CLEAN CODE

Clean Code’a Neden İhtiyaç Duyarız?

Yazılan kod zaman geçtikçe karmaşıklaşabilir. Bu karmaşıklık kodu erken bitirmek için acele etmek, kod için kullanılan isimlere özen göstermemek, fonksiyonları uzun bir şekilde yazmak, fazla yorum satırı kullanmak ve proje planlamasını yanlış yapmak sonucunda ortaya çıkar. Bu karmaşıklık yazılan uygulamanın gereksinimleri karşılayamaması, son teslim tarihlerinin uzaması ve uygulamada çökmelerin artmasına sebep olur. Ve uygulama artık kullanılamaz bir hale gelir. Clean code kodun bir düz yazı gibi açık, basit, anlaşılabilir olmasını sağlar. Bu sayede kod sadece yazan kişi tarafından değil herkes tarafından anlaşılabilir hale gelir. Bununla birlikte kod karmaşıklıktan kurtulduğu için koddaki hatalar tespit edilip düzeltilebilir. Bu yüzden karmaşıklığı gidermek için de clean code’a başvururuz.

Clean Code Nedir?

Clean code (temiz kod)la ilgili yazılımcıların birden fazla tanımı vardır. Bunlardan birkaçı şunlardır;

**Bjarne Stroustrup(C++ ‘ın mucidi) :**”Kodumun şık ve temiz olmasını seviyorum. Kodda mantık, hataların saklanmasınızorlayacak kadar düz; bağımlılıklar (dependency) bakımı kolaylaştıracak kadar minimal olmalı. Tüm istisnai durumlar (exceptions) ele alınmalı, performans optimale yakın olmalı.”

**Dave Thomas (OTI Labs’ın kurucusu):** Temiz kod, onu geliştiren yazılımcı dışında başka geliştiriciler tarafından da okunabilir ve iyileştirilebilir. Birim ve kabul testleri vardır. Anlamlı isimlendirmeleri vardır. Bir şeyin yapılması için tek bir yol vardır. Çok az bağlılığı vardır ve temiz bir Uygulama Programlama Arayüzü(API) sağlar.

**Grady Booch (Uygulamalarla Nesne Yönelimli Analiz ve Tasarım kitabının yazarı):** Temiz kod basit ve açıktır. Temiz kod, iyi yazılmış bir düzyazı gibidir. Temiz kod, asla tasarımcının niyetini gizlemez, daha çok berrak soyutlamalarla ve düz kontrol satırlarıyla doludur.

**Ron Jeffries (XP metodolojisinin 3 kurucusundan biri):** Temiz kod gereksinimleri karşılar (testleri çalıştırır). İçine koyduğumuz tüm tasarım fikirlerini ifade eder. Yineleme içermez. Bileşen sayısını en aza indirir.

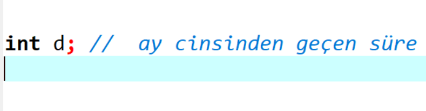
Kısacası temiz kod hataların kolayca farkedilebildiği , kodun başkaları tarafından da okunabilip düzeltilebildiği , gereksinimleri karşılayan, bileşen sayısını en aza indiren, kodun bakımını kolaylaştıran, kodun performansını arttıran ve kodu düz bir yazı gibi okunabilir hale getiren kurallardan oluşur.

Temiz kodda kullanılan kurallar: İsimlendirmeler, Koşullar, Döngüler, Yorum satırları, Fonksiyonlar, Hata Yönetimi, Nesne Yönelimli Programlama, Tasarım Prensipleri, SOLID, DRY, Tasarım Kalıpları.

**1.İsimlendirmeler:** . Kod yazarken değişkenlere, fonksiyonlara, argümanlara, sınıflara ve paketlere isim veririz. Bu yüzden isimlendirmenin iyi yapılması temiz kod için gereklidir. Peki isimlendirme nasıl iyi yapılır? Açık ve net isimlendirmeler yapılmalıdır. Genel kelimelerle isimlendirme yapmaktan kaçınmalıdır. Telaffuz edilebilir özel kelimelerle isimlendirme yapılmalıdır. Son ek veya ön eklerle isimlendirmeye ekstra bilgi katılabilir.

İsimlendirmelere birkaç örnek verecek olursak;

**Örnek 1:**



d harfi burada hiçbir şey çağrıştırmıyor. Oysaki biz ay cinsinden geçen süreyi belirtiyoruz. Burada d harfini kullanmak yerine daha anlamlı alternatifler kullanabiliriz.

Daha doğru isimlendirmeler:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Örnek 2:**

metin, beyaz tahta içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kodla ne bulmak isteniyor? Bir ne on ne? Neden girilen sayının modu alınıyor?

Daha iyi isimlendirilmiş kod bloğu:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Örnek 3:**

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu koda bakacak olursak List ne içeriyor? List’in ilk elemanının önemi nedir? 4 değerinin önemi nedir? Geri döndürülen list1 listesini nasıl kullanmalı? Bu soruların cevabını ancak kodu yazan kişi verebilir. Eğer kod şu şekilde olsaydı:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kodun basitliği değişmedi. Hala aynı sayıda işlemci ve sabit içeriyor. Fakat kod daha yalın bir hale geldi. Yapılan bu basit değişikliklerle kodda neler olduğu daha anlaşılabilir.

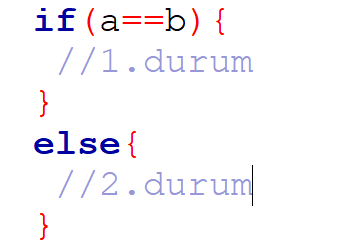
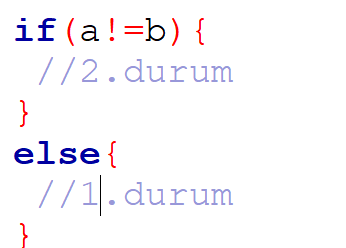
Aynı kapsamda kullanılacak iki objeye aynı ismi verilemeyeceğinden ikisinden birini değiştirme yolu kullanılır. Sayı eklemek yeterli gelmediğinde ikisinden birinin bir harfi eksilterek yapılır. Ancak ,isimler farklıysa anlamları da farklı olmalıdır.

a1, a2 gibi isimlendirmelerin anlamı yoktur. Herhangi bir amaç belirtmezler. Bu tarz isimlendirmeler kod yazan kişinin bu isimlendirmeleri ne için kullandığı hakkında fikir vermezler. metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada a1 ve a2 yerine koda uygun isimler verilebilir. metin içeren bir resim

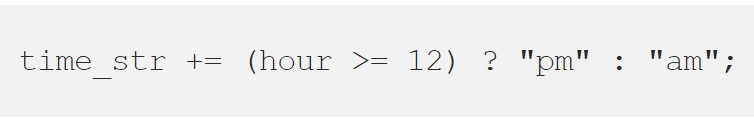
Açıklama otomatik olarak oluşturuldu Kod yazarken aranabilir isimler kullanmak kolaylık sağlar. Tek harfli isimler ya da sayı sabitleri arama yaparken sıkıntı yaratabilir. Örneğin **MAX\_CLASSES\_PER\_STUDENT** sabiti rahatlıkla bulunabilir. Ancak 5 gibi bir sayıyı bulmak zor olacaktır. Benzer şekilde e veya a harfi de yaygın kullanıldığı için her yerde karşımıza çıkabilir.  
**2-Koşullar:** Kodda hiç koşul veya kontrol akışı olmasaydı, okuması çok basit olurdu. Kodlardaki dallanmalar ve zıplamalar kodu karmaşık hale getirir. Bu yüzden bütün koşulları, döngüleri ve kontrol akışları en doğal şekilde yapılmalıdır ki karmaşıklık olmasın. **Argüman sırası:** Kodun okunabilirliği açısından argüman sırasına dikkat edilmelidir. Bir örnekle bunu pekiştirmek istersek: if( yas>=10) ve if( 10<=yas) şu iki örneğe bakarsak ilkinde eğer yaşınız 10 veya daha büyükse anlamı verir. İkincisi ise eğer 10 yaşınızdan küçük veya eşitse anlamı verir. Dildeki doğallık kodda da kullanılır. Böylece kod, okuyan için basitleşir. **İf/else blokları:**

Genel olarak ilk kod yazma şeklinin seçilmesinin şöyle iyi sebepleri olabilir; Negatif durum yerine pozitif durumla baş etmeyi tercih edersiniz. if(!debug) yerine if (debug) yazarsınız. Daha basit durumla ilk olarak başa çıkmayı tercih edersiniz. Daha ilginç veya şüpheli durumla ilk başta uğraşmayı tercih edersiniz. Diğer tarafta, bazı durumlarda negatif durum daha basit ve tehlikeli olabilir. Bu yüzden ilk olarak baş etmek gerekir.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Üçlü (Ternary) Operatörü**: Üçlü operatörü(<koşul> ? a : b), if (<koşul>) { a } else { b } gösteriminin kısa şeklidir. Okunabilirlik olarak ise tam tersi etki yapabilir. Şöyle bir örnekte üçlü operatörü daha okunabilirdir.

Eğer aşağıdaki gibi yazarsak kodu gereksiz uzatmış oluruz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu Bazı durumlarda üçlü operatörü kullanmamak daha iyi olabilir. Anlaşılmaz bir üçlü operatörü kullanmak yerine kodu normal şeklinde yazabiliriz. Temiz kodda amacımız kod satırını en aza indirmek yerine, daha iyi bir metrik olarak, insanların kodu anlama süresini en aza indirmek olmalıdır. Varsatılan olarak if/else blokları kullanılmalıdır. Sadece basitleştirilebilir kodlamalarda üçlü operatörü kullanılmalıdır.

**3-Döngüler:** Birçok dilde do-while döngüsü vardır. Do-while döngülerinde koşul sağlanmadığı durumlarda bile kod çalıştırılır. Bu gereksizdir. Bunun yerine do-while döngülerini while döngüleri şeklinde yazılabilir.

C++’ın mucidi Bjarne Stroustrup ,bu konu hakkında şöyle der: *“Deneyimime göre, do-ifadeleri hatalara ve kafa karışıklığına yol açar. Yukarıda görebileceğim şartları tercih ederim. Sonuç olarak, do-ifadeleri kullanmaktan kaçınırım.”*

**4-Fonksiyonlar:**Fonksiyonlar bir kod içindeki organizasyonun ilk satırlarıdır. Fonksiyonlar kısa olmalıdır. Eğer bir fonksiyon sadece ismindeki işlevi gerçekleştiriyorsa, o fonksiyon bir şey yapıyordur. Fonksiyonların bir amacı vardır ve onu yapmalılardır. Sadece bu işlevi ve çok iyi yapmalıdır. **Bir metot yada fonksiyon ne kadar uzun olmalıdır?** Her satır 150 karakterden az ve her fonksiyon 20 satırdan az olmalıdır. **Fonksiyon İsimlendirme:** Fonksiyon isimlendirirken kısa ve anlaşılmayan veya karışık isimler yerine uzun ve tanımlayıcı isimler kullanılmalıdır. İsim seçerken zaman harcamaktan korkulmamalıdır. Açıklayıcı isimler kullanmak zihinde tasarlanan düşünceleri ortaya çıkarır ve geliştirilmesine yardımcı olur. Fonksiyon isimleri içerdiği bilgilerle tutarlı olmalıdır. İçerisinde kullanılan değişkenler fonksiyonu çağrıştırmalıdır.

**5-Yorum Satırları:**Yorum satırları kodda açıklanamayan yerler için kullanılır. Yorum satırları kodu yazan kişinin kendini kodla ifade edememesinden dolayı ortaya çıkabilir. Bu nedenle yorum satırları koddaki kusur olarak sayılabilir. Kod yazarken eğer yorum satırı kullanacaksak öncesinde kullanmadan da kodla ifade edilip edilemeyeceğine bakmak gerekir. Koddan çıkartılabilecek gerçekler hakkında yorum yapılmamalıdır. Kötü koda yorum yazılmamalıdır. Mümkünse kodu en baştan yazmak daha iyidir. Okunabilir kod yazılırsa fazladan yorum satırlarına gerek kalmaz. metin içeren bir resim

Açıklama otomatik olarak oluşturuldu **Yasal Yorumlar:** Bazen kurumsal kodlama standartları yasal sebeplerden dolayı kodun kesin olmasını gerektirir. Örneğin telif hakkı(copyright) ve yazarlık(authorship) durumları gereklidir ve her kaynak dosyanın başına böyle bir yorum satırı koymak mantıklıdır. metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturulduSon olarak yukarıdaki örneği satır satır incelersek;

1.satırdaki yorum satırı gereksizdir çünkü kodda sayının okunduğu bellidir.

2.satırda digersayi isimlendirmesi herhangi bir şey çağrıştırmamaktadır. Bu da kafa karıştırır.

4.satırdaki yorum satırı gereksizdir. Kod açıktır.

6.satırda tutulan sayaca bir harf vermek kodda arama yapmayı güçleştirir. Yorum satırına sayaç tutulduğunu yazmak yerine j yerine sayac yazılmalıdır. Böylece kodda yorum satırına gerek kalmaz çünkü sayaç tutulduğu belli olur.

10.satırdaki yorum satırı gereksizdir. Kodda sayacın arttırıldığı görülebilir.

Yukarıdaki hataların giderildiği örnek aşağıdaki gibi olur:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Koddaki hatalar giderildiğinde kodda istenen şeyin okunan sayının faktöriyeli olduğu görülür.

Yazılımcıların temiz kod yazmakla ilgili birkaç tavsiye var bize;

“Hepimiz zaman zaman geri dönüp kodumuzu temize çekeceğimizi söylemişizdir.

Ancak o zamanlar LeBlanc’ın şu kuralını bilmiyorduk: “Sonra asla demektir (Later equals never).”

If you write bad code long enough to get an actual product out, you will deliver that product more slowly, and with more bugs, than if you had gone clean. Gerçek bir ürünü çıkarmaya yetecek kadar uzun süre kötü kod yazarsanız, o ürünü temiz gitmiş olmanızdan daha yavaş ve daha fazla hata ile teslim edersiniz. -Ron Jeffries

First, solve the problem. Then write the code. Önce sorunu çöz. Sonra kodu yaz. -John Johnson

One of my most productive days was throwing away 1000 line of code. En verimli günlerimden biri 1000 satırlık kodu çöpe atmaktı. -Ken Thompson

MEDYA BAĞLANTILARI:

KAYNAKÇA;

<https://ronjeffries.com/xprog/articles/too-much-of-a-good-thing/>

<https://medium.com/@busrauzun/clean-code-kitabindan-notlar-1-temiz-kod-derken-44e6f7a27eb0>

<https://medium.com/architectural-patterns/clean-code-temiz-kod-nedir-754fc7ccd5b8>

<https://medium.com/bili%C5%9Fim-hareketi/okunabilir-kod-sanat%C4%B1-2-2cb4cb9f5484>

<https://medium.com/@busrauzun/https-medium-com-busrauzun-clean-code-kitabindan-notlar-2-anlamli-isimlendirmeler-de309168e13e>

<https://medium.com/@busrauzun/https-medium-com-busrauzun-clean-codedan-notlar-bolum-3-fonksiyonlar-927b69bd5a47>

Deniz Kılınç Yazılım Mühendisliğinin Temelleri Ders Notları