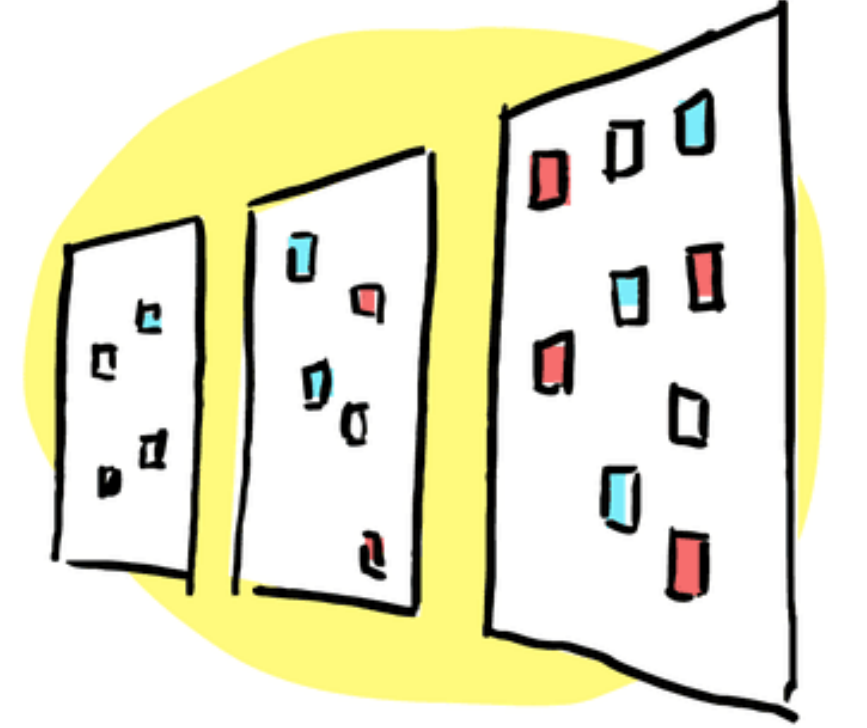


Yazılım Geliřtirmede Çevik Yöntemler 9. Hafta

15.04.2025



Mühendislik
Fakültesi
Bilgisayar Mühendisliği

Hazırlayan: Dr. Ercan Ezir

GİRİŞ

Domain Driven Development ve CI/CD süreçleri

DOMAIN-DRIVEN DESIGN(DDD)

ALAN ODAKLI TASARIM

DDD, karmaşık iş alanlarının yönetimini sağlayan bir yazılım geliştirme yaklaşımıdır.

Yazılım geliştirirken iş alanının tam anlaşılmasını hedefler.

Eric Evans tarafından 2003'te ortaya atılmıştır.

NEDEN DDD KULLANILIR?

İş ve yazılım ekipleri arasında ortak dil sağlar.

Yazılım karmaşıklığını azaltır.

Yazılımı iş süreçlerine uygun hale getirir.

DDD ÖZELİNDE TEMEL KAVRAMLAR

Alan (Domain)

Bağlam (Context)

Model (Model)

DOMAIN (ALAN) NEDİR?

İş süreçlerinin, kurallarının ve faaliyetlerinin gerçekleştiği alandır.



Örneğin, e-ticaret uygulamasının domain'i "ürün satışı" olabilir.

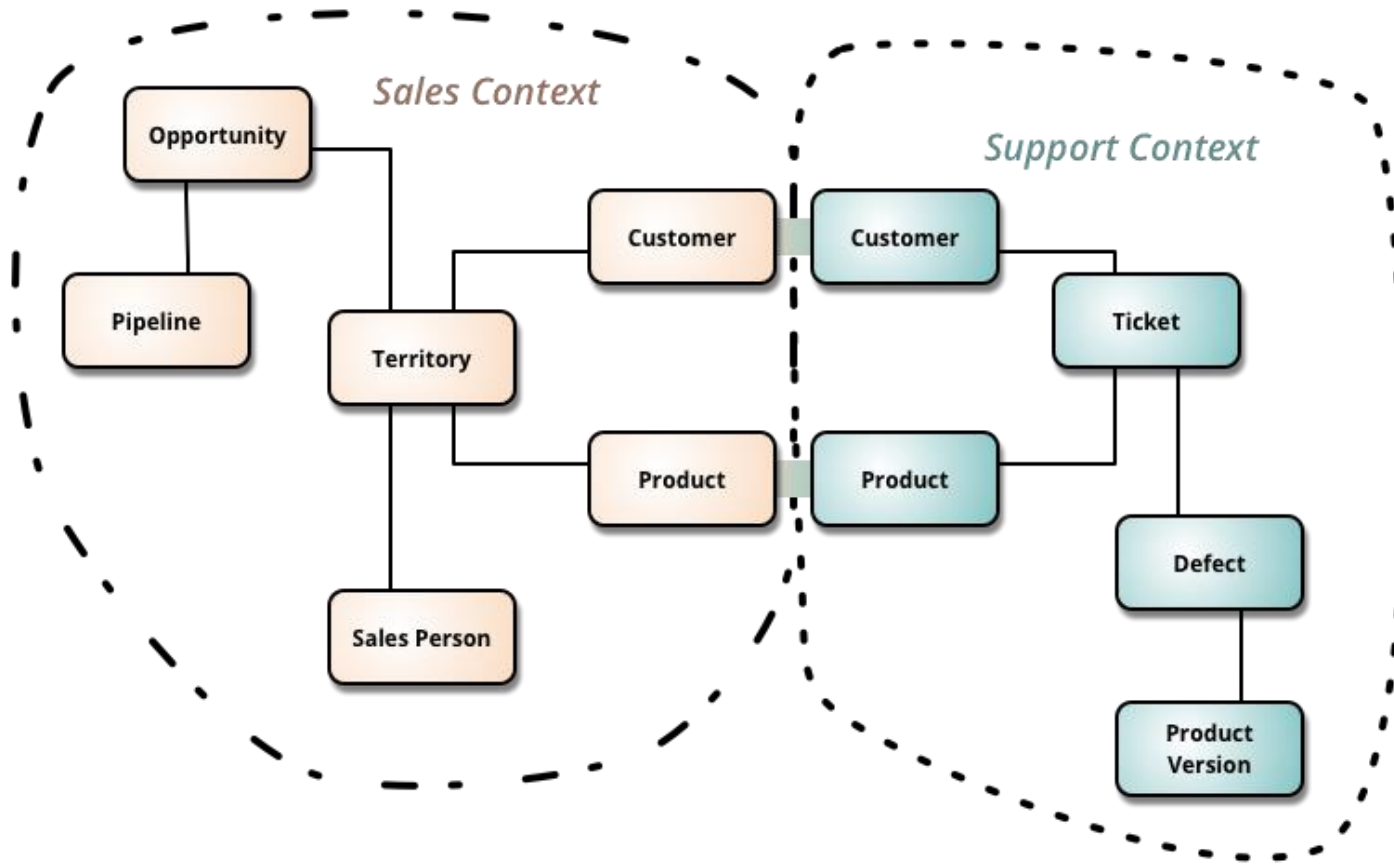
CONTEXT (BAĞLAM) NEDİR?

İş süreçlerinin
farklı bölümlerini
tanımlayan durum
ve çevredir.



Farklı bağlamlarda
aynı terim farklı
anlamlar taşıyabilir.

Örnek: **Sipariş** kavramı bir müşteri için almış olduğu ürünü, fakat ürünün tedarikini yöneten kısım için yollanacak ürünü temsil eder.



CONTEXT

MODEL NEDİR?



Domainin sistematik ve soyut bir temsilidir.



İş süreçlerini, nesneleri ve ilişkileri içerir.

DDD'NIN İKİ TEMEL TASARIMI

- Stratejik Tasarım (Strategic Design)
- Taktiksel Tasarım (Tactical Design)



STRATEJİK TASARIM

Yazılımın temel iş stratejilerine uygunluğunu sağlar.



Temel bileşenleri:
Ubiquitous
Language ve
Bounded Context.

UBIQUITOUS LANGUAGE (YAYGIN DİL)

- İş ve yazılım ekiplerinin kullandığı ortak dil.
- Terimler ve işlemler yazılım kodunda ve iş süreçlerinde aynıdır.

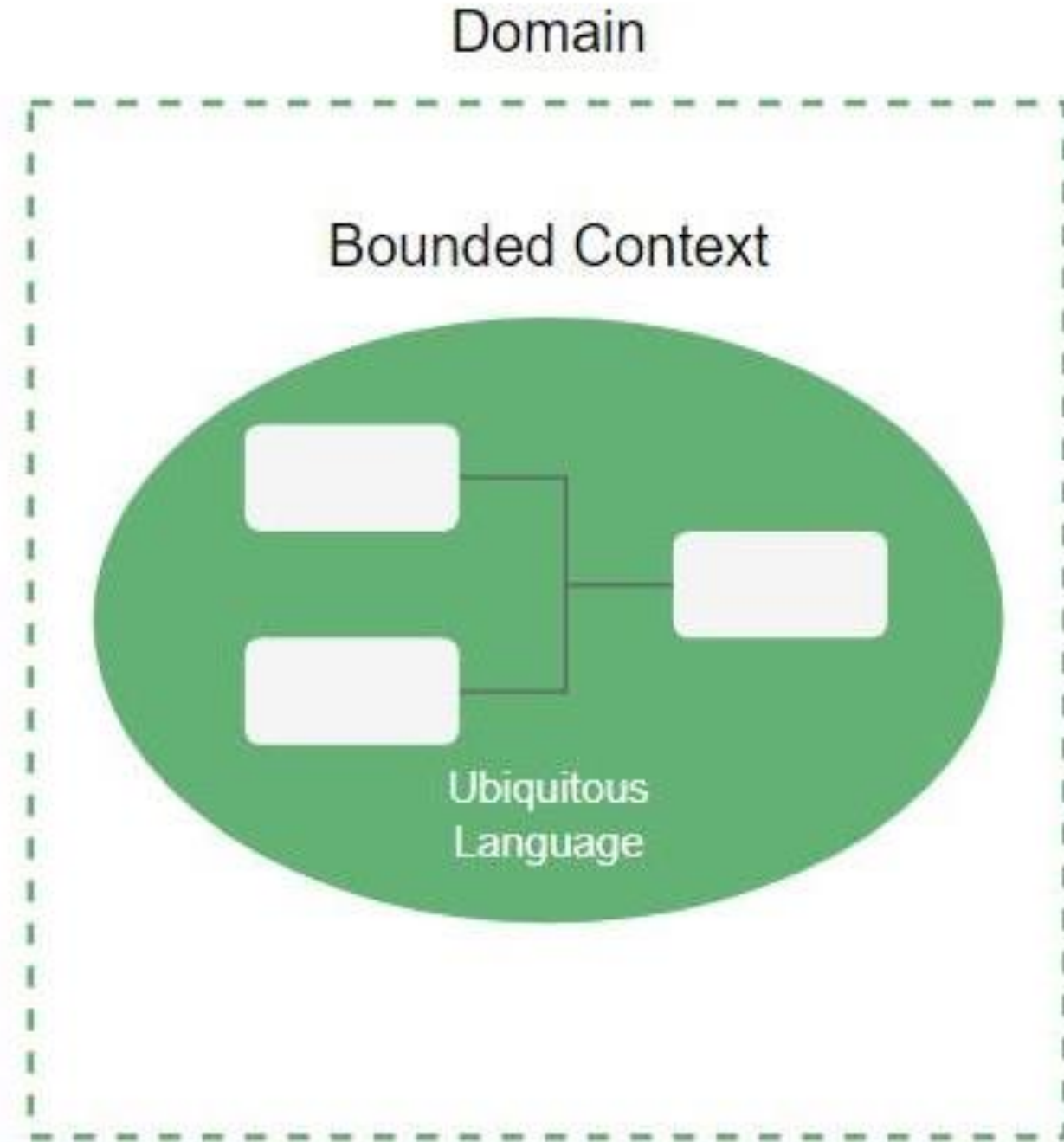
```
public class Product {  
    public int Id { get; set; }  
    public string Name { get; set; }  
    public decimal Price { get; set; }  
  
    public void UpdatePrice(decimal newPrice) {  
        Price = newPrice;  
    }  
}
```

BOUNDED CONTEXT (SINIRLI BAĞLAM)

Domainin alt
alanlara
ayrılmasıyla oluşur.

Her bağlamın
kendi tutarlılığı ve
dil kullanımı vardır.

Örnek: Sipariş
Yönetimi, Stok
Yönetimi gibi
bağlamlar.



TAKTİKSEL TASARIM

Yazılımın uygulama detaylarını içerir.

Temel bileşenler: Entity, Value Object, Aggregate, Domain Services, Domain Events.

ENTITY (VARLIK) NEDİR?

```
public class Customer {  
    public int Id { get; set; }  
    public string Name { get; set; }  
  
    public void UpdateName(string newName) {  
        Name = newName;  
    }  
}
```

- Benzersiz kimliği olan, değişebilen ve devamlılığı olan nesneler.

VALUE OBJECT (DEĞER NESNESİ)

Değerlere göre eşitliği
belirlenen, değiştirilemez
(immutable) nesneler.

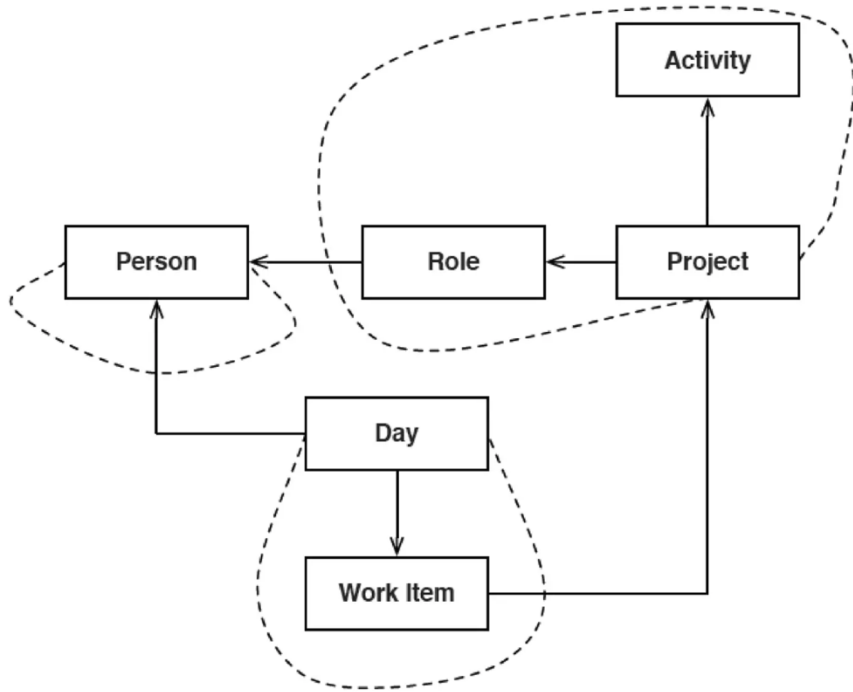
```
public class Address {  
    public string Street { get; }  
    public string City { get; }  
  
    public Address(string street, string city) {  
        Street = street;  
        City = city;  
    }  
}
```


ENTITY VS VALUE OBJECT

- Entity kimlikle ayırt edilir, Value Object ise sadece değerlerle.



AGGREGATE NEDİR?



- İlgili nesnelerin birleşiminden oluşan tutarlılık sınırıdır.
- Aggregate Root: Aggregate'i yöneten temel entity'dir.
- Aggregate Root dışarıdan erişilebilir, diğer nesneler yalnızca root üzerinden erişilir.

```
public class Order {  
    public int Id { get; set; }  
    public Customer Customer { get; set; }  
    public List<OrderItem> Items { get; set; }  
}
```

```
public interface ICustomerRepository {  
    Customer GetById(int id);  
    void Save(Customer customer);  
}
```

REPOSITORY VE SERVICES

Veri erişimini ve iş kurallarını soyutlar.

```
public class PaymentService {  
    public void ProcessPayment(Order order) {  
        // Ödeme mantığı  
    }  
}
```

DOMAIN SERVICES (ALAN HİZMETLERİ)

Birden fazla Aggregate veya Entity ile etkileşime girerek iş mantığını uygular.

```
public class OrderCompletedEvent {  
    public int OrderId { get; }  
    public DateTime CompletedAt { get; }  
  
    public OrderCompletedEvent(int orderId) {  
        OrderId = orderId;  
        CompletedAt = DateTime.Now;  
    }  
}
```

DOMAIN EVENTS (ALAN OLAYLARI)

- İş alanında önemli olayları temsil eden nesneler.

ÖRNEK DOMAIN HARİTASI



ÖRNEK DOMAIN HARİTASI

Supporting Domains

Core Domain



DDD KATMANLI MİMARİ

DDD'de uygulamalar genellikle 4 katmana ayrılır.

Presentation Layer/User Interface (Sunum Katmanı)

- Kullanıcı arayüzleri veya API'ler aracılığıyla etkileşim sağlar.

Application Layer (Uygulama Katmanı)

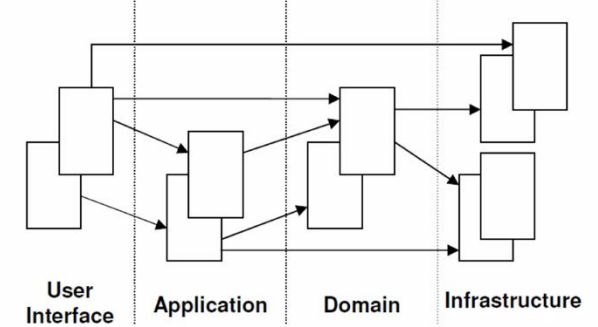
- Domain katmanı ile etkileşimi yönetir, uygulama mantığını içerir.

Domain Layer (Alan Katmanı)

- İş mantığını barındıran ana katmandır.

Infrastructure Layer (Altyapı Katmanı)

- Veritabanı ve dış servislerle iletişim sağlar.

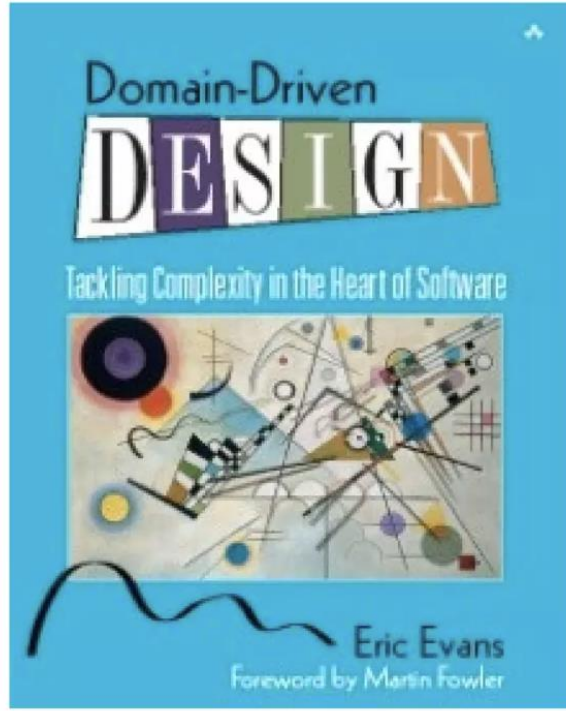


DDD'NİN AVANTAJLARI VE DEZAVANTAJLARI

Avantajlar:
Sürdürülebilirlik, ortak
dil, karmaşıklığı
yönetme.

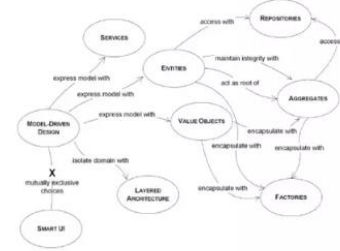


Dezavantajlar:
Başlangıçta yüksek
öğrenme eğrisi,
tasarımda daha fazla
zaman gerektirir.



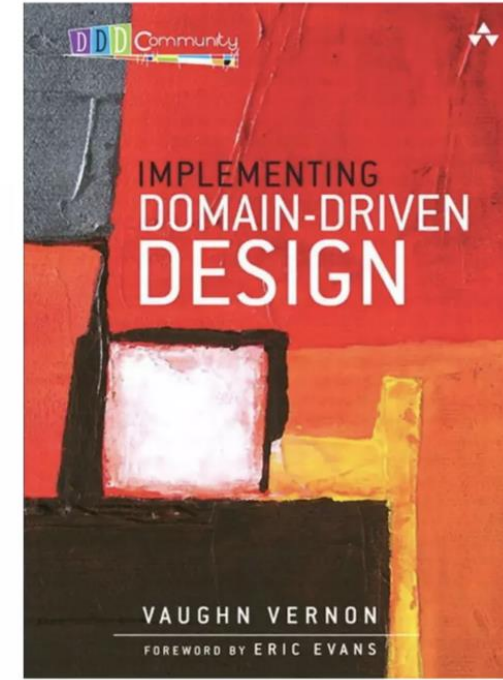
A Summary of Eric Evans' *Domain-Driven Design*

Domain-Driven Design *Quickly*



by Abel Avram & Floyd Marinescu
edited by: Don Borge Johnson, Vladimir Gilevich

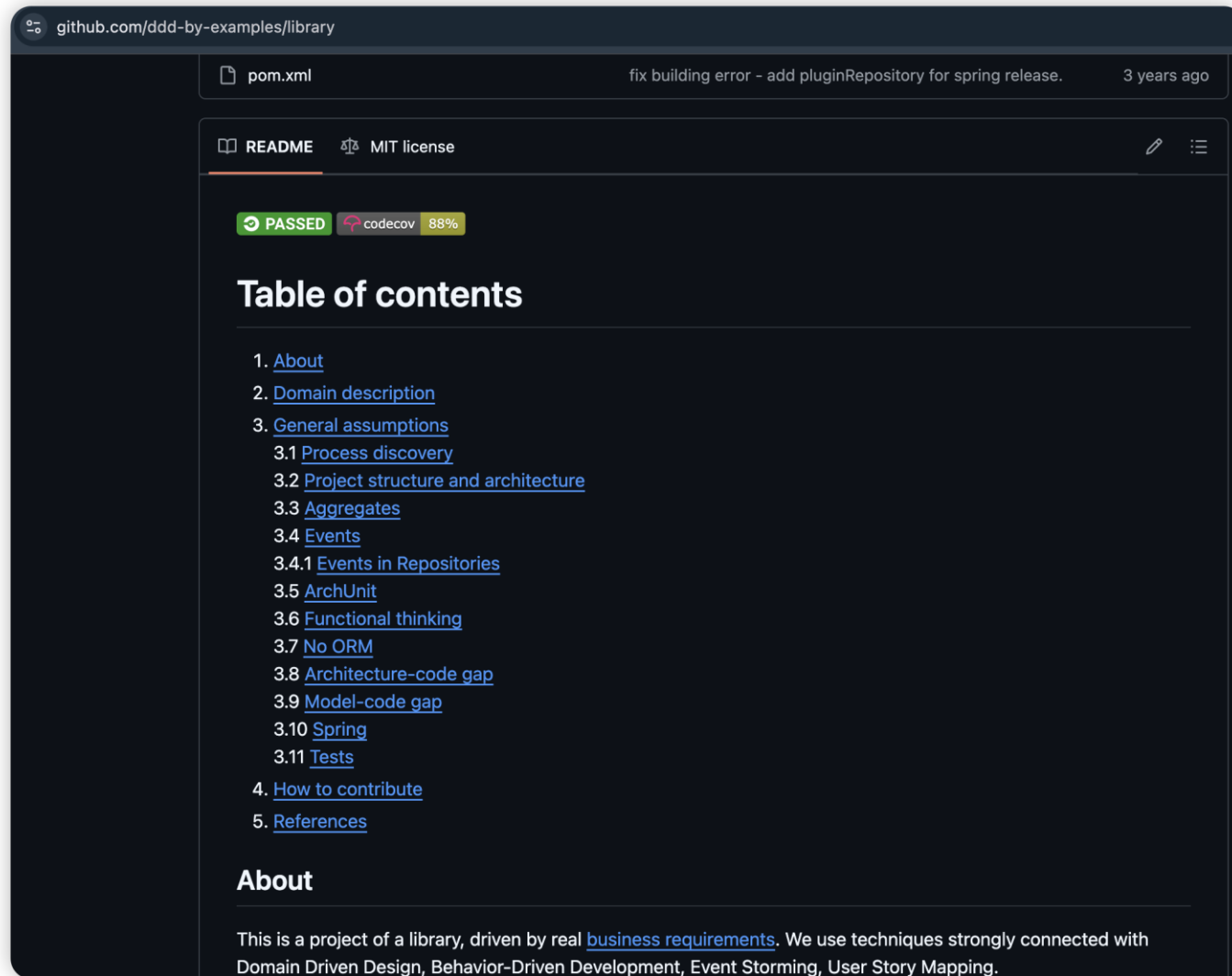
InfoQ Enterprise Software Development Series



KAYNAK KİTAPLAR

ÖRNEK DDD PROJESİ

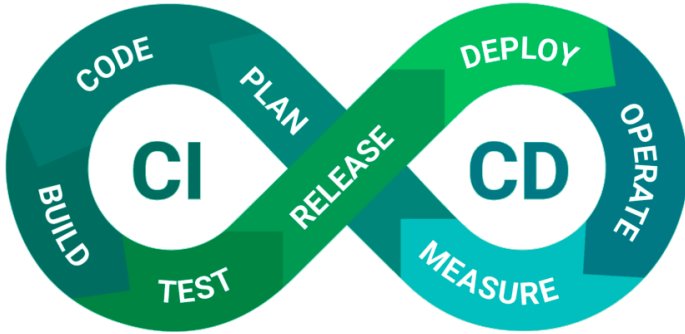
This is a project of a **library**, driven by real business requirements. We use techniques **strongly** connected with Domain Driven Design, Behavior-Driven Development, Event Storming, User Story Mapping.



The screenshot shows the GitHub repository page for 'ddd-by-examples/library'. The browser address bar displays 'github.com/ddd-by-examples/library'. The repository name 'pom.xml' is shown at the top, along with a commit message 'fix building error - add pluginRepository for spring release.' and the time '3 years ago'. Below the repository name, there are links for 'README' and 'MIT license'. A green 'PASSED' badge and a 'codecov 88%' badge are visible. The 'Table of contents' section lists the following items: 1. [About](#), 2. [Domain description](#), 3. [General assumptions](#), 3.1 [Process discovery](#), 3.2 [Project structure and architecture](#), 3.3 [Aggregates](#), 3.4 [Events](#), 3.4.1 [Events in Repositories](#), 3.5 [ArchUnit](#), 3.6 [Functional thinking](#), 3.7 [No ORM](#), 3.8 [Architecture-code gap](#), 3.9 [Model-code gap](#), 3.10 [Spring](#), 3.11 [Tests](#), 4. [How to contribute](#), and 5. [References](#). The 'About' section is partially visible at the bottom, starting with 'This is a project of a library, driven by real [business requirements](#). We use techniques strongly connected with Domain Driven Design, Behavior-Driven Development, Event Storming, User Story Mapping.'

<https://github.com/ddd-by-examples/library>

CI/CD SÜREÇLERİ



- **Continuous Integration/Continuous Delivery-Deployment**, modern yazılım geliştirme süreçlerinde kalite ve hızı artırır ve DevOps uygulamalarının vazgeçilmez parçasıdır.
- Sürekli Entegrasyon, her kod değişikliğinin otomatik test ile kontrolünü sağlar ve sistemi bozmadığından emin olmaya yarar.
- Sürekli Teslim, ürünün dağıtıma hazır tutulmasını mümkün kılar.
- Sürekli Dağıtım ise testleri geçen versiyonları otomatik olarak üretime alır.
- Bu süreçler, hızlı geri bildirim ve erken hata tespiti avantajı sunar.

SÜREKLİ ENTEGRASYON(CI)

Sürekli entegrasyon, kod değişikliklerinin anlık olarak sistemde birleştirilmesini sağlar.

Birim testleri ile hatalar erken tespit edilerek tüm ekip bilgilendirilir.

Otomatik build işlemi ile her commit, sistemin çalışır durumda kalmasını garantiler.

Programcılar, geri dönüşlerin 10 dakikadan kısa sürede alınmasıyla çalışmalarına odaklanır.

Bu yöntemin uygulanması, kod bütünlüğünü ve hızlı adaptasyonu destekler.

SÜREKLİ TESLİMAT (CD) & SÜREKLİ DAĞITIM

Sürekli teslimat, başarılı build'lerin belirli ortamlara otomatik veya yarı otomatik aktarılmasını ifade eder.

Teslimatın manuel onayla yapılabilmesi, dağıtım sürecine esneklik katar.

Sürekli dağıtımda, testleri geçen her versiyon otomatik olarak üretime alınır.

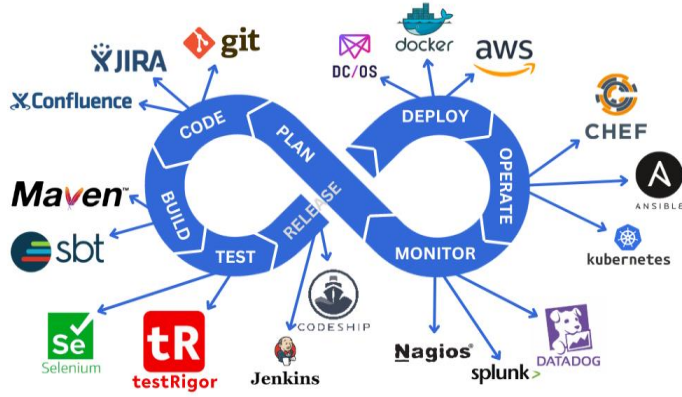
Bu model, Netflix gibi devlerin kullanımıyla örneklendirilir.

Sonuç olarak, yüksek işlevsellik ve kesintisiz müşteri deneyimi sağlanır.

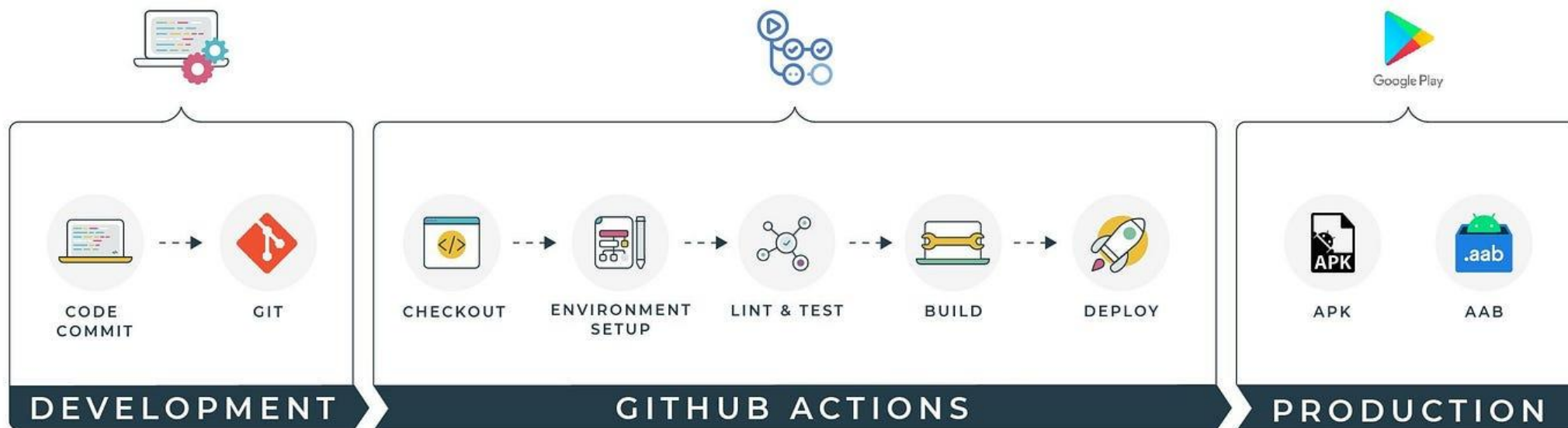
CI/CD PIPELINE VE İLKELERİ

- Pipeline, kod kontrolü, otomatik build, test ve dağıtım adımlarını kapsar.
- Tekrar eden sürümlerin destekleneceği test odaklı geliştirme prensiplerine dayanır.
- Küçük iterasyonlarla çalışmak, hataların ve yapılandırma sorunlarının anında giderilmesini sağlar.
- Otomasyonun maksimum düzeyde uygulanması, sürecin istikrarlı ilerlemesi için kritik rol oynar.
- CI/CD, sürekli geri bildirim ve ölçümlerle sistemin gerçek zamanlı izlenmesini destekler.

CI/CD ARAÇLARI VE FAYDALARI



- Jenkins, TeamCity, GitLab CI ve CircleCI gibi araçlar sektörde yaygın olarak kullanılır.
- Bu araçlar; otomatik test, gerçek zamanlı raporlama ve bulut entegrasyonu sunar.
- Verimlilik artışı, yapılandırma hatalarının azalması ve kademeli güncellemeler sağlar.
- CI/CD modelleri, Agile ve DevOps kültürüyle uyum içinde hızla yayılarak işlevselliği artırır.
- Sonuç olarak, müşteri memnuniyeti ve sistem çevikliği en üst düzeye çıkarılır.



GITHUB CI/CD

The background of the entire image is composed of numerous overlapping yellow sticky notes. Each sticky note features a simple smiley face drawn with a green marker, consisting of two dots for eyes and a curved line for a mouth. The notes are scattered across the frame, creating a textured, layered effect.

EOF*

*End of Fun/File

REFS

- <https://medium.com/goturkiye/domain-driven-design-ddd-nedir-750dc6c9641b>
- <https://www.slideshare.net/slideshow/domain-driven-design-47480496/47480496>
- <https://medium.com/empathyco/applying-ci-cd-using-github-actions-for-android-1231e40cc52f>
- <https://martinfowler.com/bliki/BoundedContext.html>