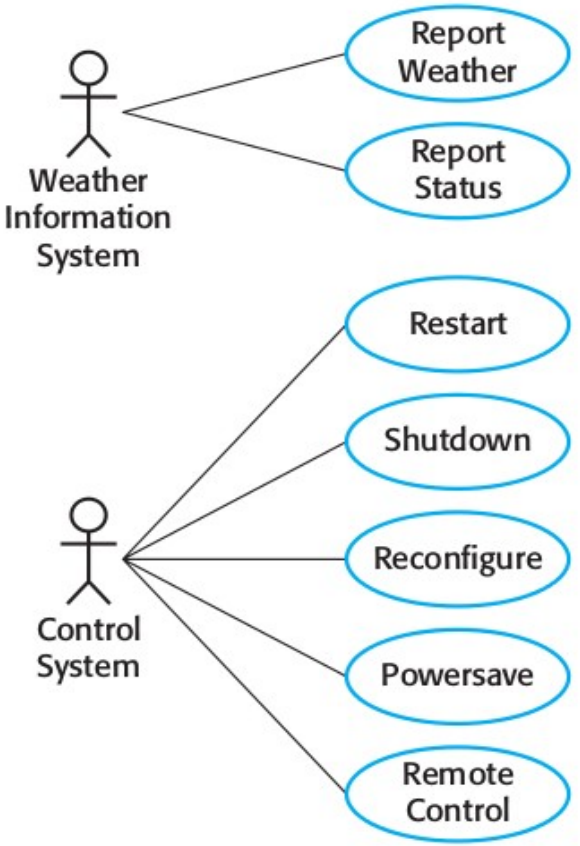
İçerik ve Etkileşim Modelleri

- Sistemin içerik modeli, geliştirilen sistemin etkileşimde olduğu diğer sistemleri göstermeye yarayan yapısal bir modeldir.
- Etkileşim modeli, sistemin kullanıldığında çevresiyle nasıl etkileşime girdiğini gösteren dinamik bir modeldir.

Meteoroloji İstasyonu Sistem İçeriği



Meteoroloji İstasyonu Kullanım Durumu



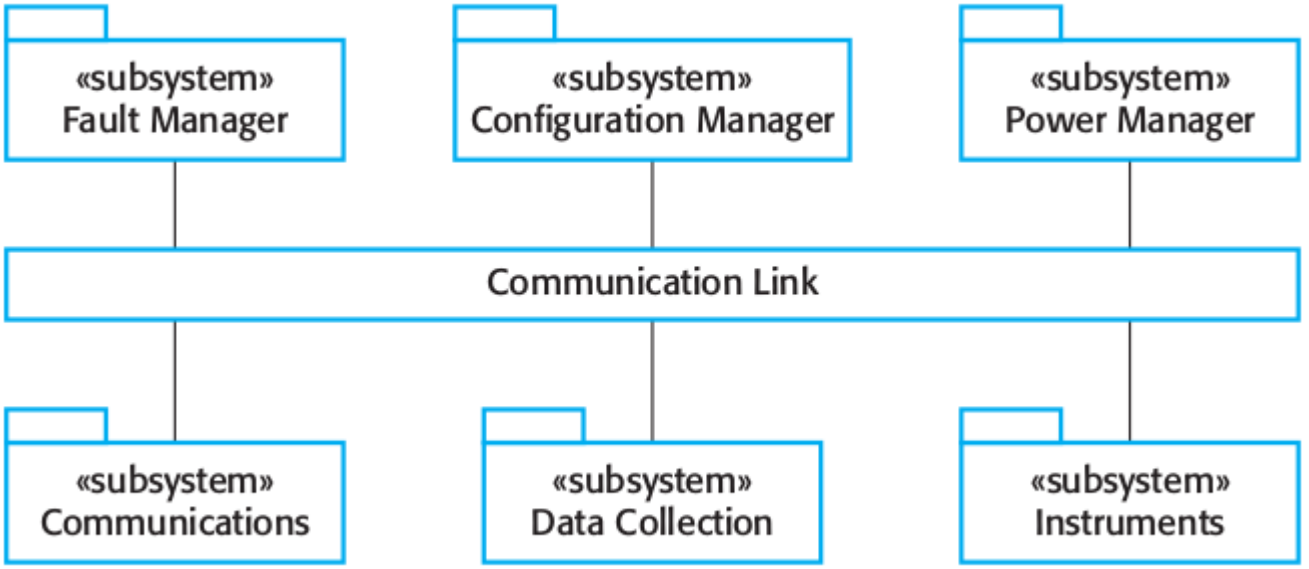
Kullanım Durumu Tanımı – Hava Raporu

Sistem	Meteoroloji İstasyonu
Kullanım Durumu	Hava Raporu
Aktörler	Meteoroloji Bilgi Sistemi, Meteoroloji İstasyonu
Tanım	Meteoroloji istasyonu veri toplama süresince cihazlardan toplanan hava durumu verisinin özetini meteoroloji bilgi sistemine gönderir. Gönderilen veri, her beş dakikalık zaman aralıklarında toplanan, maksimum, minimum ve ortalama yer ve hava sıcaklıkları, maksimum, minimum ve ortalama hava pasıncı, maksimum, minimum ve ortalama rüzgar hızı; toplam yağış ve rüzgar yönüdür.
Uyarıcı	Meteoroloji bilgi sistemi, meteoroloji istasyonu ile bir uydu iletişim hattı kurar ve veri iletimi için talepte bulunur.
Tepki	Özetlenen veri meteoroloji bilgi sistemine gönderilir.
Yorumlar	Meteoroloji istasyonlarından genelde saat başı rapor alınır, fakat bunun sıklığı bir istasyondan diğerine değişiklik gösterebilir ve gelecekte değiştirilebilir.

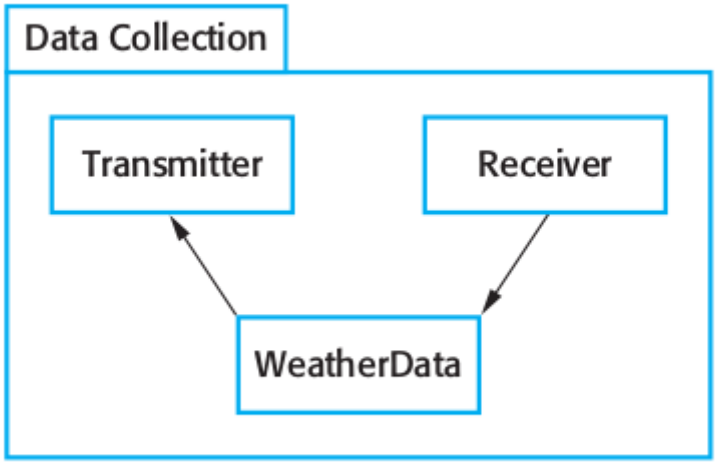
Mimari Tasarım

- Sistem ve çevresi arasındaki ilişkiler anlaşılşdıktan sonra bu bilgi sistem mimarisini tasarlamakta kullanılır.
- Sistemi ve ilişkileri oluşturan temel bileşenleri tanımlayıp sonrasında gerekirse istemci-sunucu veya katmanlı mimari gibi, mimari bir desen kullanarak bileşenleri düzenleyebilirsiniz.
- Meteoroloji istasyonu birbirleri ile mesaj yayımlayarak haberleşen bağımsız alt sistemlerden oluşmaktadır.

Hava İstasyonunun Üst Düzey Mimarisi



Veri Toplama Sisteminin Mimarisi



Nesne Sınıflarını Tanımlama

- Nesne sınıflarını tanımlamak genellikle nesneye dayalı tasarımın zor kısmıdır.
- Nesne tanımlamanın sihirli bir formülü yoktur. Tasarımcıların yetenek, deneyim ve alan bilgisine dayanır.
- Nesne tanımlama döngüsel bir süreçtir. İlk seferinde doğru tasarımı yapmanız pek mümkün değildir.

Tanımlama İçin Yaklaşımlar

- Sistemin doğal dil tasarımını temel alan gramer yaklaşımını kullanın.
- Somut şeyleri temel alın (uçak, doktor, vb.).
- Davranışsal yaklaşım kullanın ve hangi davranışta hangi nesnenin kullanılacağını temel alarak nesne tanımlaması yapın (talep, kayıt, vb).
- Senaryo-tabanlı analiz kullanın. Nesneler, özellikleri ve metodları her senaryo için tanımlanabilir.

Meteoroloji İstasyonu Tanımı

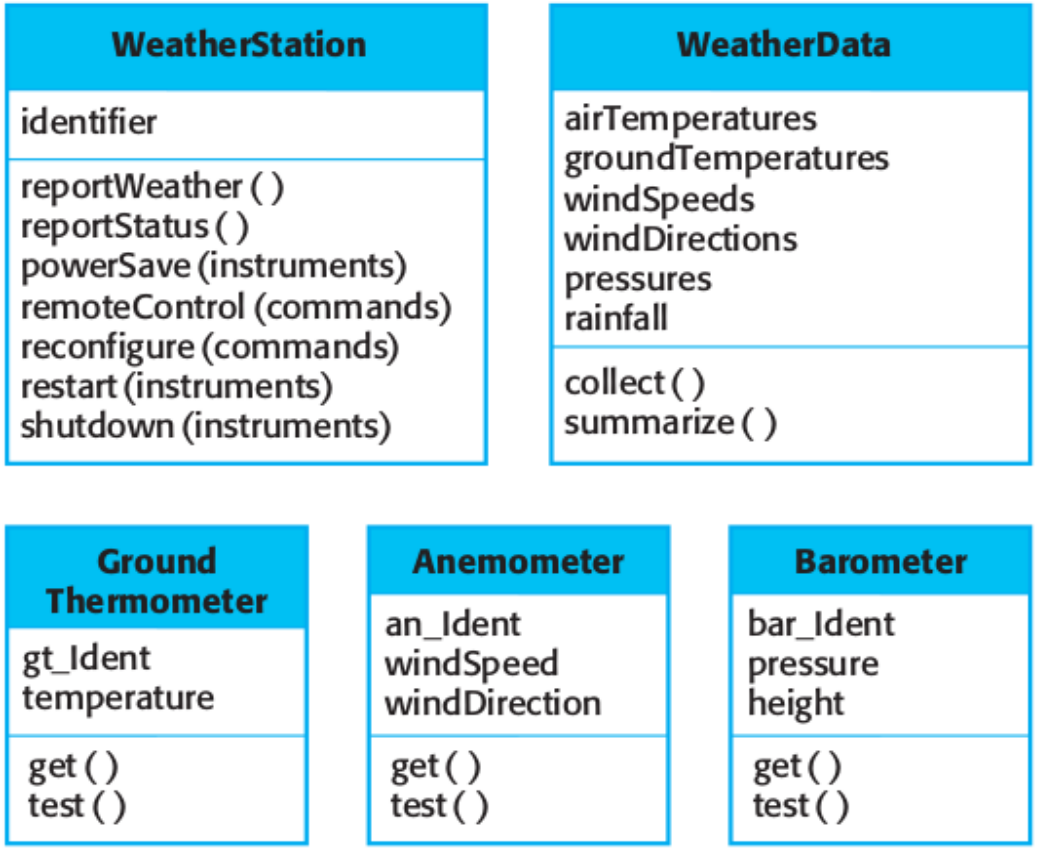
Bir meteoroloji istasyonu, veri toplayan, bir tür veri işleme yapan ve veriyi başka bir yere ileten, yazılım tarafından kontrol edilen ekipmanlardan oluşur. Bu ekipmanlar, hava ve yer sıcaklıkölçeri, basınçölçer, rüzgarölçer, rüzgar gülü ve yağmurölçer gibi cihazlardır. Veri, periyodik olarak toplanmaktadır.

Verinin gönderilmesini söyleyen bir komut geldiğinde, meteoroloji istasyonu topladığı veriyi işler ve özet çıkarır. Özetlenmiş veri ilgili bilgisayara gönderilir.

Meteoroloji İstasyonu Nesne Sınıfları

- Meteoroloji sisteminde nesne sınıfları sistemdeki somut donanım veya veri baz alınarak belirlenebilir:
  - Yer termometresi, rüzgarölçer, basınçölçer
  - Meteoroloji istasyonu
    - İstasyondan çevresine temel bir arayüz. Kullanıcı durumunda belirtilen etkileşimi yansıtır.
  - Meteoroloji verisi
    - Özetlenmiş veriyi ekipmanlardan ayırır.

Meteoroloji İstasyonu Nesne Sınıfları



Tasarım Modelleri

- Tasarım modelleri, nesneleri, nesne sınıflarını ve bileşenler arasındaki ilişkileri gösterir.
- Yapısal modeller nesne sınıfları ve ilişkiler açısından sistemin statik yapısını tanımlar.
- Dinamik modeller nesneler arasındaki dinamik etkileşimi tanımlar.

Tasarım Modellerine Örnekler

- Nesnelerin mantıksal gruplara ayrılarak uyumlu alt sistemler olarak gösterilmesini sağlayan alt sistem modelleri
- Nesne etkileşim sırasını gösteren sıralama modelleri
- Tekil nesnelerin olaylar karşısında nasıl durum değiştirdiğini gösteren durum makinesi modelleri
- Kullanım durumu, birleşme ve genelleştirme gibi diğer modeller

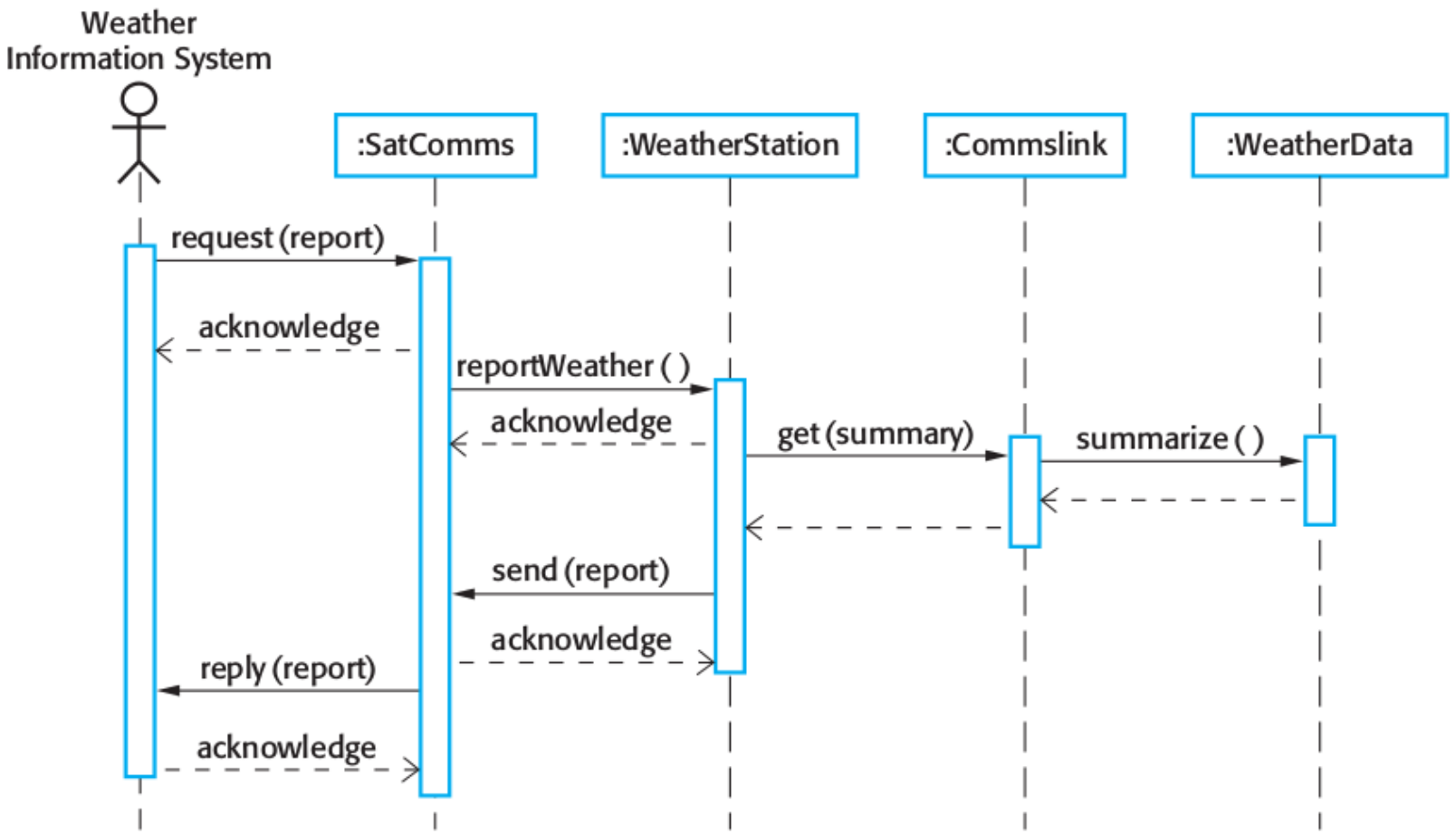
Altsistem Modelleri

- Tasarımın mantıksal olarak ilişkili nesne grupları şeklinde nasıl düzenlendiğini gösterir.
- UML'de, paketler şeklinde gösterilir. Bu mantıksal bir modeldir. Nesnelerin gerçek düzeni farklı olabilir.

Sıralama Modelleri

- Sıralama modelleri oluşan nesne etkileşimlerinin sırasını gösterir
  - Nesneyer en üstte yatay olarak yerleştirilirler;
  - Zaman dikey olarak gösterirlir dolayısıyla okuma yukarıdan aşağıya doğru yapılır;
  - Üzerinde etiketleri bulunan oklarla etkileşimler gösterilir; farklı türde oklar farklı türde etkileşimleri belirtirler;
  - Nesnenin yaşam süresinde bulunan ince dikdörtgen nesnenin sistemde mevcut olduğu zamanı belirtir.

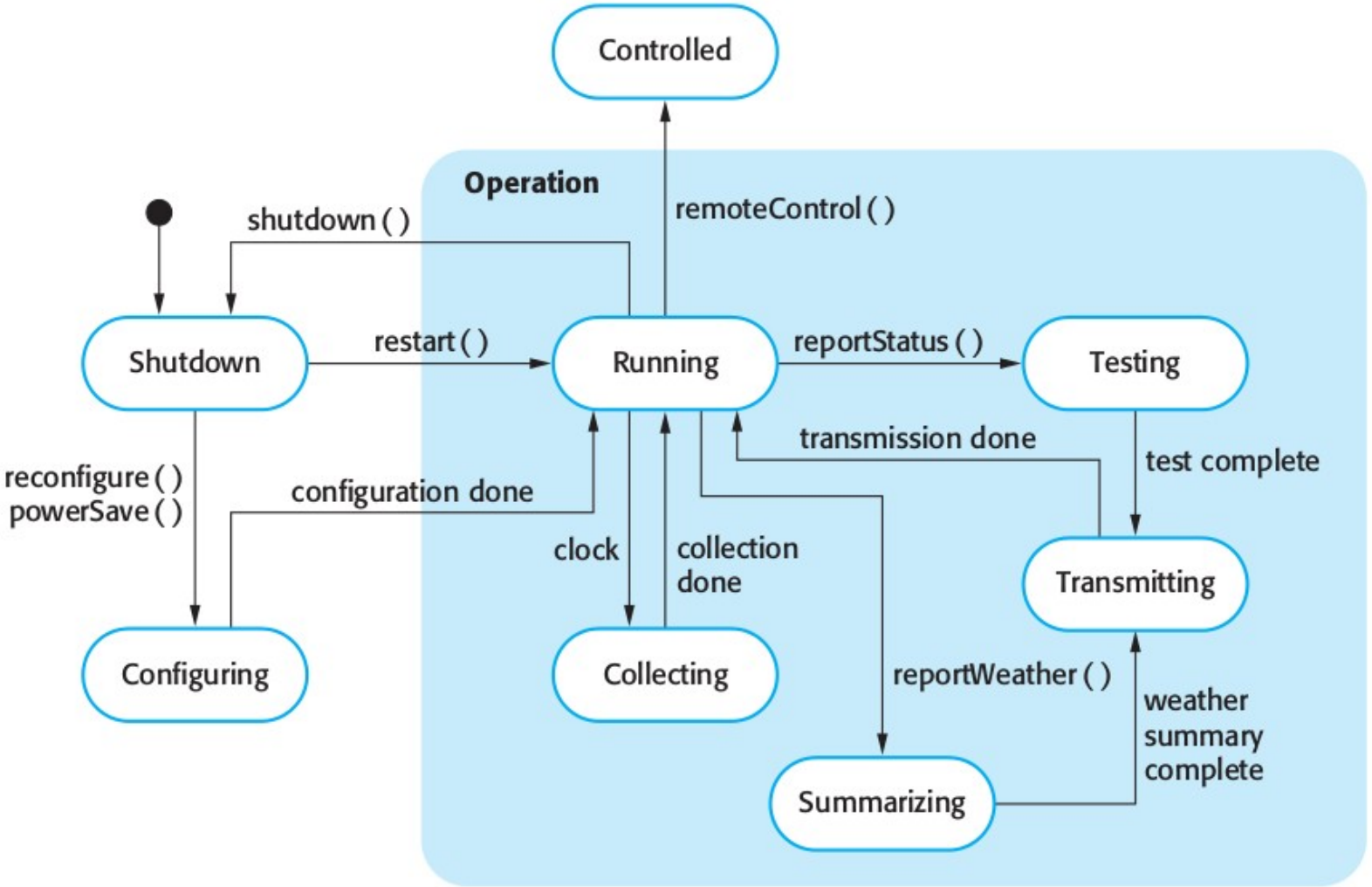
Veri Toplama İçin Sıralama Diyagramı



Durum Diyagramları

- Durum diyagramları nesnelerin farklı servis taleplerine nasıl tepki verdiğini ve bu taleplerin yarattığı durum değişikliklerini göstermek için kullanılır.
- Durum diyagramları sistemin veya nesnenin çalışma zamanı davranışlarını yüksek seviyede göstermek için oldukça kullanışlıdır.
- Sistemdeki her nesne için bir durum diyagramına ihtiyaç duyulmaz. Sistemdeki çoğu nesne basittir ve durum diyagramı kullanılması tasarıma gereksiz detay eklemiş olur.

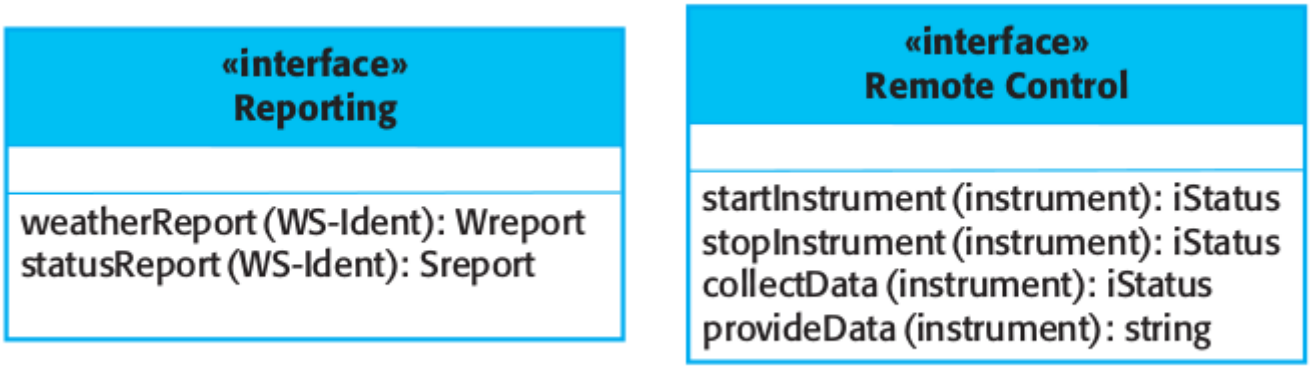
Meteoroloji İstasyonu İçin Durum Diyagramı



Arayüz Tanımlama

- Nesne arayüzleri tanımlanmalıdır böylece nesneler ve diğer bileşenler paralel olarak tasarlanabilir.
- Tasarımcılar arayüz gösterimini tasarlamamalı fakat nesnenin içine gizlemelidir.
- Nesneler birçok arayüze sahip olabilirler.
- UML arayüz tanımı için sınıf diyagramlarını kullanır ancak Java da kullanılabilir.

Meteoroloji İstasyonu Arayüzleri



Önemli Noktalar

- Yazılım tasarım ve gerçekleştirimi dönüşümlü işlerdir. Tasarımdaki detayın seviyesi sistemin ne olduğuna ve çevik ya da plan-güdümlü yaklaşım kullanılmasına göre değişir.
- Nesneye dayalı tasarım süreci, sistem mimarisi tasarımı, sistemdeki nesneleri tanımlama, farklı nesne modelleriyle sistemi tanımlama ve bileşen arayüzlerini belirleme gibi işler içerir.
- Birçok farklı model nesneye dayalı tasarım sürecinde üretilebilir. Bu modeller statik (sınıf modelleri, genelleştirme modelleri, birleşme modelleri) ve dinamik modellerdir (sıralama modelleri, durum makinesi modelleri).
- Bileşen arayüzleri dikkatli ve düzgün tanımlanmalıdır çünkü diğer nesneler onları kullanacaktır.

Tasarım Kalıpları

Tasarım Kalıpları

- Bir tasarım kalıbı, bir problem ve çözümü hakkındaki soyut bilgiyi yeniden kullanma yöntemidir.
- Bir kalıp, bir problemin tanımını ve çözümünün öz tanımıdır.
- Farklı koşullarda kullanılabilecek kadar soyut olmalıdır.
- Kalıp tanımları genellikle miras (inheritance) ve çok biçimlilik (polymorphism) gibi nesneye dayalı özelliklerden faydalanarak yapılır.

Tasarım Kalıbının Bileşenleri

- Ad
  - Kalıp için anlamlı bir tanımlayıcı
- Problem tanımı
- Çözüm tanımı
  - Somut bir tasarım değil, farklı çözümleri temsil eden bir taslak çözüm
- Sonuçlar
  - Kalıbı kullanmanın sonuçları ve zorlukları

Gözlemci (Observer) Kalıbı

- İsim
  - Observer
- Tanım
  - Nesnenin durumunun gösterimini nesnenin kendisinden ayırır.
- Problem tanımı
  - Verinin farklı şekillerde gösterilmesi gerektiğinde kullanılır. Bir nesnede değişiklik olduğunda, değişiklikten haberdar olması gereken, kendisine bağlı diğer nesnelere haber verilmesi için kullanılır
- Çözüm tanımı
  - UML gösterimine bakınız (slayt 37)
- Sonuçlar
  - Gösterimin performansını artırmak üzere yapılan iyileştirmeler anlamsızdır

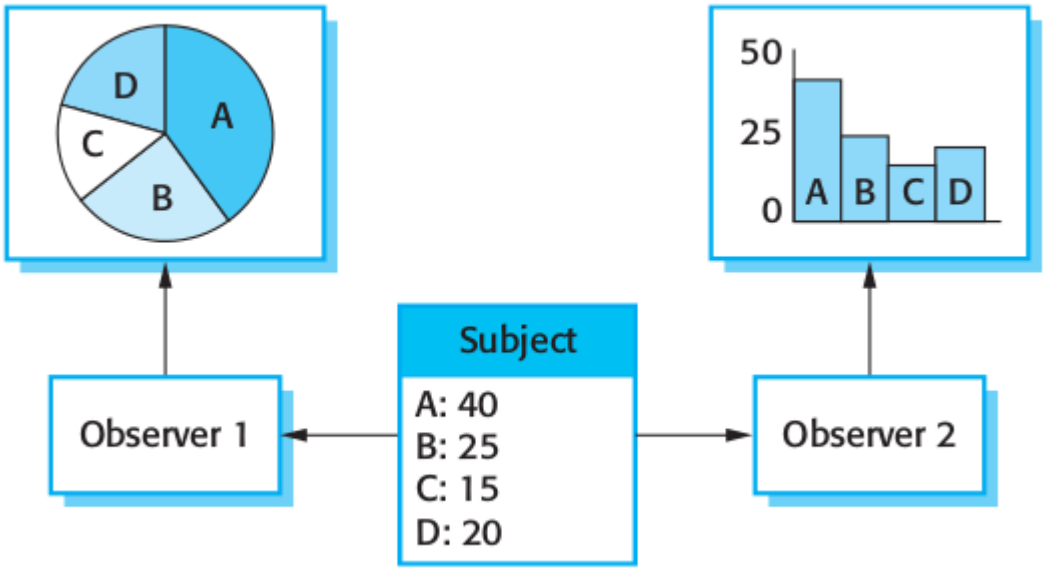
Gözlemci (Observer) Kalıbı - 1

Ad	Gözlemci
Tanım	Bir nesnenin durumunun gösterimini nesnenin kendisinden ayırır ve farklı gösterim şekillerine izin verir. Nesnenin durumu değiştiğinde, tüm gösterimler otomatik olarak bilgilendirilir ve değişimi gösterebilmek için güncellenir.
Problem Tanımı	Çoğu zaman, bir bilginin grafik veya tablo gibi farklı gösterimlerini sağlamak zorunda kalabilirsiniz. Bilgi değiştiği zaman bu gösterim şekilleri de duruma göre değişiklik gösterebilmelidir. Bu kalıp bilginin birden fazla formatta gösterilmesi gerektiği ve nesnenin bu gösterim şekillerini bilmek zorunda olmadığı durumlarda kullanılabilir.
Çözüm Tanımı	İki soyut nesne (Subject ve Observer) ile ilişkili soyut sınıfların özelliklerini miras alan iki somut nesneden oluşur (ConcreteSubject ve ConcreteObject). Soyut nesneler, tüm durumlarda kullanılan genel işlemleri içerirler. Görüntülenecek veri, Observer'ları ekleme ve çıkarma ve durum değişikliği olduğunda bildirim oluşturma gibi işlemlere izin veren Subject sınıfından türemiş ConcreteSubject sınıfında ele alınır. ConcreteObserver, ConcreteSubject'in durumunun bir kopyasını tutar ve bu kopyaların bir adımda tutulmasını sağlayan Observer'ın Update() arayüzünü gerçekleştirir. ConcreteObserver, durumu otomatik olarak görüntüler ve durum değişikliği olduğunda değişimi yansıtır.

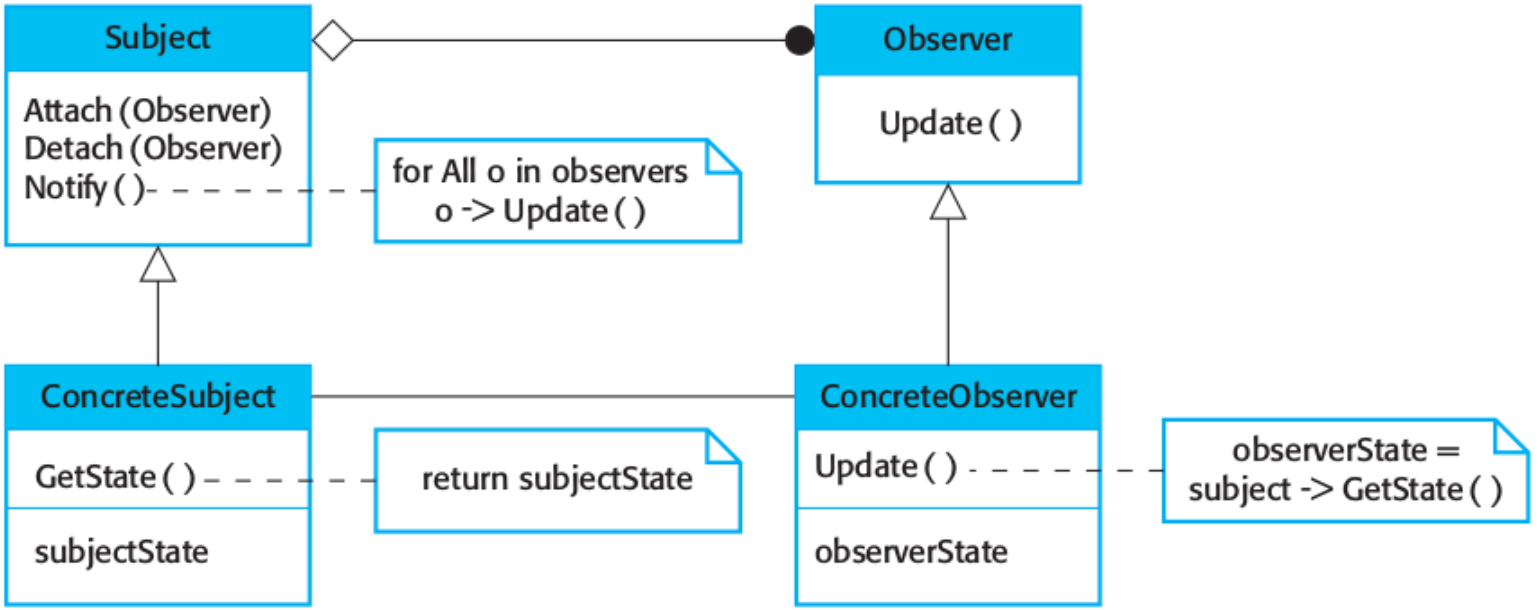
Gözlemci (Observer) Kalıbı - 2

Sonuçlar	Subject sadece soyut Observer'ı bilmektedir ve somut sınıfın detaylarından haberi yoktur. Dolayısıyla, bu nesneler arasında en az ilişki vardır. Bu bilgi eksikliğinden dolayı, gösterimin performansını iyileştirmek için yapılacak iyileştirmeler anlamsızdır. Subject'e yapılan değişiklikler, Observer'larda gereksiz bir dizi ilişkili güncellemeye de sebep olabilirler.

Gözlemci Kalıbıyla Çoklu Gösterimler



Gözlemci Kalıbının UML İle Gösterimi



Tasarım Problemleri

- Tasarımınızda kalıpları kullanmak için, karşılaştığınız tasarım problemine ilişkin geliştirilmiş bir kalıp olup olmadığını farketmeniz gerekir.
- Birçok nesneye bir nesneni durumunun değiştiğinin haber verilmesi (Observer pattern).
- Bir alt sistemin (bir grup nesne) karmaşıklığını ona erişmek isteyen diğer nesnelerden saklamak için bir arayüz oluşturma (Façade pattern).
- Koleksiyonun yapısını bilmeye gerek olmadan, bir koleksiyon içindeki elemanlara standart bir yolla erişme (Iterator pattern).
- Çalışma zamanında bir sınıfın yapısını değiştirmeden nesnelerin davranışını değiştirme (Decorator pattern).

Gerçekleştirim Konuları

- Burada çok önemli olmasına rağmen programlamaya değil, diğer gerçekleştirim konularına odaklanılmaktadır:
  - Yeniden kullanım Çoğu modern yazılım, mevcut bilşeşenler veya sistemleri kullanarak geliştirilir. Bir yazılım geliştirirken mümkün olduğunca varolan kodu kullanmaya çalışmalısınız.
  - Konfigürasyon yönetimi Geliştirim süresi boyunca, her yazılım bileşeninin sürüm bilgisini bir konfigürasyon yönetim aracında tutmak zorundasınız.
  - Konak-hedef geliştirim Üretilen yazılım genellikle geliştirildiği bilgisayarda kullanılmaz. Daha ziyade yazılım bir bilgisayarda geliştirilir ve farklı bir bilgisayarda çalıştırılır.

Yeniden Kullanım

- 1960'lardan 1990'lara kadar çoğu yeni yazılım tüm kodun yüksek seviyeli bir dilde yazılması ile geliştirilmiştir.
  - Kayde değer yeniden kullanım sadece programlama dillerinin kütüphanelerindeki fonksiyonların ve nesnelerin yeniden kullanılmasıydı.
- Maliyet ve zaman baskısı özellikle ticari ve internet tabanlı uygulamalarda bu yöntemin kullanılmasının imkansız olduğu gerçeğini gösterdi.
- Geliştirimde varolan yazılmın yeniden kullanımı yaklaşımı ortaya çıktı ve günümüzde iş uygulamaları ve bilimsel uygulamalarda genellikle kullanılmaktadır.

Yeniden Kullanım Düzeyleri

- Soyutlama düzeyi
  - Yazılımın kendisi doğrudan kullanılmaz, tasarımda daha önce denenmiş ve başarılı olmuş soyutlamalar kullanılır
- Nesne düzeyi
  - Kodu baştan yazmaktansa bir kütüphaneden nesneler doğrudan kullanılır
- Bileşen düzeyi
  - Nesnelerin koleksiyonu olan bileşenler ve nesne sınıfları uygulamada yeniden kullanılır
- Sistem düzeyi
  - Tüm uygulama yeniden kullanılır

Yeniden Kullanım Maliyeti

- Yeniden kullanım için yazılım aramak ve bulunan yazılımın bizim ihtiyaçlarımıza uygun olup olmadığına karar vermek için geçirilen zaman maliyeti
- Eğer uygunsa, yeniden kullanılacak yazılımın satın alma maliyeti. Çok geniş kapsamlı hazır uygulamalar çok yüksek maliyetli olabilmektedir
- Yazılımı kendi geliştirdiğimiz sistemin gereksinimlerine uygun olarak uyarlama ve konfigürasyon maliyeti
- Yeniden kullanılacak yazılım bileşenlerinin yeni kodumuza entegre edilmesinin maliyeti

Konfigürasyon Yönetimi

- Değişen bir yazılım sistemini yönetme işine verilen addır
- Amacı sistem bütünlüğünü sağlamaktır; böylece tüm geliştiriciler koda kontrollü bir şekilde erişebilir, yapılan değişiklikleri görebilir, derleme yapabilir

Konfigürasyon Yönetim İşleri

- Version management, where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.
- System integration, where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
- Problem tracking, where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.

Konak- Hedef Geliştirim

- Çoğu yazılım bir bilgisayarda geliştirilir fakat farklı bir makinede çalıştırılır
- Daha genel olarak, geliştirim ortamı ve çalıştırma ortamından söz edebiliriz
  - Ortam sadece donanımdan ibaret değildir
  - İşletim sistemi, veritabanı yönetim sistemi gibi yardımcı yazılımlar veya geliştirme ortamları için IDE
- Geliştirme ortamı çalıştırma ortamından farklı yazılımlara sahiptir; bu ortamlar farklı mimaride olabilirler

Geliştirim Platformu Araçları

- Derleyici ve kodu yaratıp, düzenlemek için kullanılan metin işleme sistemi
- Hata ayıklama sistemi
- UML modellerini düzenlemeye yarayan görsel düzenleme araçları
- Test araçları, örneğin Junit
- Farklı geliştirim projeleri için kodu düzenlemeye yarayan Proje destek araçları

Birleşik Geliştirim Ortamları (IDE)

- Bir IDE, yazılım geliştirmenin farklı yönlerini desteklemek için kullanılan bir yazılım aracıdır
- IDE'ler geliştirimi Java gibi belirli bir dilde yapmayı desteklemek için üretilmişlerdir. IDE sadece belirli bir dile özel de üretilmiş olabilir, belirli dil-destek araçları ile genel-amaçlı da olabilir

Bileşen/Sistem Dağıtım Faktörleri

- Eğer bir bileşen belirli bir donanım mimarisi için tasarlandıysa veya başka yazlım sistemlerine dayanıyorsa, gerekli donanım ve yazılım desteğini sağlayan bir ortamda çalıştırılmalıdır
- Uygunluğu yüksek sistemler bileşenlerinin birden fazla ortama yüklenmesi gerekebilir. Bunun anlamı, bir ortamda hata oluştuğunda o bileşenin alternetif ortamda çalışabilmesidir
- Bileşenler arasında yoğun bir iletişim trafiği varsa, onları aynı ortamda veya fiziksel olarak birbirine yakın ortamlarda çalıştırmak mantıklıdır. Bu mesaj gönderme ve alma zamanını azaltır

Açık Kaynaklı Geliştirim

- Açık kaynaklı geliştirim, yazılımın kaynak kodunun yayımlanması ve gönüllülerin geliştirim sürecinde yer almaları için davet edilmelerine dayanan bir yazılım geliştirme yaklaşımıdır.
- Kökleri Free Software Foundation (www.fsf.org)' e dayanır. Bu kurum, kaynak kodun birilerinin malı olmaması, aksine her daim kullanıcıların incelemelerine ve istedikleri gibi değiştirebilmelerine açık olması gerektiğini savunur.
- Açık kaynaklı yazılım bu fikri Internet üzerinden geniş sayıda gönüllü yazılımcıyı işe alarak genişletmiştir. Çoğu geliştirici kodun aynı zamanda kullanıcısıdır.

Açık Kaynaklı Sistemler

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

Açık Kaynak Konuları

- Geliştirilen ürün açık kaynaklı bileşenler kullanmalı mıdır?
- Yazılım geliştirme için açık kaynak yaklaşımı kullanılmalı mıdır?

İş Dünyasında Açık Kaynak

- Her gün daha fazla sayıda firma açık kaynaklı geliştirme yaklaşımını kullanmaktadır.
- Onların iş modeli yazılım ürünü satma üzerine değil ürünlere destek satma üzerinedir.
- Açık kaynak topluluğuna katılmanın yazılım geliştirme işinin daha ucuz, daha hızlı yapılacağına ve o yazılım için bir kullanıcı topluluğu yaratacağına inanırlar.

Açık Kaynak Lisanslama

- Açık kaynaklı geliştirimin temel ilkesi olan kaynak kodun açık olması, her isteyenin koda istediği şeyi yapabileceği anlamına gelmemektedir.
- Yasal olarak, kodun geliştiricisi (kişi veya kurum) hala kodun sahibidir. Açık kaynak yazılım lisansı ile kodun nasıl kullanılabileceğine dair kısıtlar ve yasal olarak bağlayıcı maddeler koyabilirler.
- Bazı açık kaynak geliştiriciler bir yazılım içinde açık kaynaklı bir bileşenin kullanılmasının o yazılımın da açık kaynaklı olmasını gerektirdiğini düşünürler.
- Diğerleri kendi kodlarının bu sınırlama olmadan kullanılabilmesini isterler. Geliştirilen sistemler tescilli olabilir ve kaynak kodu kapalı sistemler gibi satılabilir.

Lisans Modelleri

- The GNU General Public License (GPL). ‘iki taraflı’ lisans. GPL ile lisanslanmış bir yazılımı kullanıyorsanız, yazılımınızın da açık kaynaklı olması şarttır.
- The GNU Lesser General Public License (LGPL). GPL'nin bir türevidir. Kodunu açmak zorunda olmadan bazı açık kaynaklı koda bağlı bileşenler yazabilirsiniz.
- The Berkley Standard Distribution (BSD) License. Açık kaynaklı kodu alıp kullanabilir, değiştirebilir ve yazılımınızı tescilletebilir ve satabilirsiniz.

Önemli Noktalar

- Yazılım geliştirirken öncelikle varolan bir yazılımı kullanma olasılığını değerlendirmelisiniz.
- Konfigürasyon yönetimi evrilen bir yazılımın değişikliklerini kontrol etme işidir. Eğer yazılım bir takım tarafından geliştiriliyosa mutlaka kullanılmalıdır.
- Çoğu yazılım geliştirme konak-hedef'tir. Konak makinede bir IDE kullanarak yazılımı geliştirir ve çalıştırmak için başka bir makineye aktarırsınız.

Haftaya görüşmek üzere