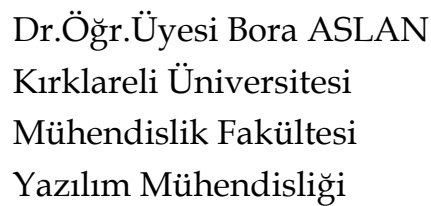


11/11/2019



# Yazılım Mühendisliğine Giriş

## UML

### Yazılım Gereksinimlerinin Analizi (Software Requirements Analysis)

Yazılım geliştirme sürecinin başarısı için kullanıcı gereksinimlerinin ve yazılım isterlerinin çok iyi anlaşılması gerekir. Bir yazılım ne kadar iyi tasarlanmış ve geliştirilmiş olursa olsun, kullanıcının isteklerini karşılayabildiği kadar başarılı olmuş sayılır. Bu sebeple tüm ürün geliştirme süreçlerinde olduğu gibi yazılım geliştirme aşamasının başında müşterinin/kullanıcının üründen beklentileri belirlenir, gereksinimler açığa çıkarılır, yazılım isterleri modellenir ve sonraki aşamalara temel olacak şekilde dokümente edilir.

Yazılım İsterleri Belirtimi belgesi oluşturulurken farklı türlerdeki isterleri tanımlamak için değişik teknikler kullanılır. Bu teknikler ile üretilen diyagramlar yazılımın gerçekleşmesi yolunda yazılımcı için oldukça aydınlatıcı olmaktadır.

- Kullanım/Kullanıcı Senaryoları (Use Cases)
- Veri Akış Diyagramları (Data Flow Diagrams)
- Varlık İlişki Diyagramları (Entity-Relationship Diagrams)
- Veri Sözlüğü (Data Dictionary)
- Nesne Diyagramları (Object, Class Diagrams)
- Aktivite Diyagramları (Activity Diagrams)
- Etkileşim Diyagramları (Interaction Diagrams)
- Durum Haritası/Şeması Diyagramları (Statechart Diagrams)

### Model Nedir Ve Neden Modelleriz?

Model, gerçeğin basitleştirilmiş halidir. Yani, karmaşık bir sistemi modelleyerek onu daha basit bir dille ifade edebiliriz; böylece geliştirmekte olduğumuz sistemi daha iyi anlayabilir ve olası hatalarımızı uygulamaya geçirmeden görebiliriz.

Aslında, kendimiz için küçük bir kulübe yapmak istersek birkaç ağaç, biraz da saman yeterli olacak ve sonunda işe yarar bir yapı çıkacaktır. Ufak tefek hatalar olsa da bu çok sorun olmayacaktır, çünkü bu sorunu başlangıçtaki isteklerimizden biraz kısararak, ya da işe en baştan başlayarak çözebiliriz. Fakat iş büyük çapta bir bina yapımına geldiğinde, o bina için mimari plan çıkarmak zorunlu hale gelir. Artık işe en baştan başlama gibi bir olanağımız yoktur, istekler biz binanın yapımına başladığımızdan sonra bile değişebilir veya müşteri yeni isteklerle gelebilir. Bu bina yapımına da kulübe gibi başlayabiliriz tabi. Ama bu işi tek başımıza yapamayacağımıza göre, binayı istenen şekilde, işe yarar bir biçimde bitirebilmek için doğru zamanda, doğru yerde

olan ve doğru bilgilere sahip çalışma arkadaşlarımız olmalı. Ancak, gerçek hayatta böyle çalışma arkadaşları pek fazla bulunmuyor.

Büyük ve karmaşık yazılım projeleri için de durum çok farklı değildir. Başarısız projelerin hepsi değişik sebeplerden ötürü, kendilerine has bir biçimde çökerken, başarılı kurumların sahip olduğu en önemli özelliklerden birisi doğru modellemenin kullanılmasıdır. Modelleme yapılırken şu 4 ilke dikkate alınmalıdır,

- **Modeller iyi seçilmeli**, doğru modeller en karmaşık problemleri çözmede yardımcı olabileceği gibi, yanlış model seçimi sizi yanıltabilir ve ilgisiz konulara odaklanmanıza neden olabilir.
- **Farklı detay seviyelerine sahip modelleriniz olmalı**, bazen binanıza üstten bakarak genel yapısını incelemeniz gerekebilir, bazen de bir odanın yer döşemesiyle ilgilenirsiniz.
- **En iyi modeller gerçekte bağlantılı olanlardır**, binanız için tasarladığınız fiziksel bir model gerçek hayatta olması gerektiği gibi davranmıyorsa, o modelin pek de değeri yoktur.
- **Yalnız bir model hiçbir zaman yeterli değildir**, örneğin bir bina yapıyorsanız, zemin planlarının yanında elektrik, ısınma ve su gereksinimleri için de planlar oluşturmalsınız

### UML ve Diyagramları

UML 1997'den bu yana 1.1, 1.3, 1.4 ve 1.5 gibi versiyonların ardından 2005'te çıkan 2.0 versiyonu ile birçok yönden geliştirilmiş, dildeki eksikler tamamlanmış ve hatalar ortadan kaldırılmıştır. Son versiyonu UML 2.1.2, 2007 Kasım ayında çıkmıştır.

Grafiksel bir dil olan UML, modelleme için değişik diyagramlar kullanır. Diyagramlar, bir sistem modelini kısmen tarif eden grafiklerdir. UML diyagramları bir sistem modelini 3 farklı açıdan ele alırlar. Modelin,

- **İşlevsel gereksinimler açısından**, kullanıcının bakış açısından sistemin gereksinimleri vurgulanır. Kullanım Senaryosu (Use-Case) diyagramını içerir.
- **Statik yapısal açısından**, nesneler, nesnelere ait özellikler ve ilişkiler kullanılarak sistemin statik yapısı incelenir. Sınıf (Class) ve Birleşik Yapı (Composite Structure) diyagramlarını içerir.
- **Dinamik davranış açısından**, nesneler arası ortak çalışmalar ve nesnelerin durumlarındaki değişiklikler gösterilerek sistemin dinamik davranışı incelenir. Sıralama (Sequence), Faaliyet (Activity) ve Durum (Statechart) diyagramlarını içerir.

UML 2.0, 3 ana bölüme ayırabileceğimiz 13 çeşit diyagram içerir. Yapısal diyagramlarda modellenen sistemde nelerin var olması gerektiği vurgulanır. Davranış diyagramlarında modellenen sistemde nelerin meydana gelmesi gerektiğini belirtir. Davranış diyagramlarının bir alt kümesi olan Etkileşim diyagramlarında ise modellenen sistemdeki elemanlar arasındaki veri ve komut akışı gösterilir.

## Yapısal Diyagramlar

- **Sınıf (Class) diyagramı**, sistemin yapısını anlatmak için sistemde var olan sınıfları, sınıfların özelliklerini ve sınıflar arası ilişkileri kullanır. Nesneye yönelik sistemleri modellemede kullanılan en yaygın diyagramdır.
- **Nesne (Object) diyagramı**, modellenen sistemin yapısının belirli bir andaki bütün ya da kısmi görünüşü tarif edilir.
- **Bileşen (Component) diyagramı**, bir yazılım sisteminin hangi tür bileşenlere ayrıldığını ve bu bileşenlerin nasıl birbiriyle ilişkili olduğunu betimler. Bir bileşen genellikle bir veya birden fazla sınıf, arayüz ve iletişime karşılık gelir.
- **Paket (Package) diyagramı**, bir sistemin hangi mantıksal gruplara bölündüğünü ve bu gruplar arasındaki bağımlılıkları betimler.
- **Dağılım (Deployment) diyagramı**, sistemde kullanılan donanımları, bu donanımların içinde yer alan bileşenleri ve bu bileşenlerin arasındaki bağlantıları gösterir.
- **Birleşik Yapı (Composite Structure) diyagramı**, bir sınıfın iç yapısını ve bu yapının mümkün kıldığı iletişimlerini tarif eder.

## Davranış Diyagramları

- **Kullanım Senaryosu (Use-Case) diyagramı**, modellenen sistem tarafından sağlanan işlevselliği sistemde yer alan aktörleri, aktörlerin sahip olduğu kullanım senaryolarını ve bu senaryolar arasındaki bağımlılıkları göstererek açıklar.
- **Durum (Statechart) diyagramı**, bilgisayar programlarından iş süreçlerine kadar birçok sistemi tarif eden standartlaşmış bir gösterimdir. Durumlar, geçişler, olaylar ve faaliyetler gösterilir.
- **Faaliyet (Activity) diyagramı**, modellenen sistemdeki iş akışını adım adım gösterir. Faaliyet diyagramı kapsamlı bir komut akışını tarif eder. Faaliyetler arası akışı gösteren Durum diyagramıdır.

## Etkileşim Diyagramları

- **Sıralama (Sequence) diyagramı**, nesnelerin birbiriyle nasıl iletişim sağladıklarını sıralı iletiler şeklinde gösterir. Ayrıca nesnelerin yaşam süreleri de gösterilir.
- **İletişim (Communication) diyagramı**, nesneler ve parçalar arasındaki etkileşimi sıralı iletiler olarak gösterir. Sınıf, Sıralama ve Kullanım Senaryoları diyagramlarındaki bilgileri kullanarak sistemin hem statik yapısını hem de dinamik davranışını gösterir.
- **Etkileşime Bakış (Interaction Overview) diyagramı**, farklı etkileşim diyagramları kullanarak, bunlar arasındaki komut akışını gösterir. Bir başka deyişle, elemanları etkileşim diyagramları olan faaliyet diyagramlarıdır.
- **Zaman Akış (Timing) diyagramı**, odağın zaman kısıtlamaları olduğu etkileşim diyagramıdır.

## UML Ne Kazandırır?

Modelleme ihtiyacımızdan kısaca bahsettikten sonra, bu ihtiyacı karşılamak üzere UML'nin nasıl ortaya çıktığından bahsettik. UML'nin modellemeye bakış açısını ve UML diyagramlarının neler

olduğunu gördüğümüze göre, artık büyük bir bina yapımına kulübe yapar gibi başlamayacağız. Peki UML kullanarak yapacağımız sistem modellemesi bize ne kazandırır?

- Takım çalışmasında yardımcı olur, UML standartlaşmış uluslararası bir dildir ve bu dili bilen herkes diyagramlardan aynı şeyleri anlar. Müşteri ve teknik sorumlular diyagramlar üzerinden rahatça iletişim kurabilirler. Ekibinizde yer alan çalışma arkadaşlarınızla uyumlu bir şekilde çalışabilirsiniz ve ekibe yeni giren bir çalışan da projeye rahatlıkla dahil edilebilir.
- Kodlamayı kolaylaştırır, UML ile uygulamanızın tasarımı analiz aşamasında yapıldığı için, modellemeniz bittikten hemen sonra kod yazmaya başlayabilirsiniz.
- Hataları en aza indirir, UML ile bütün sistem tasarlandığı için sistemde hata çıkma olasılığı azdır. Çıkan hataları düzeltmek ise çok daha kolaydır.
- Tekrar kullanılabilir bileşenleriniz artar, UML ile tüm sistem ve sistemin bileşenleri daha baştan belirlendiği için, o bölümler tekrar tekrar yazılmayacaktır.
- Program kararlılığı artar, UML ile ayrıntılı gereksinim analizleri yapıldıktan sonra senaryolar belirlenir. Senaryoların baştan belirli olması programınızı daha kararlı hale getirmenizde size yardımcı olur.

## Kullanım/Kullanıcı Senaryoları (Use Cases)

Use Case, sistemlerin işlevsel gereksinimlerini kapsamak için yazılım ve sistem mühendisliğinde kullanılan bir tekniktir. Aktörlerin (son kullanıcıların veya diğer sistemlerin) sistemle etkileşimini tanımlayan senaryolardır.

Bu analizin temel taşı olan use case, en basit şekliyle ifade etmek gerekirse, herhangi bir sistemin nasıl davrandığını ve çalıştığını gösterir. Use case'ler, bir kullanıcının spesifik bir amaca ulaşabilmek için yazılımsal bir sistem içerisinde atması gereken adımları tanımlamak için kullanılırlar. Söz konusu amacı gerçekleştirmeye yönelik tüm olası senaryoların bir arada yer aldığı bir belge olarak da düşünülebilirler.

Use Case'lerde, son kullanıcının veya alan uzmanının teknik terimlerden arınmış dili tercih edilir. İş analisti (çözümleyici) ve son kullanıcılar Use Case'lerin yazılmasında çoğunlukla birlikte çalışırlar.

1986 yılında Ivar Jacobson Use Case'leri belirtmede kullanılan görsel modelleme tekniğini ilk kez kodlamıştır. Önceleri İngilizcede usage scenarios ve usage case terimlerini kullanmış; ancak doğal kullanıma uymadıklarını düşünerek use case terimi üzerinde nihai kararını vermiştir. Türkçede ise doğal kullanıma uyduğu için usage scenarios teriminin birebir çevirisi olarak kullanım senaryoları kullanılmaktadır. Aralarında Kurt Bittner, Alistair Cockburn ve Gunnar Overgaard gibi isimlerinde bulunduğu birçok kişi, Jacobson'un başlattığı bu tekniği daha da geliştiren çeşitli katkılarda bulunmuşlardır.

1990'larda Use Case'ler, işlevsel gereksinimlerin çıkarılmasında kullanılan en yaygın uygulamalardan biri haline gelmiştir. Use Case'ler nesne yönelimli olmadıkları için doğduğu nesne yönelimli çevrelerin dışında da uygulama alanı bulmaktadır.

Use Case'ler, bir amacın veya görevin nasıl başarılacağını tanımlamaya odaklanır. Yazılım projelerinde yeni bir sistem konusunu kapsamak için Use Case'lere (bazen düzinelerce) ihtiyaç duyulur. Projenin niteliği ve durumu, Use Case'lerin ne kadar detaylandırılacağını belirler.

Use Case, harici aktörlerle sistem arasındaki etkileşimi tanımlar. Varlığın veya kişinin sistemle etkileşiminde içinde bulunduğu role aktör denir. Sistemle etkileşim içinde olan bir kişi, farklı rolleri icra ettiği için iki farklı aktörü temsil edebilir. Örneğin, kişi ATM'yi kullandığında Müşteri rolünde; ATM'nin boş para çekmecelerini doldurduğunda bir Banka Memuru rolünü oynayabilir.

Use Case'leri, sistemi bir kara kutu olarak ele alır. Sistem cevaplarını içeren sistem etkileşimleri, sistemin dışında gibi algılanır. Böylece analist, sistemin nasıl davranacağından çok ne yapması gerektiği üzerinde yoğunlaşabilir.

Use Case'lerin sağladığı kazanımlar aşağıda verilmiştir:

- Sistemin erimini, sınırlarını belirler.
- Geliştirilecek sistemin boyutu ve karmaşıklığı daha rahat canlandırılabilir.
- Use Case'ler isteklerin daha nettir ve tam olarak ifadesidir.
- Basit oluşu müşteri ile geliştirme ekibi arasında iletişime olanak tanır.
- Geliştirme aşaması için temel oluşturur.
- Sistem testi için temel oluşturur.
- Kullanıcı kılavuzu hazırlamaya yardımcı olur.

Use case analizi ise, yazılım mimarlarının ve analistlerin, geliştirilecek yazılımın gereksinimlerini elde etmek ve düzenlemek için kullandıkları bir yazılım modelleme tekniğidir. Bu analizin öncelikli amacı; sistemi kullanıcı perspektifinden tasarlamak, sistemsel davranışları bizzat kullanıcılarının diliyle ifade etmek ve olası tüm sistemsel davranışları high level seviyesinde ortaya koymaktır. Bu şekilde sistemin fonksiyonel gereksinimlerini, sistemin ne şekilde kullanılacağını, kullanıcıların sistem içerisinde bürüneceği rolleri, kullanıcıların etkilerine bağlı olarak sistemin ne gibi tepkiler vereceğini, müşteri yada kullanıcıların sistemden hangi faydaları elde edeceğini açık bir şekilde ortaya konulabilir.

### ***Use Case Diyagramı***

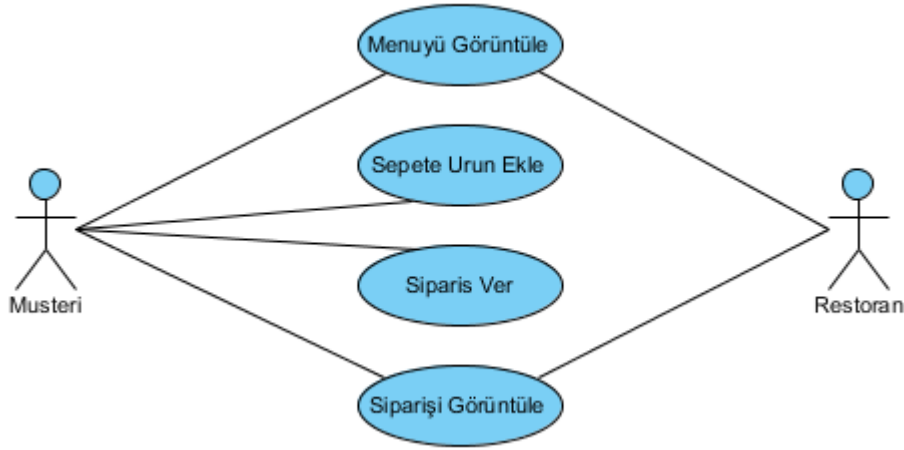
Use case diyagramı, bir uygulamanın yada sistemin kimler tarafından kullanıldığını ve söz konusu kullanıcıların bu uygulama yada sistemle neler yapabildiklerini özetlemek amacıyla kullanılan bir UML (Unified Modeling Language) diyagramıdır. Başka bir deyişle, bu diyagramın



amacı, sistemin hangi fonksiyonlarının hangi aktörler tarafından gerçekleştirileceğini ve use case'ler arasındaki ilişkileri göstermektir.

Sistemsal gereksinimler analiz edildikten sonra elde edilen fonksiyonlar use case'ler şeklinde ifade edilir. Aslında, use case'ler fonksiyonel gereksinimlerin belirli bir formata bağlı kalınarak yazılmasından başka birşey değildir. Use case'ler sistemin fonksiyonel olmayan gereksinimleriyle ilgili bilgi içermezler.

Use case'ler ve aktörler belli olduktan sonra bu ikisi birbiriyle ilişkilendirilerek use case diyagramı oluşturulur. Bir use case'le aktör arasındaki ilişki, sözkonusu aktörle use case'i birbirine bağlayan bir çizgiyle ifade edilir. Böylece hangi aktörün sistem üzerinde hangi fonksiyonu çalıştırabildiğini göstermiş oluruz. Diyagramda yer alan aktörler en az bir use case'le ilişkili olmak zorundadır. Şimdi bu söylediklerimizi daha somut bir şekilde ifade edebilmek adına aşağıdaki örneği birlikte inceleyelim:

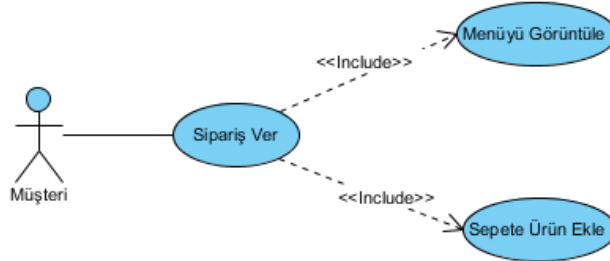


Yukarıdaki resimde, yerel restoranların online yemek satışı yapabildiği bir web sitesine ilişkin bir use case diyagramı görülüyor. Aktörler, sistemle bir şekilde etkileşime giren kullanıcı, organizasyon, cihaz yada harici yazılım sınıflarıdır demiştik. Bu diyagramdaki aktörler, sistemi kullanarak yemek siparişi vermek isteyen müşteriler ve siparişi alıp müşteriye ulaştıracak olan restoranlardır. Yatay elipslerle resmedilen use case'ler ise, bir yada birkaç aktör tarafından, belli bir amaca yönelik olarak gerçekleştirilen işlemleri ifade etmektedir. Şekilden de anlaşılabileceği gibi, müşteriler menü görüntüleme, sepete ürün ekleme ve sipariş verme işlemlerini gerçekleştirebiliyorken, restoranlar sadece menü görüntüleme ve siparişi görüntüleme işlemlerini gerçekleştirebilmektedir.

### ***Include İlişkisi***

Bazı durumlarda bir use case'in tamamlanabilmesi için, başka use case'lerin işin içerisine katılması zorunludur. Yani söz konusu use case, bir takım başka use case'ler olmaksızın tamamlanamaz. Örnek olarak müşterinin, web sitesi üzerinden sipariş verebilmek için önce sipariş vermek istediği restoranı seçip, menüyü görüntülemeli ve daha sonra sipariş etmek

istediği ürünü / ürünleri alışveriş sepetine eklemelidir. Yani, “Sipariş Ver” isimli use case’in tamamlanabilmesi, “Menüyü Görüntüle” ve “Sepete Ürün Ekle” isimli use case’lerin çalıştırılmasına bağlıdır. Include olarak adlandırdığımız bu ilişki türü, use case diyagramlarında aşağıdaki resimde aktarıldığı şekilde ifade edilmektedir. Burada okların yönü, hangi use case’in tamamlanabilmek için diğer hangi use case’lere bağımlı olduğunu gösterir.



### Extend (Genişletme) İlişkisi

Use case’ler arasındaki diğer bir ilişki türü extend olarak adlandırdığımız ilişki türüdür. Bu ilişki türü, iki use case arasında opsiyonel olarak var olabilecek bir durumu ifade etmek için kullanılır. Bu iki use case “base use case” ve “extending use case” olarak adlandırılır.

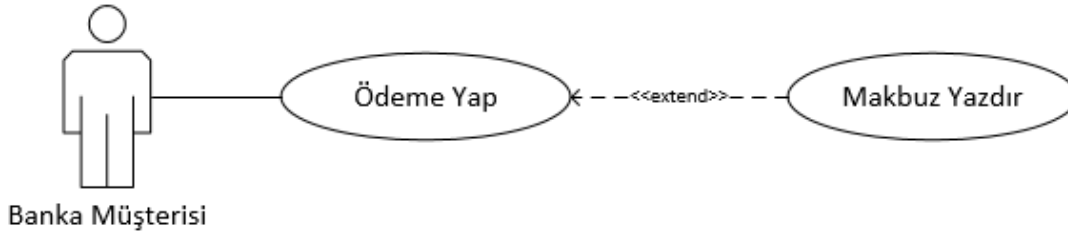
Bu ilişki, bir use case’in kapsamının başka bir use case tarafından genişletilebilmesinin olası olduğu durumlar için geçerlidir. Elimizde bir use case ve bu use case’in kapsamını oluşturan bir dizi adım olduğunu varsayarsak; sürecin bazı noktalarında, başka bir use case’in kapsamını oluşturan adımların kullanılması suretiyle, use case’in adımlarına yeni adımlar eklenmesi ve kapsamının genişletilmesi mümkün olabilir.



Örneğin müşteri siparişini tamamladıktan sonra bir hediye gönderilmesi isteniyorsa bu durumda sipariş verme use case durumu base, hediye gönder use case durumu ise extending olacaktır.

Başka bir örnek olarak farz edin ki, ATM kullanarak kredi kartı borcunuzu ödüyorsunuz. Bu use case’in ismi “Ödeme Yap” olsun. Ödemenizi yaptıktan sonra ATM size makbuz isteyip istemediğinizi soracaktır. İşlemi tamamlamak için makbuz almak gibi bir zorunluluğunuz yoktur ama ATM size böyle bir seçenek sunar. Bu örnekte, “Ödeme Yap” base use case olurken; “Makbuz Yazdır” extending use case olacaktır ve iki use case arasındaki ilişki bir extend ilişkisidir.





Extend ilişki türü, gerçekten ihtiyaç duyulmadıkça kullanılmamalıdır çünkü diyagramı okunması zor bir hale getirecektir. Diyagram üzerindeki okun yönü use case'ler arasındaki bağımlılığın yönünü gösterir. Extending use case'in, base use case'e bağlı olduğu unutulmamalıdır. Yani bir extending use case, base use case olmadan tek başına bir anlam ifade etmez.

### *Use Case Diyagramı Çizilirken Dikkat Edilmesi Gereken Hususlar*

Use case diyagramları, her ne kadar okunması ve anlaşılması kolay diyagramlar olsa da, okunması ve anlaşılması kolay bir diyagram oluşturmak her zaman bu kadar kolay olmayabilir. Bu noktada aşağıdaki unsurlara dikkat edilmesinde fayda vardır:

- En can alıcı noktalardan bir tanesi use case'leri isimlendirirken seçmiş olduğumuz kelimelerdir. Kelimeler alelade bir şekilde değil, sistemin söz konusu davranışını ve fonksiyonaltasını yansıtacak şekilde seçilmelidir. Ek olarak, use case'leri isimlendirirken genelde emir kipi kullanılır. Örnek; “Ödeme Yap”, “Makbuz Yazdır”, “Sipariş Ver” vb.
- Aktörler için söz konusu kullanıcı gruplarını yansıtan kelimeler tercih edilmelidir.
- Include ve extend ilişkileri yalnızca ve yalnızca gerekli olduğu durumlarda kullanılmalıdır. Diyagramın esas amacının gereksinimleri belirlemek olduğu unutulmamalı, bu amaç doğrultusunda basitliğin korunmasına dikkat edilmelidir.
- Gerekğinde bazı önemli hususlara açıklık kazandırmak için diyagrama notlar eklenebilir.

## Veri Akış Diyagramları (Data Flow Diagrams)

Veri akış diyagramı (Data Flow Diagram – DFD), bir bilgi sistemi içinde verinin akış şeması şeklinde görsel olarak ifade edilmesidir. Veri akış diyagramları algoritma tasarımında kullanılan akış diyagramlarına benzetmekle birlikte kullanım amaçları tamamen farklıdır. Algoritma akış diyagramlarında görselleştirilen program akışıdır, veri akış diyagramlarında ise görselleştirilen veri akımıdır. DFD'ler aynı zamanda veri işleme sistemlerinin görselleştirilmesi için de kullanılabilir.

Sistemin içinde her verinin nasıl taşındığını ve bu verilerin akışını sağlayan fonksiyonların neler olduğunu DFD'ler tarif eder. DFD, sistem analizi süresince temeli oluşturan fonksiyonların modellenmesindeki kullanılan bilgileri gösterir.

DFD'ler bir sistemin nasıl bölümlere ayrıldığını ve bu bölümler arasındaki ver akışının hangi yönde olduğunu göstermek üzere tasarlanmışlardır. DFD kavramı ilk olarak "Structured Design" adlı makalesinde Larry Constantine tarafından kullanılmıştır. Edward Yourdon "Just Enough Structured Analysis" adlı kitabında DFD oluşturma yaklaşımlarını iki temel gruba ayırmıştır:

- Yukarıdan aşağı yaklaşımı (top-down)
- Olay bölümleme yaklaşımı (event partitioning)

DFD'lerde sistemin varlıkları, süreçleri, sistemdeki veri depoları ve bunların aralarındaki ilişki kaydedilir. 4 temel simge kullanılmaktadır.

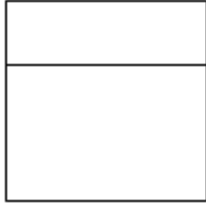
- Varlık:



- Veri Akışı:



- İşlem:

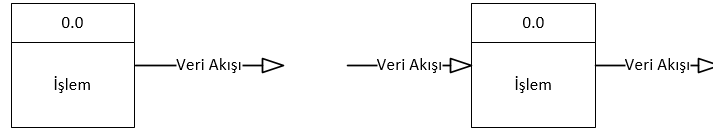


- Veri Deposu:



Sistem analisti, DFD ile sistemi rahatlıkla okuyup hâkim olabilmektedir. Her analistin aynı DFD'den aynı sonuçlara ulaşması ve doğru kararları verebilmesi için geliştirilen DFD'nin belli kurallar çerçevesinde geliştirilmesi gerekmektedir. Başlıca kurallar aşağıda gösterilmiştir. Ancak yine de unutulmamalıdır ki DFD'nin sadece aşağıdaki kurallara göre geliştirilmesi sistemdeki veri akışını tamamiyle doğru yansıttığı anlamına gelmez. Kurallara uygun olması teknik açıdan doğru olabilir ama doğru olması DFD'yi geliştiren sistem analistinin başarısına bağlıdır.

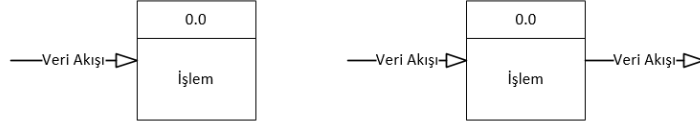
- İşlemin sadece çıkışı olmaz



Yanlış Kullanım

Doğru Kullanım

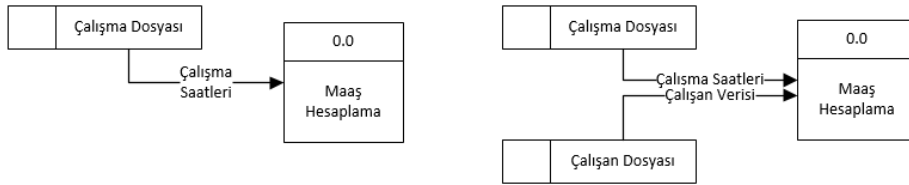
- İşlemin sadece girişi olmaz



Yanlış Kullanım

Doğru Kullanım

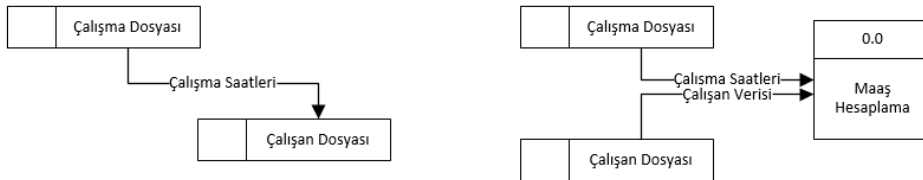
- İşlem girişleri istenen çıkışı verecek kadar yeterli olmalıdır.



Eksik Kullanım

Doğru Kullanım

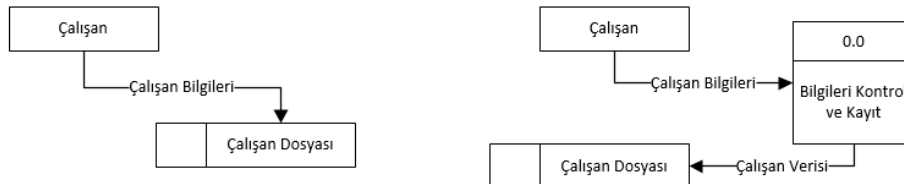
- Her veri deposu bir işlemle ilgili olmalıdır.



Yanlış Kullanım

Doğru Kullanım

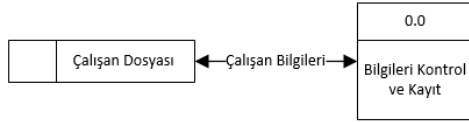
- Veri deposu bir varlıkla doğrudan ilişkide olamaz



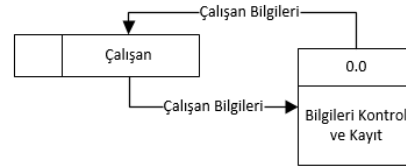
Yanlış Kullanım

Doğru Kullanım

- Veri akışı oku iki yönlü olamaz. Bir işlemle veri deposu arasında karşılıklı veri akışı varsa farklı tek yönlü oklarla gösterilmelidir.

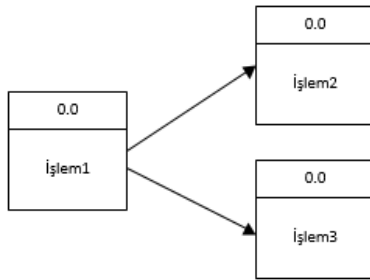


Yanlış Kullanım

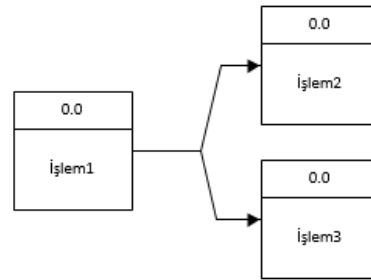


Doğru Kullanım

- Bir işlemden farklı iki işleme gidecek olan aynı veri, aynı yönde iki uçlu okla gösterilmelidir.

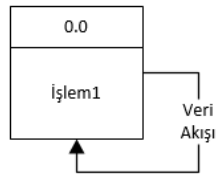


Yanlış Kullanım

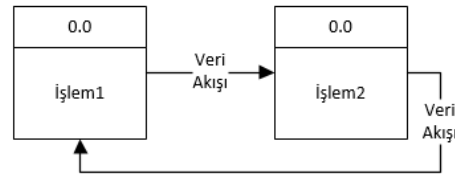


Doğru Kullanım

- Veri, hiçbir işlemten geçmeden çıktığı işleme doğrudan dönemez.

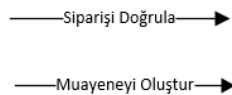


Yanlış Kullanım

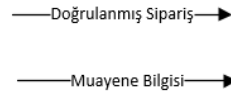


Doğru Kullanım

- Veri akış okları üzerinde gösterilen veri, sadece isim formatında olmalıdır, emir veya fiil olamaz.



Yanlış Kullanım

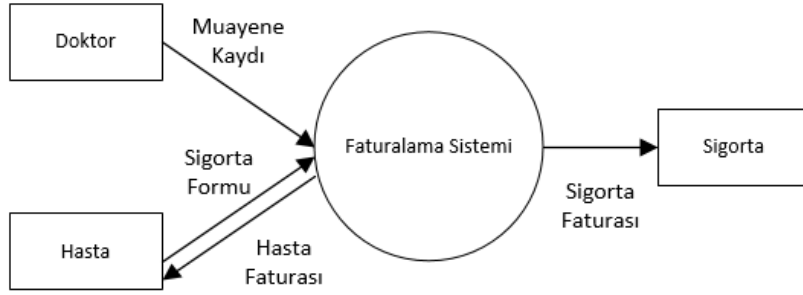


Doğru Kullanım

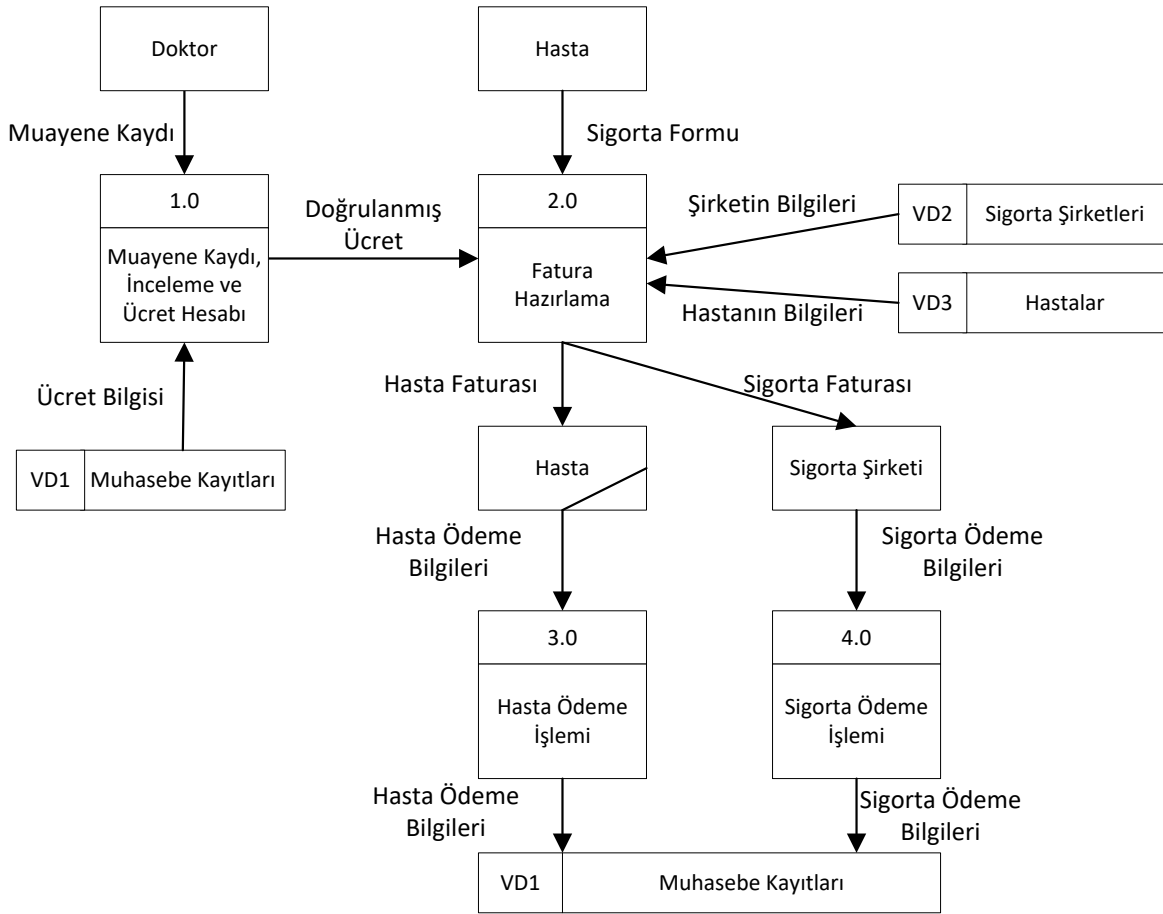
Veri akış diyagramlarında belli bir sıra izlenir. Ön inceleme sonucunda taslak veri akış diyagramı oluşturulur. Taslak veri akış diyagramı, temel alınarak birinci düzeye ikinci düzey ayrıntılı veri

akış diyagramları çizilir. Ancak bazı kaynaklarda ayrıntılı veri akış diyagramları birinci ve ikinci diye adlandırılırken bazılarında da sıfıncı ve birinci düzey olarak adlandırılır.

Sistem bir bütün olarak düşünülürse taslak veri akış diyagramı, sistem ile veri kaynaktan arasındaki ilişkiyi gösterip hangi kaynaklardan veri akışı olduğunu belirtmek için kullanılır. Aşağıda verilen taslak veri akış diyagramı, bir kliniğin faturalama sistemine örnek olarak gösterilebilir. Bu sistemde doktorlardan gelen hastanın muayene kaydı ve hastadan gelen sigorta formu verileri faturalama sistemine girerken, sistemde gerçekleştirilen işlemler sonucunda hastanın bağlı bulunduğu sigorta şirketine ve hastanın kendisine fatura bilgileri sunulmaktadır.



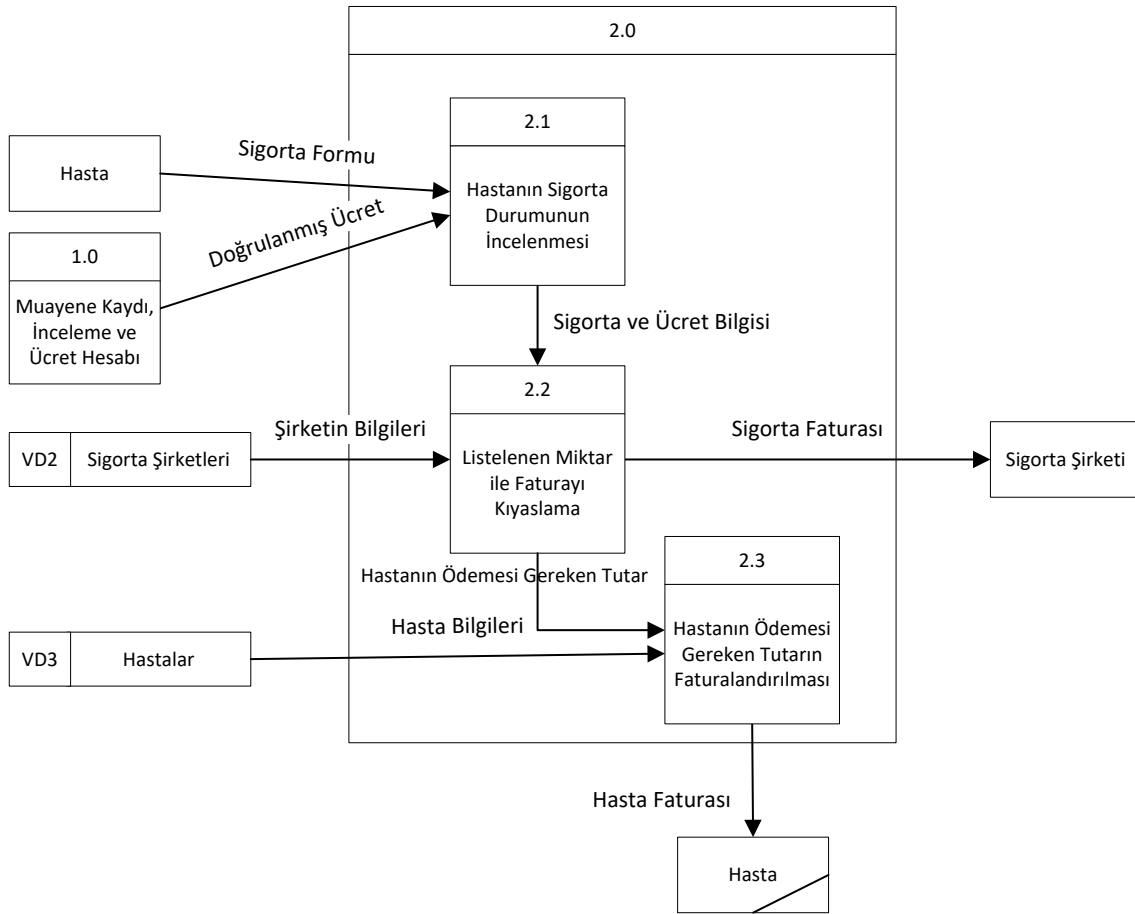
Birinci düzey veri akış diyagramında sistem içindeki alt sistemler, bu sistemlerde gerçekleştirilen işlemler, ilişkili veri kaynakları ve bu kaynaklarla işlemler arasındaki ilişkiler gösterilir. Böylece sistemdeki tüm alt sistemlerin birbiriyle ve dış kaynaklarla ilişkisi genel olarak belirlenmiş olur. Faturalama sistemi örneğinin birinci düzey veri akış diyagramı aşağıda görüldüğü gibi dört alt sistemden oluşmaktadır.



DFD'de numaralandırılan her alt sistemin hangi kaynaklarla ve alt sistemle ilişkili olduğu genel olarak görülmektedir. Bu sistemde hastaya ait özel bilgilerin daha önce hasta dosyasına kaydedildiği kabul edilmektedir. Bu nedenle faturalama sisteminde hastadan sadece sigorta formuna ait bilgiler alınmaktadır. DFD'leri okumanın en kolay yolu soldan sağa doğru okumaktır. Veri akış diyagramında veri akış oklarının birbirini kesmesini ve karmaşıklığı önlemek amacıyla aynı kaynak diyagramın farklı yerlerine yerleştirilebilir. Ancak bu kaynağın aslında farklı bir kaynak olmadığını göstermek amacıyla (gösterilen diyagramda hasta kaynağında olduğu gibi) kaynağa ait olan dikdörtgenin sağ alt köşesine eğik bir çizgi çizilir.

İkinci düzey veri akış diyagramında ise bir önceki basamakta çizilen veri akış diyagramında detaylandırılmayan alt sistemlerde gerçekleştirilen işlemler ayrıntılı olarak gösterilir. Sistem tasarımı kapsamında birinci düzey veri akış diyagramındaki her alt sistem için ikinci düzey veri akış diyagramı çizilir. Bu düzeydeki veri akış diyagramının amacı tek bir alt sistemle veri kaynakları arasındaki ilişkiyi göstermektedir. Faturalama sisteminde yer alan fatura hazırlama sisteminin ikinci düzey veri akış diyagramı aşağıda gösterilmektedir. Alt sisteme ait işlemler sistemin numarasına göre numaralandırılırlar.





## Varlık İlişki Diyagramları (Entity-Relationship Diagrams)



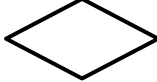

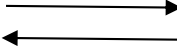
Bilgi sisteminde yer alan veri nesneleri arasındaki ilişkiler, varlık ilişki diyagramında (Entity Relationship Diagram, ER Diagram) grafiksel olarak tarif edilir. ER diyagramında kullanılan standart yapısına göre belli öğeleri ve kuralları vardır.

Veritabanı tasarımında en sık kullanılan tekniklerden bir tanesi olan ER(Entity Relationship) modeli, ilişkisel veritabanı yaklaşımının temelini oluşturmaktadır. ER modeli oluşturulacak veritabanı nesneleri arasında ilişki kurarak, nesnelerin özelliklerini ortaya koyar. Bir ER modelinde 3 temel kavram yer alır.

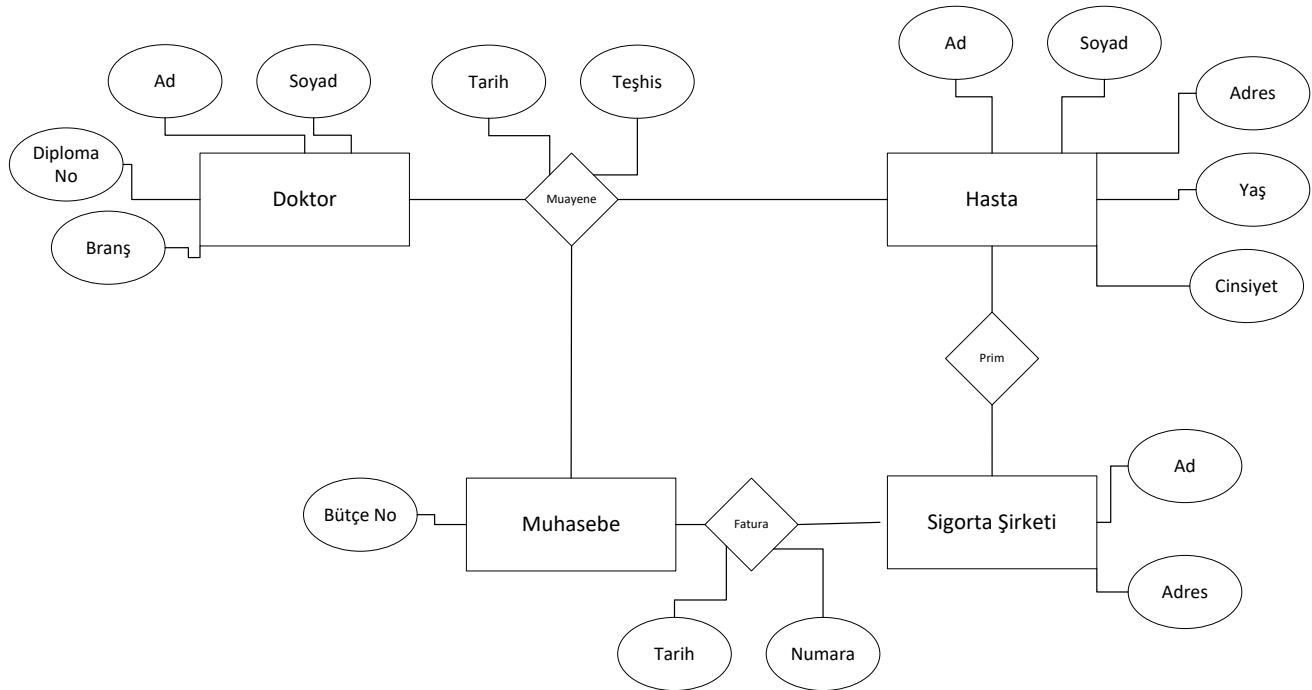
- Varlık(Entity), veritabanında oluşturulacak nesneleri temsil eden yapılardır. Genel olarak veritabanında bu nesnelere tablolar örnek verilebilir. Programlama alanında ise sınıflar(class) varlıklara birer örnektir. ER diyagramlarının temelini varlıklar oluşturur.
- Nitelik(Attribute), ER varlıklarının sahip olduğu her bir alana verilen yapılardır. Varlıkların sahip olduğu parçaları oluşturan bileşenlere denir. Veritabanı alanında örnek olarak tablo sütunları verilebilir. Programlama alanında ise sınıf üye değişkenleri(class member variable) bunun için birer örnektir.
- İlişki(Relationship), varlıklar arasında kurulan fiziksel ve mantıksal bağlantıları temsil eden yapılara denir. ER diyagramlarında varlıkları arasındaki ilişkileri tanımlar.

Genel olarak veritabanı tabloları uygulama aşamasında, sıklıkla ER diyagramlarında tasarlanan varlıkları temsil eder. Nitelikler ise her bir tablodaki sütunlardır. Ancak bu bir kural değildir. Diğer veritabanı nesneleri de ER diyagramları altında benzetilebilir.

Aşağıda ER diyagramlarında kullanılan şekiller gösterilmiştir.

- Dikdörtgen: sistemdeki doktor, hasta vb. varlıkları, 
- Çift kenarlı dikdörtgen; sistemdeki zayıf varlıkları, 
- Eşkenar dörtgen; varlıklar arasındaki ilişki setini, 
- Elips; ilişkiye veya varlığa ait özellikleri, 
- Çizgi: varlıklar arasındaki ilişki türünü ifade eder. 

Bu gösterimler çerçevesinde veri akış diyagramı bölümünde anlatılan "Faturalama Sistemi"nin ER diyagramı aşağıda çizilmiştir.



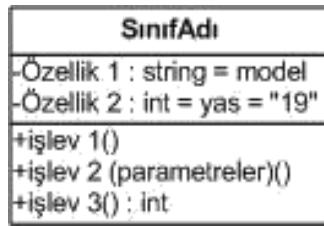
## Nesne, Sınıf Diyagramları (Object, Class Diagrams)

Sınıf tanımları yapılırken genel olarak kullanılan bir kalıp vardır. UML'de bir sınıf bir dikdörtgen içerisinde gösterilir. Dikdörtgenin en üstündeki ilk bölmede sınıfın adı yer alır. Sınıflara isim verirken, her kelimenin baş harfinin büyük olması diyagramların anlaşılabilirliğini artırır. Bir sınıfın

çeşitli özellikleri (attributes) olabilir ve bunlar dikdörtgenin ikinci bölümüne yazılır. Bir sınıfın hiç özelliği olmayabileceği gibi birden fazla özelliği de olabilir.

Sınıfların bir diğer önemli elemanı da işlevlerdir (methods). İşlevler bir sınıfta iş yapabilen elemanlardır. Bu iş başka bir sınıfa yönelik olabileceği gibi sınıfın kendi içindeki bir iş de olabilir. Sınıf diyagramlarında işlevler özelliklerin hemen altındaki bölmede gösterilir. İşlevler özelliklerden farklı olarak birtakım bilgilere ihtiyaç duyabilir veya birtakım bilgileri dışarıya verebilir ya da bunların hiçbirini yapmayabilirler.

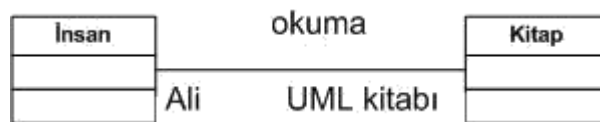
Örneğin, aşağıdaki şekilde iki özelliğe ve üç işleve sahip bir sınıf bulunmaktadır. İşlev 1 hiç bir parametre almaz ve dışarıya bilgi vermez. İşlev 2 birtakım parametreler alır ancak dışarıya bilgi vermez. İşlev 3 ise dışarıdan bilgi almaz ancak dışarıya tamsayı (integer) türünde bilgi gönderir.



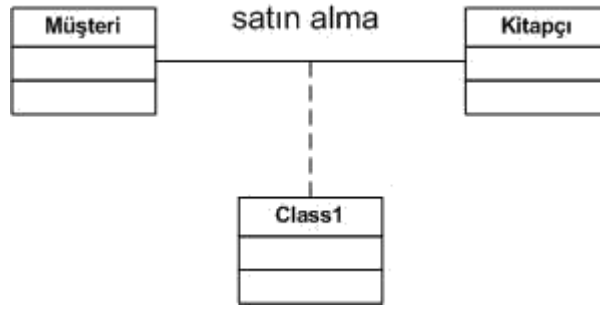
Bir sınıf diyagramında kullanılabilecek temel yapılar bunlar olmasına rağmen koşullar (constraints) ve notlar (notes) denilen elemanlar da eklenebilir. Notlar genellikle işlevler ve özellikler hakkında bilgi veren tercihe bağlı kutucuklardır. Koşullar ise sınıfa ilişkin birtakım koşulların belirtildiği ve parentez içinde yazılan bilgilerdir.

### Sınıflar Arası İlişki (Association)

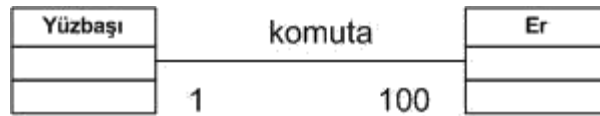
Sınıflar arası ilişki gösterilirken iki sınıf arasına düz bir çizgi çekilir ve ilişkiyi gösteren çizginin üzerine ilişkinin türü yazılır. Örneğin, Kitap ve İnsan sınıfları arasında "okuma" ilişkisi vardır. İnsan sınıfının gerçek nesnesi olan "Ali" ile Kitap sınıfının gerçek nesnesi olan "UML kitabı" arasında "okuma" ilişkisi vardır. Kısaca şöyle denir: Ali, UML kitabı okur.



Sınıflar arasındaki ilişkinin bir çizgiyle belirtebilecek kadar basit olmadığı durumlarda ilişki sınıfları kullanılır. Müşteri ile Kitapçı sınıfı arasında "satın alma" ilişkisi vardır. Fakat müşteri satın alırken Ücret ödemek zorundadır. Ücret sınıfı ile satın alma ilişkisi kesikli çizgi ile birleştirilir.



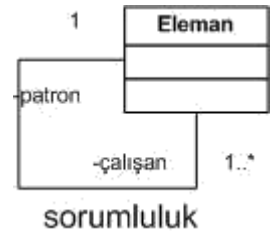
Bu tür ilişkiler bire-bir ilişkilerdir. Bire-çok ilişki durumu da söz konusudur. Bir sınıf, n tane başka bir sınıf ile ilişkiliyse bire-çok ilişki vardır. Örneğin, aşağıdaki örnekte Yüzbaşı ile Er arasında bire-yüz bir ilişki vardır. Burada 1 yüzbaşı 100 Er'e komut verebilir mantığından yola çıkılmıştır.



En temel ilişkiler:

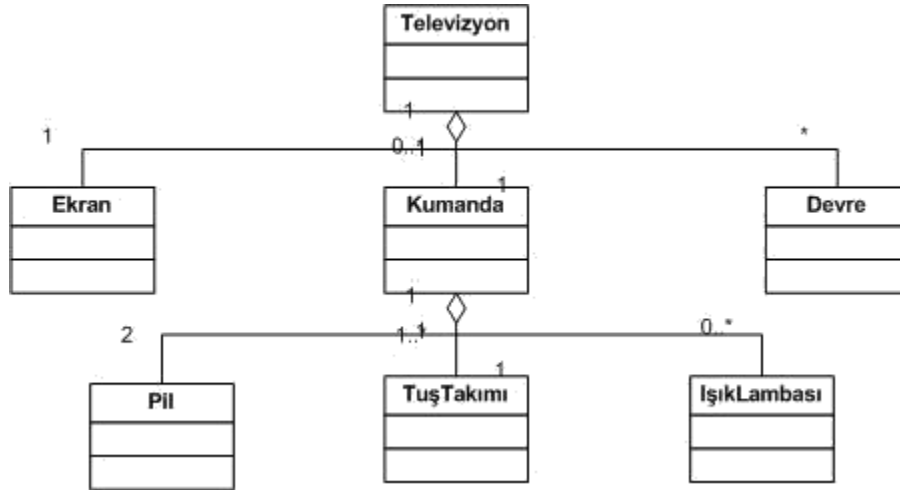
- Bire-bir
- Bire-çok

Bir sınıf kendisiyle ilişkili ise veya bir sınıfın sistemde birden fazla rolü varsa diğer bir ilişki türü olan kendine dönen (reflexive associations) söz konusudur. Patron bir elemandır, aynı zamanda kendisi gibi eleman olan birden çok çalışandan sorumludur.

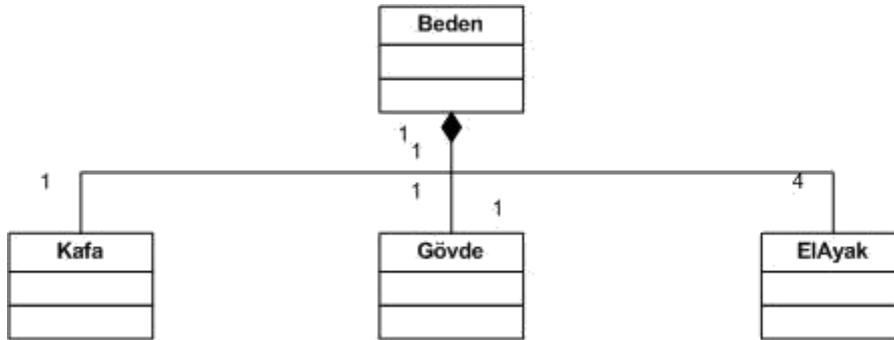


### İlişki Çeşitleri

- **İçerme (Aggregations):** İçerme bağıntısı, iki sınıf arasındaki "sahiptir" veya "içerir" türünden bağıntıları modellemekte kullanılır. Bu bağıntıda bir sınıfın nesnesi, diğer sınıfın nesnesi tarafından sahiplenilmektedir. Parça ile bütün arasında çok sıkı bir ilişki yoktur ve her bir parça ayrı olarak tek başlarına bir anlam ifade eder. İçerme ilişkisi 'bütün parça' yukarıda olacak şekilde ve 'bütün parça'nın ucuna içi boş elmas gelecek şekilde gösterilir. Örneğin, kumandanın tuş takımı, pil ve ışık lambası gibi parça elemanları vardır ve her bir parça kendi başına işlevsel bir bütünlük taşır.



- **Oluşma (Composite):** Parça-bütün ilişkilerini modellemekte kullanılırlar. Bütün nesneler yaratıldığında parçaları da yaratılmalıdır. Bütün ve bütünü oluşturan parçalar arasında sıkı bir ilişki vardır. Oluşma ilişkisi 'bütün parça' yukarda olacak şekilde ve 'bütün parça'nın ucuna içi dolu bir elmas gelecek şekilde gösterilir. Örneğin, bir insan bedeni için kafa, gövde vs. oluşturulması gereken parça nesnelerdir.

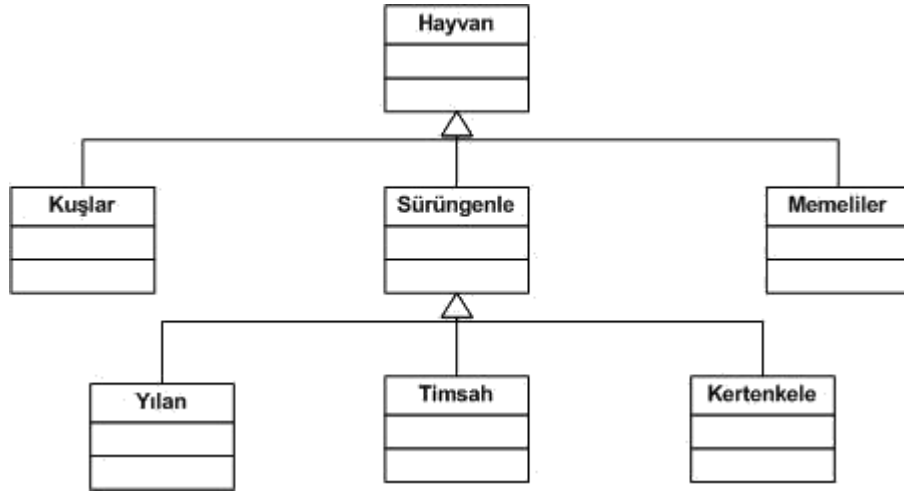


Uygulamada oluşma ve içirme bağıntıları kimi durumlarda birbirine karıştırılabilir. Çünkü, oluşma bağıntısı, içirme bağıntısının yalnızca güçlü bir biçimidir. Bir başka deyişle her oluşma bağıntısı aynı zamanda bir içirme bağıntısıdır. Belirleyici fark ise iki bağıntı türünün nesneleri birbirine bağlama gücü arasındaki farktır.

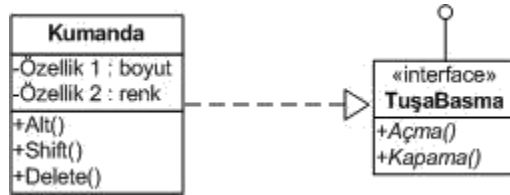
- **Türetme (Inheritance):** Bazı nesnelerin sahip olduğu ortak özellikleri her sınıfta belirtmek yerine ortak özellikler bir sınıfta toplanarak diğer sınıflar ondan türetilebilir. Hatta farklı özellikleri de ekleyerek proje daha etkin hale getirilebilir. Türetme aynı özelliklerin tekrar tekrar yazılmasını engellediği ve gruplamaya yardımcı olduğu için programcılar açısından çok önemlidir.

Örneğin, Hayvan, Memeliler, Sürüngenler, Kuşlar şeklinde sınıfların olduğu düşünölsün. Memeliler, Sürüngenler ve Kuşlar sınıflarının farklı özellikleri olduğu halde hepsinin Hayvan olmasından dolayı birtakım ortak özellikleri vardır. Bu yüzden Memeliler, Sürüngenler ve Kuşlar

birer Hayvandır denir. Yani kısacası Memeliler, Sürüngenler ve Kuşlar, Hayvan sınıfından türemiştir ve herbirinin kendine özgü özellikleri vardır denir.



- Arayüz (Interface): Bazı durumlarda bir sınıf sadece belirli işlemleri yapmak için kullanılır. Herhangi bir sınıfla ilişkisi olmayan ve standart bazı işlemleri yerine getiren sınıfa benzer yapılara arayüz denir. Arayüzlerin özellikleri yoktur. Yalnızca bir takım işleri yerine getirmek için başka sınıflar tarafından kullanılırlar. Örneğin, bir TuşaBasma arayüzü yapılarak istenirse Kumanda sınıfında, istenirse de Klavye sınıfında kullanılabilir.



## Veri Sözlüğü (Data Dictionary)

Veri sözlüğü (Data Dictionary), bilişim maddelerinin içeriğini dil kurallarına uygun olarak tanımlamak amacı ile düzenlenmektedir. Sadece diyagramların çizilmesi ile bilişim alanının analizi tamamlanmış olamaz. Kısa isimler olarak belirlenen veri akışı ve işlemlerin bir sözlük içerisinde tanımlanması gerekmektedir. Veri sözlüğü, diyagramların sahip olduğu işlem spesifikasyonlarında geçen her verinin tanımlarından oluşmaktadır. Bileşik veriler, öğelerine göre; basit veriler ise anlamlarına göre tanımlanmalıdır. Böylece bir veri sözlüğü veri akışlarının, dosyaların ve veriye uygulanan işlemlerin tanımlarından oluşmaktadır.

Veri Sözlüğü, veriler için merkez bellektir. Geliştiriciye veri akışını ve belleğini tanımlamasına basit isimlerle, veri akış diyagramlarını okunabilir tutarak izin verir. Birçok iletişim problemini çözer. Proje üzerinde çalışan herkes kullanılan kelimelerin ve terimlerin tam anlamını bilir. Ayrıca farklı kullanıcıların aynı şey için farklı isimler kullanma problemini de çözer. Veri sözlüğü, çeşitli birimler arasında kullanılan kurumsal verileri ve veri setlerini anlam, kaynak, kullanım, ilişki,



format, vb. öznitelikleri ile tanımlayan merkezi elektronik bir depodur. Aşağıda 2 örnek verilmiştir.

Entity Name	Entity Description	Column Name	Column Description	Data Type	Length	Primary Key	Nullable	Unique
Employee	An employee is someone who work in a company	CompanyID		integer	10	false	false	false
		ID	For the unique identification of employee records.	integer	10	true	false	false
		name	Name of the employee.	varchar	255	false	true	false
		jobtitle	The position of the employee in a company.	varchar	50	false	true	false

Varlık	Ad	Tip	Açıklama
Öğrenci	Numara	Sayı	Dersi alan öğrencinin numarası
	Adi	Metin	Dersi alan öğrencinin adı
	Soyadi	Metin	Dersi alan öğrencinin soyadı
	Kisim_No	Sayı	Dersi alan öğrencinin sınıfı
	Odev_Notu_1	Sayı	100 üzerinden not
	Odev_Notu_2	Sayı	100 üzerinden not
	Quiz_1	Sayı	100 üzerinden not
	Quiz_2	Sayı	100 üzerinden not
	Uygulama_Notu	Sayı	100 üzerinden not
	Proje_No	Sayı	Öğrencinin Çalıştığı proje grubunun numarası
	Proje_Notu	Sayı	100 üzerinden not
	Ara_Sınav	Sayı	100 üzerinden not
	Final_Sinavi	Sayı	100 üzerinden not
	Ortalama	Sayı	Bütün notların ağırlıklı ortalaması
	Basari_harfi	Metin	Ortalamanın katalog karşılığı

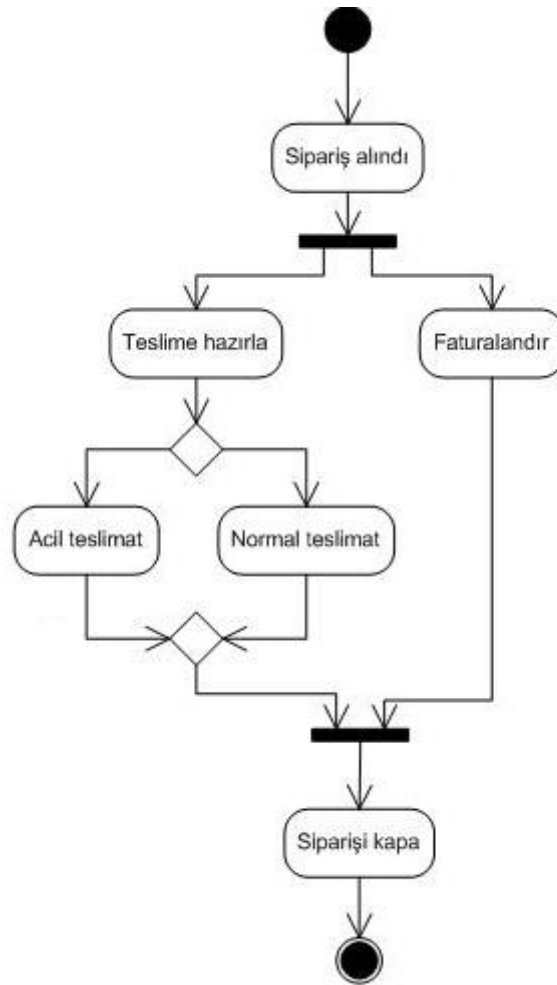
## Aktivite Diyagramları (Activity Diagrams)

Aktivite diyagramları, sistemin akış yönünden davranışını betimler. Aktivite diyagramları bir şeyin durumlarını temsil etmesi niteliği ile Use Case diyagramlarına benzer. Fakat Use Case diyagramları gerçekleşen eylemler neticesinde oluşan durumları nitelerken etkinlik diyagramları ise koşullu ya da paralel ilerleyen eylemleri belirtirler.

Bir aktivite diyagramı, aktiviteler arası kontrol akışını modelleyerek sistemin dinamik doğasını gösterir. Bir aktivite, sistemdeki bazı sınıfların bir operasyonunu gösterir. Bu operasyon sonucu sistemin durumunda değişiklik olmaktadır. Tipik olarak aktivite diyagramları bir iş akışını, bir iş sürecini veya bir iç operasyonu modellemek için kullanılırlar.


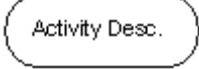
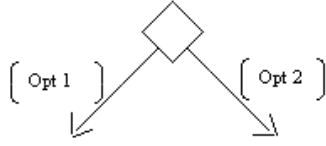
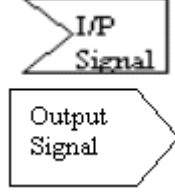
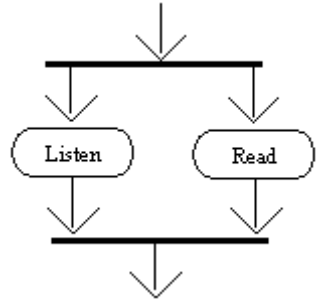


Aktivite diyagramları sistem içerisindeki eylemlerin akışını gösterir. Diyagram yukarıdan aşağıya doğru okunur ve dallanmalar (bölünmeler) paralel eylemleri ve durumları belirtir. Dallanma, aynı anda birden fazla sürecin işlediğini gösterir.

Aşağıda, sipariş işleminin olası bir aktivite diyagramı görülmektedir. Diyagram, sistem akışı içerisinde eylemlerin akışını göstermektedir. Sipariş alındıktan sonra eylemler iki farklı eylem kümesine ayrılmaktadır. Bir yönde sipariş hazırlanmakta ve gönderilmekte, diğerinde ise faturalandırma işlemi yerine getirilmektedir. Sipariş hazırlama dalında teslim metodu duruma göre tespit edilmektedir. Duruma göre Acil teslimat ya da Normal teslimat işlemi gerçekleştirilir. Son olarak paralel yürüyen aktiviteler siparişi kapatmak üzere birleşirler.

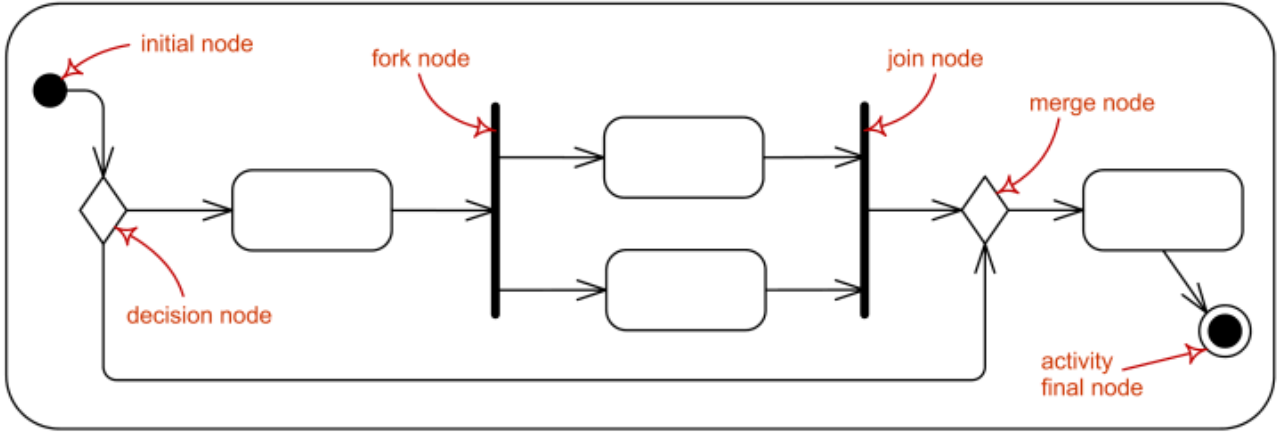


Yukarıdaki diyagramda 1.Aktivite'den sonra bir çatal yer almakta ve dallanma meydana gelmektedir. Bu, 2.Aktivite ve 3. Aktivite'nin aynı anda gerçekleştiğini göstermektedir. 2.Aktivite'den sonrada bir dallanma vardır. Dallanma belirli koşullar altında hangi eylemlerin gerçekleşeceğini gösterir. Tüm dallar bir noktada koşullu durumda işletilecek eylemlerin sonunu belirtmek üzere birleşirler. Tüm paralel aktiviteler sonuca ulaşmadan önce birleşmek zorundadır.

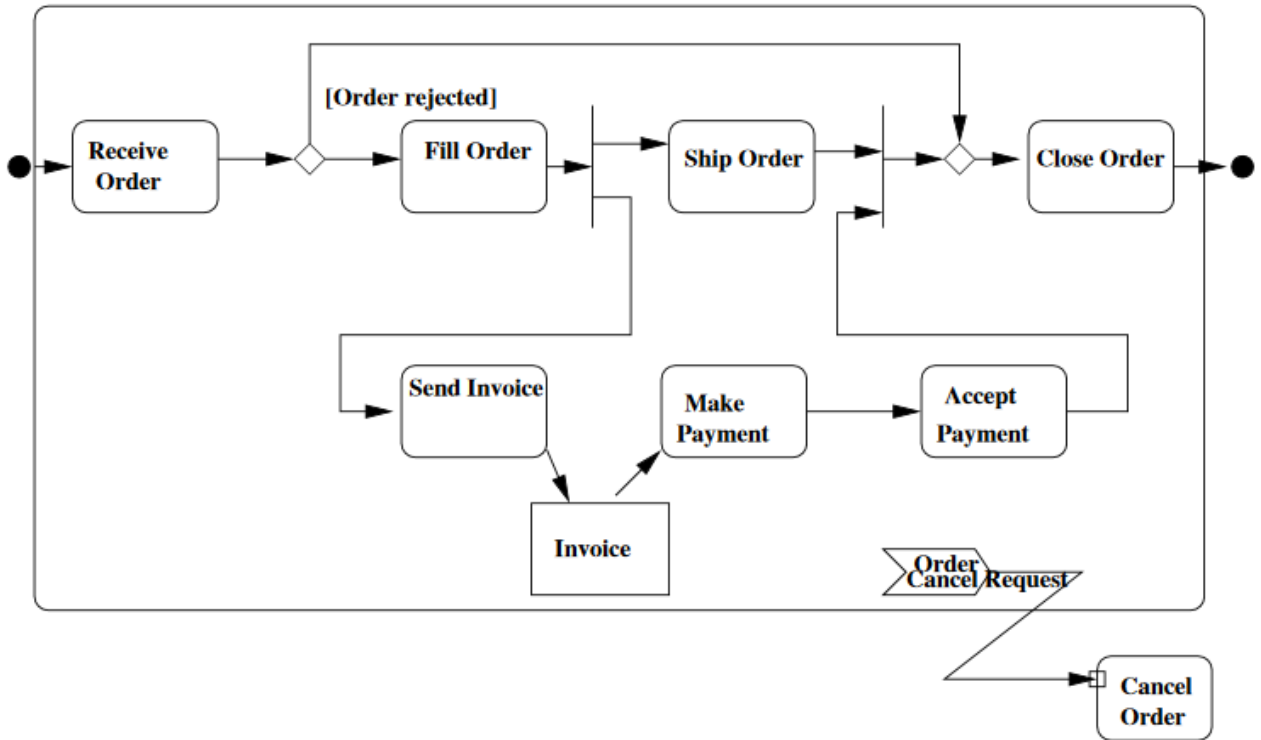
Aktivite diyagramlarında kullanılan temel şekiller ve açıklamaları aşağıda verilmiştir:

Şeklin Tanımı	Kullanılan Şekil
<b>İlk Durumu (Initial State):</b> Diyagramın başlangıcını ifade eder. İçi dolu daire şeklinde simgelenir.	
<b>Aktivite (Activity):</b> Kenarları oval dörtgen olarak simgelenir. Şeklin içerisine aktivitenin kısa tanımı yapılır.	
<b>Dallanma (Decision):</b> Akış şemalarındaki karar verme işlemine benzer bir görevdir. Eşkenar dörtgen ile simgelenir. Karar uçlarına kararın adı parantezler içerisinde yazılabilir.	
<b>Sinyal (Signal):</b> Aktivite bir mesaj gönderdiği veya aldığı zaman bu simge kullanılır. 2 çeşit sinyal olabilir: <ul style="list-style-type: none"> <li>Giriş Mesajı</li> <li>Çıkış Mesajı</li> </ul>	
<b>Paralel Aktiviteler (Concurrent Activities):</b> Bazı aktivitelerin aynı anda paralel veya senkron yürümesi gerebilir. Örneğin öğretmeni dinlemek ve aynı zamanda tahtaya bakmak bu tip 2 aktivitedir. Bu işlem koyu renkli kalın bir çizgi ile belirlenir. Bu çizgiye senkronizasyon barı (çubuğu) denir. Senkronizasyon ayrıca çatallanma (forking) ve birleşme (joining) olarak ta adlandırılmaktadır	
<b>Nesne Akışı (Object Flow):</b> Nesne akışı aktiviteler tarafından nesnelerin oluşturulması yada nesnelerde gerçekleşen değişiklikleri ifade eder. aktivite ile nesne arasındaki veri akışı okunun faaliyetin nesneyi oluşturduğu yada etkilediği anlamına gelmektedir. Nesneden faaliyete doğru çizilen bir nesne akış oku ise aktivitenin durumunun o nesneyi kullandığını göstermektedir.	
<b>Son Aktivite (Final Activity, Final State):</b> Diyagramın son aktivitesi bu simge ile çizilir.	

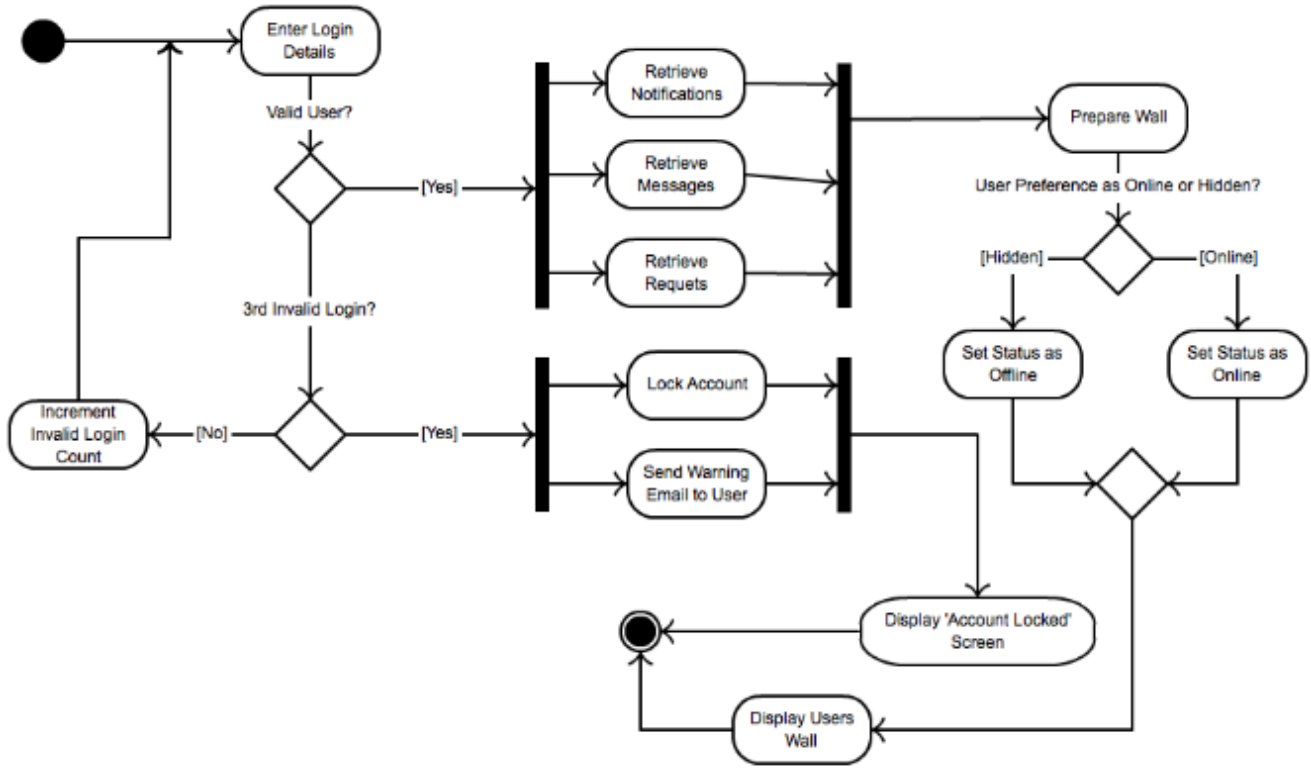
Bazı durumlarda aktivite diyagramı aşağıdaki gibi yatay olarak da tasarlanabilir. Bu durumda diyagram soldan sağa doğru okunur.



Bir sipariş sisteminin aktivite diyagramı aşağıdaki gibi çizilebilir.



Facebook uygulamasının oturum açma modülünün aktivite diyagramı yukarıdaki gibi çizilebilir.



Çalışma Sorusu:

Örnek olarak bir ATM makinesine kartı yerleştirip para çekme senaryosunun aktivite diyagramını çiziniz.

## Etkileşim Diyagramları (Interaction Diagrams)

Nesnelerin birbirleri ile olan ilişkilerini ve etkileşimlerini anlamak için hazırlanan diyagramlardır. 2 adet iç diyagramdan oluşur:

- İletişim Diyagramı (Communication diagram –eski ismi İşbirliği Diyagramı (Collaboration Diagram) ): Nesneler arasındaki etkileşimi bir grafik olarak ortaya koyar.
- Sıralı Diyagram (Sequence Diagram): Nesneler arası etkileşimler oluştukları sıra ile yukarıdan aşağıya doğru çizilirler.

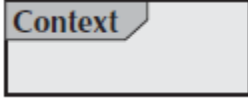
### İletişim Diyagramları (Communication Diagrams):

İletişim diyagramı bir sistemde bulunan sınıflar, nesneler ve aktörler arasındaki dinamik ilişkileri mesajlar aracılığı ile gösterir. Bu bağlamda etkilenen sınıflar arasındaki mesaj değiş tokuşu yaşam çizgileri ile belirlenir. İletişim diyagramlarının kullanımına örnek olarak bir servis şirketinin servis formunda bulunan bilgileri içeren sınıfları ve o sınıflardan alıntı ve sorgulama işlemini yapan mesajları gösteren diyagram gösterilebilir. Eski ismi işbirliği diyagramlarıdır

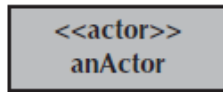
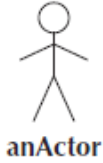
(Collaboration Diagram). Özet olarak iletişim diyagramları kimin kimle nasıl iletişime geçtiğini ve ilişkide olduğunu belirler.

İletişim diyagramlarında kullanılan şekiller aşağıda verilmiştir:

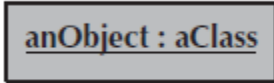
- Çerçeve (frame): Diyagram üzerinde isim yazan bir çerçeve içerisinde çizilir.



- Aktör (actor): Sistemi kullanan veya sistemden etkilenen insanlar çubuk adam olarak çizilir. Eğer aktör insan değil bir nesne ise bu durumda kutu içerisinde ismi << >> arasında belirtilerek yazılır.



- Nesne (object): Belirli görevleri gerçekleştirmek için tanımlanan yapı bloklarıdır. Nesneler, tasarım ve modellemenin en küçük yapı taşı olduğu için hemen hemen her tür diyagramda bulunurlar. İletişim diyagramlarında nesneler dikdörtgen ve bu dikdörtgen içinde nesne ismi olacak şekilde temsil edilir. Önemli noktalardan birisi ise nesne isminin :NesneIsmi formatında olmasıdır.



- Mesaj (Message): Mesajlar bir nesneden diğer bir nesneye doğru çizilen simgelerden oluşmaktadır. Nesnelerin birbirleri ile haberleşmesi, birbirlerine komut göndermesi çizilmiş oklarla gösterilmektedir. İletilen mesaj okun üzerine yazılır. Bir nesne başka bir nesneye mesaj gönderebileceği gibi kendisine de mesaj gönderebilir (recursion - öz yineleme). Genellikle bir sıra numarası verilerek mesaj iletilir. Mesaj bir fonksiyon olduğundan dolayı sonuna ( ) işaretleri eklenir.

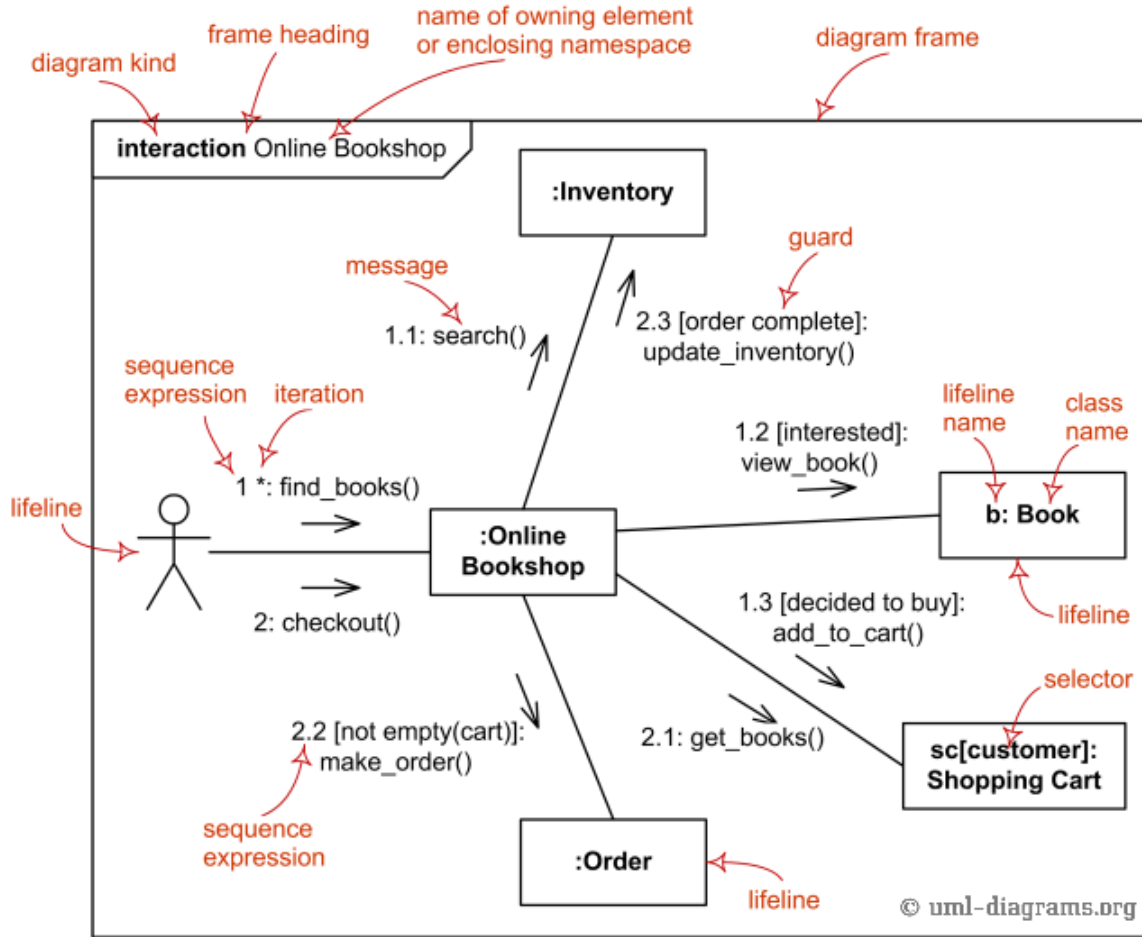
SeqNumber: aMessage →

- Koşullu mesaj (guard condition): Bir mesajın iletimi veya uygulanması bir koşula bağlı ise bu durumda koşullu mesaj kullanılır. Koşul [ ] arasına yazılır ve ardından mesaj adı : işaretinden sonra gelir. Genellikle bir sıra numarası verilerek mesaj iletilir.

SeqNumber: [aGuardCondition]: aMessage →

Aşağıda bir kitap dükkânının bazı işlevlerini yerine getiren bir iletişim diyagramı gösterilmektedir.





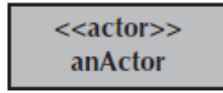
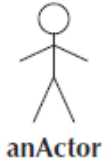
### Sıralı Diyagramlar (Sequence Diagrams)

"Sequence" kelime anlamı olarak "birbirini takip eden, ardışıl olan, peşi sıra, sıralı" anlamlarına gelmektedir. Sıralı diyagramlar, nesnelerin peşi sıra etkileşimde bulunmalarını ve nesnelerin zaman boyutunda birbirleri ile nasıl ilişkiye ve iletişime girdiklerinin tespiti için kullanılır. Buradaki zaman bağılı olmasından kastımız, nesnelerin gerçekleştirdikleri aktivitelerin peşi sıra gerçekleşmesi ve bu peşi sıralığın belirlenen zaman dilimleri içerisinde meydana gelmesidir. Kısacası bu diyagram aktörler ile sistem arasındaki etkileşimin zaman çizgisi üzerinde gösterilmesini hedeflemektedir. Analiz aşamasında çıkarılan Use Case diyagramlarının daha gelişmiş ve rafine edilmiş hali olarak görülebilir.

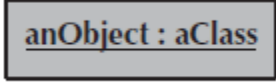
Her use case durumu ayrı ayrı olarak sıralı diyagrama dönüştürülebilir. Bu sebeple oldukça zaman alabilir. Fakat her use case durumunun detaylı olarak yapılmasından dolayı geliştirme sürecini oldukça kolaylaştırır.

Bir "sequence" diyagramı temel olarak nesnelerden(objects), mesajlardan (messages) ve zaman eksenlerinden(timeline) oluşmaktadır. Bu diyagramlarda kullanılan bazı şekiller aşağıda verilmiştir.

- Aktör (actor): İletişim diyagramlarındaki gibidir.



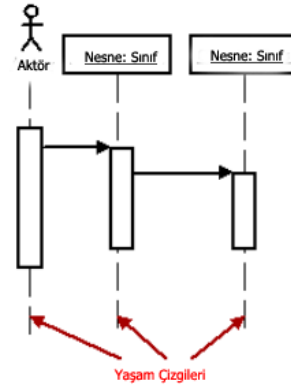
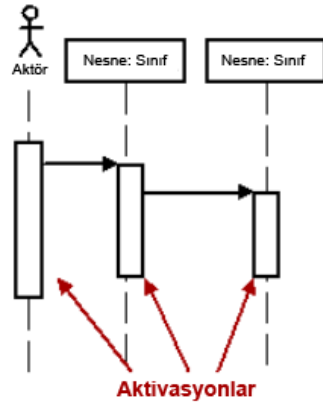
- Nesne (object): İletişim diyagramlarındaki gibidir.



- Zaman Çizgisi (lifeline): Kesik çizgiler ile etkileşimlerin zaman dönemleri belirlenir.

Her bir nesnenin altından çıkan ve kesik kesik olan çizgi nesnenin zaman çizgisini(timeline) belirtmektedir. "Sequence" diyagramlarındaki zaman olgusu bu çizgilerle belirtilmektedir. Kesik çizginin en altı teorik olarak sonraki zamanı göstermektedir.

Zaman çizgisi üzerinde bulunan ince ve uzun dikdörtgenler ise o nesnenin o zaman içerisinde meydana getirdiği aktivitedir(activation). Yine teorik olarak dikdörtgenin uzunluğu aktivitenin ne kadar zaman aldığı ile doğru orantılıdır. Yani uzun süren bir aktivite daha uzun dikdörtgenle gösterilir.

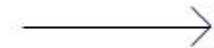


- Mesaj (Message): İletişim diyagramlarındaki gibidir.

SeqNumber: aMessage →

Birkaç tip mesaj bulunmaktadır.

1 - Basit(Simple) Mesaj Tipi: Bu mesaj tipi basit anlamda bir nesnenin, akış kontrolünü diğer bir nesneye verdiği durumlarda kullanılır. Sık kullanılan bir mesaj tipi değildir.

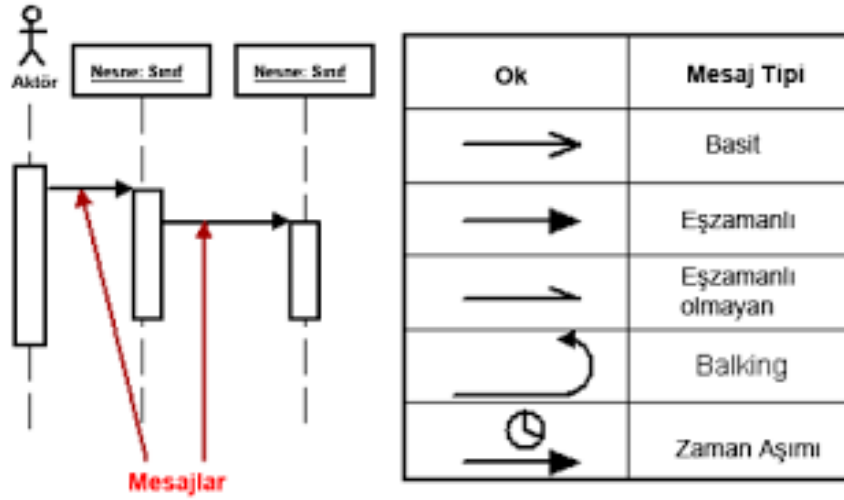


2 - Senkron (Synchronous) Mesaj Tipi : Bir nesnenin mesajı gönderdikten sonra, zaman çizgisinde devam edebilmesi için karşı nesneden cevap beklenmesi gereken durumlarda

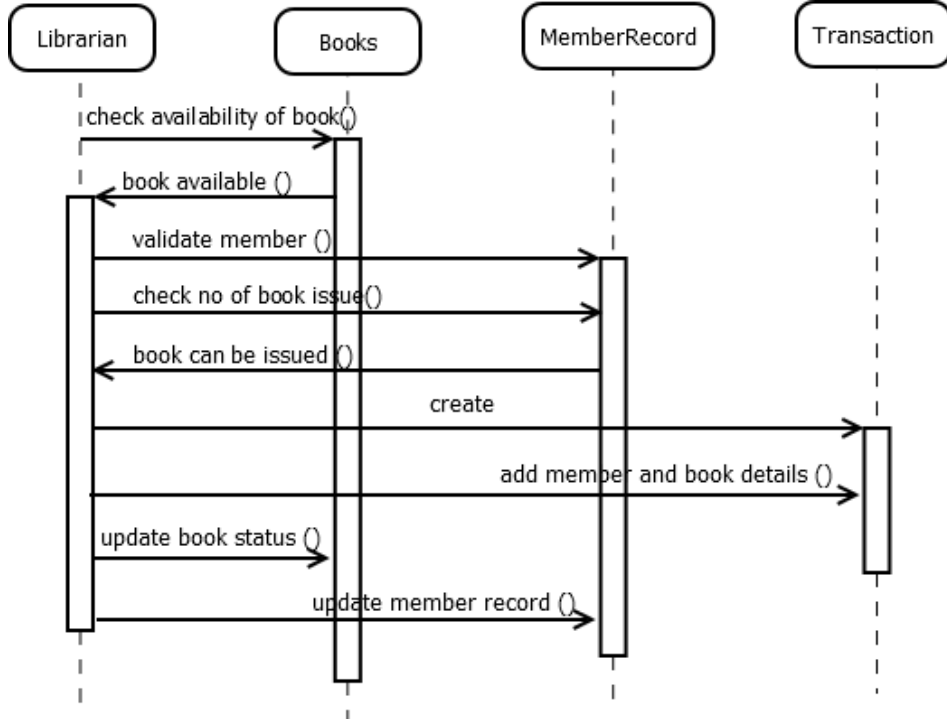
kullanılır. Varsayılan mesaj tipi olduğundan sıklıkla bu mesaj tipi kullanılmaktadır. Gösterimi UML diyagramlarında aşağıdaki gibidir.



3 - Asenkron (Asynchronous) Mesaj Tipi : Senkron mesajların aksine bir nesnenin mesajı gönderdikten sonra, zaman çizgisinde devam edebilmesi için karşı nesneden cevap beklenmesi gerekmiyorsa kullanılır. Sıklıkla asenkron işlemesi gereken komut zincirlerinde kullanılmaktadır.



Kitap teslim işlemi için bir sıralı diyagram aşağıdaki şekilde verilebilir.



Bir online yemek satışı için basit bir sıralı diyagram aşağıda verilmiştir.

