

# Hafta 1: Nesneye Yönelimli Düşünme → Nesne nedir?

Nesneler kodu düzenli tutarlar.

" " esnek tutar.

" " kodun yeniden kullanılmasına izin verir.

Kedi miyav = new Kedi(); #nesne türete

Gurbet Günpören

## Hafta 2: Yazılım Sürecinde Gereksinim ve Tasarım

Yazılım tasarımı ⇒ Bir müşterinin isteklerini ve gereksinimlerini uzun vadede istikrarlı ve sürdürülebilir olan, geliştirilebilir ve daha büyük bir sistemin parçası olabilecek çalışma koduna dönüştürme işlemidir.

Yazılım mimarının görevi ⇒ Ürün ile müşteri ve mühendis ekipleri arasındaki ara yüz olmaktır.  
Temel amacı, müşterinin ihtiyacına, müşterinin sahip olduğu bütçe dahilinde hizmet etmektir.

Bir process jinedir (iterative)

① Gereksinim (Requirements) ⇒ müşteri veya kullanıcı isteğine bağlı olarak bir üründe uygulanması gereken şartlar veya yeteneklerdir.

② Tasarım (Design) ⇒ Kavramsal tasarım ve teknik t.

Kavramsal tasarım ⇒ Bir ürünün nasıl çalışacağını göstermek ve tartışmak için basit bir yol sağlayarak müşteriler ve kullanıcılar ile tasarım kararlarını netleştirmeye yardımcı olur.

Teknik t. ⇒ Gözümün teknik ayrıntılarını tanımlamak için kavramsal tasarımlar ve gereksinimler üzerine kuruludur.

Başlam ve Sorular ⇒ Başlam, tasarımdaki kalitenin dengesine karar verirken önemli bilgiler sağlar.

Sınıf sorumluluk işbirliği (CRC) ⇒ 3 bölümden oluşur. sınıf adı, sınıf sorumlulukları ve ortak çalışanlar.

Class Name
Responsibilities (Sorumluluklar)
Collaborates (İşbirlikçiler)

### CRC Avantajları

\* Kavramsal bir sistemi kolayca anlamamızı sağlar.

Daha " " tasarımlar oluşturmamıza izin verir.

Pahalı işlen kaynakları gerektirmez.

CRC temel olarak, bir takımın farklı kişilerin birlikte çalışarak tasarımı geliştirmelerini ve takımın herkesin katkıda bulunmalarını sağlayan bir beyin fırtınası aracıdır.

CRC, Extreme Programming ve UML gibi modelleme dilleriyle birlikte kullanılabilir.

## Hafta 3: Tasarım Prensipleri

Soyutlama ⇒ bir kavramın temel davranışlarını açıklar

① Temel özellikler (Attributes) ⇒ zaman içinde kaybolmayan özelliklerdir. Değerleri değişebilmesine rağmen, öz niteliklerin kendileri değişmez.

② Temel davranışlar veya sorumluluklar

İyileştirme - kapsülleme (Encapsulation) ⇒ 3 önemli fikir vardır:

① Paketleme ⇒ Değerleri ve davranışları kendi kendine yetebilen bir nesneye dönüştürülebilir.

② Açığa çıkarma: Bir nesnenin belirli bir verisini ya da fonksiyonunu kendisiyle paketleme yeteneğidir.

③ Kısıtlama: Bir nesnenin belirli bir verisini ya da fonksiyonunu kendisiyle paketleme yeteneğidir.

Sınıf ismi
Özellikler (Attributes)
Davranışlar (Operations)
- private + public

Öğrenci	Ders
Öğrenci ismi Öğrenci no Bölümü Ders kayıtları Ders çıkarma	Ders ismi Ders kodu Ders içeriği "Eğitimi"

Kart Okuyucu Sınıfı	Collaborates
Responsibilities Kart oluşturuldu a tıme bildir	ATM
Karttan bilgiyi okur	Kart
Kartı çıkar	
Kartı tut	



## (Hafta 6) Aynıştırma (Decomposition)

Bir bütün alır ve farklı bölümlere ayırır. Persini de yapar.


3 for 11/12 vordr!

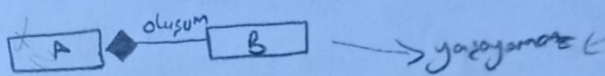
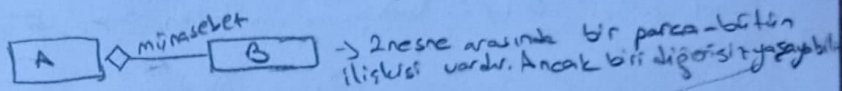
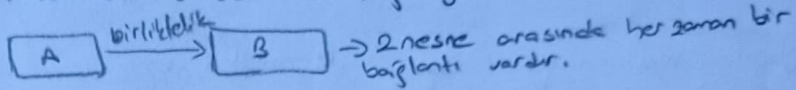
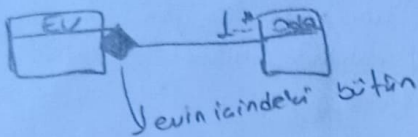
① Birliklilik → iki nesne arasında, bir süre birbirleriyle etkileşime girebilecek gerçek bir ilişki olduğunu gösterir.  
Birbirlerine bağımlı değildir. (Müşteri: otel  $0^* \dots 0^*$ )

④ Oluşum elde etmek için verilerimiz veya nesnelerimiz arasında kurabileceğimiz ilişki türleri birliktelik veya münasebet olabilir.

③ Minusket  $\Rightarrow$  parçalar time ait olsa da, bağımsız olarak da var olabiliyor.

Ölçüm  $\Rightarrow$  Bir bütün, parçaları var olmadan var olamaz ve bütün, eğer yok edilirse, o zaman parçaları da yok edilir. Ev örneği

  $\rightarrow$  2 nesne arasında her zaman bir ilişki vardır.



## Java Erişim Seviyeleri

Public : her yerden erişebilir.

public : her yerden erişilebilir.  
private : sadece sınıf içinden erişilebilir.

protected sınıf içinde ve türetilen sınıflar içinde erişilebilir.

private, sadece sınıf içinde kullanılabilir.  
protected, sınıf içinde ve türetilen sınıflar içinde erişilebilir.  
( kapsüllülmüş, türetilen ve aynı paket içindeki sınıflardan erişilebilir.

Hafta 5

Hafta 5 Kalıtım → İlgili sınıfların tek bir üst sınıfa genelleştirilmesine izin verir ve yine de alt sınıfların aynı nitelik ve davranış kümesini korumasını sağlar.

**Hafta 5** Kalıtım → İlgili sınıfların tek bir üst sınıfa ait olmasını sağlar. Sınıfların aynı nitelik ve davranış kümesini paylaşmasını korumasını sağlar. İyi bir tasarımda, modüller birbirleriyle uyumludur ve bu nedenle kolayca bağlanabilir ve yeniden kullanılabilir. Tasarımın karmaşıklığını değerlendirmek için 2 öbektir kullanılır; Coupling (bağlama) ve cohesion (yapışma).

Tasarımın karmaşıklığını değerlendirmez için 2000

### Coupling (Bağlama)

Coupling (Bağlama)  
Bağlama, bir model ile diğer modeller arasındaki karmaşıklığa odaklanır. Bağlama 2 uç nokta arasında dengelenebilir ısıtıcı b. ve parçaları bağlama  
↳ Kütü tasarımlarında bulunulacak şartlar şunlardır: Derece, kolaylık ve esneklik

Başlangıçta, bir model ile ilgili olarak, bir hipotez oluşturulur. Bu hipotez, bir dizi değişkenin bir arada çalıştığı bir süreçten geçtiğini varsayar. Bu hipotez, bir dizi değişkenin bir arada çalıştığı bir süreçten geçtiğini varsayar. Bu hipotez, bir dizi değişkenin bir arada çalıştığı bir süreçten geçtiğini varsayar.

Bağlamı değerlendirmek için potansiyel bağlantıların sayısı, bağlantıların türleri ve diğerleri arasındaki bağlantı sayısıdır.

Derece: Model ve diğerleri arasındaki bağlantı sayısıdır. Model ile ilişkili olduğu ile ilgilidir. Model ile diğerleri arasındaki bağlantı sayısıdır. Model ile ilişkili olduğu ile ilgilidir.

Kolaylık: "İkili ile bu model için ne kadar değiştirilebilir olduğunu gösterir."

Esnellik: Diğer modüllerin bu modül için ne kadar değiştiğini

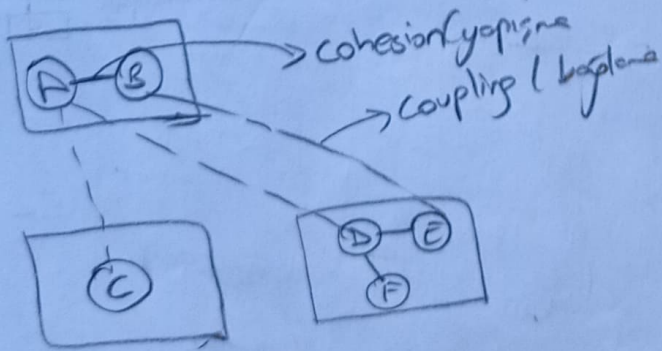
Cohesion (Yapışma): Bir model içindeki karmazıklığa odaklanır ve bir modelin sorumluluklarının netliğini gösterir. Bir tasarımcı için bir tasarımcı h.c.dur. Bir tasarımcıya ait olan ve başka hiçbir tasarımcıya ait olmayan veya açık bir amaç.

Cohesion (Yapışma): Bir model içindeki elemanların birbiriyle ilişkisini gösterir. Birbirine atıfta bulunan elemanlar arasında ilişkiyi gösterir.

Cohesion (Yapışma): Bir modül içi birlikteliğini gösterir. İyi bir tasarım h.c.dur.  
High cohesion: Bir görevi yerine getiren ve başka hiç bir şey yapmayan veya aynı bir amacı olan bir modülün h.c.vardır.  
Low cohesion: Bir modülün başka bir amaca sahip olması low.c. vardır.

Low cohesion: Modüller net olmayan bir amaç varsa low c. vardır.

low cohesion: Modülün net olmayan bir amacı varsa low c. vardır.



Siki baplaması → kök türemesi  
\* jersde " → yi "

\* Yüksek yapıma → iyi tasarım  
düşük " → kötü "



## Hafta 6 Tasarım Kalıpları

Nesneye y.p. jeri başlayan programalar için ortak dil oluşturmaları, ilk kullanan Christopher Alexander <sup>→ mimar</sup> T.k. yüksek cohesion (yapışma) ve düşük coupling (bağlama) yapıları kurulumamıza yardımcı olur,

### Tasarım Kalıpları

Finding responsibilities  
Highly Cohesive Objects  
Lowly Coupled Objects

"Değişkenlik gösterecek tüm özellik ve davranışlar, bir araya getirir ve ayrı kalıplarda tut,

Setter: Bu metod davranış ile sınıf içinde saklı olan metod ve değişkenlere ulaşabilir ve bunları değiştirebiliriz.

Strateji? Tasarım Kalıpları: Değişkenlik gösteren algoritmaları ve davranışları bir araya getirip (encapsulation) algoritmaları geliştirilebilir hale getirmek. Böylece arayüzün arkasına toplanan ve ileri de geliştirilmeye açık davranışların kontrolsüz değiştirilmesinin önüne geçer.

### Tasarım prensipleri:

- 1) Değişkenlik gösterecek davranışları bir araya getir.
- 2) Kalıtım yerine kompozisyon kullan
- 3) Uygulanmaya program yazmak yerine, arayüze yazılım yap

Görölenci tasarım  
Bir nesnenin durumlarında değişiklik olduğunda, bu değişikliklerden haberdar olmak isteyen diğer nesnelere haber verilmesi gerektiği durumlarda kullanılır.

### Tasarım kalıplarının Temel Kuralları

- 1) Uygulanmaya değil de arayüze program yazılmalı.
- 2) Nesneler arasında esnek bağ oluşturulmalı nesneler kesinlikle birbirlerine somut bağ oluşturmamalı.
- 3) Değişken ve gelişmeye açık olan bölümler tespit edilmeli ve bir çatı altında toplanmalı.
- 4) Kalıtım yerine kompozisyon kullanılmalı.
- 5) Tasarım geliştirmeye açık olmalı fakat değiştirilmeye kapalı olmalı.
- 6) Somut sınıflar kullanılmalı. Somut sınıflar yerine daha çok soyut sınıflar kullanılmalı. <sup>→ abstract veya interface</sup>

### Görölenci Tasarım Kalıbı → 1-n ilişkisi var <sup>Chatbot örneği</sup> <sup>→ arayüzcü vs. kullanıcı ilişkisi</sup>

Medyator Tasarım Kalıbı → Nesnelerin birbirleri ile iletişiminin koordinasyonunu yönelebilen tasarım kalıbıdır.  
Görölencide n nesnesinin birbirleri ile iletişimde problem yaşayabilme ihtimaline karşılık Medyator tasarım k. bu sorunu çözmektedir.

Hafta 9 Dekorasyon tasarım kalıbı → Kalıtım yönteminin tersine konfigürasyonu belirleme işini belirleme esnasında değil de geliştirme esnasında yapıyoruz.

Tasarım Kuralı: Uygulanmaya program yazmıyoruz. Arayüze uygulama yapıyoruz. Böylece çalışma (run-time) esnasında istediğimiz değişikliği rahatlıkla yapabiliyoruz.  
wrap: dekorasyon eklene <sup>→ monitor, harddisk örneği</sup>

Yalnızlık - Tekil Tasarım Kalıbı Bir nesnenin sadece ve sadece bir kez yaratılmasını sağlar. <sup>→ örnek veritabanı oluşturma</sup>  
Singleton Design Pattern

Nesne private yapılır. private veritabanıbağlantı() { }  
Her private değişkende olduğu gibi, private yapılandırıcıya erişmek için public bir metoda (getter) ihtiyacımız var.  
public static veritabanıbağlantı getVeritabanınesne() { return new ... (); }  
public class test {  
 psurm;  
 veritabanıbağlantı.getVeritabanınesne(); } }







## Öğrenci

```

- notortalanasi : float
- lisansprogrami : String
+ getNotortalanasi() : float
+ Lisansprogrami : String
+ setNotortalanasi() (float)
+ setLisansprogrami (String)

```

```

public class Öğrenci {
    private float notortalanasi;
    private String lisansprogrami;
}

```

```

class test {
    psum {
        Öğrenci ahmet = new Öğrenci();
        ahmet.notortalanasi = 3.2;
        " " . lisansprogrami = "BMÜ"
        Sout ( ahmet, notortalanasi );
    }
}

```

```

ahmet.setNotortalanasi (3.2);
Sout (ahmet, getNotortalanasi);

```

Arayüz için implements kullanılır  
(interface)

Coupling (Bağlama) → Bir modül ile diğer modüller arası karmaşıklık

- ① bir modül diğer modüllere çok sayıda parametre veya arayüz aracılığıyla bağlanırsa,
  - ② bir modüle karşılık gelen modülleri bulmak zorlaşır
  - ③ " " modül sadece belirli diğer modüllere bağlanabilir ve değiştirilemez ise
- Sistemin sıkıca bağlandığını ve kötü bir tasarıma sahip olduğunu düşünebiliriz.

Cohesion (yapışma) bir modül içindeki karmaşıklığa odaklanın

Tarih ve İşyeri Amirinin İmzası

...../...../20...



Yazılım mimarisi, sistemde hangi öğelerin bulunduğunu, her bir öğenin hangi işlevi olduğunu ve her bir öğenin birbirleriyle nasıl ilişkili olduğunu tanımlar.

- Sistemin amacı
- Sistemin kullanıcıları
- Kullanıcılar için en önemli olan nitelikler (qualities)
- Sistemin çalışacağı yer.

Kruchten's 4+1 View Model → Bir yazılım sisteminin tüm davranışını ve gelişimini yakalamak için çoklu bakış açıları gereklidir.

1- Yazılım işlevselliği → Bir sistemin müşterinin istediği amacı yerine getirmek için ne yaptığını içerir. logical view (mantıksal görünüm) olarak adlandırılan bir perspektif ile ilişkilendirilir.

2- Sistemin Verimliliği → Sistemin performansı ve ölçeklenebilirliğini etkiler. Mantıksal görünümdeki nesnelerin uyguladığı süreçlere odaklanmak, process view (süreç görünümü) olarak adlandırılan bir perspektifte yol açar.

3- Yazılım, development view (geliştirme görünümü) de içerebilir. Bu bakış açısı, yazılımın hiyerarşik yapısı gibi uygulama hususlarına odaklanır.

4- Yazılımın bir başka perspektifi physical view (fiziksel görünüm) görülebilir. Yazılım, etkileşime giren ve konuşturulması gereken fiziksel bileşenlere sahip olacaktır. Bu farklı öğeler ve bunların konuşturulması arasındaki iletişim, sistemin çalışma şeklini etkiler.

Mantıksal, süreç, geliştirme ve fiziksel görünümle Philippe Kruchten'in 4+1 Görünüm Modelini oluşturur.

### Logical View (Mantıksal görünüm)

Genellikle sistemin nesnelerini içerir.

UML sınıf diyagramı oluşturulabilir.

Sınıf diyagramı, sınıfların nasıl etkileşimde bulunduğunu ve verilerin bir veritabanında birbirleriyle nasıl ilişkili olması gerektiğini göstermeyi kolaylaştırır.

### Process View (Süreç görünümü)

Fonksiyonel olmayan gereksinimlerin elde edilmesine odaklanmaktadır.

İşlem görünümü ayrıca, mantıksal görünümdeki nesnelere karşılık gelen işlemleri de sunar. UML diyagramlarından botlar, aktivite (activity) diyagramı ve

Aktivite diyagramı → Bir sistem için işlemleri veya aktiviteleri gösterebilir.

Dağılım " → Nesnelerin birbirleriyle nasıl etkileşime girdiğini, bu yöntemlerin nasıl uygulandığını ve hangi sırada olduğunu gösterir.

Tarih ve İşyeri Amirinin İmzası

...../20...

Yazılım işlevselliği → Logical view (mantıksal görünüm)  
Sistemin verimliliği → process view (süreç görünümü)  
Hiyerarşik yapısı → development view (geliştirme " ) → tanımlara, bütçe, iş atamaları



## Development View (Geliştirme Görünümü)

Geliştirme görünümü hiyerarşik yapıyı ve proje yönetimini kapsar. Yapılım geliştirme ayrıntıları ve bunu desteklemek için neyin dahil olduğu ile ilgilenmektedir. Zamanlara, bütçeler ve iş atomları gibi yönetim ayrıntılarını da kapsar.

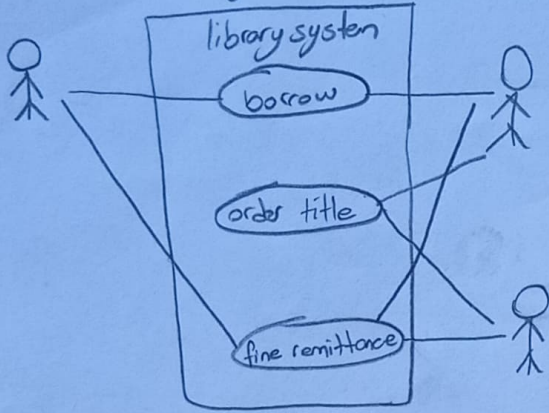
## Physical View (Fiziksel görünüm)

Maniksal, süreç ve geliştirme görünümündeki öğelerin sistemi çalıştırmak için farklı düğümlere veya donanımlara nasıl eşlenmesi gerektiğini ele alır.

! Bir sistemin fiz. ile ilgili en etkili UML diyagramlarından biri dağılım (deployment) diyagramıdır. Bir sistemin parçalarının donanım veya yürütme (execution) ortamlarına nasıl dağıtıldığını ifade etmektedir.

## USE CASE

Use case diyagramı, bir dizi use case hakkında genel bilgi sağlar.



UML Use Case diagram

## Senaryolar

Bir sistemin kullanım durumlarıyla veya kullanıcı görevleriyle uyumludur ve diğer dört görünümün birlikte nasıl çalıştığını gösterir.

Her senaryoda, nesneler ve işlemler arasındaki etkileşimin sırasını tanımlayan bir komut dosyası (script) vardır.

### Bu script:

- Maniksal görünümde tanımlanan arahtar nesneleri,
- Süreç görünümünde açıklanan süreçleri,
- Geliştirme " tanımlanan hiyerarşisi,
- Fiziksel " belirtilen farklı düğümleri içerir.



## Görölenci Pasarm Kalibi

interface  
Yayinci

Attributes

```

public void aboneEkle (Abone abone)
public void aboneSil ( " " )
public void yayinla ()

```

interface

Abone

Attributes

```

public void pnceel yayinla ()
public void aboneEkle ()
public void aboneSil (Yayin,

```

HavaDurumu

Attributes

private String yayin tipi

Operations

```

package HavaDurumu ()
public String toString ()
public Abone [0..*] getAbone ()
public void setAbone (Abone abone [0..*])

public void aboneEkle (Abone abone)
public void aboneSil (Abone abone)
public void yayinla

```



# Hafta 11 - Mimari Tasarım

(3)

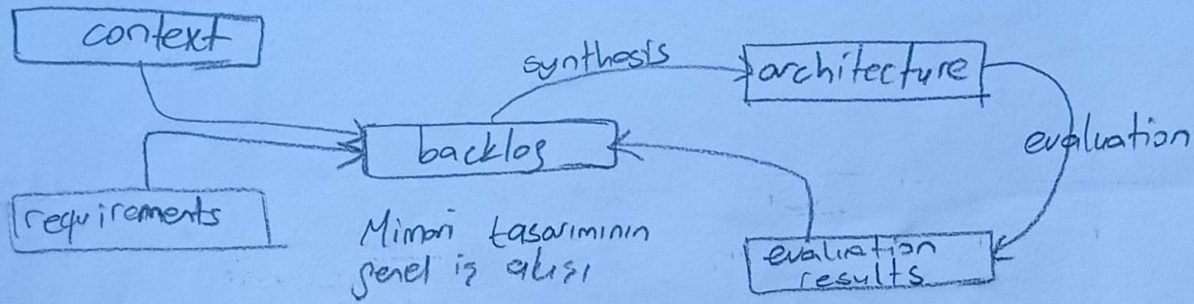
Tasarım süreci yaratıcı bir süreçtir ve tasarımcıların kalitesi ve uzmanlığı başarısı için kritik bir belirleyicidir.

## ADD → Attribute Driven Design

ADD sürecinin girdisi gereksinimlerdir, yukarıdan aşağıya ayrıştırma (decomposition) işlemi olarak tanımlanmaktadır.

- Her iterasyonda daha fazla ayrıştırma için bir veya birkaç bileşen seçilir, ilk iterasyonda yalnızca bir bileşen vardır: Sistem
- Kalite ötel. senaryolarından, mevcut adımda ele alınacak önemli bir kalite özelliği seçilmiştir.

Lütfiye sistemi → bir sunum katmanı  
bir iş mantığı ve  
bir veri katmanı



Tasarım sürecinin sonucunu belgelemek için teknikleri kullanır: kararlar, tercihler, ortaya çıkan tasarım.

## Yatırım Mimarısının Kalite Ölçütleri

Kalite ölçütleri, bir sistemin tasarımını, çalışma zamanı performansını ve kullanılabilirliğini ölçmek için kullanılan sistemin ölçülebilir özellikleridir. Kalite ötel. standartları.

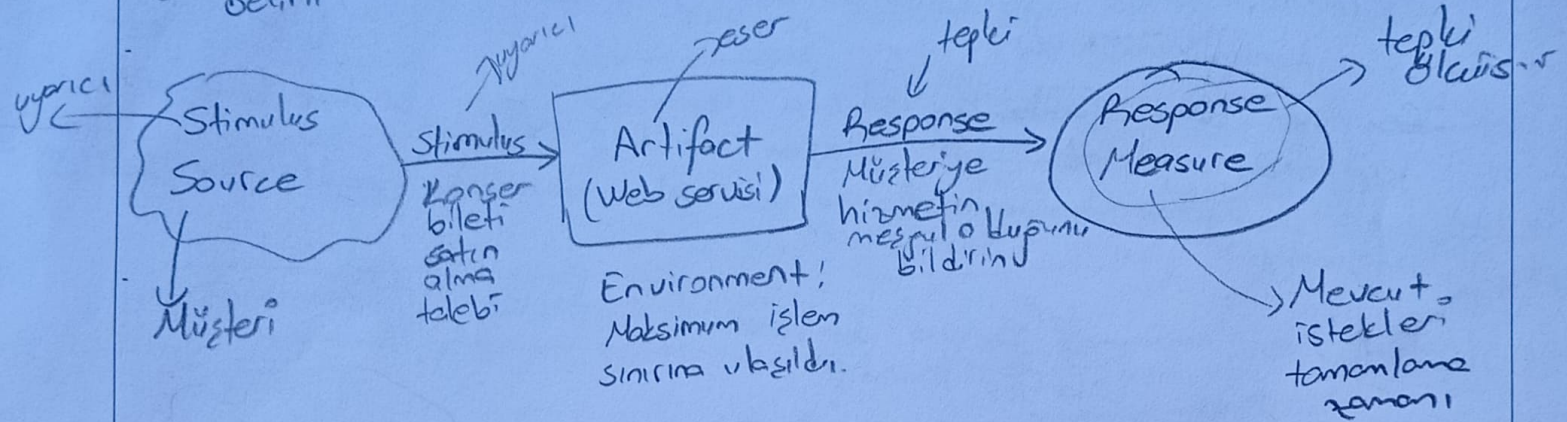
1. Sürdürülebilirlik: Sistemimizin kolaylıkla değişiklik yapma yeteneğine sahip olması.
2. Tekrar Kullanılabilirlik: " " fonksiyonlarının veya bölümlerinin bir başka sistemde ne ölçüde kullanılabilirliği.
3. Esneklik: Bir sistemin gereksinim değişikliğine ne kadar iyi uyum sağlayabileceğidir.
4. Değiştirilebilirlik: " " değişikliklerle başa çıkma, yeni birleşme veya mevcut bir parçanın fonksiyonelliği kaldırma yeteneği.
5. Test Edilebilirlik: Testler hızlı, kolay bir şekilde yapılabilirliğinden ve bir kullanıcı arayüzü gerektirmeden sistemler test edilmelidir. Arına tespit edilir, sistem serbest bırakılmadan önce düzeltilmelidir.

Kullanıcı Tarafından dikkate alınacak kalite özellikleri ne kadar iyi kurtulduğu tespit edilebilir.

1. Erişilebilirlik: Sistemin belirli bir süre boyunca çalıştığı süre. Bir sistemin kullanılabilirliği çalışma süresiyle ölçülür. böylece sistem hataları, yüksek yükler veya pürüzlenmeler gibi sorunlardan.
2. Birlikte çalışabilirlik: Sisteminin iletişimleri araları ve verileri harici sistemlerle paylaşma yeteneği.
3. Güvenlik: Sistemin hassas verileri yetkisiz ve yetkisiz kullanımlara karşı koruma yeteneği.



- Herhangi bir sistemi karakterize etmek için kullanılan genel bir sengyo
- Belirli " " " " " Somut " "



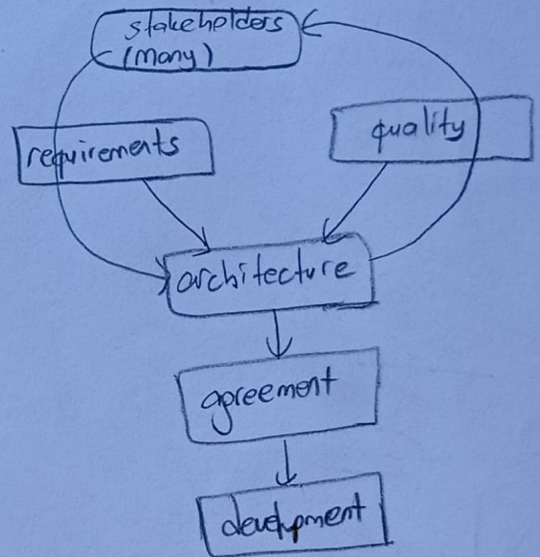
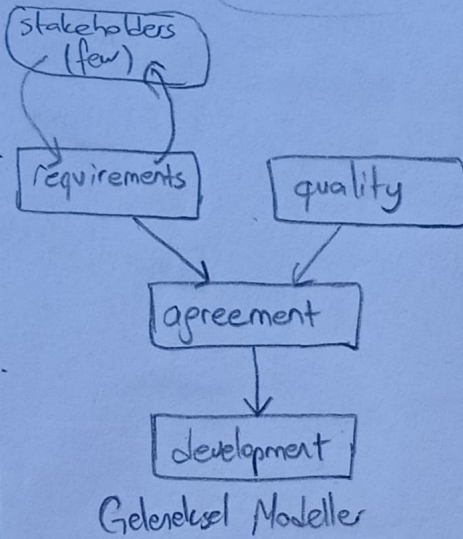


## Yazılım Mimarısının Amacı

" " yazılım sistemlerinin büyük ölçekli yapısı ile ilgilidir.

Gereksel tasarım ise döndürür, bir takım gereksinimler verildiğinde, bu gereksinimleri karşılayan bir sistemin nasıl oluşturulacağını odaklanır.

Y.M. bir yandan fonksiyonel ve kalite gereksinimlerinin tartışılmasını ve değerlendirilmesini, diğer yandan olası çözümleri içerir.



Yatırım mimarisi içeren süreç modelleri