



1906003172019

## Tasarım Desenleri

Dr. Öğr. Üy. Önder EYECİOĞLU  
Bilgisayar Mühendisliği



Hafta	İşlenecek Konu
1	Bölüm 1: Yazılım Tasarımı, Yazılım Örüntüleri ve UML 1.1. Yazılım geliştirme süreci 1.2. Yazılım Tasarımı ve Tasarım Örüntüleri 1.3. UML
2	1.3.1. Sınıf Diyagramı (Class Diagram) 1.4. Java Dökümantasyonu (Javadoc)
3	Bölüm 2: Nesnel Kavramlar 2.1. Modülerlik 2.2. Soyutlama (abstraction) 2.2.1. Genelleştirme ve Özelleştirme 2.3. Sınıf Tasarımı
4	2.3.1. Bilgi Saklama 2.4. Kalıt (inheritance) 2.5. Tür Değiştirme (Type Casting)
5	2.6. Soyut Sınıf (Abstract Class) 2.7. Soyut Metotlar 2.8. Polimorfizm (polimorphism)
6	2.9. Arayüz (interface) 2.10. Nesnenin Yaşam Döngüsü 2.11. Sınıf ve Nesne Verileri 2.12. Java'da Diğer İlgili Konular
7	2.12.1. Java Nesnelerinin Kopyalanması Veya Klonlanması 2.12.2. Paket

Hafta	İşlenecek Konu
8	2.12.3. Java Kütüphaneleri 2.12.4. Java Collections 2.12.5. Java Generics 2.12.6. Java Nesnelerinin Karşılaştırılması
9	Bölüm 3: Tasarım Örüntüleri 3.1. Tasarım Örüntüsü Nedir? 3.2. İncelenen Tasarım Örüntüleri ve Tasarım Kategorileri
10	3.3. Gözlemci Örüntüsü (Observer Pattern) 3.4. Dekorator Örüntüsü (Decorator Pattern) 3.5. Strateji Örüntüsü (Strategy Pattern) 3.5.1. Soyut Fabrika Örüntüsü (Abstract Factory Pattern)
11	3.6. Tekli Örüntü (Singleton Pattern) 3.7. Komut Örüntüsü (Command Pattern) 3.8. Adaptör Örüntüsü (Adapter Pattern)
12	3.9. Fasat Örüntüsü (Façade Pattern) 3.10. Kalıp Metodu Örüntüsü (Template Method Pattern) 3.11. İterasyon Örüntüsü (Iterator Pattern)
13	3.12. Komposit Örüntüsü (Composite Pattern) 3.13. Durum Örüntüsü (State Pattern) 3.14. Proksi Örüntüsü (Proxy Pattern)
14	Genel Tekrar



## DERSİN HEDEFLERİ

- UML, bilgi saklama, sınıf tasarımı, kalıtım (inheritance), arayüz ve soyut sınıflar, dinamik nesne ilişkilendirilmesi, tasarım örüntüleri, uygulama çerçeveleri, uygulama programlama arayüzleri (API).

## Bölüm Hedefi

Yazılım tasarımı, yazılım geliştirme sürecinde en önemli adımlardan birisidir. Bu bölümde yazılım geliştirme sürecinde yazılım tasarımının önemi anlatılacak, yazılım örüntülerinin yazılım tasarımında önemi açıklanacak ve ayrıca nesnesel yazılım tasarımında kullanılan önemli bir görsel tasarım dili olan Unified Modelling Language (UML) anlatılacaktır.

Yazılım geliştirme süreci belirli bir gereksinimi karşılayacak veya bir problemi çözecek işlemlerin sayısal ortamlarda gerçekleştirilmesi için gerekli yazılım tasarım aşamaları olarak tanımlanır.

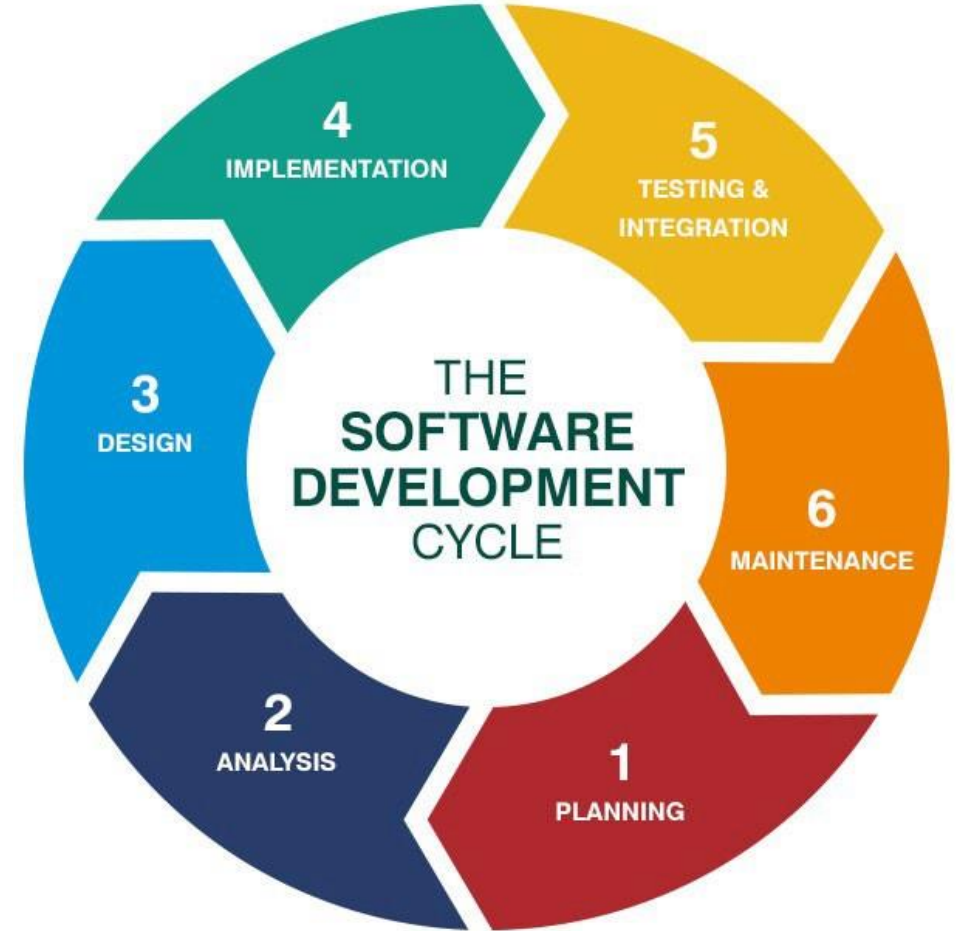
Yazılım, bir mühendislik projesidir.

Yazılım Geliştirme Yaşam Döngüsü (SDLC) bu iki yaklaşımın (tüketim ürünü ve mühendislik ürünü yaklaşımlarının) ortasındadır.

Uzun yıllar boyunca yazılımın yaşaması istenilir.

Yazılım Geliştirme Yaşam Döngüsü (SDLC) kavramı bir döngüsel (cyclic) yaklaşımdır. Bir sarmal hareket halinde olan bir yaklaşımdır. SDLC aşamaları aşağıdaki şekilde sıralanabilir.

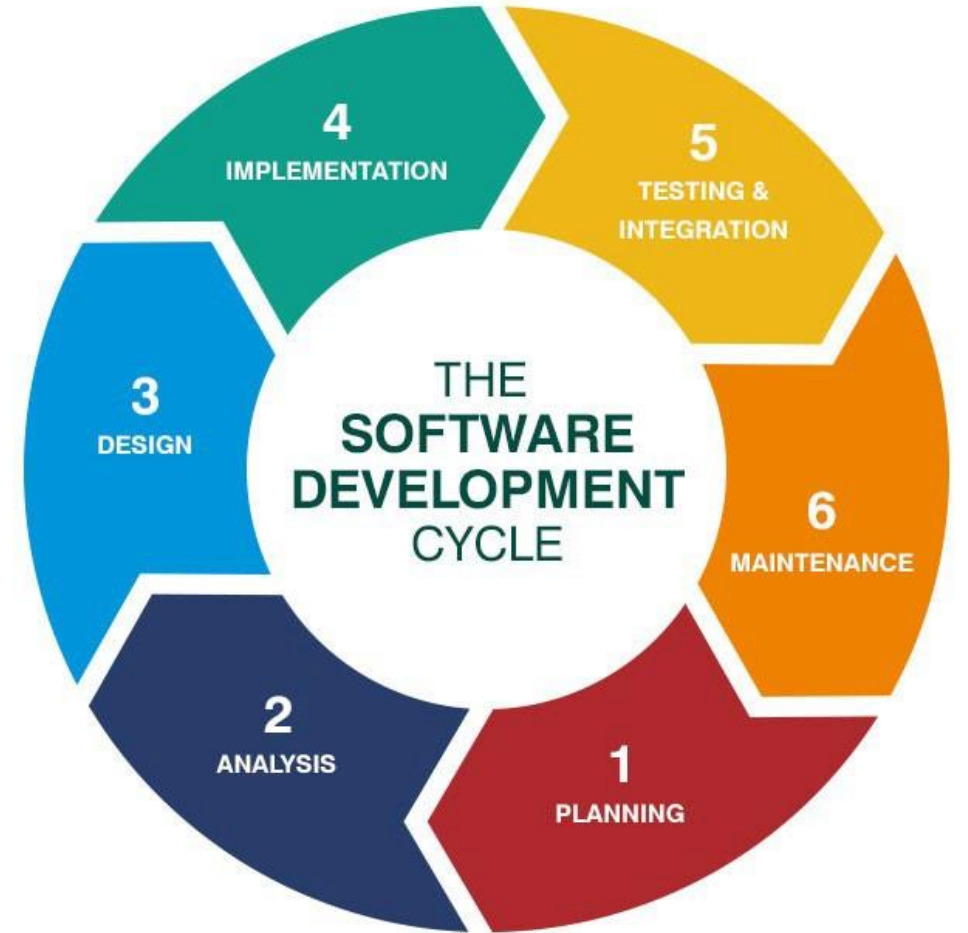
1. Planlama
2. Analiz -Tanımlama
3. Tasarım
4. Geliştirme Uygulama
5. Entegrasyon ve testler
6. Bakım



## 1. Planlama:

Plan aşaması aslında proje yönetimlerinde en önemli aşamasıdır. Diğer adımların da başlangıç aşamasıdır. İşin projelendirildiği, fikrin ortaya çıkarıldığı ve fikrin tartışıldığı aşamadır.

Personel ve donanım gereksinimlerinin çıkarıldığı, fizibilite çalışmasının yapıldığı ve proje planının oluşturulduğu aşamadır.



## 2. Analiz –Tanımlama

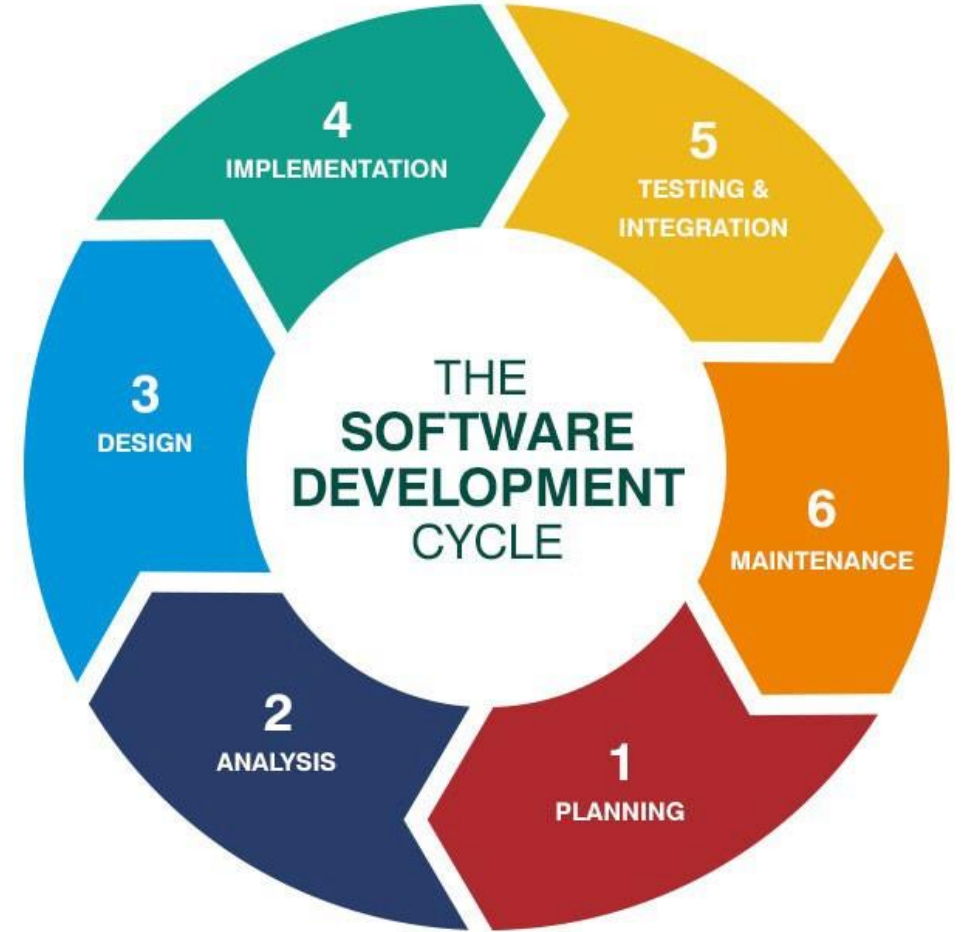
Sistem gereksinimlerinin ve işlevlerinin ayrıntılı olarak çıkarıldığı aşama. Var olan işler incelenir, temel sorunlar ortaya çıkarılır

Genelde istenilen fikrin ne olduğu ve temel tanımlar üzerinde konuşulacak kavramların tanımlandığı aşamadır.

Çünkü aynı fikirden bahsedilmiyorsa farklı sonuçlara varılabilir. Bu tanımlar üzerine bir tasarım vardır. Bu analiz aşaması olarak da düşünülebilir.

Problemin tanımlandığı veya yaşam döngüsünün tanımlandığı sistemin veyahut yazılımın tanımlandığı aşamadır.

Tanım aşamasında projede nelerin istenildiği ile ilgili analiz çalışmaları da yapılabilir.





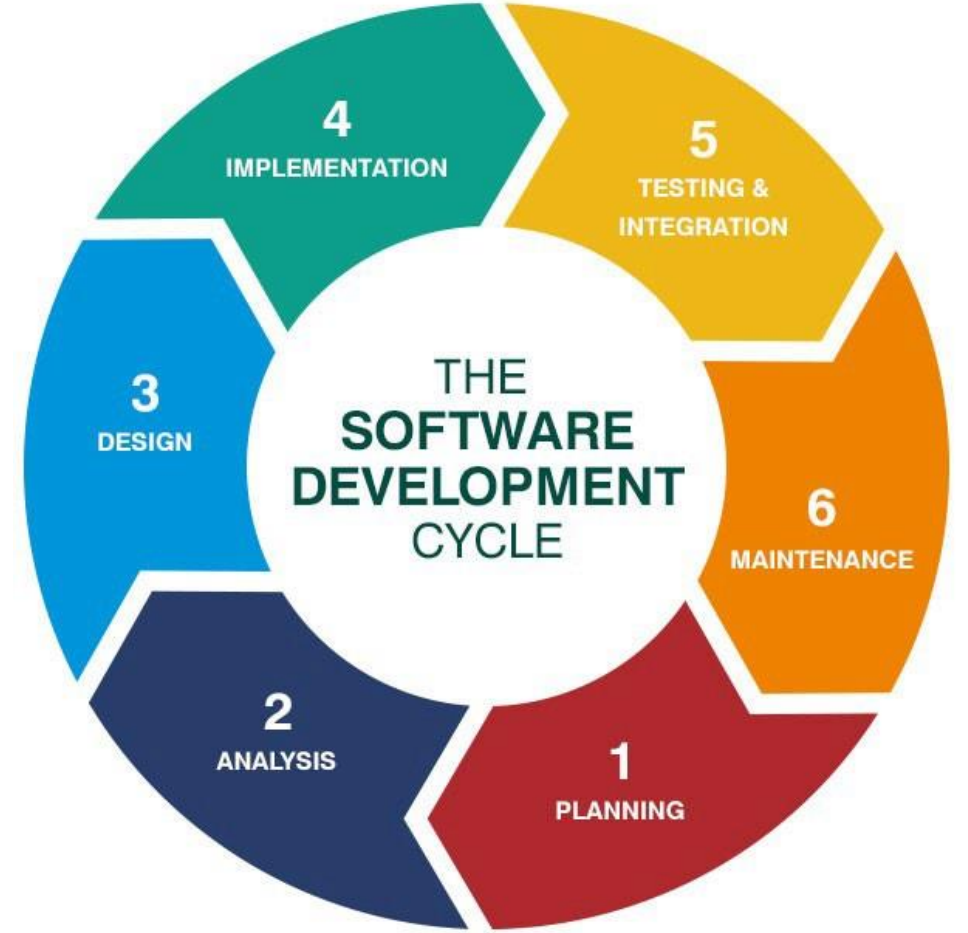
### 3. Tasarım

Belirlenen gereksinimlere yanıt verecek yazılım sisteminin temel yapısının oluşturulduğu aşamadır.

**mantıksal;** önerilen sistemin yapısı anlatılır,

**fiziksel;** yazılımı içeren bileşenler ve bunların ayrıntıları.

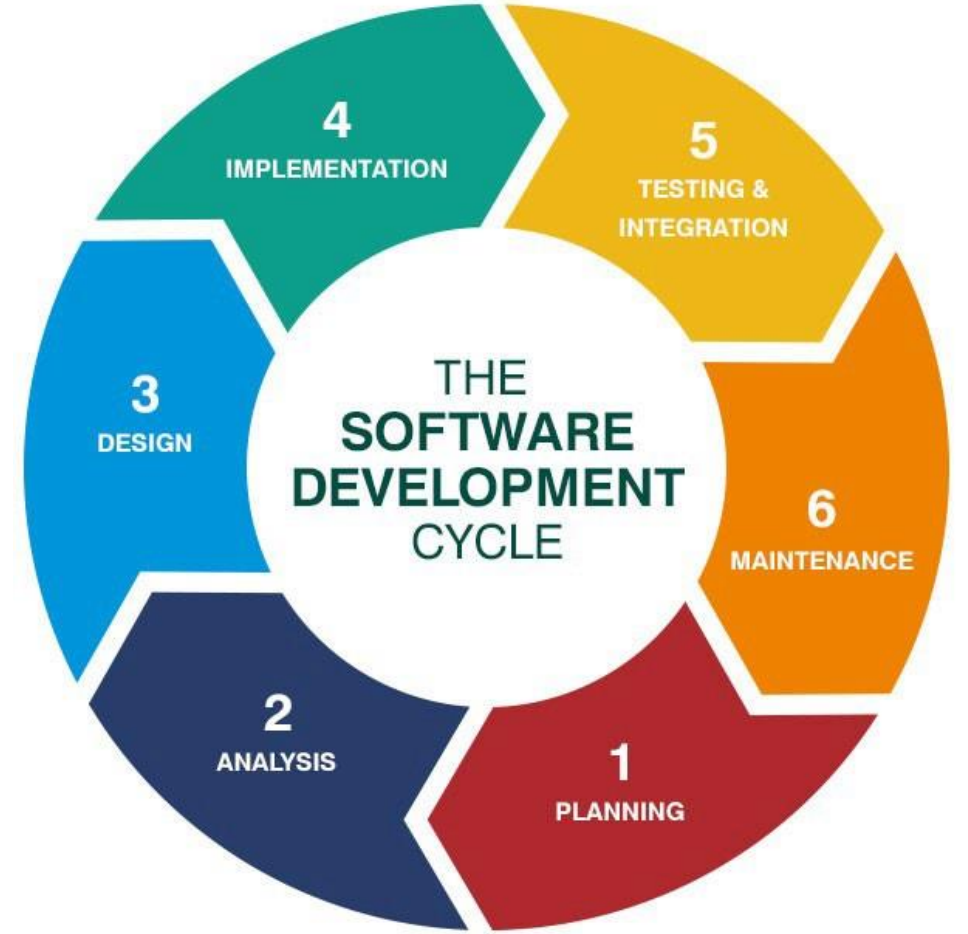
Yazılımımızın veya sistemimizin tasarımları yapılır. Projeleri çizilir. İnşaat projeleri gibi düşünülebilir. Planlama ve tanımlaya göre bir tasarım çizilir. Kararlar verilir, seçimler yapılır, örneğin yazılımın ekranları, ekranlarda neler bulunacağı, hangi ekranlara nasıl geçileceği, fonksiyonel olarak hangi adımların oluşturulacağı, hatta yazılımın bileşenleri, modülleri bu aşamada tasarlanır. Bir sonraki adım olan uygulama/geliştirmeye bütün kararlar verilmiş olarak geçilmesi beklenir. Geliştirme aşamasında herhangi bir soru veya karar bırakılmaz.



#### 4. Geliştirme Uygulama

Bu aşama, yazılım projeleri için kodlama olarak düşünülebilir veya sistemin yaşatılmaya başlandığı ilk örneklerinin çıkmaya başladığı aşama olarak düşünülebilir.

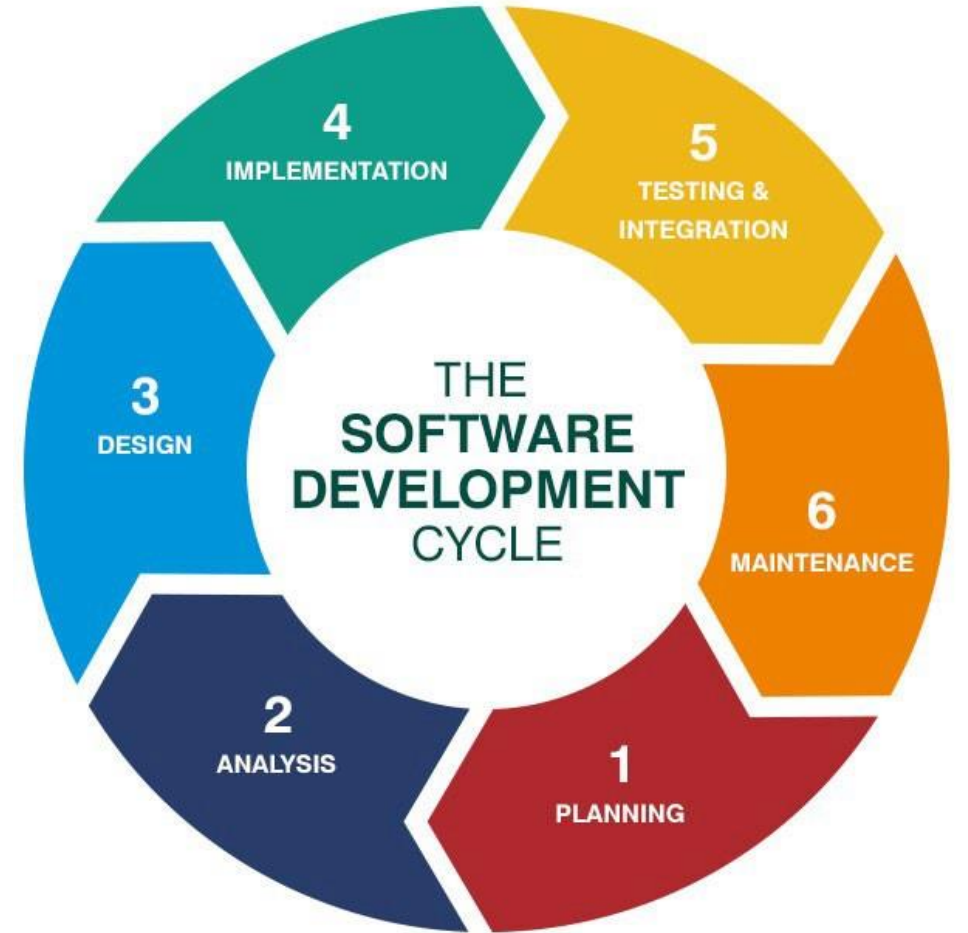
Daha önceden tasarım aşamasında karar verilen ortama uygun olarak, yine tasarım aşamasında verilen kararlar doğrultusunda projenin gerçekleştirilmesine başlanır. Yazılım Projelerinin kodlandığı aşamadır.



## 5. Entegrasyon ve testler

Sistemin artık gerçek hayata geçtiği aşamadır. Sistem veya yazılım hayat içinde bir yer edinir.

Bir şirkette kullanılacak yazılımdan bahsediliyorsa şirketle ilgili birimlerin yazılımla ilgili eğitimlerin alınması, donanımlarının temin edilmesi, bağlantılı olduğu birimlerin buna entegre edilmesi, başka yazılımlarla bir entegrasyonu varsa bunun sağlanması ve veri tabanı bağlantıları gibi pek çok adım bu aşamada ele alınır.



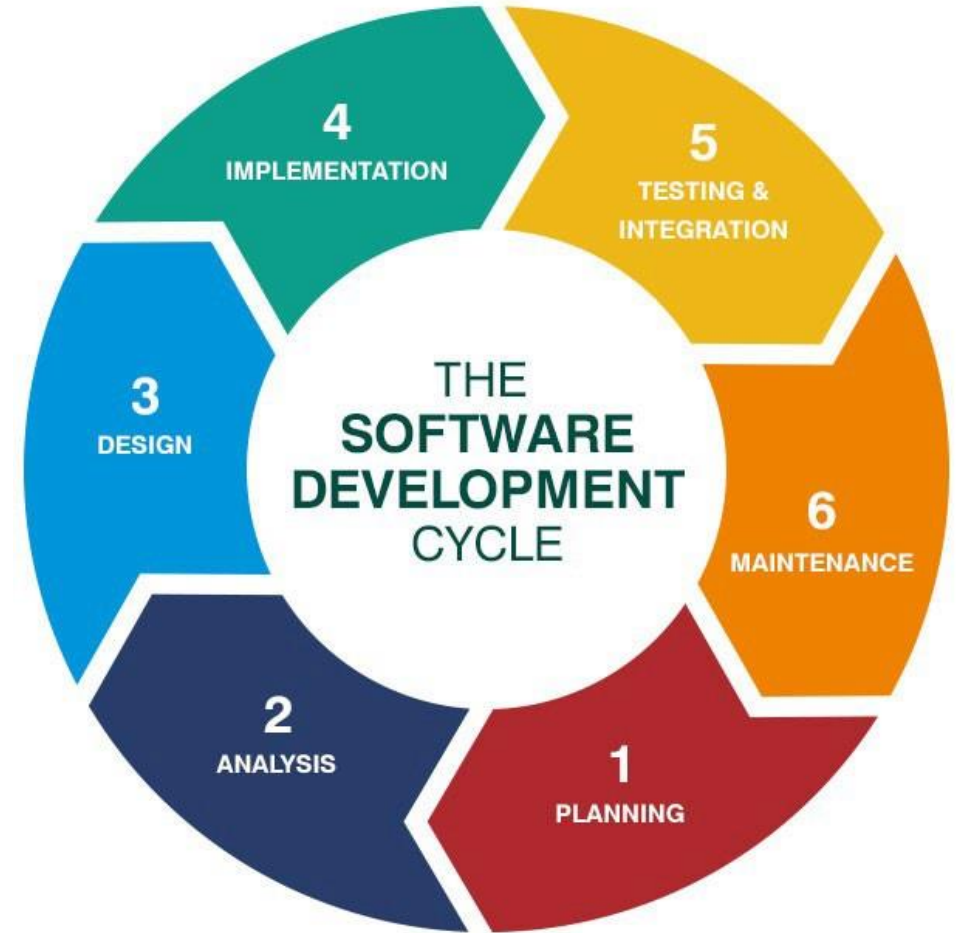
## 5. Entegrasyon ve testler

Test aşaması yazılım geliştirme çevriminde her aşamada yapılmalıdır!

Yazılım sürecinde ilerledikçe, ortaya çıkabilecek hataların giderilme maliyeti üstel olarak artar.

Aksi gibi, hataların büyük çoğunluğunun kökenleri isteklerin belirlenmesi ve tasarım aşamalarındaki sorunlara dayanır.

Kalite ve maliyet açısından erkenden, ve sık sık test yapılmalıdır



## 6. Bakım

Bu aşama yazılım tesliminden sonra gerçekleştirilen aşamadır.

Hata giderme ve yeni eklentiler yapma aşamasıdır. Yazılımın faaliyete geçirilmesinden sonra sistemde yapılan değişikliklerdir.

Yazılım hatalarının düzeltilmesi:

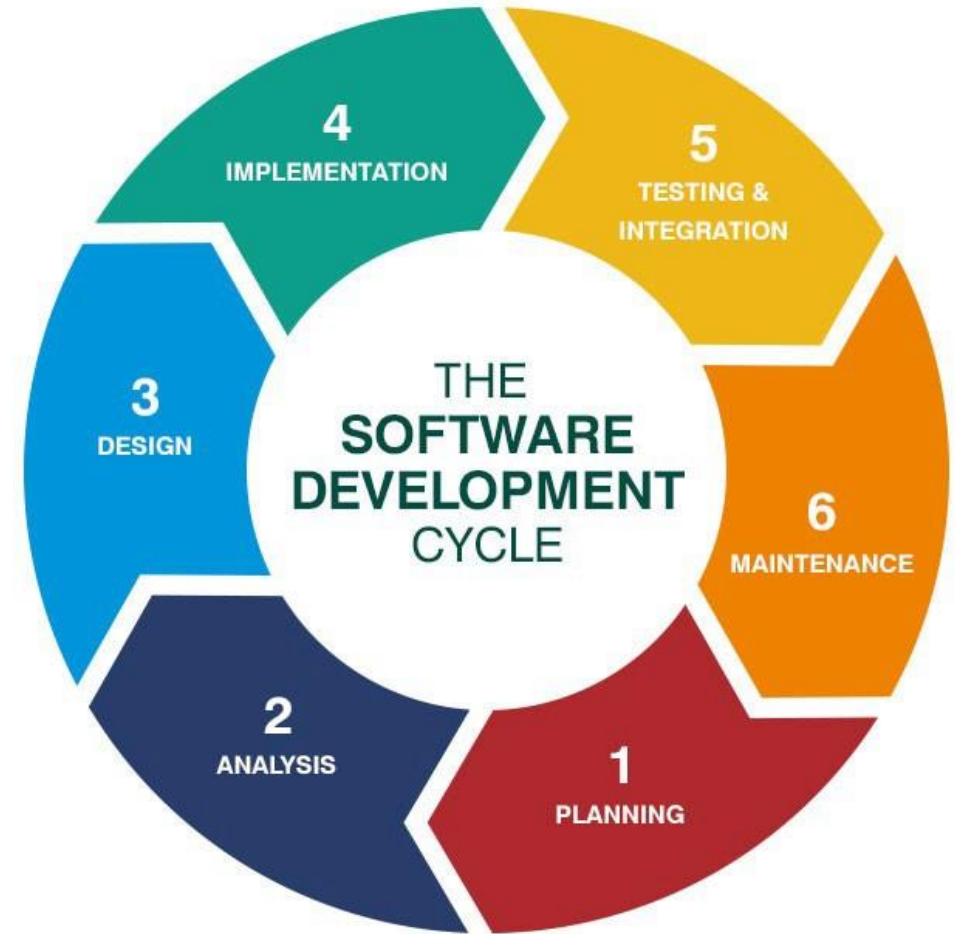
Kodlama hataları

Tasarım hataları (!)

Gereksinim ve analiz hataları (!!)

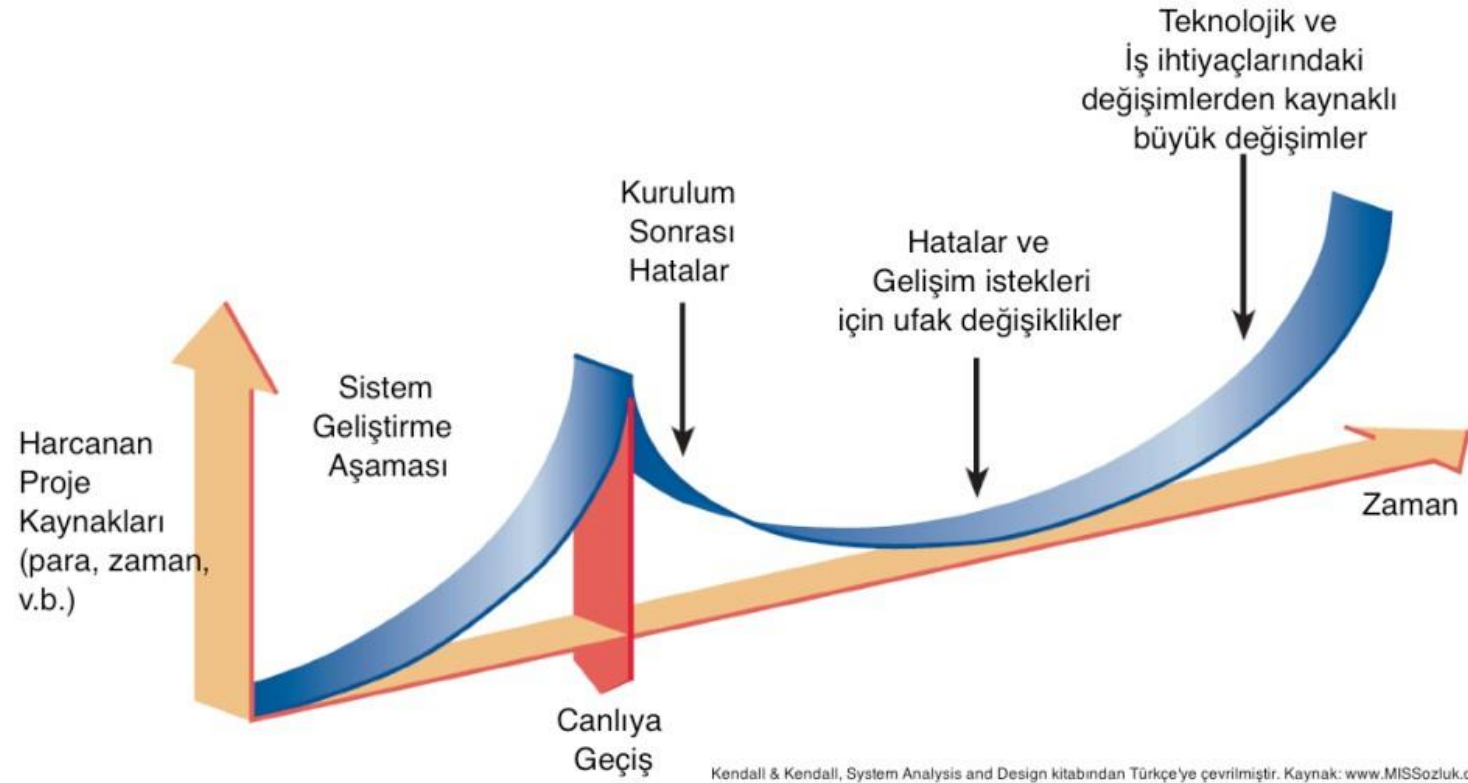
Sistemin işlevlerini değiştirme veya işlevlere eklemeler/çıkarmalar,

Yazılımın farklı bir ortama taşınması (programlama dili, işletim sistemi, donanım, iklim, vb.)





## Kendall, SDLC Proje maliyet Grafiği



Yazılım geliřtirmede temel alınan çeřitli modeller vardır:

- řelale (Waterfall) modeli
- Prototip oluřturma
- Spiral modeli
- Dördüncü nesil teknikler



Tekrar

İlk yayınlanan yazılım geliştirme süreç modelidir  
Proje yönetim modelleri genelde şelale modeli (waterfall model) ile başlar. Yazılım mühendisliğinde ilk anlatılan modellerden birisi budur.

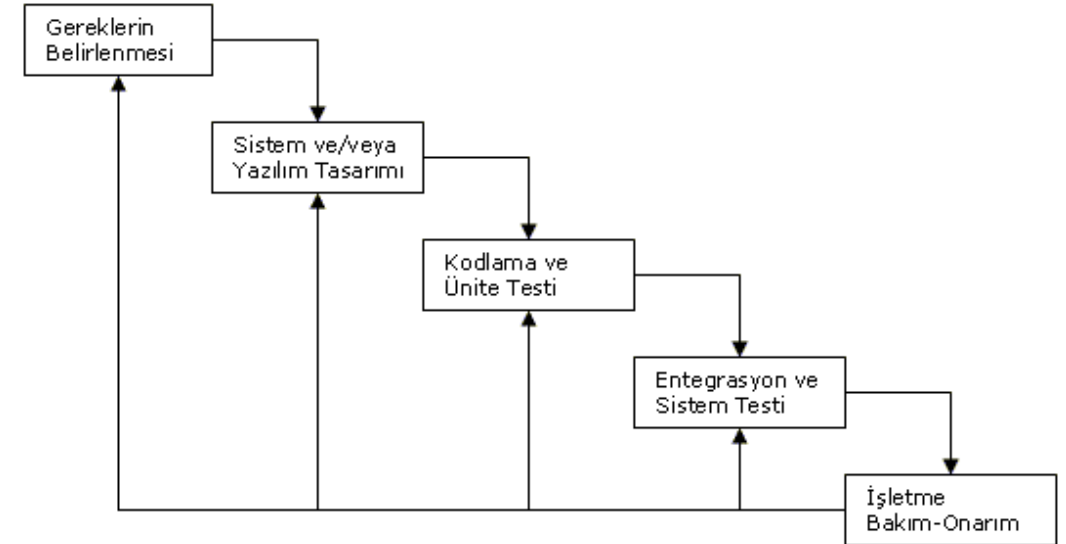
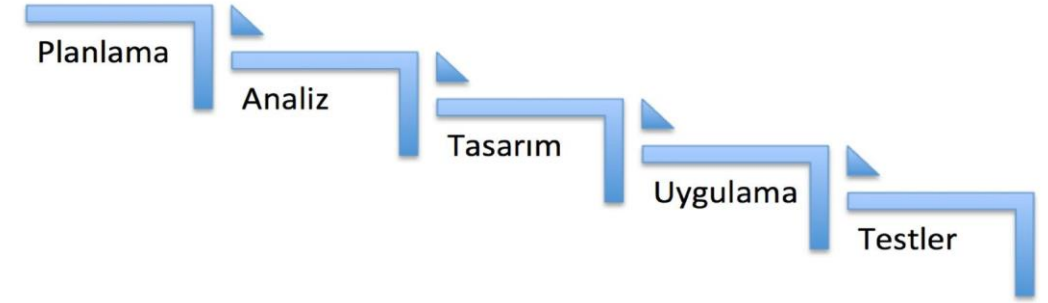
Statiktir.?!?

Aşamaları:

Bir sonraki aşama bir önceki aşama bitmeden başlamamaktadır.

Dokümantasyon odaklıdır. (Maliyetli)

Problemler son aşamaya bırakılır.

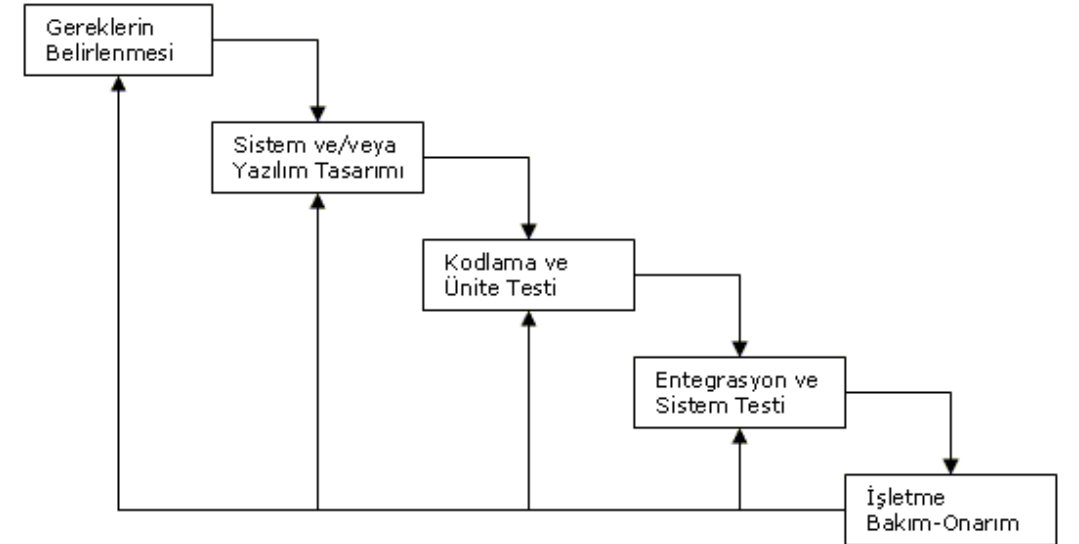
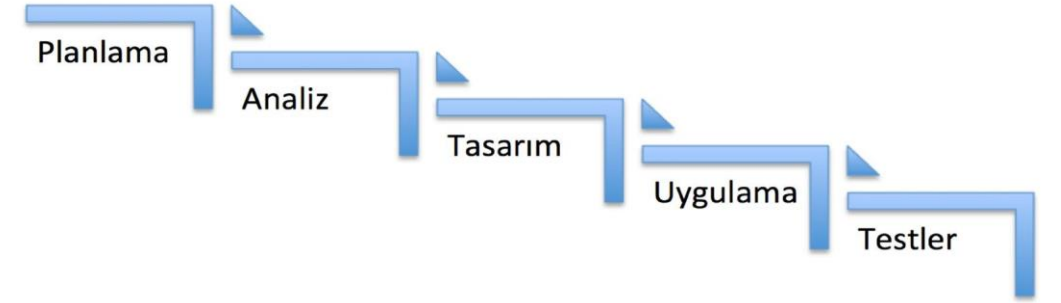




Yaşam döngüsü temel adımları **baştan sona en az bir kez** izleyerek gerçekleştirilir.

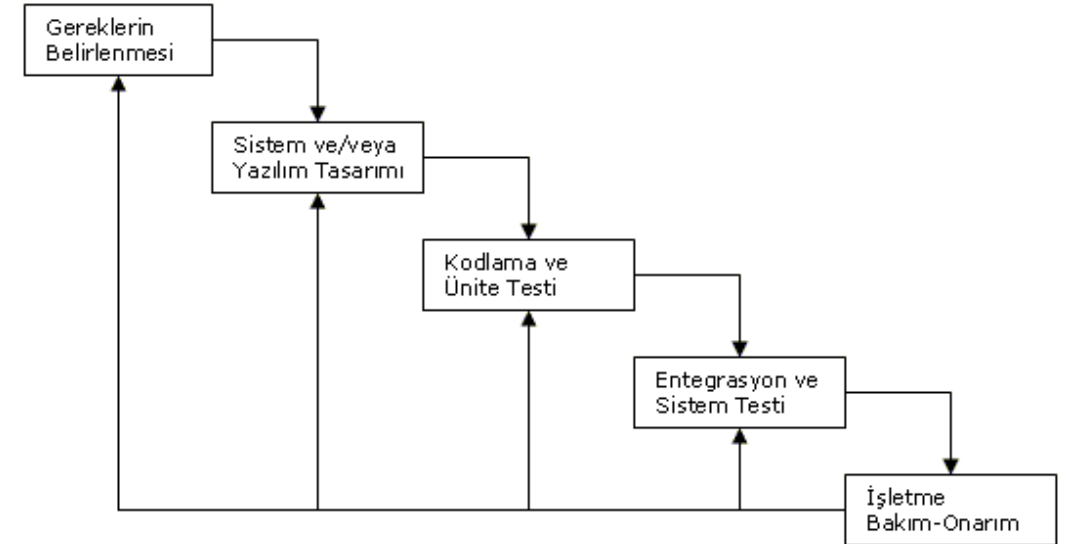
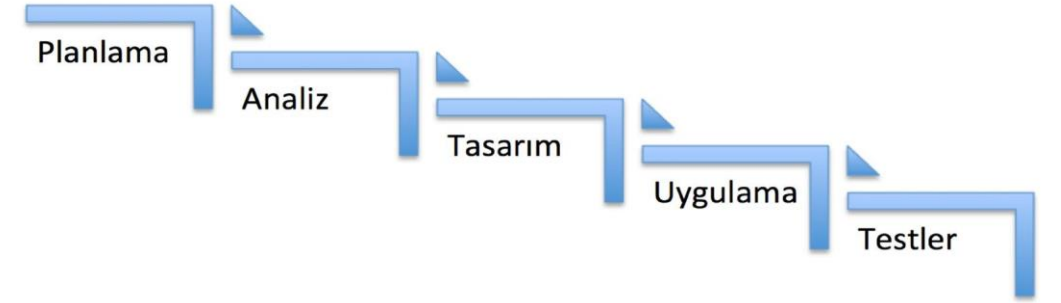
Barok modelin aksine **belgeleme** işlevini ayrı bir aşama olarak ele almaz ve üretimin doğal bir parçası olarak görür.

**İyi tanımlı** projeler ve **üretimi az zaman gerektiren** yazılım projeleri için uygun bir modeldir.



Projeyi bu modelde olduđu gibi esnek olmayan safhalara ayırmak doğru değildir. Bu nedenle Şelale modelinin müşteri ihtiyaçları tam ve çok iyi anlaşıldığında kullanılması tavsiye edilmektedir. Sonuç olarak büyük sistemlerde, mühendislik projelerinde kullanılmaktadır.

En basit, en etkili ve diğer modelleri temeli olan yöntemdir.



## ELEŖTİRİ:

Ŗelale modelinin en zayıf yönü olan ve geri dönüŖe izin vermeyen yapı olmasıdır.

David Parnas, “Mantikli bir sistemi aldatmak” (Parnas, 1986):

“Çoğu sistemin detayları ancak sistem üzerinde çalışmaya başladıktan sonra anlaşılmaktadır.

Yapılan işlemlerin bazıları, hata yaptığımızı göstermekte ve geri dönmemizi gerektirmektedir”

Bilişim teknolojileri konusunda tecrübesi olmayan paydaşların teknolojik olarak neler yapılabileceğinden haberdar olmaması yüzünden isteklerinin değışime açık olması

Proje ihtiyaçlarının zaman içerisinde değışime açık olması

Proje paydaşlarının isteklerini doğru şekilde aktaramaması

Proje ekibinin acemiliğı veya kodlama öncesi aşamalardaki kaynak eksiklikleri (nitelik ve nicelik anlamındaki eksikler)

Projenin ilerideki şeklinin önceden kestirilememesi

## ARTILARI

iki kere ölçüp bir kere biçmek yaklaşımının kurumda uygulanmasını sağlaması

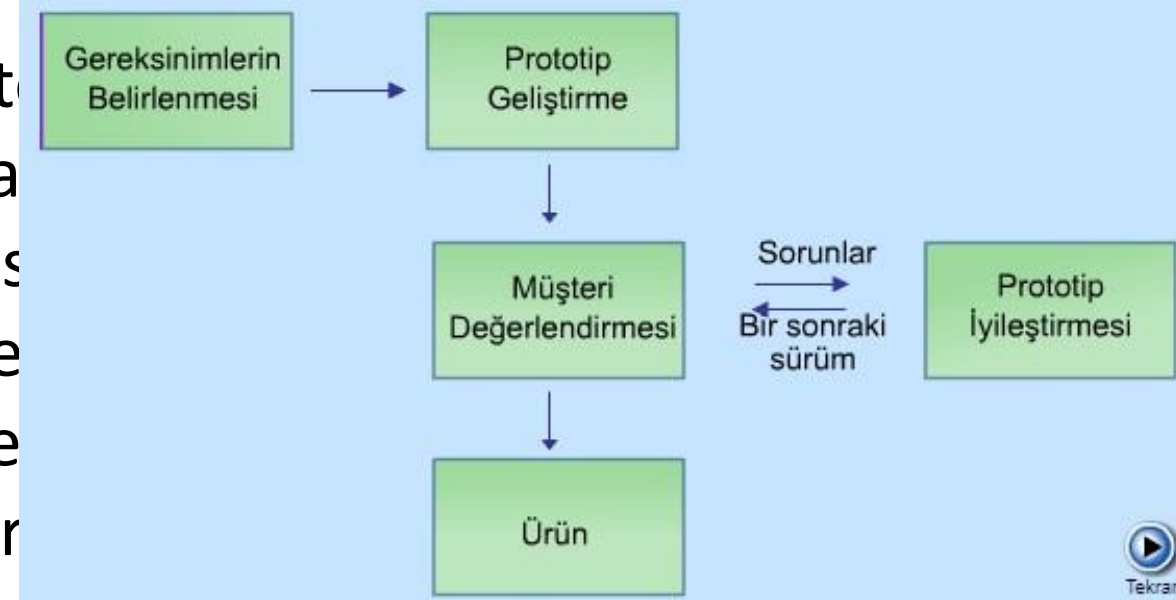
Proje dokümantasyonu hazırlanmaya zorlaması

Hataların hangi aşamalarda ve kimin tarafından kaynaklandığının bulunmasını kolaylaştırması

Her sonraki aşamaya geçmeden önce, önceki aşamanın tamamlanması, bu sayede önceki aşamada ayrılan kaynakların serbest kalması ve farklı projelere aktarılabilmesi

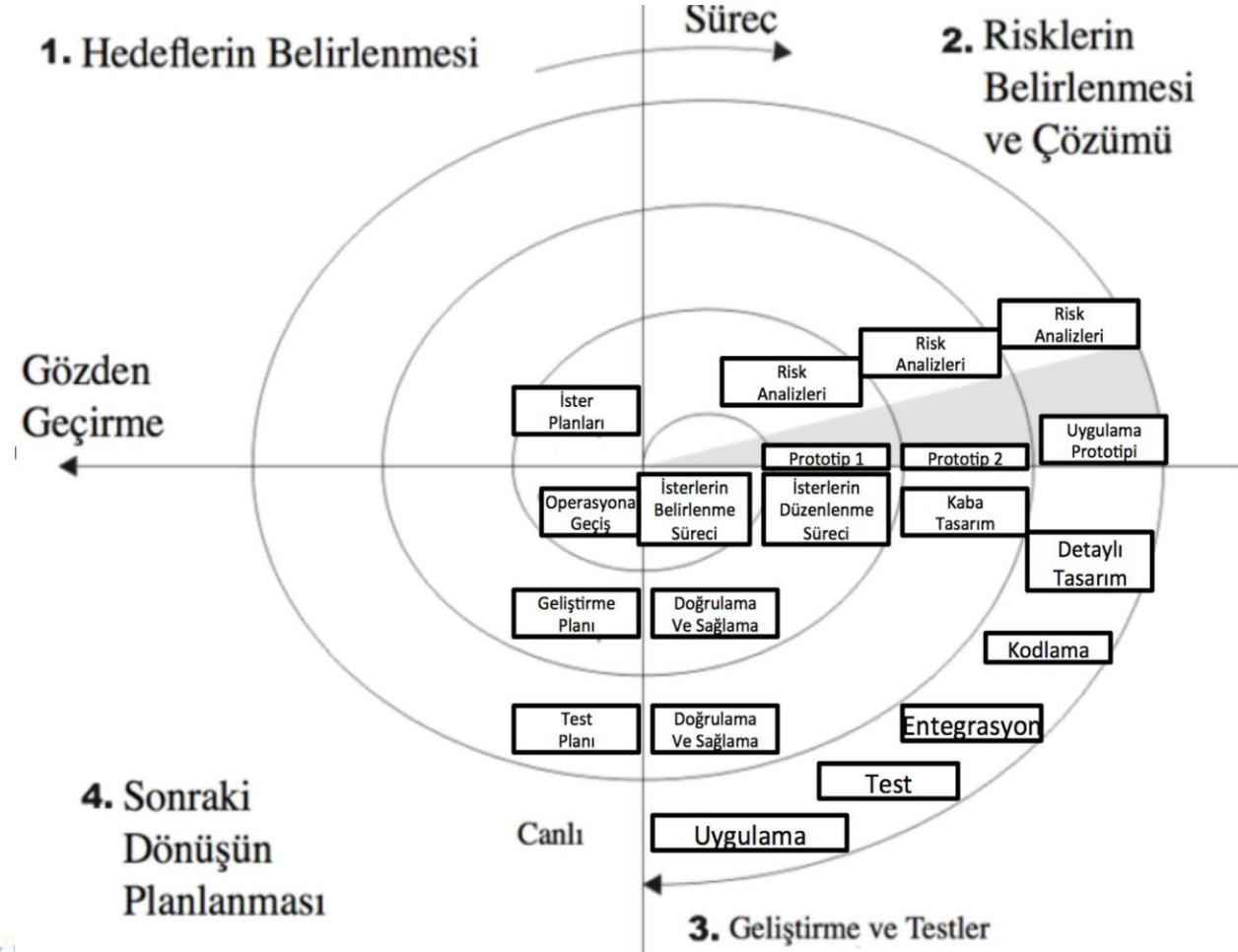
Genellikle, müşteri yazılımın genel hatlarını belirler, fakat girdi, işlemler ve çıktı konusunda ayrıntılı olarak bilgi sağlamaz. Bunun yanı sıra, geliştirici algoritmanın kesinliğinden, işletim sisteminin uygunluğundan emin olamaz. Bu tür durumlarda prototip modeli en iyi yaklaşımdır. Prototip oluşturma, geliştiricinin yapacak olduğu yazılımın bir modelini yaratması işidir.

Sistem analist, kullanıcılarla birlikte çalışarak ve prototip oluşturmaya bir geliştirme tekniği olarak kullanarak, organizasyon içindeki ilk veya temel sistem gereksinimleri (müşteri) prototip ile çalışarak beğenmedikleri özellikleri hakkında sistem analist, kendisine verilen geri besleme üzerinde iyileştirmeler yapar ve yeni versiyon sunar. Bu iterasyonlu süreç, kullanıcılar kadar sürer.



Spiral model, diğer modellerden farklı olarak süreci oluşturan aşamalardan tekrar tekrar geçilmesini ve her geçişte projenin ilerleme kat etmesini hedeflemektedir. Bu spirali de 4'e böldüğümüzde 4 aşaması vardır.

1. Hedeflerin belirlenmesi
2. Risklerin belirlenmesi ve çözümü
3. Geliştirme ve testler
4. Sonraki dönüşün planlanması



## 1. Planlama

Üretilcek ara ürün için planlama, amaç belirleme, bir önceki adımda üretilen ara ürün ile bütünleştirme

## 2. Risk Analizi

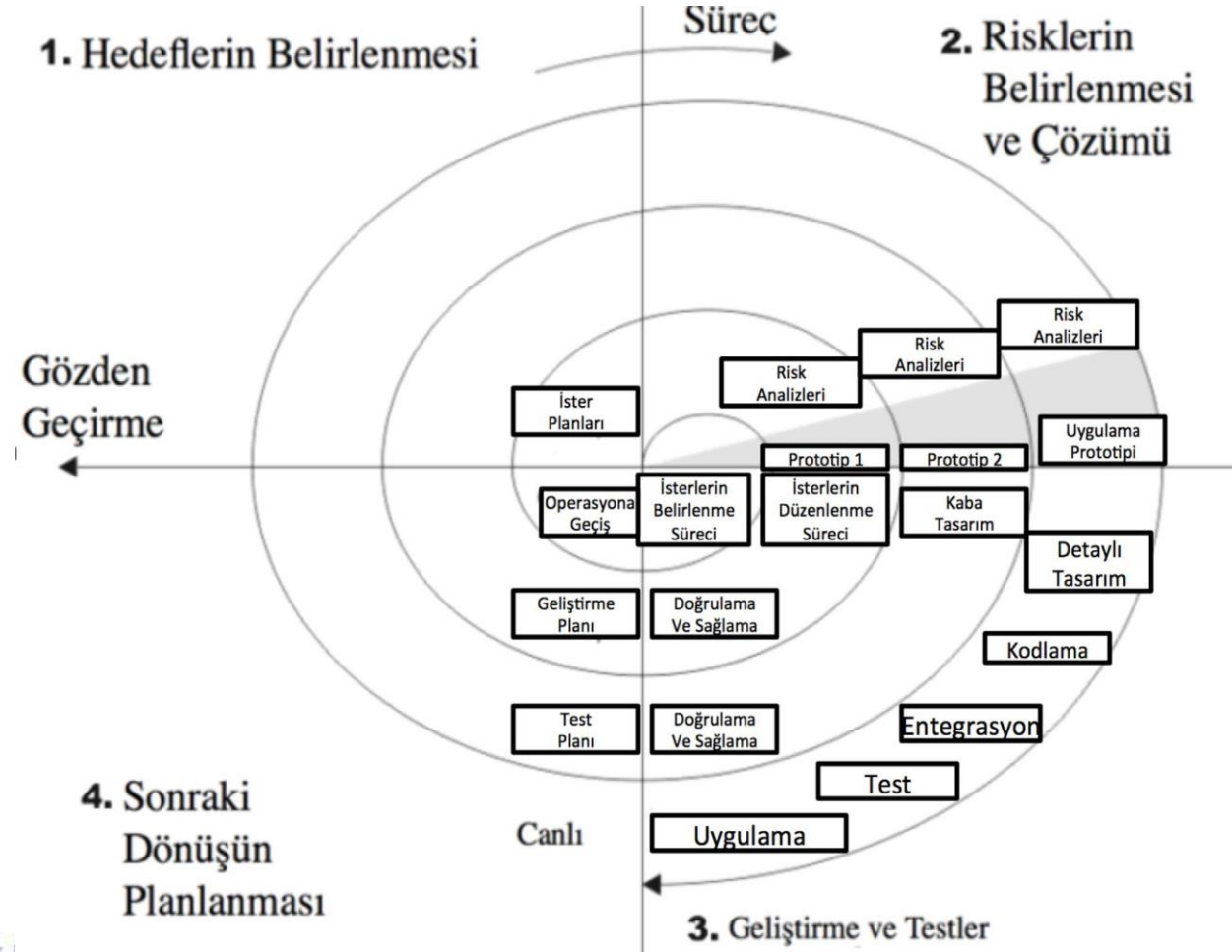
Risk seçeneklerinin araştırılması ve risklerin belirlenmesi

## 3. Üretim

Ara ürünün üretilmesi

## 4. Kullanıcı Değerlendirmesi

Ara ürün ile ilgili olarak kullanıcı tarafından yapılan sına ve değerlendirmeler





Modeli oluşturan her döngü (sistemin olabilirliği, ihtiyaç tanımlama, tasarım,...) amaçların, sınırların tanımlanması, risklerin belirlenmesi, gelişim ve geçerliliğin sağlanması, bir sonraki seviyenin planlanması, aşamalarına tabi tutulur.

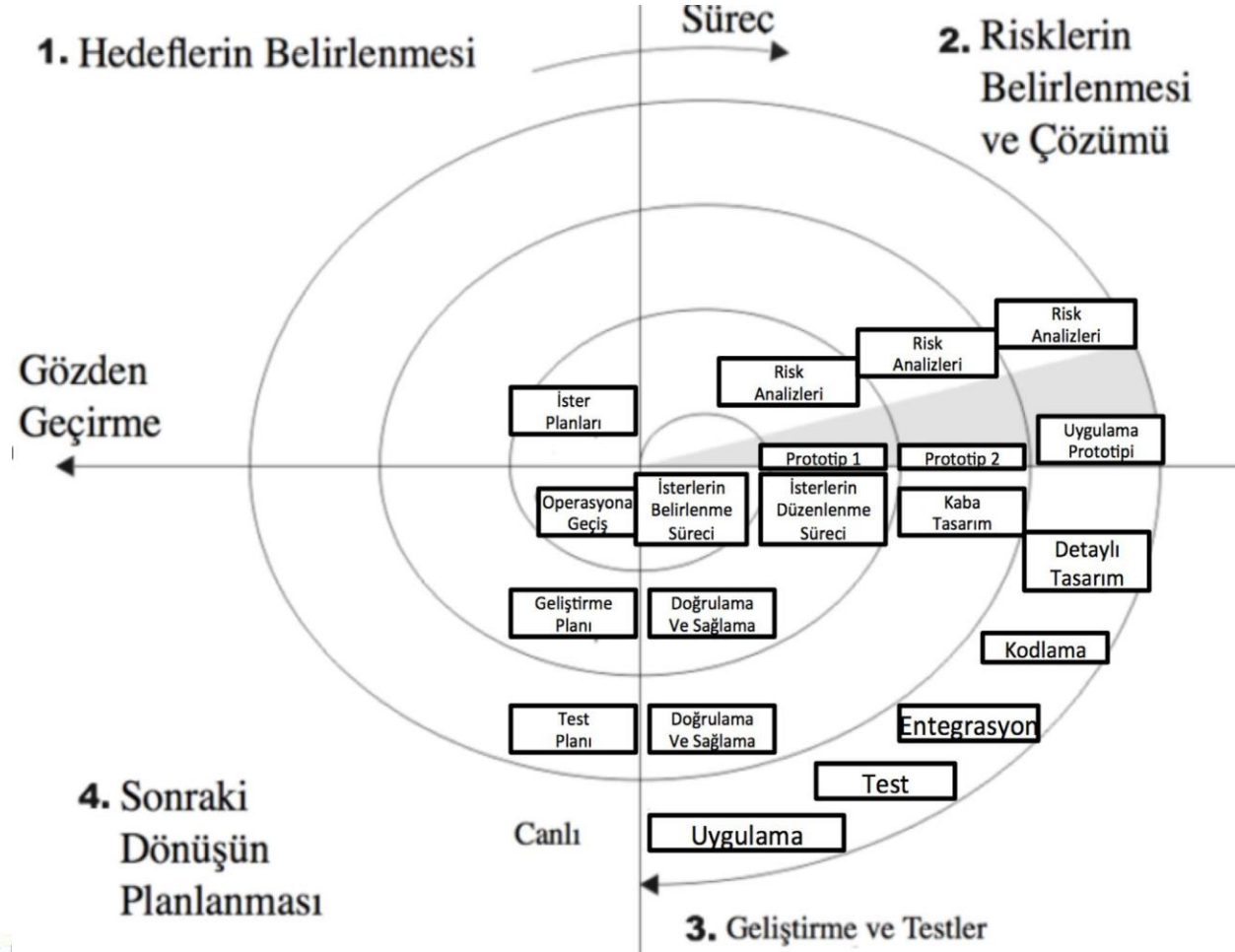
Risk Analizi Olgusu ön plana çıkmıştır.

Her döngü bir fazı ifade eder. Doğrudan tanımlama, tasarım,... vs gibi bir faz yoktur.

Yinelemeli artımsal bir yaklaşım vardır.

Prototip yaklaşımı vardır.

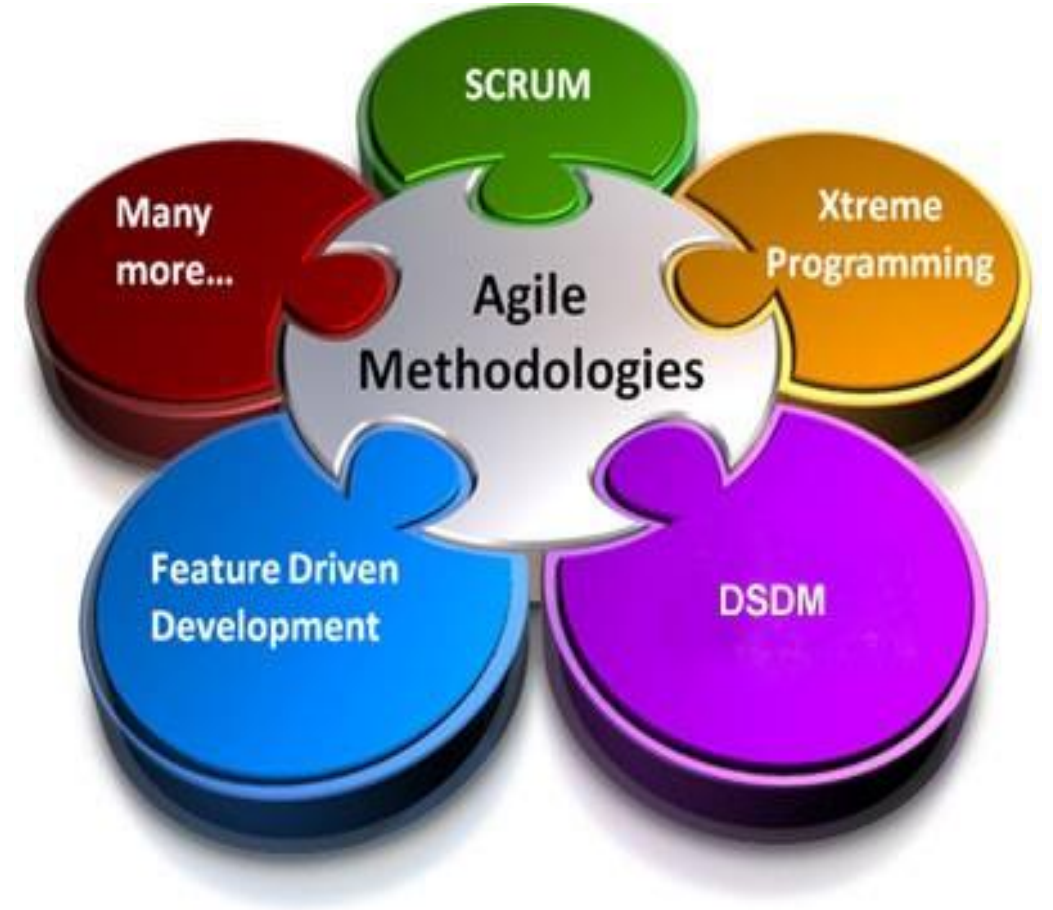
Odaklandığı nokta, bir sonraki riskleri azaltmaktır. Risk yönetimi kullanılması, farkını oluşturur (Sommerville, 2000).



Dördüncü nesil teknikler başlığı, birçok yazılım aracını içerir. Bu araçlar yüksek düzeyde yazılımcının yazılımın karakteristiklerini belirlemesini sağlar. Araç ayrıca otomatik olarak, geliştiricinin spesifikasyonlarına göre kodu da hazırlar.

Bu teknikler, doğal dile en yakın yazılımın belirlenmesi üzerine yoğunlaşır.

Dördüncü nesil teknikleri destekleyen araçlar genel olarak şunlardır; veritabanı sorgusu için prosedürel olmayan diller, rapor yaratma, veri işleme, ekran tanımlama, kod yaratma, yüksek düzeyli grafikler. Tüm bu araçlar mevcuttur fakat spesifik alanlarda kullanılmaktadırlar.



Müşteri, mühendislik işini değerlendirir ve önerilerde bulunur. Bu verilere göre, tekrar planlama ve risk analizi aşamaları gerçekleştirilir. Risk analizinin sonucuna göre, projeye devam edilir ya da durdurulur.

Bu model de diğer modeller gibi her derde deva değildir. Her müşteriyi bu modelin kontrollü olduğuna ikna edebilmek zordur. Modelin uygulanması, risk kestirim deneyimi gerektirir. Eğer büyük bir risk farkedilmezse, problemler şüphesiz olarak oluşur. Sonuç olarak bu model yenidir, diğer iki model kadar sıklıkla kullanılmamaktadır.



Bilgi sistemlerinin tasarımında 3 yaklaşımdan söz edilebilir. Bunlar:





- Veri akışı yaklaşımı
- Veri yapısı yaklaşımı
- Nesne yönelimli yaklaşım



Tekrar

Veri akışı yaklaşımı, her bir yazılım birimini belirli girdileri belirli çıktılara dönüştüren bir birim olarak ele alır. Bu yaklaşımın en belirgin aracı Veri Akış Diyagramları'dır.

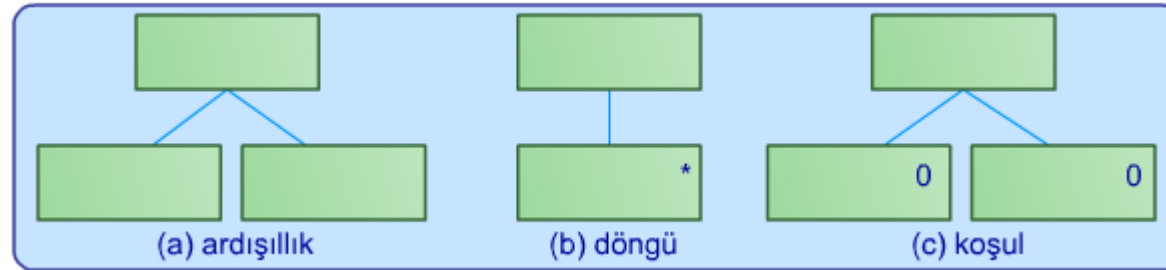
Veri Akış Diyagramları'nda kullanılan simgeler gösterilmektedir.

	Süreç
	Veri Akışı
	Veri kullanıcısı ya da veri kaynağı
	Kütük

İşletmecilik yazılımlarında işlenen verilerin çeşitliliğinde var olan karmaşıklık, yazılımın veri yapısı dikkate alınarak geliştirilmesini gerektirir. Veri yapısı yaklaşımının temel felsefesi de yazılım yapısının, işlenecek veri yapısıyla uyumlu olması gereğidir.

Bu yaklaşımda veri yapılarının belirtiminden yola çıkılır ve aşamalı olarak ayrıntılandırma yapılır. Tasarımcı veri kümelerine ve alt kümelerine odaklanır.

### Veri Yapısı Diyagramları



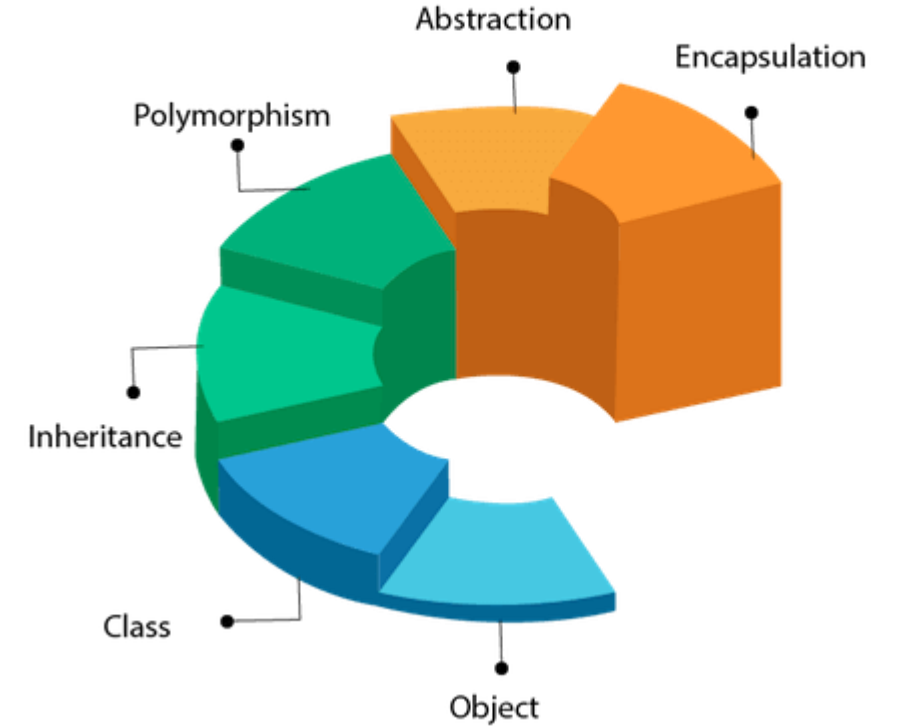
## OOPs (Object-Oriented Programming System)

Nesne yönelimli yaklaşım gerçek dünyayı nesneler ve aralarında alıp verdikleri mesajlar ile modeller.

Yaşadığımız dünyada herşey birer nesnedir. Örneğin kişiler, kütükler, sistemler, hepsi birer nesnedir.

Her nesnenin bir çalışma tarzı ve kendine has dışa yönelik davranışları vardır. Bir nesne ile interaksiyona girmek için, nesnenin içinde ne bulunduğunu bilmemize gerek yoktur.

Belirli arayüzler (Interface) üzerinden mesajlar göndererek, nesne ile bağlantı kurulur. Nesne bu mesajı alır ve gerekli işlemleri yapar.





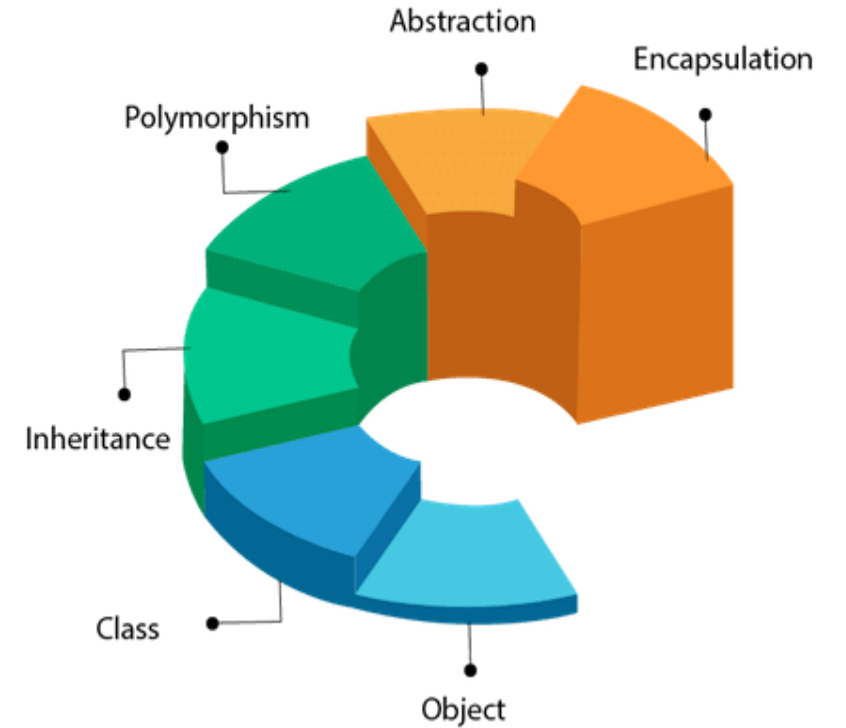
Bu yaklaşımda ifade edilmek istenen, sistem geliřtirmede her ne kadar veri ve prosesler birbirinden bağımsız, ayrı bir şekilde tutulsa da, veri ve proseslerin birlikte paketlenerek tutulması anlamlıdır.

Nesne yönelimli tasarımın amacı, sistem elemanlarını daha fazla **tekrar kullanılabilir (reusable)** hale getirerek sistem kalitesini iyileřtirmek ve sistem analiz ve tasarımının üretkenliğini artırmaktır.

Nesne yönelimliliğın ardındaki bir diğér anahtar fikir, **kalıtsallık (inheritance)**'tır. Nesneler, yapısal ve davranıřsal karakteristikleri paylařan nesne grupları řeklinde tanımlanan **nesne sınıfları (object classes)** řeklinde düzenlenirler.

Kalıtsallık, mevcut sınıfların bazı karakteristiklerini paylařan yeni sınıfların (class) yaratılmasına olanak tanır.

## OOPs (Object-Oriented Programming System)



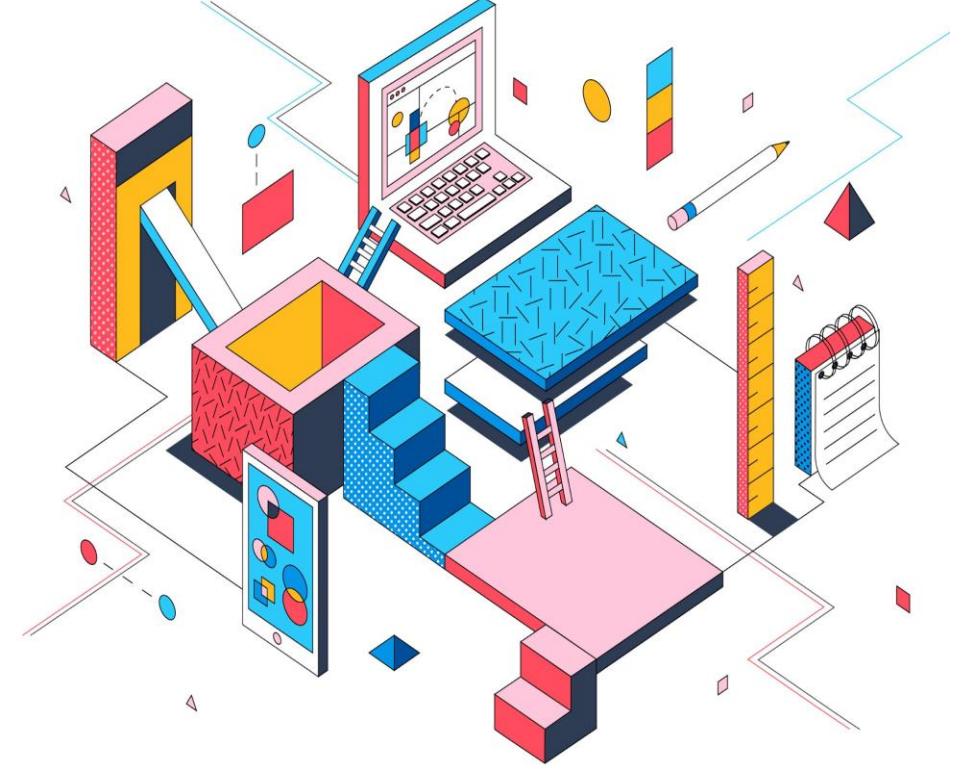


- Yazılım tasarımı, üretilecek yazılımın ana bileşenlerini, bu bileşenlerin özelliklerini ve bileşenler arasında ilişkileri belirler.
- Tasarım ne kadar kusursuz veya mükemmele yakın olursa, bu tasarım kullanılarak elde edilen ürün de o kadar mükemmele yakın bir ürün olur. Zaman içerisinde her mühendislik dalında en iyi çözümler ortaya konmuş, bunlar sonradan gelen mühendisler ve tasarımcılar tarafından tekrar tekrar kullanılmıştır ve daha iyiye götürülmüştür.



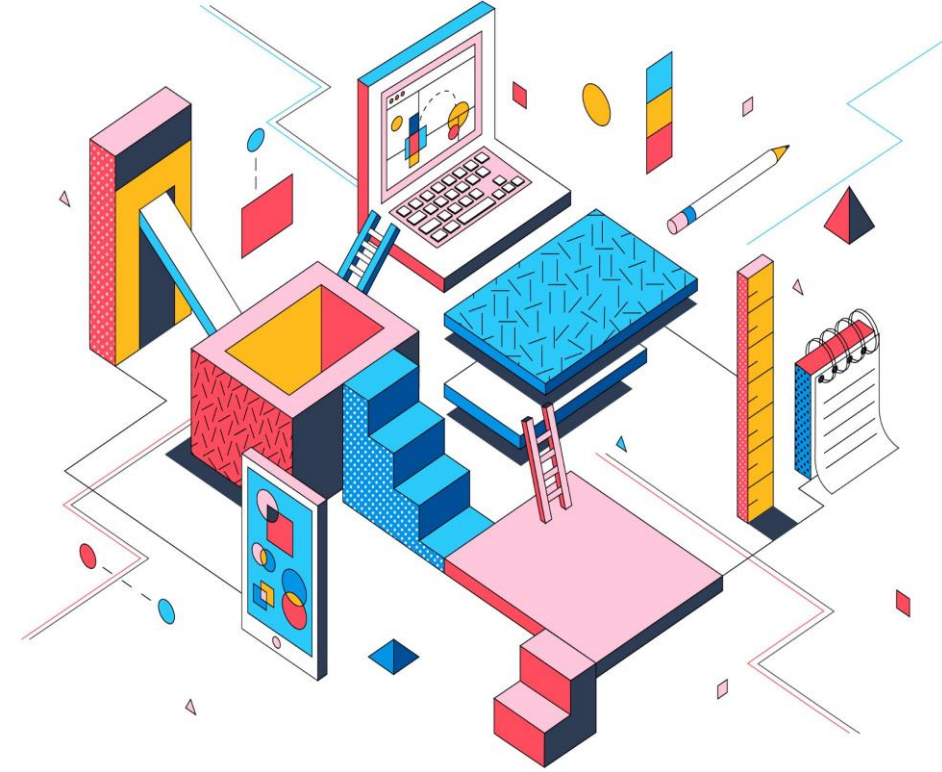


- Geliştiriciler ayrıca, donanım / yazılım stratejisi, kalıcı veri yönetimi, global kontrol akışı, erişim kontrolü politikası ve sınır koşullarının ele alınması gibi sistemi oluşturmak için stratejiler seçerler.
- Sistem tasarımının sonucu, bir alt sistem ayrışmasını ve bu stratejilerin her birinin açık bir tanımını içeren bir modeldir.
- Sistem tasarımı algoritmik değildir.



Sistem tasarımı, her biri sistemin ayrıştırılmasıyla ilgili genel sorunun bir bölümünü ele alan çeşitli etkinliklere ayrılır:

- **Tasarım hedeflerini tanımlayın.** Geliştiriciler, optimize etmeleri gereken sistemin niteliklerini belirler ve önceliklendirir.
- **İlk alt sistem ayrışmasını tasarlayın.** Geliştiriciler, kullanım durumu ve analiz modellerine göre sistemi daha küçük parçalara ayırır. Geliştiriciler, bu etkinlik sırasında standart mimari stilleri başlangıç noktası olarak kullanır.
- **Tasarım hedeflerini ele almak için alt sistem ayrışmasını hassaslaştırın.** İlk ayrışma genellikle tüm tasarım hedeflerini karşılamamaktadır. Geliştiriciler, tüm hedefler karşılanana kadar bunu hassaslaştırır.



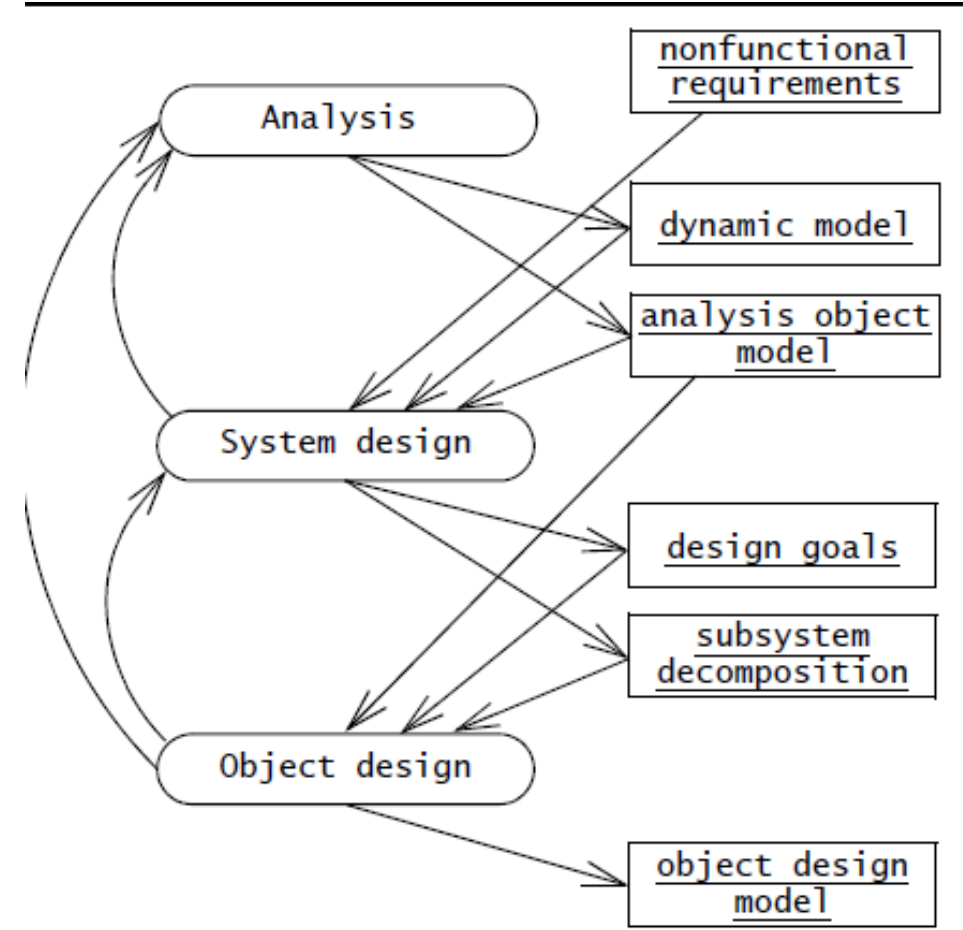
Sistem tasarımı sistemin iç yapısı, donanım konfigürasyonu veya daha genel olarak sistemin nasıl gerçekleştirilmesi gerektiği hakkında bilgi sağlama yönünden ilk adımdır. Sistem tasarımı aşağıdaki ürünlerden oluşur:

- Geliştiricilerin optimize edeceği sistemin niteliklerini tanımlayan tasarım hedefleri
- Alt sistem sorumlulukları, alt sistemler arasındaki bağımlılıklar, donanım alt sistemi eşlemesi ve denetim akışı, erişim denetimi ve veri depolama gibi temel politika kararları açısından alt sistem ayrıştırmasını tanımlayan yazılım mimarisi
- Sistem yapılandırması, başlatma, kapatma ve istisna işleme sorunlarını açıklayan sınır kullanım durumları.





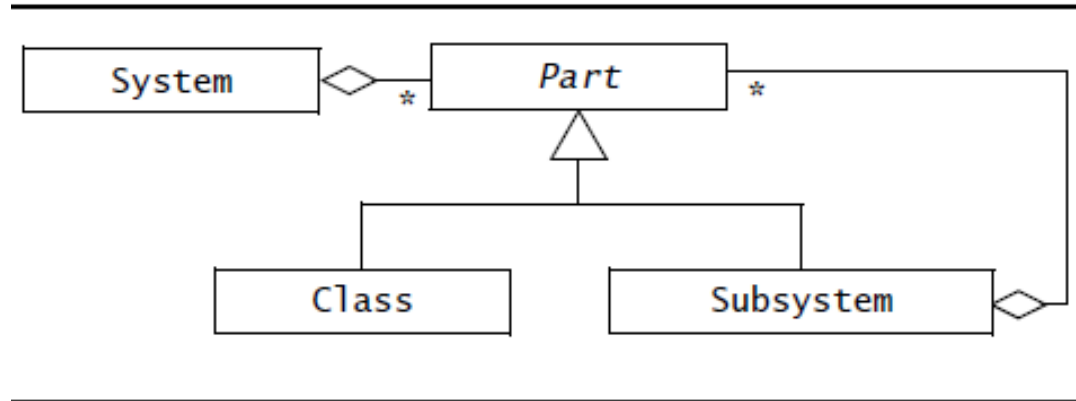
- Dizayn amaçları fonksiyonel olmayan gerekliliklerden üretilirler. Programcılar Sistem Dizaynı sırasında *ödünleşimler* (trade-off) hakkında karar verirken bu amaçlar onlara yol gösterir.
- Dizayn amaçlarından daha az öncelikli olanlar, öncelikli olanlara feda edilebilir veya önemsenmeyebilirler. Altsistem tasarımı sistem dizaynının esas işini oluşturur.
- Geliştiriciler sistemi karmaşıklıkla başa çıkmak için yönetilebilir parçalara ayırır: her alt sistem bir ekibe atanır ve bağımsız olarak gerçekleştirilir. Bunun mümkün olması için, geliştiricilerin sistemi çözerken sistem çapında sorunları ele almaları gerekmektedir.



- Yazılım Tasarım Seviyeleri
- Yazılım tasarımı üç düzeyde sonuç verir:
- **Mimari Tasarım** - Mimari tasarım, sistemin en yüksek soyut versiyonudur. Yazılımı, birbiriyle etkileşim içinde olan birçok bileşeni olan bir sistem olarak tanımlar. Bu seviyede, tasarımcılar önerilen çözüm alanı fikrini alırlar.
- **Üst Düzey Tasarım** - Üst düzey tasarım, mimari tasarımın 'tek varlık-çok bileşenli' kavramını alt sistemlerin ve modüllerin daha az soyut görünümüne böler ve bunların birbirleriyle etkileşimini gösterir. Üst düzey tasarım, sistemin tüm bileşenleriyle birlikte modül formlarında nasıl uygulanabileceğine odaklanır. Her bir alt sistemin modüler yapısını ve aralarındaki ilişkiyi ve etkileşimi tanır.
- **Ayrıntılı Tasarım** - Ayrıntılı tasarım, önceki iki tasarımda bir sistem ve alt sistemleri olarak görülenlerin uygulama kısmı ile ilgilenir. Modüller ve uygulamalarına yönelik daha ayrıntılıdır. Her modülün mantıksal yapısını ve diğer modüllerle iletişim kurmak için arayüzlerini tanımlar.

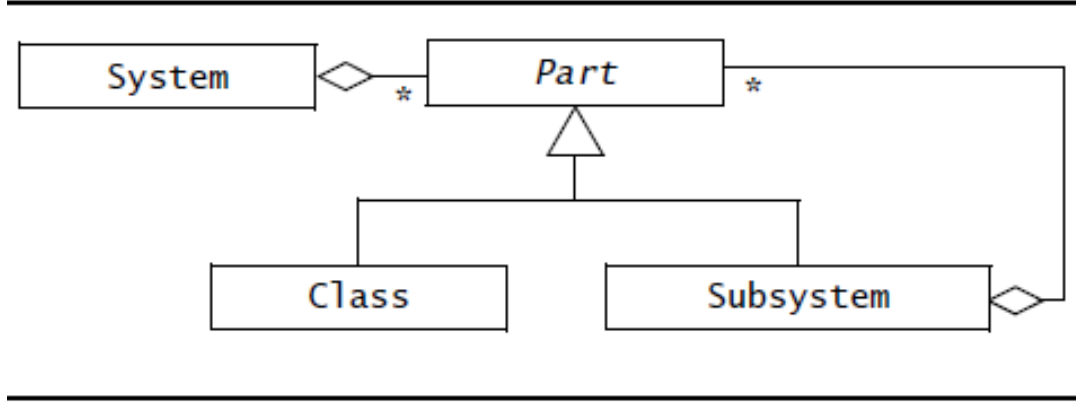
## 1.2.1. Altsistemler ve Sınıflar

- Uygulama alanındaki işimizi kolaylaştırmak için daha küçük parçalar olan sınıf kavramını üretmiştik. Benzer şekilde, çözüm alanındaki karmaşıklığı azaltmak için sistemi daha küçük olan parçalara böleriz (altsistemler). Bu altsistemler çözüm alanındaki sınıflardan bir veya birkaçını içermektedir. Karmaşık altsistemlerde ise, bu yöntemi özyinelemeli olarak uygulayarak bir altsistemi daha basit altsistemlere bölmekteyiz.





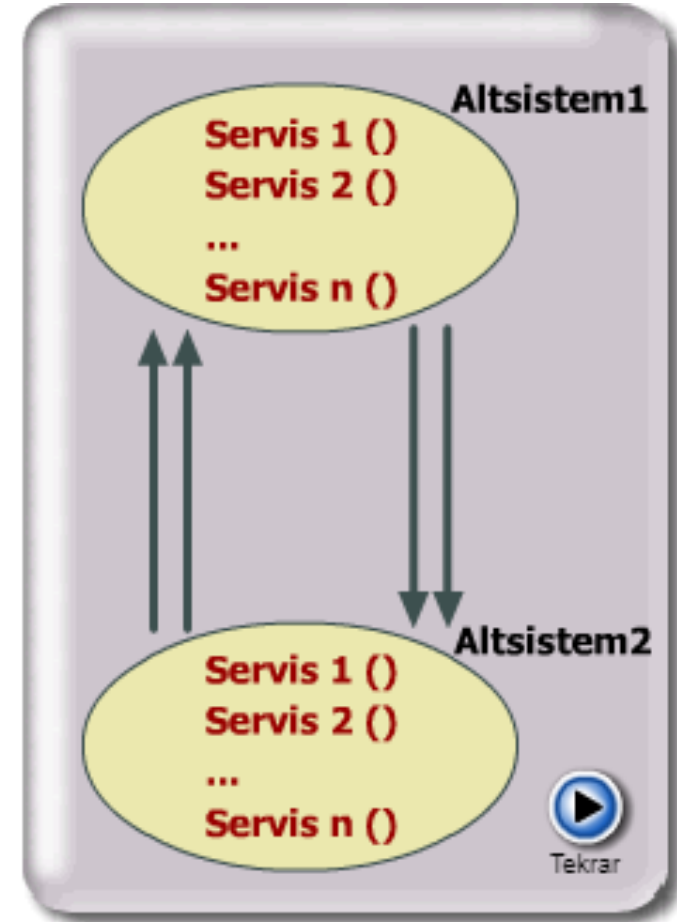
## 1.2.1. Altsistemler ve Sınıflar



- Birçok programlama dili (Java ve Modula-2) altsistemleri modellemek için içlerinde yapılar bulundurmaktadırlar. Diğer programlama dilleri ise, örneğin C veya C++ gibi altsistemler belirli olarak modellenmez, o durumda programcılar sınıfları gruplamak için önceden geliştirilmiş taktikler uygularlar. Altsistemler ister belirlenmiş olsun ister olmasın, başka takımlar tarafından programlanacakları için iyi dökümente edilmiş olmaları gerekmektedir.

## 1.2.2. Servisler ve Altsistem Arayüzleri

- Altsistemler, diğer altsistemlere sağladıkları servislerle şahsiyetlenirler. Bir servis aynı ortak amaç için bir araya gelmiş bağlantılı işlemlerin kümesidir.
- Bir altsistemin diğer altsistemler tarafından kullanabilecek işlemleri o altsistemin arayüzünü oluşturur. Altsistem arayüzü, aynı zamanda uygulama programcısı arayüzü olarak da adlandırılır, işlemlerin isimlerini, parametrelerini, parametre tiplerini ve döndürdüğü değerleri içerir. Sistem Dizaynı her altsistemin verdiği servislere odaklanır yani işlemleri, parametrelerini ve yüksek seviye davranışlarını numaralandırır. Oysa nesne dizaynı altsistem arayüzünü tanımlamaya odaklanır, yani parametrelerin tipine ve her işlemin döndürdüğü değerlere odaklanır.



Şekil 3. Bir altsistemin diğer altsistemler tarafından kullanabilecek işlemleri o altsistemin arayüzünü oluşturur.

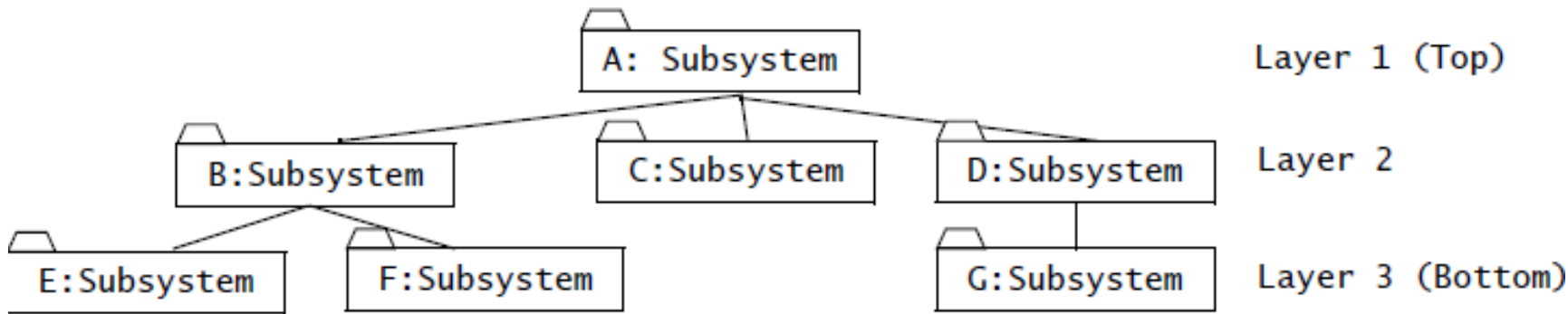
### 1.2.3. Bağlaşım ve Evreuyumluluk (Coupling ve Coherence)

- **Bağlaşım**, iki alt sistem arasındaki bağımlılıkların sayısıdır. İki alt sistem gevşek bir şekilde bağlanmışsa, bunlar nispeten bağımsızdır, bu nedenle alt sistemlerden birinde yapılan değişikliklerin diğeri üzerinde çok az etkisi olur. İki alt sistem güçlü bir şekilde birleştirilirse, bir alt sistemde yapılan değişikliklerin diğeri üzerinde bir etkisi olabilir.
- Bir alt sistem ayrışmasının arzu edilen bir özelliği, alt-sistemlerin makul olarak gevşek bir şekilde bağlanmış olmasıdır. Bu, bir alt sistemdeki hataların veya gelecekteki değişikliklerin diğeri alt sistemlerde yarattığı etkiyi en aza indirir.
- **Uyum**, bir alt sistemdeki bağımlılıkların sayısıdır. Bir alt sistem, birbiriyle ilişkili birçok nesne içeriyorsa ve benzer görevleri yerine getiriyorsa, onun birleşimi yüksektir. Bir alt sistem bir dizi ilgisiz nesne içeriyorsa, onun birleşimi düşüktür. Bir alt sistem ayrışmasının arzu edilen bir özelliği, yüksek bir bütünlüğe sahip alt sistemlere yol açmasıdır.



## 1.2.4. Katmanlar ve Parçalar

- Bir sistemin **hiyerarşik bir ayrışması**, sıralı bir katman kümesi sağlar. **Bir katman**, muhtemelen başka bir katmandan hizmetlerle gerçekleştirilen, ilgili hizmetleri sağlayan alt sistemlerin bir gruplandırmasıdır.



**Kapalı mimariye**, örnek OSI katman modeli.

**Açık mimariye** örnek, Java için Swing kullanıcı arabirimi araç takımıdır.

## 1.2.5. Yazılım Mimarisi

- Sistemin karmaşıklığı arttıkça, sistemin nasıl yapılandırıldığının önemi artar. Eğer uygulama aşamasına başlanmışsa, kötü bir yapılandırmayı değiştirmek çok zor olabilir. Çünkü böyle bir yeniden yapılandırma birçok altsistemin arayüzlerinin değişimini getirecektir.
- Bu probleme getirilen çözüm, yazılım mimarisi kavramının ortaya atılmasıdır. Yazılım mimarisi, sistemin yapılandırılmasını, genel kontrol akışını, hata ayıklama yöntemlerini ve altsistem haberleşme protokollerini içerir.

- ▶ Havuz (Repository) Mimarisi
- ▶ Model/Bakış/Kontrol (MBK) Mimarisi
- ▶ İstemci/Sunucu (Server/Client) Mimarisi
- ▶ Peer-to-Peer Mimarisi
- ▶ Boru-Filtre (Pipe-Filter) Mimarisi

- Tasarım desenleri; yazılım tasarımı, problemlerde karşımıza sıkça çıkan ortak sorunları çözmek için oluşturulmuş desenlerdir. Tasarım desenleri, yazılım sürecinde uygulanan çözümlerin esnekliği ve tekrar kullanılabilirliği ile de ilgilenmektedir.
- *Her desen, çevremizde tekrar tekrar ortaya çıkan bir sorunu açıklar ve daha sonra bu soruna çözümün uygulanmasını, bu çözümü iki kez aynı şekilde yapmadan milyonlarca kez kullanabileceğiniz şekilde tanımlar.*

—Christopher Alexander

- Bu örüntülerin kullanımı ile aşağıdaki faydalar elde edilir:
  - 1) Yazılım tasarımı kolaylaştırır ve tasarımcıya zaman kazandırır
  - 2) Elde edilen tasarım en iyi çözümleri kullanıyor olur ve gerçekleştirimi kolaylaşır
  - 3) Elde edilen tasarım güncel ve ileride daha kolay güncelleştirilebilir bir çözüm olur.



Tasarım deseni referans kitabı **Design Patterns - Elements of Reusable Object-Oriented Software'e** göre, üç kategoride sınıflandırılabilen 23 tasarım deseni vardır: Yaratıcı, Yapısal ve Davranışsal desenler.

## Types Of Design Patterns

### Creational

- 1.Singleton
- 2.Factory
- 3.Abstract Factory
- 4.Builder
- 5.Prototype

### Structural

- 6.Adapter
- 7.Composite
- 8.Proxy
- 9.Fly Weight
- 10.Facade
- 11.Bridge
- 12.Decorator

### Behavioural

- 13.Template Method
- 14.Mediator
- 15.Chain Of Responsibility
- 16.Observer
- 17.Strategy
- 18.Command
- 19.State
- 20.Visitor
- 21.Iterator
- 22.Interpreter
- 23.memento

SN	Desen ve Açıklama
1	<b>Oluşturma Kalıpları</b> Bu tasarım kalıpları, nesneleri doğrudan new operatörü kullanarak somutlaştırmak yerine, oluşturma mantığını gizlerken nesneler oluşturma bir yolunu sağlar. Bu, belirli bir kullanım durumu için hangi nesnelerin oluşturulması gerektiğine karar vermede programa daha fazla esneklik sağlar.
2	<b>Yapısal Örüntüler</b> Bu tasarım desenleri, sınıf ve nesne bileşimi ile ilgilidir. Kalıtım kavramı, arayüzler oluşturmak ve yeni işlevler elde etmek için nesneleri oluşturma yollarını tanımlamak için kullanılır.
3	<b>Davranış Kalıpları</b> Bu tasarım kalıpları özellikle nesneler arasındaki iletişimle ilgilidir.
4	<b>J2EE Kalıpları</b> Bu tasarım kalıpları özellikle sunum katmanı ile ilgilidir. Bu modeller Sun Java Center tarafından tanımlanır.

