

Mimari Tutarlılığı Nasıl Sağlarım?

İhsan Baran SÖNMEZ

Software Engineer @Tubitak Bilgem YTE



Merhaba!

- @Tubitak Bilgem YTE'2019
 - Yazılım Mimarileri
 - Sanat Tarihi ve Kahve

Sunum Planı

01

PROBLEM

Mimari tutarlılığın sağlanamama nedenleri

02

SONUÇLARI

Tutarlılığın sağlanamamasının sonuçları

03

ÖNERİLER

Bu durumla karşılaşmamak için yapılabilecekler

04

ARCHUNIT

Archunit çözümü



01

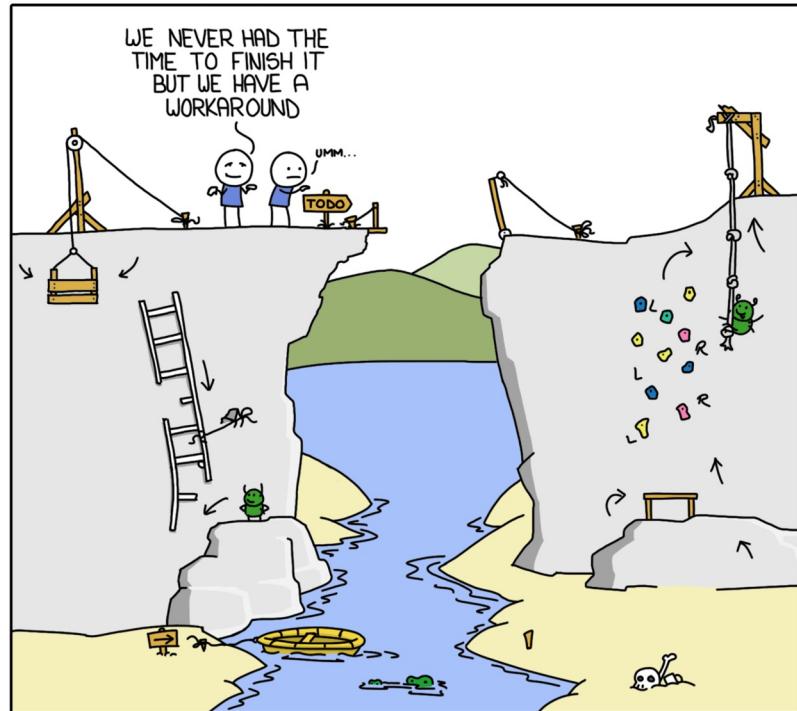
PROBLEM

Neden Tasarım Kuralları İhmal Ediliyor?

Süre Baskısı

Proje ilerledikçe, kodun daha hızlı yazılması gerekebilir ve bu durum tasarım kurallarının ihmal edilmesine yol açabilir.

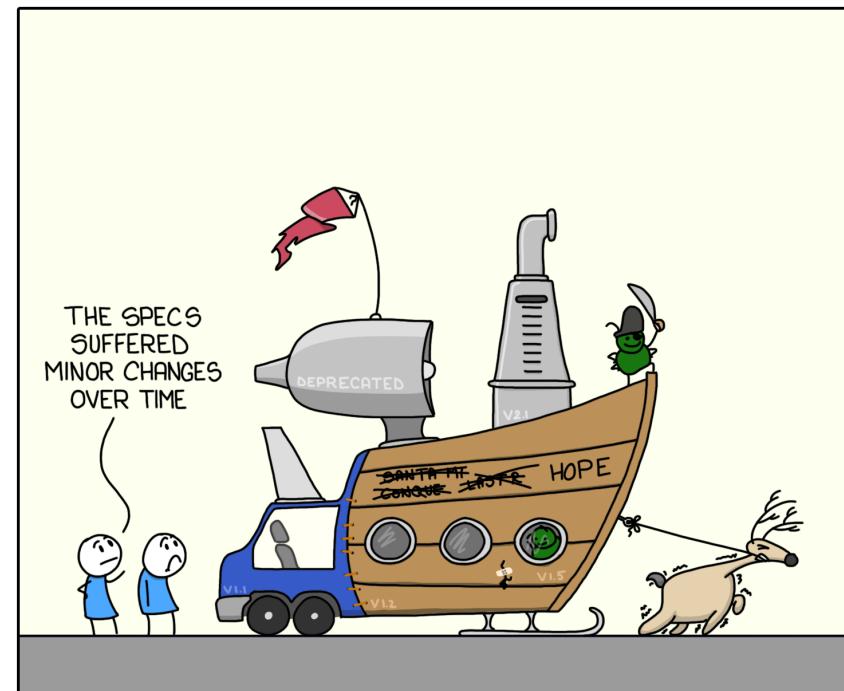
WORKAROUND



Değişen İhtiyaçlar

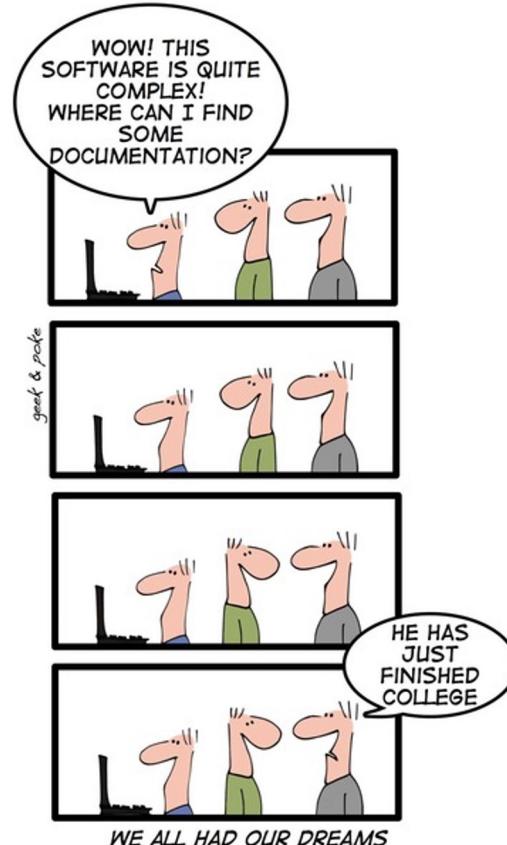
Yazılım projelerinin doğası gereği gereksinimler zaman içinde değişebilir.
Başlangıçta tasarlanan bir sistem, gereksinimlerin değişmesiyle gereksinimleri karşılayamaz hale gelebilir.

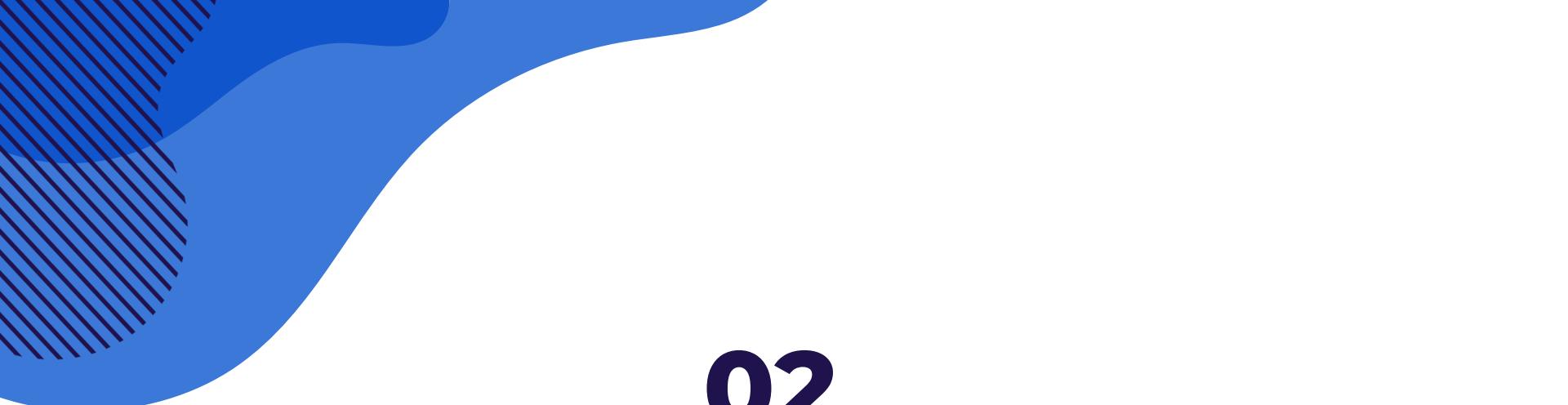
PIVOTING



Dokümantasyon Eksikliği

Başlangıçta belirlenen tasarım kurallarının yeterince dokümante edilmemiş olması da bir sorun olabilir.





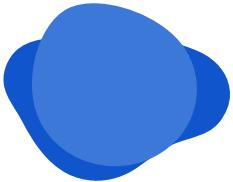
02

SONUÇLARI

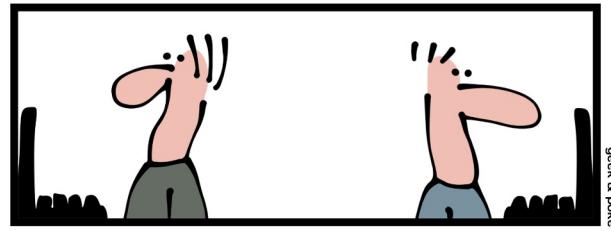
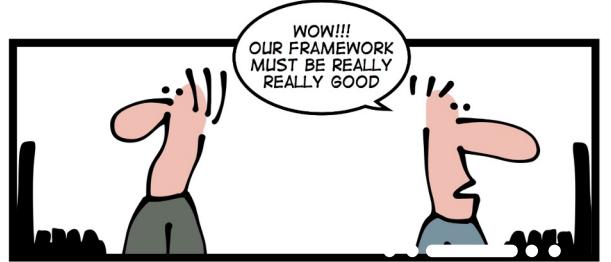
Tasarım Kurallarının İhmal Edilmesinin Sonuçları



Kod Karmaşıklığı



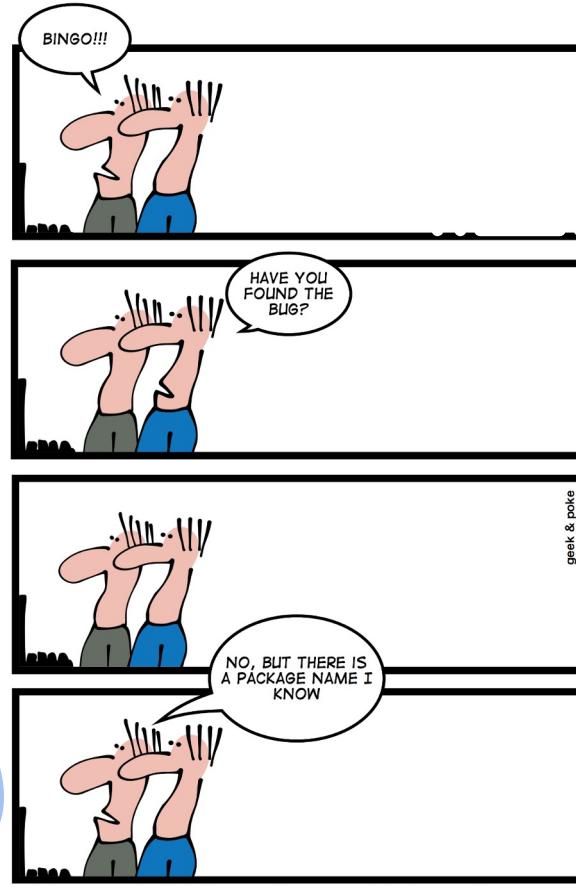
Karmaşık bir kod tabanı, bakım ve geliştirme süreçlerini zorlaştıracaktır. Aynı zamanda geliştiricilerin de verimliliğini düşürür.



See "The Law of Leaky Abstractions" from Joel Spolsky
(<http://www.joelonsoftware.com/articles/LeakyAbstractions.html>)
Over 10 years old but still true.

Hataların Artması

Karmaşık kod, hataların ortaya çıkma olasılığını artırır. Tasarım kurallarının göz ardı edilmesi veya ihmal edilmesi, kodun istenmeyen yan etkilere sahip olmasına ve hataların tespit edilmesinin daha zor olmasına neden olabilir.



geek & poke

Güncelleme Sorunları

Yeni özellikler eklerken veya hataları düzeltirken, karmaşık bir kod tabanının üzerinde çalışmak daha riskli ve zaman alıcı olabilir.

QUICK FIX



İletişim Zorlukları

Karmaşık kod, ekip içi iletişimini olumsuz etkileyebilir. Ekip üyeleri, kodun mantığını ve işleyişini anlamak için daha fazla zaman harcarlar.

SIMPLY EXPLAINED



03

ÇÖZÜM ÖNERİLERİ

Tasarım Kurallarını Korumanın Yolları

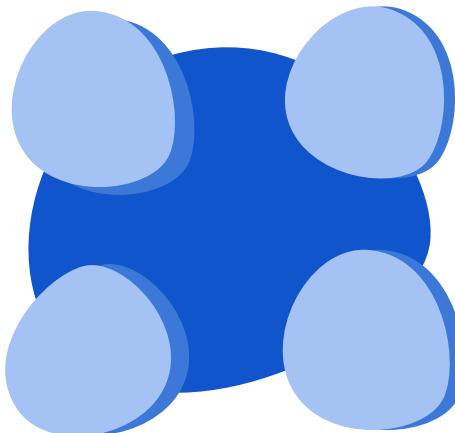
Çözüm Önerileri

Sürekli Denetim

Tasarım kurallarının proje sürecinin her aşamasında düzenli olarak denetlenmesi büyük önem taşır.

Eğitim ve Farkındalık

Ekip üyelerinin tasarım kurallarının önemini ve projenin başarısı üzerindeki etkisini anlamaları oldukça önemlidir.



İşbirliği ve İletişim

Tasarım kurallarına uymak için ekip içi iletişim ve işbirliği teşvik edilmelidir.

Sürekli İyileştirme

Proje gereksinimleri değişikçe, tasarım kuralları da güncellenmelidir.



Otomasyon

Tasarım kurallarının otomatik olarak kontrol edilmesi, insan hatalarını en aza indirir ve sürekli uyumluluğu sağlar. ArchUnit gibi araçlar, bu otomasyon görevlerini etkili bir şekilde yerine getirebilir.

04

ArchUnit

ArchUnit Nedir?

ArchUnit, yazılım projelerinde mimari bütünlüğü koruma amacıyla kullanılan bir test çerçevesidir.



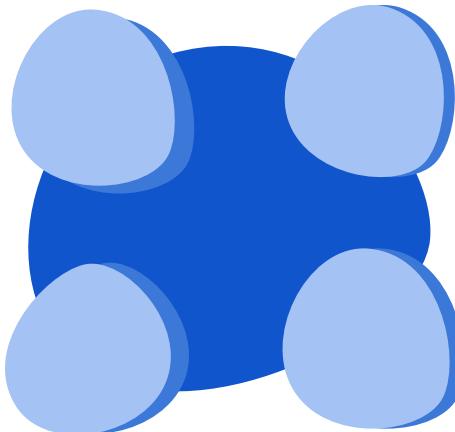
ArchUnit Ne Sağlar?

Tasarım Kuralları Oluşturma

Projenizin mimari gereksinimlerine uygun tasarım kuralları tanımlayabilirsiniz.

Sürekli Entegrasyon

Temelde test olduğu için sürekli entegrasyon süreçlerine kolaylıkla eklenir.



Otomatik Kontrol

ArchUnit, tanımladığınız tasarım kurallarına uymayan kod parçalarını tespit eder.

Dokümantasyon

Testlere eklenen açıklamalarla beraber yaşayan bir dokümantasyon oluşturur.



```
@ArchTest
static ArchRule test =
    classes()
        .that().resideInAPackage("..service..")
        .and().areAnnotatedWith(MyService.class)
        .should().haveSimpleNameStartingWith("Service");
```

ArchUnit Test Anatomisi

Sınıflar

Hangi sınıflara testin uygulanacağı tanımlanır

1

Birim

Kuralların hangi birim üzerinde çalışacağı tanımlanır

2

Kural

Birimler üzerinde hangi kuralın çalışacağı tanımlanır

3

```
@AnalyzeClasses(packages = "com.mycompany.myapp")
public class MyArchitectureTest {

    @ArchTest
    public static final ArchRule myRule =
        classes()
            .that().resideInAPackage("..service..")
            .should().onlyBeAccessed().byAnyPackage("..controller..", "..service..");
}
```

Neleri Kontrol Edebilirim?

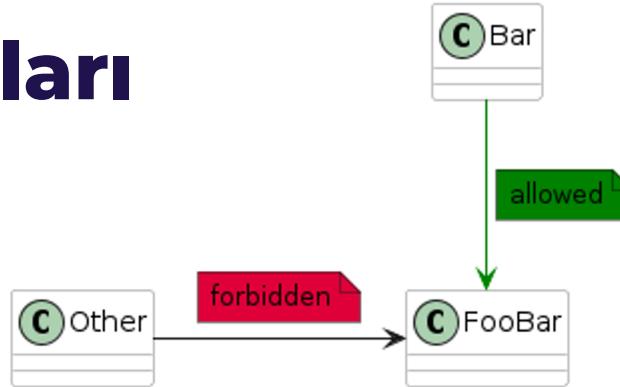
- Paketlerin Bağımlılıkları
- Sınıfların Bağımlılıkları
- İsimlendirme Kontrolleri
- Kalıtım Kontrolleri
- Anatasyon Kontrolleri
- Ön tanımlı Mimari Kontrolleri

Paket Bağımlılıkları



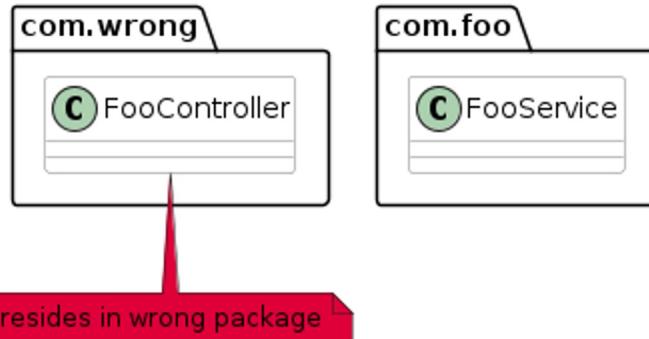
```
noClasses().that().resideInAPackage("..source..")  
    .should().dependOnClassesThat().resideInAPackage("..foo..")
```

Sınıf Bağımlılıkları



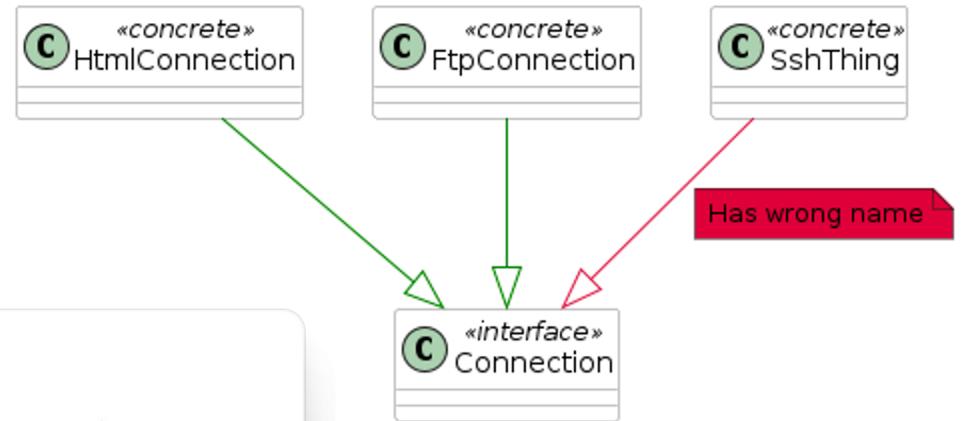
```
classes().that().haveNameMatching(".*Bar")
    .should().onlyHaveDependentClassesThat().haveSimpleName("Bar")
```

Sınıf Bağımlılıkları



```
classes().that().haveSimpleNameStartingWith("Foo")  
    .should().resideInAPackage("com.foo")
```

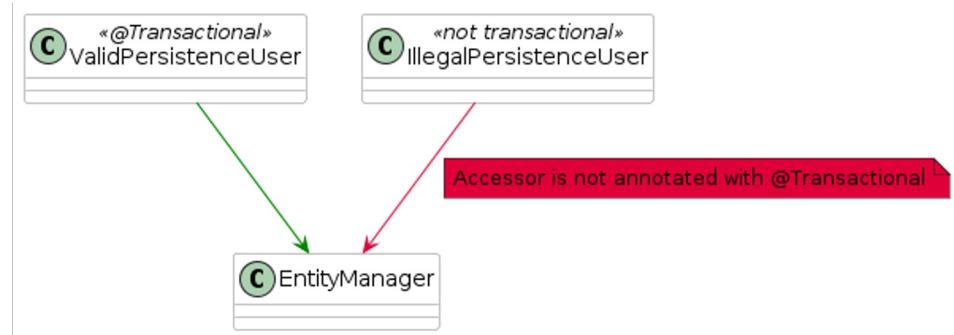
Kalıtım Kontrolleri



...

```
classes().that().implement(Connection.class)
    .should().haveSimpleNameEndingWith("Connection")
```

Anatasyon Kontrolleri



```
classes().that().areAssignableTo(EntityManager.class)  
.should().onlyHaveDependentClassesThat().areAnnotatedWith(Transactional.class)
```

Mimari Kontroller - Katmanlı

```
layeredArchitecture()  
    .consideringAllDependencies()  
    .layer("Controller").definedBy("..controller..")  
    .layer("Service").definedBy("..service..")  
    .layer("Persistence").definedBy("..persistence..")  
  
    .whereLayer("Controller").mayNotBeAccessedByAnyLayer()  
    .whereLayer("Service").mayOnlyBeAccessedByLayers("Controller")  
    .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Service")
```

Mimari Kontroller - Hexagonal

...

```
onionArchitecture()  
    .domainModels("com.myapp.domain.model..")  
    .domainServices("com.myapp.domain.service..")  
    .applicationServices("com.myapp.application..")  
    .adapter("cli", "com.myapp.adapter.cli..")  
    .adapter("persistence", "com.myapp.adapter.persistence..")  
    .adapter("rest", "com.myapp.adapter.rest..");
```

Yaşayan Dokümantasyon



```
@ArchTest
static ArchRule methods_in_the_controller_layer_should_return_API_response_wrappers =
    methods()
        .that().areDeclaredInClassesThat().resideInAPackage("..anticorruption..")
        .and().arePublic()
        .should().haveRawReturnType(WrappedResult.class)
        .because("we do not want to couple the client code directly to the return
                 types of the encapsulated module");
```

DEMO

Teşekkürler!