# Full-Stack Automation Engineer

QA

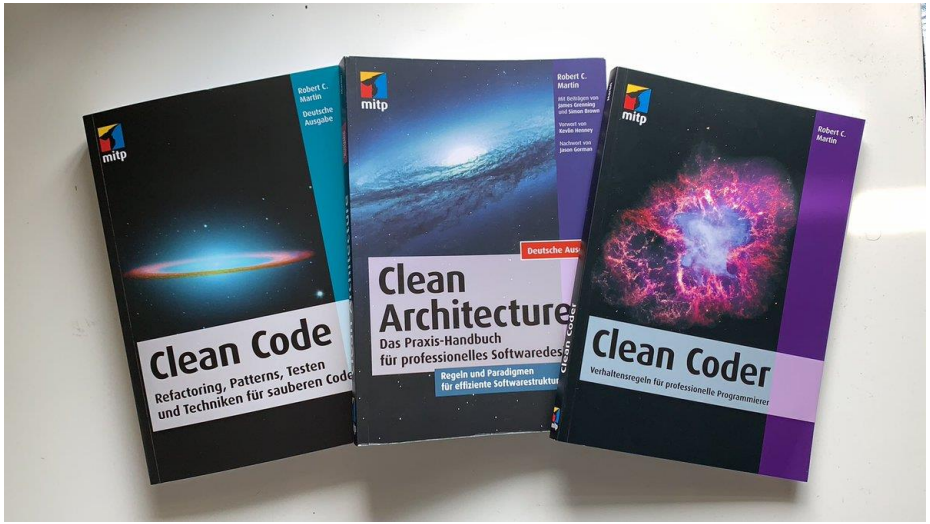
# Temiz Kod İçin 15 Önemli Öneri

- Karşınıza karmaşık, düzensiz ve not içermeyen kodlar çıktığında neler hissettiğinizi çok iyi anlıyorum. Bunu önlemek için "temiz kod" önerilerini ve ipuçlarını bu yazıda bulacaksınız.

- Günümüzde sıfırdan başlanarak ticari bir yazılım geliştirilmesi, özellik, arayüz ve protokol çeşitliliğinin getirdiği kompleksite artışı ve pazara çıkış sürelerinin giderek kısalması nedeniyle neredeyse imkansızdır. Bu yüzden **mevcut kod parçalarının** doğrudan ya da değiştirilerek kullanılması bir zorunluluk haline gelmiştir. Yeniden kullanılacak kod parçaları kurum içinde üretilmiş olabileceği gibi, açık kaynak kodları ya da satın alınmış kodlar olabilir. Bu kod parçalarının **verimli ve güvenli** olarak kullanılabilmeleri için kolay anlaşılabilir ve değiştirilebilir olmaları çok önemlidir. Kurum içinde üretilmiş de olsa, kodun ilk geliştiricisinin uzun zaman önce ayrılmış olması, bir üst düzey yöneticinin kodu olması durumunda, sıklıkla “koddaki şu açıklamada (// yyyy) ne demek istemiştiniz” gibi sorular sorma şansının olmaması, ya da kendi kodu da olsa altı ay sonra detayları hatırlama zorluğu, kodla yeni sahibini başbaşa bırakacaktır.

Tüm bu nedenlerle yazılımcıların aşağıdaki konuları anlaması ve **dikkat etmesi** oldukça önemlidir;

- İyi kod ve kötü kod arasındaki fark,
- İyi kod yazmak ve kötü kodu iyi koda dönüştürmek,
- İyi isimler, iyi fonksiyonlar, iyi nesneler (object) ve iyi sınıflar (class) oluşturmak,
- Maksimum okunabilirlik için kodu formatlamak,
- Kod yapısını bulanıklaştırmadan hata işlemek (error handling),
- Birim test yapmak ve test odaklı geliştirme (TDD).



# Temiz Kod

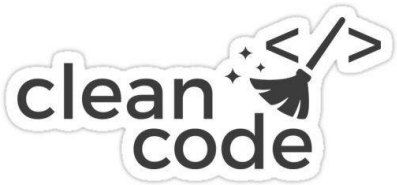
## 1) Niyeti ortaya koyan isimler kullanın

Değişken, fonksiyon ya da sınıf ismi, neden var olduğunu, ne yaptığını, ve nasıl kullanıldığını anlatmalıdır.

*int g; // gun olarak gecen zaman* yerine

*int gunOlarakGecenZaman;*
*int atamadanBeriGecenGun;*
*int degisiklikten BeriGecenGun;*
*int gunOlarakDosyaYasi;*

gibi isimler seçilmelidir.



# Temiz Kod

## 2) İsimde yanlış bilgilendirmeden kaçının

Bir grup hesaba gerçekten liste değilse *hesapListesi* demeyin. *hesapGrubu*, *birGrupHesap* ya da sadece *hesaplar* daha iyi olacaktır.

Yanıltıcı isimlendirmeye en uç örnek değişken ismi olarak küçük L ya da büyük o kullanmak olacaktır.

```
int a = l;
if (O == l)
a = Ol;
else
l = Ol;
```

Kullanılan editör ve fonta bağlı olarak küçük L birle, büyük o sıfırla karıştırılabilir.

# Temiz Kod

## 3) İsimleri anlamlı olarak farklılaştırın

İsimleri numara ekleyerek farklılaştırmak yanlış bilgi vermekten öte hiç bilgi içermez.

```
public static void copyChars(char a1[], char a2[]) {
for (int i = 0; i < a1.length, i++) {
a2[i] = a1[i];
}
}
```

Argüman olarak *kaynak* ve *hedef* kullanılırsa bu fonksiyon daha anlaşılır olacaktır.

# Temiz Kod

## 4) Telaffuz edilebilir isimler kullanın

Telaffuz edilebilir isimler kullanmak anlaşılırlığı kolaylaştıracağı gibi **takım içi iletişimi** de kolaylaştıracaktır. Telaffuz edemiyorsanız, üzerine tartışmanız da bir hayli zor olacaktır.

*olsyagsdsn* (oluşturma tarihi, yıl, ay, gün, saat, dakika, saniye) değişkeninden bahsederken "Buradaki olsyag se de se ne değişkeni int olmalı" benzeri konuşmalar sıkça duyulacaktır çünkü programlama sosyal bir aktivitedir.

# Temiz Kod

## 5) Aranabilir isimler kullanın

Tek harflik isimler ve numerik sabitlerin metin içinde aranması kolay değildir. *OGRENCI_BASINA_EN_COK_SINIF*'ı bulmak *7*'yi bulmaktan kolaydır. Tek harflik isimler sadece kısa metodların içindeki lokal değişkenler için kullanılabilir.

Aşağıdaki iki kod parçasını karşılaştırınız.

```
for (int j = 0; j < 34; j++) {
t += (g[j]*4)/5;
}
```

VS

```
int reelGunSayisi = 4;
const int HAFTALIK_CALISMA_GUNU = 5;
int toplam = 0;
for (int j = 0; j < GOREV_SAYISI; j++) {
int reelGorevGunleri = gorevKestirimi[j] *
reelGunSayisi;
int reelGorevHaftalari =
(reelGunler/HAFTALIK_CALISMA_GUNU);
toplam += reelGorevHaftalari;
}
```

# Temiz Kod

## 6) Kodlamadan kaçının

İsim uzunluğu sınırlaması olan eski dillerde bu kuralı çiğnemek zorunluydu. Fortran, ilk harfin türü belirten bir kod olması kuralı nedeniyle **kodlamaya zorlayan bir dildi.** Ancak günümüzün güçlü tür kontrolü yapan modern dillerinde böyle bir sınırlama bulunmamaktadır.

*int iTelefonNumarasi;*
*yerine*
*int telefonNumarasi;*

kullanılırsa değişkenin *long* yapılması gerekince ismin değişmesi gerekmeyecektir.
Sınıf ve fonksiyon üye isimlerinde de *'_'*, *'m_'* ya da *'its'* benzeri kodlamalara gerek yoktur çünkü sınıf ve fonsiyonlarınız bunlara gerek duymayacak kadar kısa olmalıdır.
Ayrıca üye isimlerini renklendiren bir editör kullanılmalıdır.

# Temiz Kod

## 7) Zihinsel eşlemelerden kaçının

Döngü sayaçlarına *i* ya da *j* gibi isimler verilebilir çünkü bu geleneksel bir uygulamadır. Ancak diğer çoğu durumda tek harfli bir isim kötü bir seçimdir.

Yazılımcılar genelde akıllı insanlardır. Akıllı insallar bazen gösteriş yapmayı severler. *r*'nin *url*'nin sunucu ve şema çıkarılmış küçük harf versiyonu olduğunu hatırlayabiliyorsanız açıkça çok akıllı olmalısınız. **Akıllı yazılımcı ile profesyonel yazılımcı** arasındaki farklardan biri, profesyonelin, açıklığın paha biçilmez olduğunu bilmesidir.

CLEAN
CODE

# Temiz Kod

## 8) Sınıf isimleri

Sınıf ve obje isimleri, *musteri, WikiSayfası, Hesap, HesapCozumleyici* gibi isim ya da isim cümleleri olmalıdır. Sınıf ismi fiil olmamalıdır.

## 9) Metod isimleri

Metod isimleri, *odemeyiErtele, sayfayiSil, sakla* gibi fiil ya da fiil cümleleri olmalıdır.

## 10) Her kavram için tek kelime kullanın

Farklı sınıfların benzer metodları için *Sakla, Yaz* ve *Kaydet* kullanmak kafa karıştırıcıdır. Bir tanesi seçilip tüm kodda bağlı kalınmalıdır.

# Temiz Kod

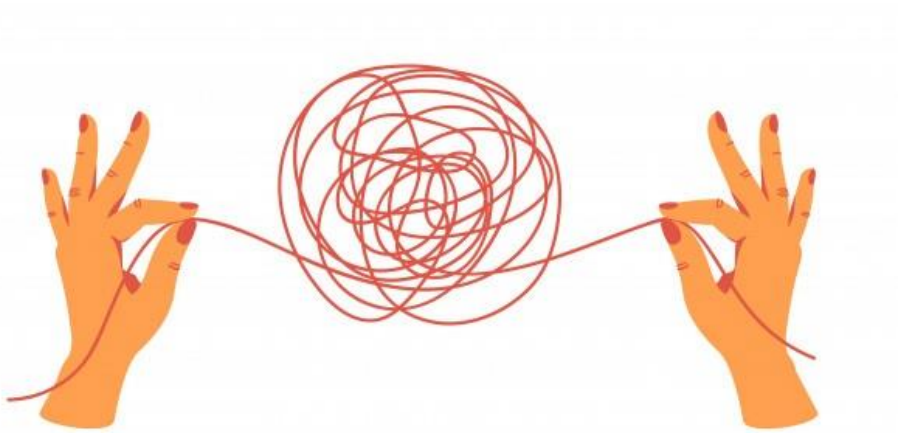
## 11) Kelime oyunu yapmayın

Aynı kelimeyi farklı fikirler için kullanmak bir kelime oyunudur. *Topla*'yı ele alalım. Bir sınıfta tuşlanan numaraların toplanması, bir diğerinde mevcut iki değerin toplanması ya da eklenmesi işlemi için kullanılması **kafa karıştırıcı olacaktır.**

# Temiz Kod

## 12) Çözüm alanına ait isimler kullanın

Kodunuzu okuyacak kişilerin yazılımcı olacağını unutmayın. Bilişim terimlerini, algoritma isimlerini, örüntü isimlerini, matematik terimlerini kullanmaktan çekinmeyin. Zorunlu olmadıkça sorun alanına ait terimler kullanmamalıyız çünkü yazılımcıların ikide bir müşteriye gidip isimlerin ne anlama geldiğini sormalarını istemeyiz. *HesapVisitor* Visitor örüntüsüne aşina bir yazılımcıya çok şey ifade edecekken, *HesapCetveli* fazla fikir vermeyebilir.



# Temiz Kod

## 13) Sorun alanına ait isimler kullanın

Yazılım jargonundan bir isim bulunamıyorsa, sorun alanından bir isim kullanılabilir. Yazılımcı en azından bir alan uzmanı bulup ne anlama geldiğini sorabilir.

## 14) Anlamlı bağlamlar ekleyin

*ad, soyad, cadde, sokak, numara, sehir ve postakodu* isimli değişkenleriniz olduğunu düşünelim. Topluca bakıldığında bir adres oluşturdukları anlaşılabilir. Fakat sadece kodun bir yerinde *numara*'nın tek başına kullanıldığını görürseniz ne anlarsınız? Ön ekler kullanarak bağlam ekleyebilirsiniz: *adresAd, adresSoyad, adresCadde*... Daha iyisi *Adres* isimli bir yapı ya da sınıf oluşturmak olacaktır. Böylece derleyici de bu değişkenlerin **daha büyük bir oluşumun elemanları** olduklarını bilecektir.

# Temiz Kod

## 15) Gereksiz bağlam eklemeyin

“Clarusway IT Merkezi” adlı hayali bir uygulamada, her sınıf isminin önüne CITM eklemek kötü bir fikirdir. Açıkcası kullandığınız araçlara karşı haksızlık yapmış olursunuz. C’ye basın ve sistemdeki tüm sınıfların bir kilometrelik listesi karşınızda.

## BONUS: Muzip olmayın

İsimlendirme çok akıllıca olursa sadece yazarın espri anlayışını paylaşanlar tarafından ve şaka hatırlandığı süre boyunca anlaşılabilecektir. *KutsalElBombasi* isimli bir fonsiyonun ne olduğunu kim bilebilir? Gerçekten çok muzipçe ama bu durumda *OgeleriSil* daha uygun bir isim olacaktır.

# THANKS!

## Any questions?

Garry T.
Full-Stack Automation Engineer
garry@clarusway.com