KISS NEDİR?

Keep it Simple Stupid. Yapılan işi basit ve çözüm odaklı bir yol ile sonuçlandırmak. Yazılımcı mevcut bir sorunu çözmek için komplike yollar seçmekten kaçınmalı işin özünü kaçırmadan problemi çözmeye ve işi bitirmeye odaklanmalıdır. Kurulan yapı elden geldikçe esnek tutulmalıdır.

KISS Prensibi en basit çözüm ihtiyacın olan çözümdür mottasının kalıplaşmış halidir.

1960'lı yıllarda Amerikan donanmasında ilk olarak ifade edilen bu prensip günümüzde en önemli programlama prensiplerinden biridir. Bir problemi çözerken olabilecek en basit ve yalın çözümü seçmeyi önerir. Hatta o kadar basit olmalıdır ki ilk bakışta "Bunu bir aptal bile yapar ve anlar" demeliyiz.

KISS Prensibi , karmaşık çözümlerin daha "zekice" çözümler olduğu fikrini reddeder. Çoğu mühendis/yazılımcı karmaşık çözümler bularak, karmaşık yapılar inşa ederek "zekice" işler yaptığını düşünür. Dışarıdan bakan bir insan ne kadar zor anlar ise o kadar artı değer ürettiğini sanır. Ancak zor olan basitleştirmektir.

YAGNI NEDİR?

You Aren't Gonna Need It.

You Ain't Gonna Need It.

YAGNI, Extreme programlamanın prensiplerinden biridir. Gelecekte lazım olacak düşüncesi ile bir takım özelliklerin geliştirilmemesi gerektiğini savunur. Şimdi bir örnek üzerinden bu prensibi anlamaya çalışalım.

Diyelim ki projemize bir "oturum açma" özelliği ekliyoruz. O anda bizden istenen şey kullanıcıların kullanıcı adı ve şifreleri ile giriş yapabilmelerini sağlayan bir özellik. Fakat biz "ilerde lazım olur" düşüncesi ile facebook ile de oturum açılmasını sağlayabilecek şekilde geliştiriyoruz arka taraftaki methodlarımızı. İşte tam bu noktada YAGNI diyor ki "YAPMA". Peki neden?

Yazılımcıların o an kullanmayacakları özellikleri geliştirmesinin en yaygın sebebi o özelliği o an geliştirmenin daha ucuz olacağını düşünmeleridir. Örneğimizden gidecek olursak geliştirici şöyle düşünür: "Şimdiden ben arka tarafta facebook entegrasyonunu yapayım. Yarın bir gün zaten istenecek bu benden. O zaman bu yazdıklarımı değiştirmem gerekmez." Fakat bu özellik yazıldığı an itibari ile "varsayımsal özellik" kategorisindedir ve hiç bir zaman sizden talep edilmeyecek olabilir. Bu noktada bazıları bunun planlama olduğunu ve iyi analizler sonucu bu tip geliştirmeler yapılabileceğini iddia edebilir. Ne var ki gerçek hayat tecrübeleri bunun çok uzağındadır ve geliştiriciler yanlış öngördükleri varsayımsal özelliklerin geliştirme maliyetine katlanmak zorunda kalır. Bu özellik için harcanan analiz, programlama ve test etme süreleri boşa harcanmış olur.

YAGNI prensibinin ne zaman geçerli olduğu konusu biraz göreceli fakat kısaca şöyle denebilir: Varsaydığımız özelliğin büyüklüğü arttıkça YAGNI prensibinin önemi artar. 1-2 saatlik bir iş yapılacaksa ve bu ilerde bir kaç günlük bir geliştirmeyi karşılayacaksa bunda çok sakınca olmayabilir fakat bu süre uzadıkça riskleriniz artar.

Ayrıca YAGNI kodun kolay değiştirilebilecek şekilde yazılmaması anlamına gelmez. Aksine YAGNI bir Extreme Programming prensibidir ve XP'nin en önemli pratiklerinden biri Refactoring'dir. XP kodun kolay değiştirilebilir olmasını öğütler. Bol bol refactoring yaparak kodumuzu temiz tuttuğumuzda, methodlarımızı parametrik yazdığımızda zaten özellikleri sonradan eklemek bize çok maliyet getirmez. Bu anlamda YAGNI esnek kod yazma ile çatışmaz, uyum içerisindedir.

SOLİD NEDİR?

SOLID yazılım prensipleri; geliştirilen yazılımın esnek, yeniden kullanılabilir, sürdürülebilir ve anlaşılır olmasını sağlayan, kod tekrarını önleyen ve Robert C. Martin tarafından öne sürülen prensipler bütünüdür.

S — Single-responsibility principle

Bir sınıf (nesne) yalnızca bir amaç uğruna değiştirilebilir, o da o sınıfa yüklenen sorumluluktur, yani bir sınıfın(fonksiyona da indirgenebilir) yapması gereken yalnızca bir işi olması gerekir.

O — Open-closed principle

Bir sınıf ya da fonksiyon halihazırda var olan özellikleri korumalı ve değişikliğe izin vermemelidir. Yani davranışını değiştirmiyor olmalı ve yeni özellikler kazanabiliyor olmalıdır.

L — Liskov substitution principle

Kodlarımızda herhangi bir değişiklik yapmaya gerek duymadan alt sınıfları, türedikleri(üst) sınıfların yerine kullanabilmeliyiz.

I — Interface segregation principle

Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturmalıyız.

D — Dependency Inversion Principle

Sınıflar arası bağımlılıklar olabildiğince az olmalıdır özellikle üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır.

CLEAN CODE NEDİR?

Kodun temiz olması, kodu yazan geliştirici dışında ekiptekilerin kodu kolay şekilde anlayabilmesi ve geliştirme yapabilmesidir.

- readability, (Basitçe okunup anlaşılıyor ise)
- changeability, (Basitçe değiştirilebiliyor ise)
- extensibility (Basitçe genişletilebiliyor ise)
- maintainability. (Basitçe bakım yapılabiliyor ise)

Tasarım Kuralları

- Konfigüre edilen veriyi kodun içerisinde derinlerinde değilde rahat erişilebilen değiştirilebilen kısımda bulundur.
- if/else veya switch/case koşulları yazmak yerine polymorphism tercih et.
- multi-threading kodları ayrıştır.
- Her kod yapısını konfigürasyonlu ve dinamik hale getirmekten kaçının.
- Dependency Injection (Bağımlılık Enjekte Etmeyi) kullanın.
- Bir sınıf doğrudan sadece bağımlılıklarını bilmelidir yasasını takip et.

SPAGETTİ CODE NEDİR?

Bilgisayar kodlamasında, bir kodun okunabilirliğinin düşük olması, yani kod takibinin zor olması durumunda, koda verilen isimdir.

Genellikle yapısal programlama dillerinde (structured programming languages) fonksiyonların bulunması ile birlikte GOTO veya JMP gibi, kodun içerisinde bir yerden başka bir yere atlayan komutların kaldırılması mümkündür.

Bu tip komutların kullanılması durumunda, kodun hem diğer programcılar tarafından okunabilirliği düşer, hem de kodun karmaşıklığı kestirilemez bir hâle gelebilir.

Spagetti kodu engellemek için 3 temel adım kullanılır:

1. Kodda bulunan ve koşula bağlanması istenen satırlar için if bloğu kullanılır
2. Kodda bulunan ve tekrarlanması istenen satırlar için döngü kullanılır
3. Kodda bulunan ve parametrize edilmesi istenen satırlar için (aynı satırların farklı değerlerle çalışması isteniyorsa) fonksiyon kullanılır.

Yukarıdaki bu üç adım tamamlandıktan sonra kodda herhangi bir GOTO satırı kalmamalıdır.

DUMMY CODE NEDİR?

Kukla Değişken. Sahte Değişken.

Çoğunlukla regresyon analizlerinde kullanılan bir değişkendir. Kukla, dummy veya sahte olarak adlandırılmasının nedeni; aslında veriler içerisinde yer almaması, araştırmacı tarafından, var olan bir değişkene göre sonradan oluşturulmasıdır. Sahte değişken oluşturulmasının nedeni, regresyon analizlerinde en az iki adet sürekli değişkene ihtiyaç duyulmasıdır. Örneğin elimizde iki farklı puan olur.

Bunlardan birisi arttığında diğeri de artıyor mu? Ya da azalıyor mu? Birinin, diğeri üzerinde anlamlı bir etkisi var mı? Bunu tespit etmeye çalışırız.

Bir kategorik değişkenin sürekli değişken üzerindeki etkisini tespit etmek istediğimizde burada bir işlem yapmamız gerekmektedir. Bu işlem de kukla değişken oluşturma işlemidir.

1. Öncelikle kategorilerimizi birer ayrı değişken olarak oluşturuyoruz. Kategorilerin adlarını yeni değişkenlerin adları olarak belirliyoruz ve kaç kategorimiz varsa o kadar değişkenimiz oluyor.

2. Her bir katılımcıyı, oluşturduğumuz yeni değişkenlerde bulunuyorsa 1 bulunmuyorsa 0 olarak kodluyoruz. Örneğin burada 1. katılımcı yardımcı doçent, o zaman yrd. doçent adındaki değişkene 1 yazıyoruz. Diğer değişkenlere 0 yazıyoruz.

Böylece kukla değişkenlerimiz oluşturulmuş oluyor. Bundan sonrasında artık regresyon analizi yapılabiliriz.

FİFO NEDİR?

FIFO, "First-In, First-Out" (İlk Giren, İlk Çıkar) anlamına gelen bir kısaltmadır ve genellikle envanter yönetimi ve veri yapısı gibi alanlarda kullanılan bir kavramdır.

FIFO, bir sıra veya kuyruk yapısını ifade eder. Bu yapıda, ilk giren öğe veya veri, ilk çıkan öğe veya veri olur. Yani, en eski öğe veya veri ilk olarak ele alınır ve işleme tabi tutulur. Bu şekilde, veriler veya öğeler sırayla işlenir ve kuyruğun sonunda bekleyen öğeler işleme alınmadan önce öncekilerin tamamlanmasını bekler.

FIFO prensibi veri yapıları içinde de kullanılır. Örneğin, bir kuyruk (queue) veri yapısı, FIFO prensibini temel alır. Yeni bir öğe kuyruğun sonuna eklenirken, kuyruğun başındaki öğe çıkarılır. Bu şekilde, en eski eklenen öğe önce çıkarılır ve sıra ile işlenir.

LİFO NEDİR?

LIFO, "Last-In, First-Out" (Son Giren, İlk Çıkar) anlamına gelen bir kavramdır. LIFO, bir veri yapısı veya işlem sırası düzeni olarak kullanılır.

LIFO, veri yapısı olarak bir yığın (stack) olarak düşünülebilir. Yığına yeni bir öğe eklendiğinde, en son eklenen öğe en üstteki konuma yerleştirilir. Aynı şekilde, bir öğe çıkarıldığında (kaldırıldığında), en üstteki öğe çıkarılır ve alttaki öğelerin erişimi mümkün hale gelir. Yani, en son eklenen öğe veya veri, ilk olarak çıkarılır.

LIFO prensibi genellikle veri işleme veya depolama yöntemlerinde kullanılır. Örneğin, bir çağrı yapma sırasında LIFO mantığı kullanılırsa, en son arayan kişiye öncelik verilir ve en son arayan kişiyle önce iletişim kurulur. Aynı şekilde, LIFO mantığına dayanan veri yapısı olan yığın, bir hesap makinesinde geçerli olan hesaplama sırasını takip etmek için kullanılabilir. En son giren sayı veya işlem, ilk olarak kullanılır.