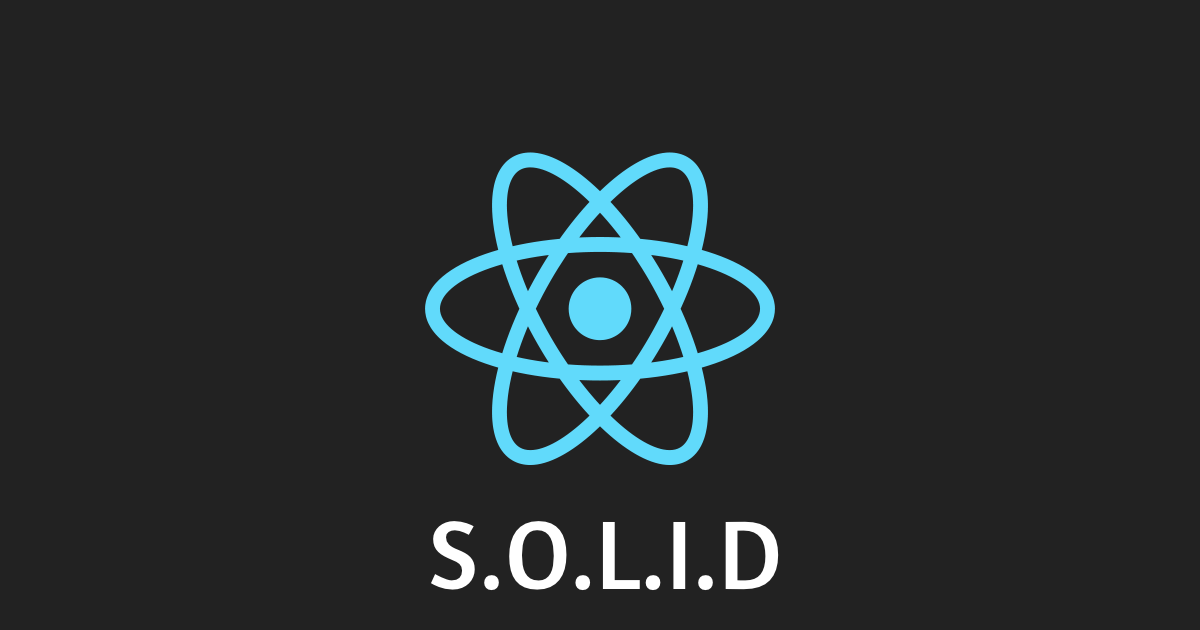
|Software Development is not Jenga Game|



**SOLİD PRENSİPLERİ**

**“Yazılımda sürdürebilirliği sağlayan kurallar bütünü”**

**Nedir S O L I D ?**

SOLİD, projemizin sürdürebilirliğinin olması,yeni teknolojilere ve eklentilere açık olması,yapılacak değişikliğe farklı yerleri etkilemeyip,geliştirmelere açık olmasıdır.

Robert Cecil Martin nam-ı diğer Uncle Bob’un “**Yazılımda Kötü Mimari Tasarımı Sorununa Çözüm**” olarak ortaya çıkan **Bağımlılık Yönetimi**dir. (Dependency Managament)

**Bağımlılık Yönetimi;**

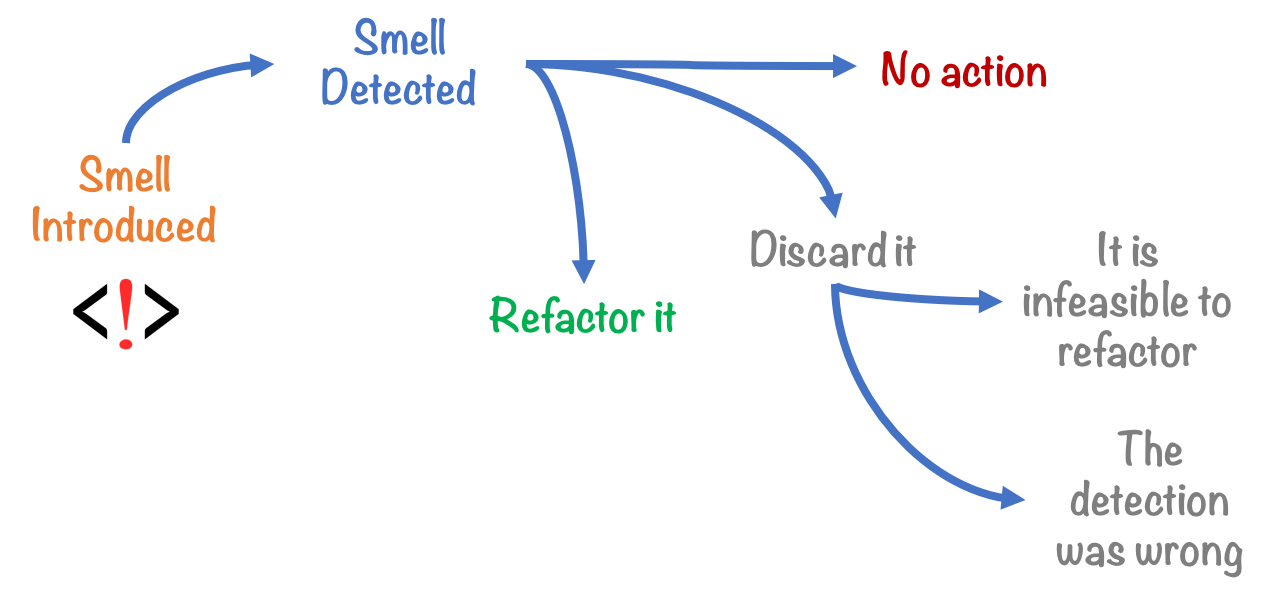
Kötü oluşturulmuş bir projede,**genişletilebilirliğin** neredeyse mümkün olmaması **sabit**(immobility:geliştirilen modüllerin tekrar kullanıma uygun olmaması) ve **kırılgan** (fragility: yapılacak olan bir değişikliğin,başka bölümleri etkilememesi) bir yapının olmasından dolayı projemizin sürdürülebilirliğinin olmaması durumudur.

Her bileşenin kendi sorumluluk alanına sahip olması,başka bir bileşen ile sıkı sıkıya bağlı olmaması bağımlılık yönetiminin en önemli özelliğidir.

Bağımlılıklarının azaltılması ve görevlerin teklileştirilmesi,yazılımın bakımını ve genişletilmesini kolaylaştırır.

-Solid prensiplerini açıklamadan önce değinmemiz gereken bir kaç kavram var.

**Refactoring**; yazılımı daha basit,daha anlaşılır,değiştirilmesi daha kolay bir hale getirmek amacıyla iç yapısında yapılan ve yazılımın dış davranışlarını etkilemeyen değişkenlerdir.



**Code Smell**

— What? How can code "smell"??  
— Well it doesn't have a nose... but it definitely can stink!

Yeniden yapılandırılmaya ihtiyaç duyulan kodlar için kodun kötü koktuğu söylenir. Kokular,koddaki temel tasarım ilkelerinin ihlali sonucu ortaya çıkar.Bir kaç ihlal durumu;

* Bir çok işten sorumlu üst sınıflar
* Koplaya yapıştır yöntemiyle oluşmuş kod tekrarları
* Iç içe geçmiş karmaşık (if-else,for,while)yapılar
* Bir gün lazım olur düşüncesiyle yazılmış kodlar
* Sınıf olmayı haketmeyen sınıflar

Bütün bunlara dikkat etmek her yazılımcının ana görevi.

Gelelim prensiplere!

**Single Responsibiliyt Principle**

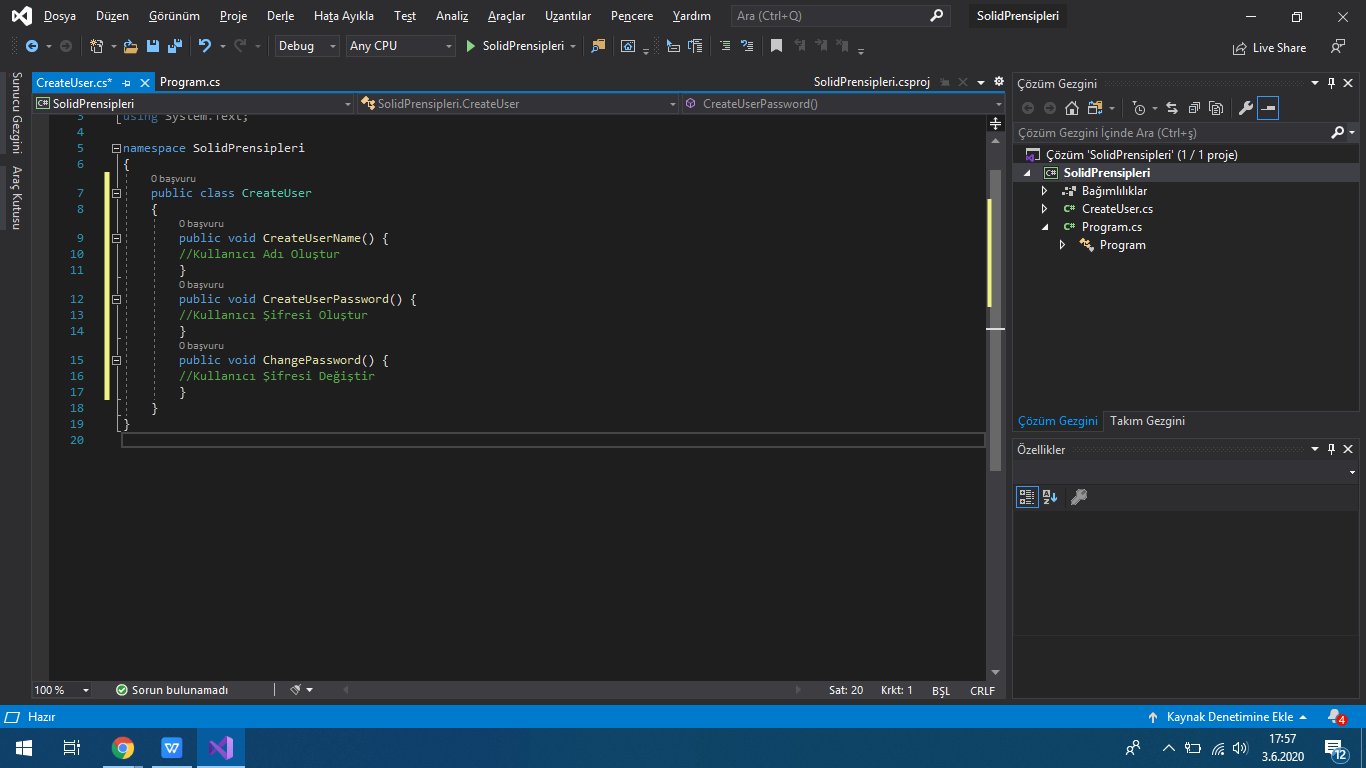
Bir class veya function’a sadece **tek bir sorumluluğu** yerine getirilmesine dayanmaktadır.Birden fazla sorumluluk tek bir class/function’a yüklendiği zaman kod büyür ve karmaşık bir yapı halini alır. Karmaşık ve aşırı bir kodun yönetimi zordur ve bu yüzden **kırılganlığı** ortaya çıkar. Aynı zamanda **esnekliği** bir o kadar da azdır.

Bu prensip göz ardı edildiğinde,kendisi ile ilgili sorumlulukları yerine getiremediği gibi,uyapılan herhangi bir değişiklikte başka bir yer etkilenebilir ve yeniden kullanılabilirliği düşürülmüş olur.

**Basit bir örnekle** açıklamak istersek alanında yapması gerekenden daha fazlası yapılması istediği taktirde bir noktada alanında da hatalara neden olması.

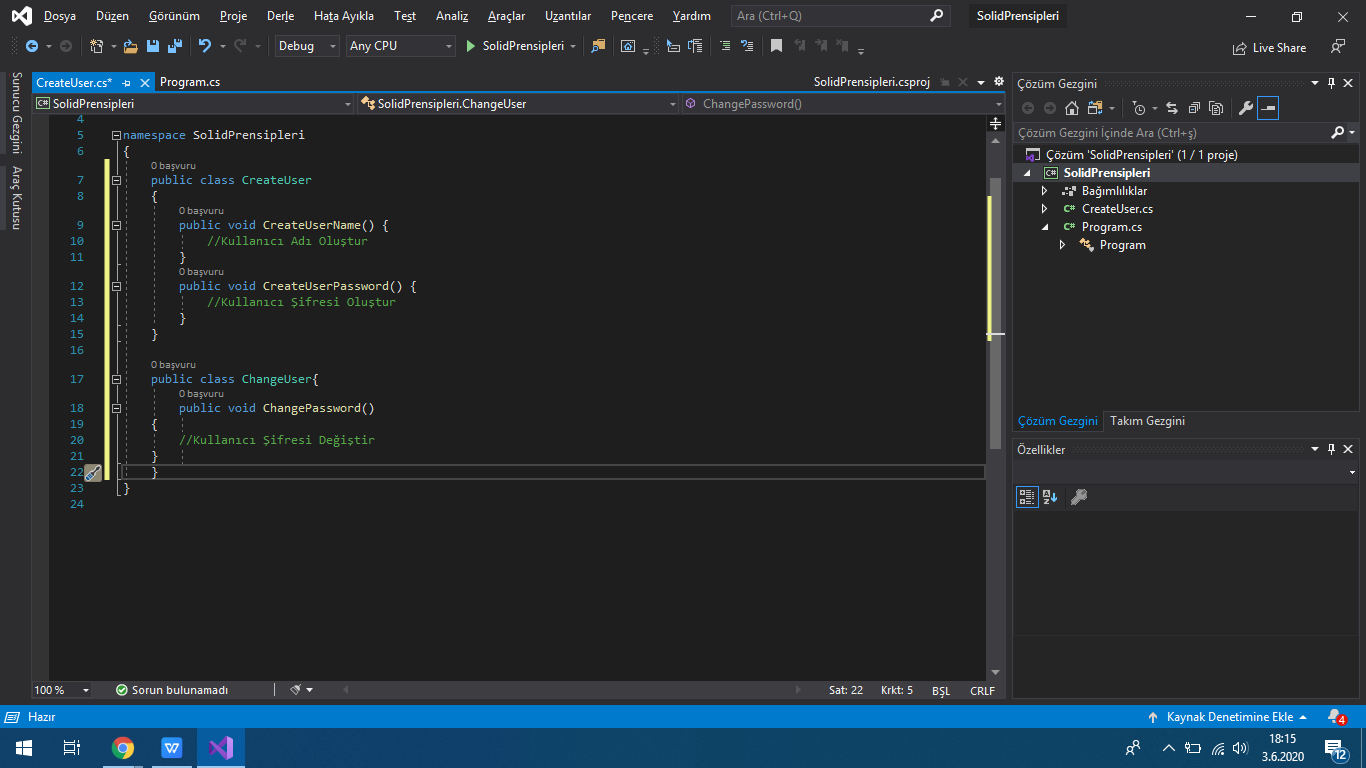


Kod Örneği ile



Kullanıcı oluşturmak üzere olan sınıfımızda Kullanıcı Adı Oluştur,Kullanıcı Şifresi oluştur ve Şifre Değiştir Metotları vardır.

Kodumuzu incelediğimizde kullanıcı oluşturma class’ının içerisinde bu class içerisinde farklı bir görevi olan şifre değiştirme metotu mevcuttur. Single Responsibility Prensibinde amacımız her class ya da function kendi görevini yerine getirmesiydi. Kodumuzun doğru hali aşağıdadır.



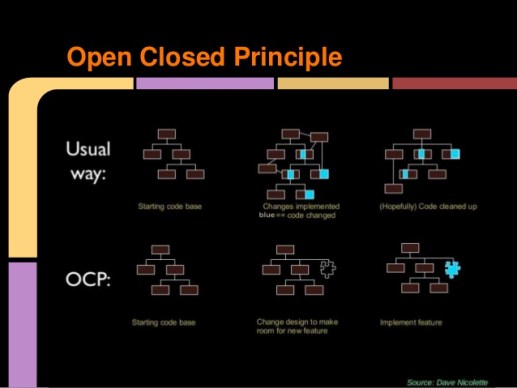
**Open/Closed Principle**

Projemizde kullandığımız varlıkların(class,function,methot) gelişime açık değişime kapalı olma prensibidir. Bir varlığın davranışını değiştirmeden onu genişletmeye dayalıdır.

Bu prensip;sürdürülebilir ve tekrar kullanılabilir yapıda kod yazmanın temelini oluşturur.

**Open** varlık için yeni davranışlar eklenilmesini sağlar. Gereksinimler değiştiğinde,yeni gereksinimlerin karşılanabilmesi için bir varlığa yeni veya farklı davranışlar eklenebilir olmasıdır.

**Closed** bir varlığın temel özelliklerinin değişime kapalı olmasıdır.



**Basit Bir Örnekle**

Herkesin kendine göre çalışma alanları olduğunu varsayalım. Herkes kendi alanında uzman bir şekilde çalışıyor. Günümüz şartları ve gelişim açısından her gün yeni bir bilgiye maruz kalıp bu yönde kendimizi geliştirmemiz gerekiyor. Eğer kendimizi geliştirmezsek rekabetin bir adım arasında kalabiliriz. Eğer kendimizi geliştirme yolunda alanımızı da değiştirirsek,farklı alanlara doğru yol alabiliriz. Gelişime açık olup, değişime kapalı olmalıyız.

****Liskov ‘s Substitution Principle****

**Kodlarımızda herhangi bir değişiklik yapmaya gerek duymadan alt sınıfları,türetildikleri sınıfların yerine kullanabilmeliyiz.**

**Türeyen sınıf (alt sınıflar) ana (üst) sınıfın tüm özelliklerini ve methotlarını aynı işlevi gösterecek şekilde kullanabilme ve kendine ait yeni özellikler barındırabilmelidir.**

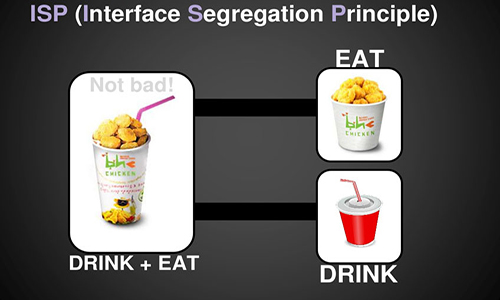


**Basit Bir Örnekle**

Eğitim-Öğretim yılları içerisinde belirli bir müfredat dahilinde Öğretmen-Öğrenci ilişkisiyle eğitim verilmektedir.Öğretmen’i üst sınıf olarak düşünürsek kendi bünyesinde barındırdığı bilgileri bir alt sınıf olan Öğrenci’ye aktarmakla görevlidir. Bir öğretmenin bir konu üzerinde ki bilgi birikimini öğrencisiyle paylaştığı düşünülürse,öğretmenin tüm özellikleri öğrenciye geçecektir.

**İnterface Segregation Principle**

Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturmayı tercih etmemizi açıklayan prensiptir.

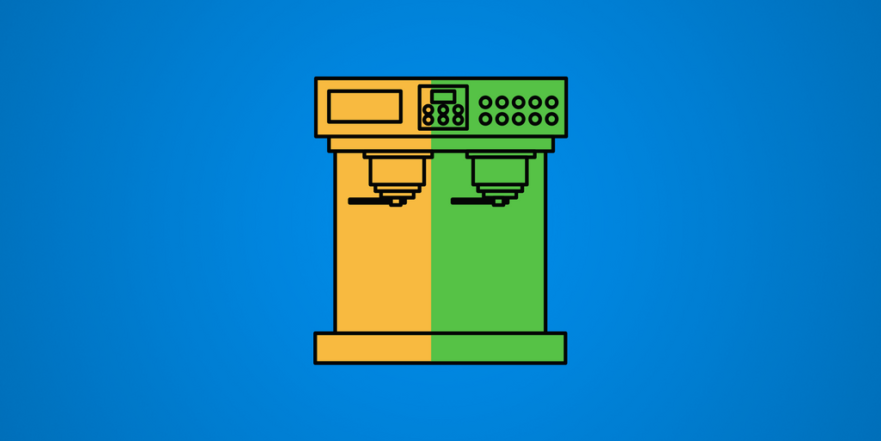


**Basit Bir Örnekle**

Bir proje ortaya çıkarmak için ekip oluşturulmuştur. Öncelikle projenin planlanmasıyla başlayıp ekip üyelerine bu şekilde yapılması gereken görevler verilir. Bir kişinin bütün projeyi yapmasını beklemek yerine,görev dağılımı ile iş bölümünü sağlayıp hem zamandan tasarruf edebilir hem projemizi daha kolay bir şekilde bitirebiliriz.

**Dependency Inversion Principle**

Bir sınıfın,metodun ya da özelliğin,onu kullanan diğer sınıflara karşı bağımlılığı en aza indirgenmelidir.Bir alt sınıfta yapılan değişiklik üst sınıfı etkilememelidir.



**Basit Bir Örnekle**

Buzdolabımızın kapağının bozulduğunu varsayalım. Teknik servis çağırdığımızda buzdolabı kapağımızın değişimini talep etmek istiyoruz fakat buzdolabı kapağının bu dolaba ait olduğunu ve yerini değiştirebileceğimiz başka kapak olmadığı cevabını alıyoruz. Bu yüzden kapağı değiştirmek isterken buzdolabını da değiştirmek zorunda kalıyoruz.

**Microsoft Build 2020 Yenilikleri**

**AZURE YENİLİKLERİ**

Microsoft, Azure ARC özellikli Kubernetes’in herkese açık önizlemeisini müşterilerin Kubernetes kümelerini Open Shıft’te dahil olmak üzere Azure Stack hublarında ölçeklendirebilmelerini,organize edebilmelerini ve yönetebilmelerini sağlıyor.

Mıcrosoft ve SUSE Linux Enterprise Server’ı Azure ARC özellikli bir sunucu olarak destekleyerek müşterilerin kolalaştırmasına olanak sağlıyor.

Yeni Azure Stack Hub güncellemeleri filo ve kaynak yönetimini basitleştirecek hale geldi.

Azure Stack Hub Filo Yönetimi, müşterilere tüm Azure Stack Hub dağıtımları için Azure’da tek bir görünüm ve yönetim sağlıyor.

ManageIQ bulut operatörlerini artık Azure Stack hubdaki kaynakların yönetmelerine ve Azure Stack Hub’ı yönetmek için Redhat teknik araçlarını kullanmalarına izin veriyor.

ManageIQ,Ibm ve Redhat tarafından desteklenen bir platform.

Azure Stack’te Aks Kaynak Sağlayıcısı,müşterilerin Azure Stack Hub’da Kubernetes kümelerini otomatik olarak oluşturmaları için sağlanan bir hizmettir.

Microsoft herhangi bir ölçek için Apılere sahip hızlı NoSQL veritabnı olan Azure Cosmos DB için eklentileri var.Bu eklentilrt müşterilere benzersiz ihtiyaçlarına uygun fiyatlandırma modelleri sağlamak,uygulama geliştirme döngüsünü kısaltmak ve iş yüklerini etkili bir şekilde yönetmek için tasarlanmıştır.

**.NET**

.Net 5 sürümü için mobil,web,masaüstü,yapay zeka,makine öğrenimi,büyük veri ve loT ile proje geliştirebiliyoruz. Daha az bellek kullanan,daha küçük,daha hızlı çeşitli bulut ve web uygulamalarına sahip olacaktır.   
 Geliştiricilerin JAvaScript yerine c# ile etkileşimli web UI’leri oluşturmasına izin veren ASPçNET Blazor,artık WebAssembly’ı de destekliyor.

**Takip Edilmesi Gereken Kişiler**

**Fatih Kadir Akın**

Toolio şirketinde Senior Software

JavaScript hakkında kitabı var

[github.com/f](https://github.com/f)

**Engin Demiroğ**

Udemy’de en çok oy alan eğitmen

Java,Python,C#,Html,Flutter alanında uzman

github.com/engindemirog

**HACKATON**

**Flutter Hackathon**

27-28th June 2020

**United Game Jam**

#evdekal Oyun Geliştir, 5-7 Haziran

Game Jam’ler oyun geliştirme amaçlı yapılan 48 saat süren bir oyun geliştirme etkinliğidir. United Game Jam’de tüm oyun geliştiriciler online olarak bir araya gelir,ekiptler kurar ve oyun geliştirirler.