

Yazılım Tasarımı ve Mimarisi

Amaç

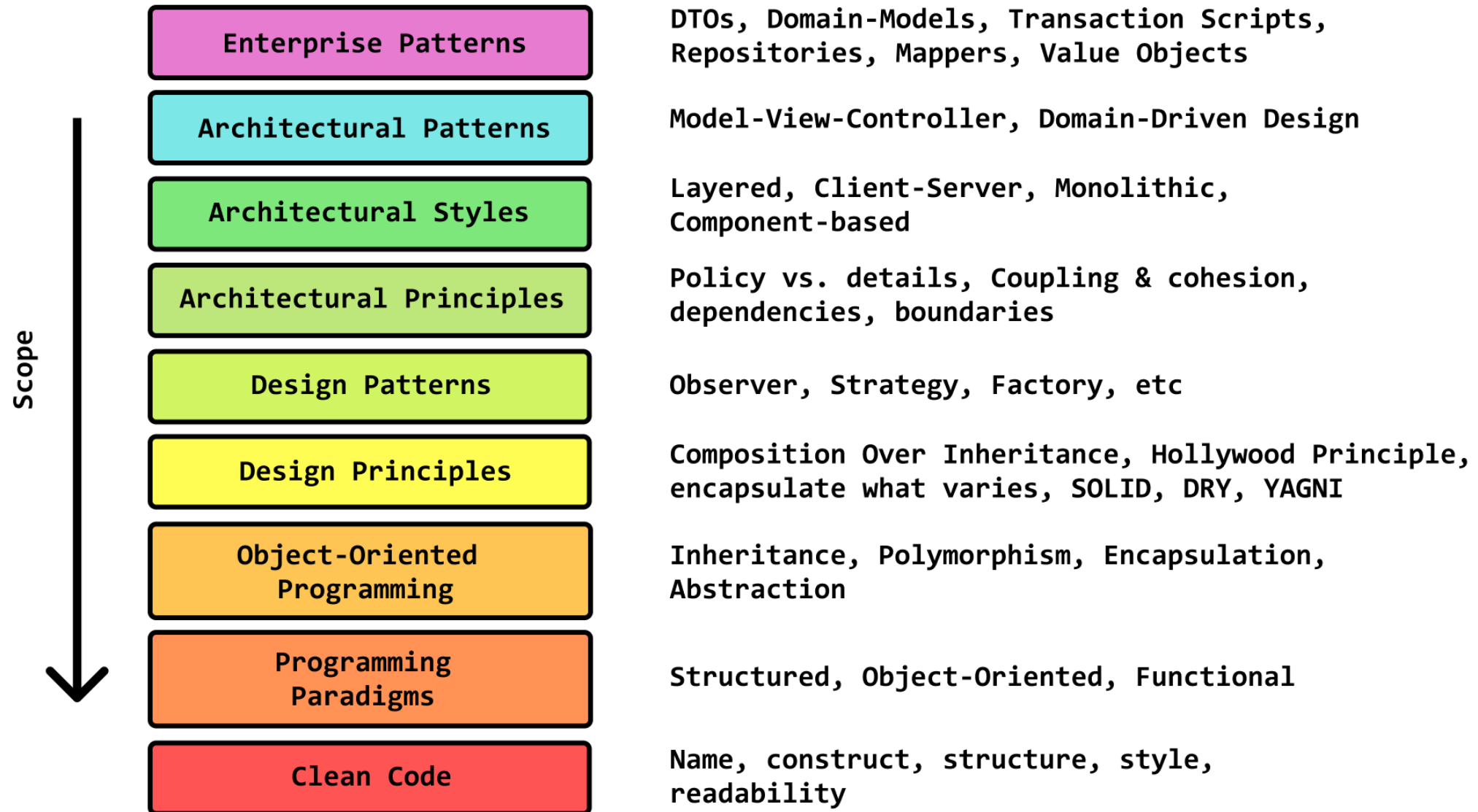
- Tasarım kalıpları üzerinden yazılım tasarımı disiplini kazandırmaktır. Dolayısıyla;
 - Yazılım tasarımında soyut düşünme/modelleme,
 - Yazılım tasarımının temel problemleri, kavramları ve teknikleri,
 - Sık rastlanan yazılım problemleri üzerine düşünme,
 - Kalıpların iş alanlarındaki problemlere uygulanması,

- Yazılım karmaşıklığı ve değişim (software complexity and change)
- Birliktelik ve bağımlılık (cohesion & coupling) ve anti-patternları
- SOLID, GRASP, Demeter prensipleri
- Nesne rolleri
- Yaratımsal, Yapısal ve Davranışsal kalıplar
- Null Object vb.diğer kalıplar



The Software Design & Architecture Stack

Khalil Stemmler
@stemmlerjs



Yığının Akış Haritası İçin;

- <https://khalilstemmler.com/articles/software-design-architecture/full-stack-software-design/>

Yazılım Geliřtirmenin Özellikleri

- Yazılım ürünleri karmaşıktır.
- Takım içinde iyi bir iletişim gerektirir.
- Modüller arası uyum gerektirir.
- Yazılım sürekli deęiřeceęi için esnek olmalıdır.

- Yazılım Geliştirme Aşamalarının Maliyetleri:
 - İsteklerin Çözümlemesi (Requirements): %3
 - Tasarım: %8
 - Kodlama: %7
 - Sınama: %15
 - Bakım: %67
- Hataların %85'i isteklerin çözümlenmesi ve tasarım aşamalarında gerçekleşmektedir.

- Hataların Giderilme Maliyeti (Belirlendikleri aşamaya göre)
 - İsteklerin Çözömlenmesi (Requirement): 1 Birim
 - Tasarım: 1.5-2 Birim
 - Kodlama: 5-10 Birim
 - Sinama: 10-15 Birim
 - Bakım: 15-100 Birim

Yazılım Tasarımı

- Eğer bir yazılım projesinde başarıyı elde etmek istiyorsanız doğru olan yazılım mimarisini kurmak zorundasınız.
- Tasarım, Sistem Analizi çalışması sonucunda üretilen mantıksal modelin fiziksel modele dönüştürülmesidir.
- Yazılım mimarı, yazılım projesini baştan sona tasarlar, gerekli yerlerde devreye girer ve yazılım süreci boyunca tasarladığı planın dışına çıkılmamasını sağlar.
- Mimari tasarım sürecinin çıktısı bir sistemin haberleşen bileşenler olarak nasıl yapılandığını anlatan bir mimari modeldir.
 - Soyutlama
 - İyileştirme
 - Modülerlik
- Yapı, arayüz ve süreç tasarımı önce veri tasarımı yapılmalıdır. En son süreç tasarımı yapılır.

Yazılım Mimarisi

- Yazılım Mimarisi, bir bilgisayar sistemini meydana getiren yazılım bileşenleri, görünür dış özellikleri ve bunlar arasındaki ilişkilerdir. Yazılımdaki karmaşıklığın giderilebilmesi için önceleri veri yapıları kullanılırken, 1980'lerden sonra yazılım mimarisi kullanılmaya başlanmıştır.
- Mimari tasarım yazılım sisteminin nasıl yapılandırılması gerektiğini anlamak ve sistemin genel yapısını tasarlamaktır.
- İyi bir yazılım için doğru seçilmiş yazılım mimarisi gerekmektedir. Ayrıca kalite üzerinde büyük etkisi vardır.
- Bir mimari tasarım dökümanı ürüne genel bir bakış, mimari spesifikasyonlar ve tasarım gerekçesi gibi bölümlerden oluşur.

- Yazılım mimarisi sistemin güvenliğini, performansını, evrimleşebilirliğini, kullanılacak donanım gibi sonradan değiştirilmesi maliyetli olacak durumları belirler.
- Mimari tasarım genellikle ürün tasarımı sırasında başlamalıdır. Yazılım sürecindeki Tasarım bölümünün ilk aşamasıdır.
- Yazılım mimarisi çok boyutludur;
 - Mantıksal boyut: Sistemin ana hatları, nesneler ve sınıflar arası ilişki.
 - İşleyişsel boyut: Sistemin çalışma zamanı.
 - Geliştirme boyutu:Sistem hangi parçalardan oluşacağı.
 - Fiziksel boyut: Hangi donanımdan oluşacağı ve hangi yazılımın hangi işlemcide çalışacağı.

- Gereksinim mühendisliği ile mimari tasarım süreçleri arasında ciddi örtüşme vardır.
- Gereksinim mühendisliği sürecinin bir parçası olarak, sistem işlev ve yeteneklerinin büyük çaplı bileşenler ve alt sistemlerle bağlandığı soyut bir sistem mimarisi önerilebilir.
- Sistem mimarisi üzerindeki en büyük etki fonksiyonel olmayan gereksinimlerdir.

Yazılım Mimarisinin Avantajları

- Üst seviye soyutlama ile daha anlaşılabilir olur. (Ayrıntılı tasarıma doğru soyutlanma derecesi azalır)
- Karmaşık sistemler daha iyi analiz edilir.
- Paydaşlar için iletişim ortamı sağlar.
- Gerçekleştirilecek güncellemelerin ne olacağı, nasıl yapılacağı ve etkinliği sağlanır.
- Yazılım alt bileşenlere ayrıldığı için hem işin büyüklüğü öngörülmüş olur, hem de takım içinde görev paylaşımı kolaylaşır.
- Yazılımın geliştirilebilirliği öngörülmüş olur.
- Performans, sağlamlık, dağıtılabilirlik ve bakım kolaylığı sağlar
- Yazılımın testi kolaylaşır.
- Yazılımın aşamalı teslimi kolaylaşır.
- Benzer bileşenler başka yazılımlar için de tekrar kullanılabilir.

- Programın mimari modelinin iki farklı kullanımı vardır;
- Sistem tasarımını tartışmak
- Tasarlanmış bir mimariyi belgelemek

- Çoğu büyük sistem, sistem yazılımının çok sayıda bilgisayar üzerine dağılmış olduğu dağıtık sistemlerdir.
- Yazılım sisteminin mimarisi mimari bir desene dayanıyor olabilir.
- Bir mimari desen, istemci sunucu yapılanması veya katmanlı mimari gibi bir sistem organizasyonunun betimlemesidir.
- İşlevsel olmayan gereksinimler ile yazılım mimarisi arasındaki yakın ilişki nedeniyle seçilecek stil seçimi sistemin işlevsel olmayan gereksinimlerine bağlıdır.
 - Eğer performans kritik bir gereksinim ise,
 - Eğer güvenlik kritik bir gereksinim ise,
 - Eğer emniyet kritik bir gereksinim ise,
 - Eğer erişilebilirlik ve bakım kolaylığı kritik bir gereksinim ise,

Mimari İin

- Mimari taktikler
- Mimari desenler
- Referans mimarileri
- Bařkalarının geliřtirdiđi bileřenler

Mimari Tasarım Kalıpları

- Desen Mühendisliği (Pattern Engineering) sistematik ve disiplinli bir şekilde verilen kısıt ve sağlanan kaynaklarla kalıpları tespit etme, kullanma, bakım yapma yaklaşımıdır.
- Hangi kalıbın ne zaman kullanılacağını proje kısıtları ve sağlanan kaynaklar belirler. Kalıp mühendisliği de yazılım mühendisliğinde hem tasarım aşamasında tasarım kalıpları olarak hem de mimari tasarımda mimari kalıplar olarak kendini göstermektedir.

Sık Kullanılan Mimari Modeller

- Katmanlı Mimari
- Boru (Kanal) & Filtre Mimarisi
- İstemci - Sunucu Mimarisi
- Yinelemeli İçerme Mimarisi
- Bileşen Tabanlı (Ambar/Depo) Mimarisi
- Model - View - Controller
- Servis Odaklı Mimari
- Microservis Mimarisi

Katmanlı Mimari Tasarım Kalıbı

- N-katmanlı (n-tier) mimari olarak da bilinir. Ayrıştırma ve bağımsızlaştırmanın bir başka yoludur.
- Her katman farklı bir fonksiyonu yerine getirir ve genel olarak kendisinden bir alt ve bir üst katman ile iletişim kurar. Her katman bir üst katmana hizmet eder.
- Katmanlı mimari kalıbında eğer bir katman sadece bir alt katmandan servis alabiliyor ise bu tür sistemlere kapalı katmanlı sistem denir. Bu ideal tasarımıdır. Performans gibi kaygılardan dolayı bir katman kendinden iki ve daha fazla altta olan katmanlara da erişiyor ise buna açık katmanlı sistem denmektedir.
- Kolay geliştirilip test edilebilir fakat performans, çeviklik ve genişleyebilme açısından kısıtlıdır.

Kanal ve Filtre (Boru ve Süzgeç) Tasarım Kalıbı

- Veri işleme için yaygın kullanılan bir mimaridir. Sisteme giren veri, Filtre dediğimiz veri üzerinde işlem yapan bileşenler ve bu bileşenler arasında veriyi aktaran kanallardan oluşur.
- Güçlü bir mimaridir.
- Bu mimaride ayrı ayrı filtreleme bileşenleri bulunur. Filtreler arasında da yerel bağlantı bulunur. Bu bağlantılara pipe denir.
- Her filtre kendisine gelen veriyi işler ve sıradaki filtreye aktarır.
- Filtreler verileri zenginleştirebilir, detayları gizleyebilir, başka bir gösterime dönüştürebilir.
- Farklı işlemler farklı filtreler halinde geliştirilerek sistem daha modüler hale gelebilir. Ayrıca filtrelerin farklı amaçlarla kullanımı ile yeniden kullanılabilirliği artar.

İstemci Sunucu Mimari

- İstemci isteği yapan, sunucu ise isteği karşılayan kısımdır.
- Sunucu, istemciden gelen isteği kabul ettiğinde, istemciyle belirli bir protokol üzerinden bağlantı kurar.
- Sunucu istemciden gelen isteklerin kaydını tutuyorsa buna oturum denir.
- Sistem bir servisler kümesidir. Her bir servis ayrı bir sunucu tarafından sağlanır. İstemciler bu servislerin kullanıcılarıdır ve servislerden yararlanabilmek için sunuculara ulaşırlar.
- Paylaşılan veritabanı farklı lokasyonlardaysa, farklı istemciler tarafından ulaşılacak isteniyorsa ve sistem üzerindeki yük değişkense tercih edilir.
- Esas paylaşılan servis sunucuda yer alır ve bu sunucudan servis alacak istemciler istekte bulunarak sunucudan servis alırlar. İstemci ayrı çalışan bir uygulama olabileceği gibi WEB üzerinde hizmet alan bir web uygulaması da olabilir.

Yinelemeli İçerme Tasarım Kalıbı

- Büyük bir sistemi alt sistemlere oradan da daha alt bileşenlere bölecek şekilde tasarlar. Burada her bir sistem kendi alt parçalarını içerir. Bu kalıp aynı zamanda Bütün Parça tasarım kalıbı ya da Birleşik tasarım kalıbı olarak bilinmektedir. Örneğin bir çizim şekillerden oluşur.

Bileşen Tabanlı (Ambar) Mimari

- Sistem küçük bileşenler ve bu bileşenleri bir arada tutan bileşen yöneticisinden meydana gelir.
- Tüm veriler, tüm sistem bileşenlerinin erişebildiği merkezi bir havuzda yönetilir.
- Uzun süre saklanması gereken büyük miktarda veri olduğunda kullanılır.
- Yazılım sistemi yeniden kullanılabilir, birleştirilebilir ve paketlenmiş bileşenlerden oluşur. JavaBean, .NET, WEB Servisleri vb. gibi. Bağlantılar bileşenlerin diğer bileşenlerle nasıl iletişim kuracağını belirler.
- Bileşen etkileşimi metot çağırma, uzaktan metot çağırma, yayımlama, mesaj tabanlı iletişim ve benzeri protokoller aracılığı ile yapılır.
- Yeniden kullanılabilirlik, test edilebilirlik ve genişleyebilirlik açısından önemli avantajlar sağlamaktadır.

Model Sunum Kontrolcü (Model Görünüm Kumanda - Model View Controller-MVC) Tasarım Kalıbı

- En çok bilinen ve sık sık kullanılan şablondur.
- İş mantığını ve verileri sunum detaylarından ayırarak bağımsız olarak değişebilmelerine olanak tanır.
- Kullanıcı ara yüzü tabanlı yazılımlarda çok yaygın olarak kullanılır.
- Sunum ve etkileşimi sistem verisinden alır ve ayrı katman olarak değerlendirir.
- Pek çok Web tabanlı sistemdeki etkileşim yönetiminin tabanını oluşturur.
- Temel amaç veri ve kullanıcı arayüzünü birbirinden ayırmaktır. Veri ile kullanıcı ara yüzü arasındaki aracılığı kontrolcü sağlar. ASP .Net MVC, Rubby on Rails, Django gibi kütüphaneler yapılarını bu kalıp üzerine kurmuşlardır.

ServisTabanlı Mimari

- Değiştirilmesi zor olan eski monolitik bağımsız uygulamalardan, iş hedeflerine ulaşmak için birlikte çalışmak üzere birleştirilen yeniden kullanılabilir kurumsal bir yapıya geçiş için mimari bir yaklaşımdır.
- Burada farklı kabiliyetler servis olarak geliştirilmekte ve istemcilerin istekleri bu servislere iletildiğinde ilgili servis diğer servislerden tanımlı ara yüzler üzerinden hizmet olarak istemcinin ihtiyacını karşılamaktadır.

Mikro Servis Mimarisi

- Birden fazla mimari tasarım kalıbını bir arada kullanan bir referans mimaridir. Mikro servis referans mimarisinde mimari kalıp dili önerilmiştir.
- Sistem birbiri ile iletişim halinde olan servislerden oluşur. Bu servisler HTTP/ Rest çağrıları ile birbirleri ile doğrudan konuşabilirler ya da AMQP gibi yayımla abone ol tabanlı ara katmanlar üzerinden iletişim sağlarlar.
- Servislerin mikro olmasından çok anlamlı ve tek bir iş mantığı sağlıyor olmasıdır. Aynı zamanda yönetilebilir küçük bir takım tarafından geliştirilebilir olması önem verilen bir noktadır.
- Temelde bileşenlere ayırma kalıbı olarak iki yaklaşım önerilmektedir (Kabiliyete göre ya da alt domainlere göre).
- İletişim tararım kalıpları olarak mesajlaşma, uzaktan metot çağırma ve alana özel haberleşme kalıpları bulunmaktadır.

Tasarım Desenleri

- Yaratımsal Kalıplar
 - **Tekil Kalıp - Yegane (Singleton Design Pattern):** Bir nesneden sadece bir tane oluşturmak.
 - **Fabrika Metodu (Factory Method):** Nesne oluşturmayı soyutlamak.
 - **Soyut Fabrika Metodu (Abstract Factory Pattern):** Nesne ailesi oluşturmayı soyutlamak.
 - **İlk örnek Kalıp - Örnek (Prototype Design Pattern):** Bir örnek nesneden kopyalama ile yeni nesne üretmek.
 - **Yapıcı Kalıp (Builder Design Pattern):** Karmaşık nesne oluşturma sürecini kurgulamak.

.

Yapısal Kalıplar

- **Adaptör Kalıp (Adapter Pattern):** Sisteme yeni bir özelliğin eklenmesini kolaylaştıran bir tekniktir. Sadece bir sınıfa özel olan arayüzleri diğer sınıflarla uyumlu arayüzler haline getirir. Birbirleri ile çalışamayan sınıflara da birbirleri ile çalışma imkanı sunarlar.
- **Köprü Kalıp (Bridge Pattern):** Soyutlamayı uygulamadan (implementation) ayırarak ikisinin birbirinden bağımsız çalışmasını sağlar. Kapsülleme, toplama (aggregation) ve de kalıtımı kullanarak programın görevlerini birbirlerinden farklı sınıflara aktarırlar.
- **Bileşik Kalıp (Composite Pattern):** Birbirlerinden farklı olan bir grup nesnenin sanki tek nesneymiş gibi davranmalarını sağlarlar. Nesneler bir ağaç yapısında birleştirilip proje genelindeki parça bütün ilişkisi yeniden düzenlenir.
- **Dekorator Kalıp (Decorator Pattern):** Nesneye dinamik olarak yeni özellikler kazandırır. Projede runtime esnasında bir sınıfın işlevlenebilirliğinin genişletilmesini mümkün kılar. Bunun için yeni dekorator sınıflar geliştirilir.
- **Vitrin Kalıp (Facade Pattern):** Çok geniş boyutlardaki kod parçalarını sadeleştirilmiş arayüzlere indirgenilmesini sağlar.
- **Sineksiklet Kalıp (Flyweight Pattern):** Hafıza kullanımını en aza indirebilmek için verileri kendisine benzer olan nesnelerle paylaşan nesnelerdir. Çok sayıda aynı türden nesneler yaratılacağına tek bir nesneden görsel nesneler yaratarak kalabalık bir nesne yapısı kurulmasına olanak sağlar.
- **Vekil Kalıp (Proxy Pattern):** İnternet bağlantıları, yüksek miktarda hafıza kullanımları, yapım aşaması zorlayıcı olan nesnelerin tek bir arayüz tarafından kullanılmalarını olası kılar.

Davranışsal Kalıplar

- **Komut Kalıbı (Command Pattern):** Bir nesne üzerindeki işlemlerin yapılmasındaki süreci bilmediğimizde ya da kullanılmak istenen nesnenin tanınmadığı durumlarda kullanılır.
- **Hatıra Kalıbı (Memory Pattern):** Bir nesnenin, daha önce sahip olduğu durumlardan birine tekrar dönüştürülmesinde kullanılır.
- **Strateji Kalıbı (Strategy Pattern):** Bir işlemin yerine getirilmesi için birden fazla metot kullanıldığında ihtiyaca göre birinin seçilmesini sağlayan kalıptır.
- **Tekrarlayıcı Kalıbı (Iterator Pattern):** Bir listede yer almakta olan nesnelerin sırasıyla, listenin yapısını ve çalışma tarzının uygulamanın diğer kısımları ile olan bağlantılarını en aza indirmek için uygulamadan soyutlamaktır.
- **Durum Kalıbı (State Pattern):** Bir nesnenin metotları, sahip olduğu özellikler değiştiği zaman değiştirilebilir. Bu durumda nesnenin sahip olduğu sınıf yapısı değiştirilmiş olacaktır.

- **Sorumluluk Zinciri Kalıbı (Chain of responsibility Pattern):** Client ile isteği yerine getiren arasındaki bağımlılık azaltılır.
- **Aracı Kalıbı (Mediator Pattern):** Nesnelerin yönetimi ve aralarındaki iletişimin merkezi bir noktadan sağlanması ve yönetilmesi için kullanılmaktadır.
- **Gözlemci Kalıbı (Observer Pattern):** Nesnenin durumunun değişikliklerinden haberdar olunmak istendiğinde.
- **Şablon Kalıbı (Template Pattern):** Algoritma için gerekli işlemler soyut olarak tanımlanır. Alt sınıflar algoritma için gerekli işlemi kendileri gerçekleştirerek, kullanılan algoritmanın kendi istekleri doğrultusunda çalışmasını sağlayabilmektedirler.
- **Ziyaretçi Kalıbı (Visitor Pattern):** Sınıf hiyerarşisinde yer alan sınıflar üzerinde değişiklik yapmadan, bu sınıflara yeni metotların eklenmesini kolaylaştırır.

Kaynak

- <https://khalilstemmler.com/articles/software-design-architecture/full-stack-software-design/>
- www.akademi.itu.edu.tr/buzluca
- <https://www.horato.com/tr/post>
- www.wikipedia.org
- www.medium.com
- www.arakatman.com
- Ian Sommerville, Yazılım Mühendisliği