Yazılım Mimarisi, Tasarımı ve Kalıpları

Yazılım Mimarisi Nedir? (What is Software Architecture?)

- Yazılım mimarisi, bir yazılım sisteminin organizasyonunu (organization) ve bileşenlerinin etkileşimini (interaction) tanımlar.
- Mimari, sistemin temel yapısını (fundamental structure) ve davranışını (behavior) belirler, ayrıca bileşenlerin nasıl etkileşeceğini ve tasarım ilkelerini (design principles) açıklar.

Yazılım Mimarisi Tasarımının Önemi (Importance of Software Architecture Design)

- İyi tasarlanmış bir mimari, paydaşların (stakeholders) ihtiyaçlarını dengeleyerek ekip üyeleri arasında iletişim (communication) sağlar.
- Mimari, geliştirme sürecinde erken tasarım kararlarını (early design decisions) temsil eder ve bu kararlar, uygulama gereksinimlerinin (requirements) değişmesi durumunda esneklik (agility) sağlar.

Mimari Tasarımın Ürünleri (Products of Architectural Design)

- Yazılım Tasarım Belgesi (Software Design Document - SDD), mimari diyagramlar (architectural diagrams) ve UML diyagramları (Unified Modeling Language diagrams) gibi çeşitli belgeler, yazılım tasarımını paydaşlara iletmek için kullanılır.
- SDD, tasarımın nasıl uygulanacağına dair teknik spesifikasyonlar (technical specifications) içerirken, mimari diyagramlar bileşenlerin etkileşimlerini gösterir. UML diyagramları ise yapı (structure) ve davranışı (behavior) temsil eder.

Yazılım Tasarımının Temelleri (Fundamentals of Software Design)

- Yazılım tasarımı, yazılımın yapısal bileşenleri (structural components) ve davranışsal özelliklerinin (behavioral attributes) belgelenmesi sürecidir.
- Modüler bir yapı (modular structure) oluşturmak için yapılandırılmış tasarım (structured design), yazılım problemlerini daha küçük çözüm öğelerine (solution elements) ayırır.

Birleşik Modelleme Dili (Unified Modeling Language - UML)

- UML, karmaşık yazılım sistemlerinin mimarisini (architecture), tasarımını (design) ve uygulamasını (implementation) görsel olarak temsil etmek için kullanılan standart bir modelleme dilidir.
- UML diyagramları (UML diagrams), yazılım geliştirme süreçlerinde iletişimi (communication) kolaylaştırır ve kodlama öncesi planlama (planning) yapmayı sağlar.

Davranışsal Modeller (Behavioral Models)

- Davranışsal modeller, bir sistemin ne yaptığını (what a system does) açıklarken, nasıl yaptığını (how it does it) açıklamaz.

- Durum geçiş diyagramları (state transition diagrams) ve etkileşim diyagramları (interaction diagrams), sistemin davranışını (behavior) ve nesneler arasındaki ilişkileri (relationships between objects) modellemek için kullanılır.

Nesne Yönelimli Analiz ve Tasarımın Temelleri

- OOAD, yazılım sistemlerini analiz etme ve tasarlama yaklaşımıdır ve nesne yönelimli programlama dilleri (object-oriented programming languages) kullanılarak geliştirilir.
- Nesneler (objects), veri (data) ve bu veriye ilişkin davranışları (behaviors) içerir.

Sınıflar ve Nesneler

- Sınıf (class), nesnelerin genel bir versiyonudur ve nesneler, sınıflardan (classes) türetilen özel örneklerdir (instances).
- Sınıflar, nesnenin genel özelliklerini (attributes) ve yöntemlerini (methods) tanımlar, ancak nesne oluşturulduğunda (instantiation) bu özellikler belirli değerlere atanır.

Sınıf Diyagramları (Class Diagrams)

- Sınıf diyagramları, nesne yönelimli tasarımda (object-oriented design) yazılım sisteminin yapısını (structure) gösterir.
- Her kutu bir sınıfı temsil eder ve sınıflar arasındaki ilişkileri (relationships) gösterir; alt sınıflar (subclasses), üst sınıfların (parent classes) özelliklerini miras alır (inheritance).

Bu içerik, uygulama mimarisi yaklaşımlarını ve bileşen tabanlı mimarileri açıklamaktadır.

Bileşenlerin Özellikleri (Characteristics of Components)

- Bileşenler (Components), yeniden kullanılabilir (reusable), değiştirilebilir (replaceable), bağımsız (independent), genişletilebilir (extensible), kapsüllenmiş (encapsulated) ve bağlamdan bağımsız (non-context specific) olmalıdır.
- Her bir bileşen, diğer bileşenlerle birlikte çalışan kapsüllenmiş bir işlevsellik birimidir (unit of encapsulated functionality).

Bileşen Tabanlı Mimari (Component-Based Architecture)

- Bileşen tabanlı mimari, tasarımın mantıksal bileşenlere (logical components) ayrılmasını sağlar ve bağımsız bileşenlerin (independent components) bir arada çalışmasını tanımlar.
- Bileşenler, nesnelerden (objects) oluşur ve hizmetler (services), bileşenlerin bir türüdür (type of component).

Hizmetler ve Dağıtık Sistemler (Services and Distributed Systems)

- Hizmetler, bağımsız olarak dağıtılabilen (deployed independently) ve birden fazla sistem (multiple systems) tarafından yeniden kullanılabilen işlevsellik birimleridir.

- Dağıtık sistemler (distributed systems), farklı makinelerde (different machines) bulunan birden fazla hizmetin etkileşimde bulunduğu sistemlerdir ve kullanıcıya tek bir sistem gibi görünür (single coherent system).

Hizmetler (Services) ve bileşenler (Components) arasındaki ilişki şu şekilde tanımlanabilir:

- **Bileşenler (Components)**, uygulama içinde belirli işlevsellikleri kapsayan bağımsız birimlerden oluşur. Her bileşen, diğer bileşenlerle birlikte çalışarak uygulamanın genel işlevselliğini sağlar.
- **Hizmetler (Services)**, bileşenlerin bir türüdür ve bağımsız olarak dağıtılabilirler. Hizmetler, belirli bir iş ihtiyacını karşılamak için tasarlanmış işlevsellik birimleridir ve genellikle birden fazla sistem tarafından yeniden kullanılabilir.
- **İlişki**: Hizmetler, bileşenlerden oluşur. Yani, bir hizmet, bir veya daha fazla bileşeni içerebilir ve bu bileşenler, hizmetin sağladığı işlevselliği destekler. Hizmetler, bileşenlerin bir araya gelerek daha büyük ve karmaşık işlevsellikler sunmasını sağlar.

Bu şekilde, bileşenler, hizmetlerin yapı taşlarını oluştururken, hizmetler de bileşenlerin bir araya gelerek sunduğu daha yüksek seviyeli işlevsellikleri temsil eder.

Hizmetler (Services) ve bileşenler (Components) arasındaki ilişkiyi daha iyi anlamak için bazı örnekler üzerinden gidelim:

Örnek 1: E-Ticaret Uygulaması

- **Hizmet (Service)**: **Ödeme İşleme Servisi (Payment Processing Service)**
    - **Bileşenler (Components)**:
        - **Ödeme Geçidi Bileşeni (Payment Gateway Component)**: Kullanıcıların kredi kartı bilgilerini güvenli bir şekilde işlemek için kullanılır.
        - **Fatura Oluşturma Bileşeni (Invoice Generation Component)**: Kullanıcının yaptığı alışverişin faturasını oluşturur.
        - **Hesap Doğrulama Bileşeni (Account Verification Component)**: Kullanıcının hesabının geçerliliğini kontrol eder.

Örnek 2: Sosyal Medya Uygulaması

- **Hizmet (Service)**: **Kullanıcı Profili Servisi (User Profile Service)**
    - **Bileşenler (Components)**:
        - **Profil Bilgileri Bileşeni (Profile Information Component)**: Kullanıcının ad, soyad, profil resmi gibi bilgilerini saklar.
        - **Arkadaş Listesi Bileşeni (Friends List Component)**: Kullanıcının arkadaşlarının listesini yönetir.
        - **Gizlilik Ayarları Bileşeni (Privacy Settings Component)**: Kullanıcının gizlilik ayarlarını yapılandırmasına olanak tanır.

Örnek 3: Bankacılık Uygulaması

- **Hizmet (Service)**: **Hesap Yönetimi Servisi (Account Management Service)**
    - **Bileşenler (Components)**:
        - **Hesap Bilgileri Bileşeni (Account Information Component)**: Kullanıcının hesap bakiyesi ve işlem geçmişi gibi bilgileri gösterir.
        - **Para Transferi Bileşeni (Money Transfer Component)**: Kullanıcının başka bir hesaba para transferi yapmasını sağlar.
        - **Hesap Güvenliği Bileşeni (Account Security Component)**: Kullanıcının hesabını korumak için güvenlik önlemleri alır.

İlişkiyi Anlamak

Bu örneklerde, her hizmet belirli bir işlevselliği yerine getirirken, bu işlevselliği destekleyen bir veya daha fazla bileşen içerir. Hizmetler, bileşenlerin bir araya gelerek sunduğu daha karmaşık işlevsellikleri temsil eder. Örneğin, "Ödeme İşleme Servisi", kullanıcıların ödemelerini güvenli bir şekilde gerçekleştirmelerini sağlarken, bu hizmetin içinde yer alan bileşenler, bu sürecin farklı yönlerini yönetir.

Bu içerik, yazılım mimarisi (software architecture) ve farklı mimari desenler (architectural patterns) hakkında bilgi vermektedir.

Yazılım Mimari Desenleri (Architectural Patterns)

- **2-Tier Mimari (2-Tier Architecture)**: İki katmanlı bir modeldir; istemci (client), sunucuya (server) veri veya hizmet talep eder. Örnek: Metin mesajlaşma uygulamaları (text messaging apps).
- **3-Tier Mimari (3-Tier Architecture)**: Üç katmandan oluşur: sunum katmanı (presentation tier - kullanıcı arayüzü), uygulama katmanı (application tier - iş mantığı) ve veri katmanı (data tier - veri yönetimi). Örnek: Web uygulamaları (web apps).

Diğer Mimari Desenler (Other Architectural Patterns)

- **Peer-to-Peer (P2P)**: Dağıtık bir ağ yapısıdır; her düğüm (node) hem istemci hem de sunucu olarak çalışır. Örnek: Kripto paralar (cryptocurrencies - Bitcoin, Ethereum).
- **Olay Tabanlı Mimari (Event-Driven Architecture)**: Olayların (events) üreticileri (producers) ve tüketicileri (consumers) arasında iletişim kurar. Örnek: Araç paylaşım uygulamaları (ride-sharing apps - Lyft, Uber).
- **Mikroservisler (Microservices)**: Uygulamayı modüler bileşenlere (modular components) ayırır; her bir bileşen API (Application Programming Interface) aracılığıyla iletişim kurar. Örnek: Sosyal medya siteleri (social media sites).

Bu mimari desenler, yazılım sistemlerinin tasarımında tekrar eden çözümler sunar ve bazıları bir arada kullanılabilir.

2-Tier Mimari (2-Tier Architecture)

2-Tier mimari, iki katmandan oluşan bir yazılım mimarisi modelidir. Bu modelde, istemci (client) ve sunucu (server) arasında doğrudan bir iletişim vardır. İstemci, sunucudan veri veya hizmet talep eder ve sunucu bu talepleri karşılar.

- **Katmanlar**:
    - **İstemci Katmanı (Client Layer)**: Kullanıcı arayüzü (user interface) burada bulunur. Kullanıcı, uygulama ile etkileşimde bulunur ve sunucuya talepler gönderir.
    - **Sunucu Katmanı (Server Layer)**: Veritabanı (database) ve iş mantığı (business logic) burada yer alır. Sunucu, istemciden gelen talepleri işler ve gerekli verileri geri gönderir.
- **Örnek**: Metin mesajlaşma uygulamaları, burada bir istemci mesaj gönderir ve sunucu bu mesajı alır ve ilgili alıcıya iletir. Ayrıca, veritabanı istemcileri (database clients) de bu mimariyi kullanarak veritabanı sunucularına bağlanabilir.

3-Tier Mimari (3-Tier Architecture)

3-Tier mimari, üç katmandan oluşan daha karmaşık bir yazılım mimarisi modelidir. Bu model, uygulamanın farklı bileşenlerini ayrı katmanlarda organize ederek daha iyi bir yapı sağlar. Her katman, yalnızca doğrudan üstteki veya alttaki katmanla iletişim kurar.

- **Katmanlar**:
    - **Sunum Katmanı (Presentation Tier)**: Kullanıcı arayüzü burada bulunur. Kullanıcı, uygulama ile etkileşimde bulunur ve bu katman, kullanıcıdan gelen talepleri alır.
    - **Uygulama Katmanı (Application Tier)**: İş mantığı burada işlenir. Bu katman, sunum katmanından gelen talepleri alır, gerekli işlemleri yapar ve sonuçları sunum katmanına iletir.
    - **Veri Katmanı (Data Tier)**: Verilerin depolandığı ve yönetildiği yerdir. Uygulama katmanı, veri katmanına erişerek gerekli verileri alır veya günceller.
- **Örnek**: Bir web uygulaması, sunum katmanında kullanıcı arayüzü ile başlar, uygulama katmanında kullanıcı giriş bilgilerini işler ve veri katmanında kullanıcı verilerini depolar. Bu yapı, her katmanın bağımsız olarak güncellenmesine ve yönetilmesine olanak tanır.

Özetle:

- 2-Tier mimari, istemci ve sunucu arasında doğrudan bir iletişim kurarken, 3-Tier mimari, uygulamanın farklı bileşenlerini ayrı katmanlarda organize ederek daha iyi bir yapı sağlar. Bu, uygulamanın bakımını ve genişletilmesini kolaylaştırır.

Uygulama Ortamları (Application Environments)

- Uygulama ortamları, bir uygulamanın çalışması için gereken donanım (hardware) ve yazılım (software) kaynaklarının birleşimidir. Bu, uygulama kodu (application code), yazılım yığını (software stack), ağ bileşenleri (network components) ve fiziksel veya sanal donanımı içerir.

Ön Üretim Ortamları (Pre-Production Environments)

- Ön üretim ortamları, uygulamanın üretim için hazırlandığı çeşitli platformlardır. Yaygın ön üretim ortamları şunlardır:
    - **Geliştirme Ortamı (Development Environment)**: Uygulamanın aktif olarak kodlandığı platformdur.
    - **QA Ortamı (Quality Assurance Environment)**: Uygulamanın bileşenlerinin test edildiği ortamdır.
    - **Staging Ortamı (Staging Environment)**: Üretim ortamına en yakın olan, ancak genel kullanıcılar için tasarlanmamış bir ortamdır.

Üretim Ortamı (Production Environment)

- Üretim ortamı, uygulamanın tüm kullanıcılar için çalıştığı, donanım ve yazılımın tamamını içeren ortamdır. Bu ortam, güvenlik, güvenilirlik ve ölçeklenebilirlik gibi işlevsel olmayan gereksinimleri dikkate almalıdır.

Dağıtım Seçenekleri (Deployment Options)

- Uygulama ortamları, iki ana dağıtım seçeneği ile dağıtılabilir:
    - **Yerinde Dağıtım (On-Premises Deployment)**: Sistem ve altyapı, organizasyonun fiziksel konumunda bulunur. Güvenlik ve kontrol açısından avantaj sağlar, ancak genellikle daha pahalıdır.
    - **Bulut Dağıtımı (Cloud Deployment)**: Üç tür bulut dağıtım modeli vardır:
        - **Açık Bulut (Public Cloud)**: Bulut sağlayıcısının donanımında, internet üzerinden erişilen altyapıdır.
        - **Özel Bulut (Private Cloud)**: Tek bir organizasyon için ayrılmış bulut altyapısıdır.
        - **Hibrit Bulut (Hybrid Cloud)**: Hem açık hem de özel bulutların avantajlarını bir arada sunan bir modeldir.

Bu içerikte, uygulama ortamlarının türleri, işlevleri ve dağıtım seçenekleri hakkında temel bilgiler sunulmaktadır.

Üretim Ortamı (Production Environment) ile Ön Üretim Ortamı (Pre-Production Environment) Arasındaki Farklar:

1. **Amaç**:
    - **Üretim Ortamı**: Uygulamanın son kullanıcılar tarafından erişildiği ve kullanıldığı ortamdır. Gerçek verilerle çalışır ve uygulamanın tüm işlevselliğini sunar.
    - **Ön Üretim Ortamı**: Uygulamanın geliştirilmesi, test edilmesi ve üretime hazırlanması için kullanılan ortamdır. Gerçek kullanıcılar tarafından kullanılmaz.

2. **Kullanıcı Erişimi**:
    - **Üretim Ortamı**: Genel kullanıcılar (son kullanıcılar) tarafından erişilebilir.
    - **Ön Üretim Ortamı**: Sadece geliştiriciler, test uzmanları ve belirli ekip üyeleri tarafından erişilebilir.
3. **Veri Kullanımı**:
    - **Üretim Ortamı**: Gerçek ve canlı verilerle çalışır. Kullanıcıların verileri burada depolanır ve işlenir.
    - **Ön Üretim Ortamı**: Genellikle sahte (dummy) veya test verileri kullanılır. Gerçek verilerin kullanılmaması, testlerin güvenli bir şekilde yapılmasını sağlar.
4. **Güvenlik ve Stabilite**:
    - **Üretim Ortamı**: Yüksek güvenlik ve stabilite gerektirir. Kullanıcıların verileri korunmalı ve uygulama sürekli çalışır durumda olmalıdır.
    - **Ön Üretim Ortamı**: Güvenlik önlemleri alınsa da, üretim ortamındaki kadar katı olmayabilir. Test süreçleri sırasında hatalar ve kesintiler kabul edilebilir.
5. **Performans ve Yük**:
    - **Üretim Ortamı**: Yüksek performans ve ölçeklenebilirlik gerektirir, çünkü çok sayıda kullanıcı aynı anda erişebilir.
    - **Ön Üretim Ortamı**: Performans testleri yapılabilir, ancak gerçek kullanıcı yükü ile karşılaşmaz.

Özetle: Üretim ortamı, son kullanıcıların eriştiği ve gerçek verilerle çalıştığı bir ortamken, ön üretim ortamı uygulamanın geliştirilmesi ve test edilmesi için kullanılan, genellikle sahte verilerle çalışan bir ortamdır.

Bu içerik, üretim ortamında (production environment) uygulama dağıtımı için gerekli olan bileşenleri tanımlamayı amaçlamaktadır.

Bileşenlerin Tanımı (Definition of Components)

- Üretim ortamında yaygın olarak kullanılan bileşenler arasında güvenlik duvarı (firewall), yük dengeleyici (load balancer), web sunucuları (web servers), uygulama sunucuları (application servers) ve veritabanı sunucuları (database servers) bulunmaktadır.
- Her bileşenin belirli bir işlevi vardır; örneğin, güvenlik duvarı ağlar arasındaki trafiği izlerken, yük dengeleyici ağ trafiğini birden fazla sunucu arasında dağıtır.

Güvenlik Duvarı ve Yük Dengeleyici (Firewall and Load Balancer)

- Güvenlik duvarı (firewall), ağlar arasındaki trafiği izleyen ve belirli güvenlik kurallarına göre veri akışını engelleyen bir güvenlik cihazıdır.

- Yük dengeleyici (load balancer), sunucular arasında ağ trafiğini verimli bir şekilde dağıtarak sunucu aşırı yüklenmesini önler.

Sunucu Türleri (Types of Servers)

- Web sunucuları (web servers), web sayfaları ve dosyalar gibi içerikleri istemcilere sunar.
- Uygulama sunucuları (application servers), iş mantığını (business logic) çalıştırarak istemcilere uygulama hizmeti sağlar.
- Veritabanı sunucuları (database servers), verilerin depolanmasını ve akışını kontrol eden bir veritabanı yönetim sistemi (DBMS) ile çalışır.

Bu özet, üretim ortamında uygulama dağıtımı için gerekli olan temel bileşenleri ve işlevlerini açıklamaktadır.

Canarying, yazılım dağıtımında kullanılan bir stratejidir ve yeni kod değişikliklerini veya güncellemeleri, sistemin tamamına yaymadan önce küçük bir kullanıcı grubuna (canary users) sunmayı içerir. Bu yöntem, yeni kodun sistemde beklenmedik sorunlara yol açıp açmadığını test etmek için kullanılır.

Canarying'in Temel Özellikleri:

- **Risk Azaltma:** Yeni bir özellik veya güncelleme, tüm kullanıcılar yerine sadece belirli bir grup üzerinde test edilerek, olası hataların etkisi minimize edilir.
- **Geribildirim Alma:** Canary kullanıcıları, yeni özellikler hakkında geri bildirim vererek, geliştiricilerin sorunları hızlı bir şekilde tespit etmesine ve düzeltmesine yardımcı olur.
- **Aşamalı Dağıtım:** Eğer canary sürümü başarılı olursa, güncelleme daha geniş bir kullanıcı grubuna yayılabilir. Eğer sorunlar tespit edilirse, güncelleme geri alınabilir (rollback).

Örneğin, bir yazılım şirketi yeni bir özellik geliştirdiğinde, bu özelliği önce sadece %5'lik bir kullanıcı grubuna sunabilir. Eğer bu grup sorunsuz bir deneyim yaşarsa, özellik daha geniş bir kitleye açılır. Ancak, eğer sorunlar yaşanırsa, bu durum tüm kullanıcıları etkilemeden düzeltilebilir.

Canarying, yazılım geliştirme sürecinde güvenliği artırmak ve kullanıcı deneyimini iyileştirmek için etkili bir yöntemdir.