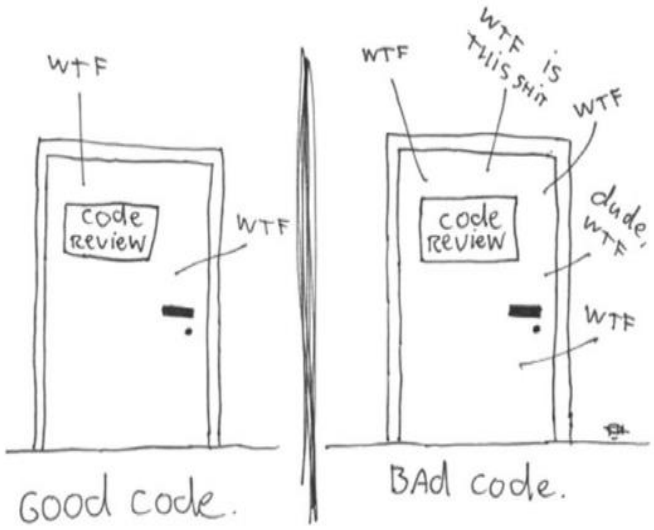
Robert C. Martin

CLEAN CODE

A Handbook of Agile Software Craftmanship

Kitabın Amacı:

→ Kitabı bitirdikten sonra, iyi kod ve kötü kod arasındaki farkı söylemek mümkün olacak. İyi kod yazmayı öğreneceğiz. Ve kötü kodu iyi koda nasıl dönüştüreceğimizi bileceğiz.



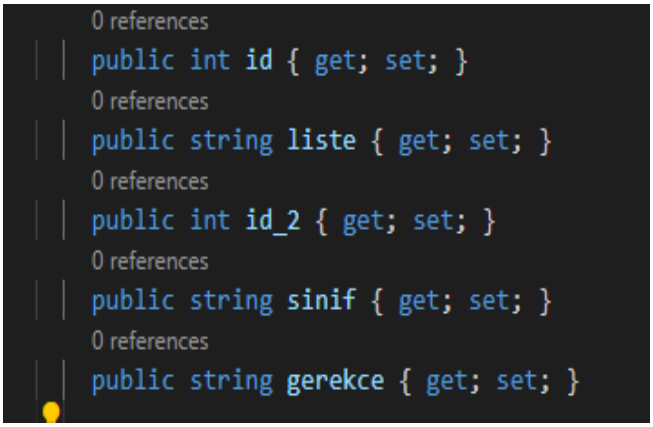
Bölüm 1-Temiz Kod:

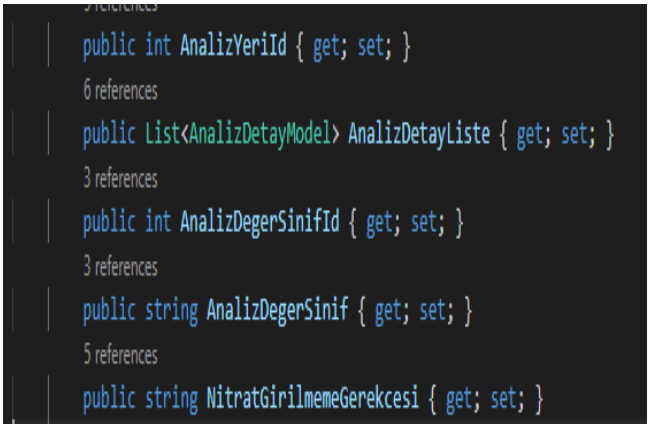
→ Temiz kod yazmanın en büyük gerekliliği kişinin ve ekibinin işini kolaylaştırmaktır. Peki temiz kod dediğimiz şey nedir? Temiz kod kısaca basit, berrak ve açıktır. İyi yazılmış bir düz yazı gibidir. Temiz kod nasıl olur? Kodda mantık, hataların saklanmasını zorlayacak kadar düz; bağımlılıklar (dependency) bakımı kolaylaştıracak kadar minimal olmalı. Tüm istisnai durumlar (exceptions) ele alınmalı, performans optimale yakın olmalı.

Bölüm 2- Anlamlı İsimler:

→ Yapılan en çok hatalardan birisi de verilen anlamsız isimlerdir. Bazen kısıtlı zamandan bazen de dikkatsizlikten dolayı yazılımcılar bu hatayı göz ardı edebiliyor fakat bu durum onların başına daha çok dert açıyor.

Kodda verilen değişken isimleri ne amaçla kullanıldığını belirtmelidir çünkü üzerinden zaman geçtikten sonra veya bir başkası tarafından kod incelediğinde onun ne olduğunu veya neden kullanıldığını direk anlamalıdır aksine anlamak için ayrıca bir zaman kaybetmemelidir(anlaya bilirse tabii😊).





Yukarda bulunan iki farklı veri kayıtları aslında durumu özetliyor. Soldakini verilerin anlamsızlığı kodun ilerleyen boyutlarında oldukça sorun teşkil ediyor çünkü bu kod belki aylar sonra bile kullanılacak veya el değiştirecek peki o durumda ne olacak? Söylenecek tek söz kalıyor “WTF!!!”.

Yapılması gereken 2. ekrandaki gibi cümle dahi tutsa tanımlama ismi tam olarak temsil ettiği şey yazılmalıdır.

NOT: Kelimeler arasına “_” koyulması gerektiğini düşünüyorum. Örneğin “Analiz_Yeri_Id” çünkü okunmasını daha açık hale getiriyor.

Bölüm 3-Fonksiyonlar:

→Fonksiyonlar, herhangi bir programdaki organizasyonun ilk satırıdır. Bu sebeple olabildiğince dikkatli davranılmalı ve temiz yazılmalıdır çünkü yine temel sebebimiz olan anlaşılırlık devreye giriyor. Temiz bir fonksiyon nasıl olmalıdır?

1) Kesinlikle uzun olmamalıdır. Yani birkaç işlemden fazlası yapılmamalıdır. Kısaca olabildiğince kısa yazılmalıdır.

2) Karışık iç içe işlemler olmamalıdır. Örneğin iç içe ifler, iç içe döngüler kodun okunurluğunu zorlaştırır bu sebeple iç içe kod yazmak yerine fonksiyonlara bölmeliyiz.

3) Birden fazla işlemi birden çok yerde yapmamak gerekiyor. Mesela bug kontrolü için try-catch yöntemini kullandığımızı düşünelim. Bug yaratabilecek her işlem altına try-catch yazılmaktansa bir fonksiyon oluşturularak bu try-catch yöntemini o fonksiyonun içine koymalı ve kontrol yapılacak her alanda o fonksiyonu çağırmalıyız.

4) Fonksiyon isimleri yaptığı işe uygun ve açıklayıcı konulmalıdır.

Bölüm 4- Yorum Satırları:

→Yazdığımız kodun ne yaptığını daha kolay anlaşılması için yorum satırı koyarak açıklamalıyız. İyi bir yorum satırı nasıl olmalıdır?

1)Gereksiz, yanıltıcı ve fazla laf kalabalığı yapan sözcüklerden kaçınılmalı, sade ve gerektiği yerde kullanılmalıdır.

2)Bilgi verici ve net olmalıdır.

3)Yapılacaklar tarzında yorum satırları gerektiği yerlere konulmalıdır. Örneğin, bu işlevin geleceğinin ne olması gerektiğini açıklayan bir yorum satırı bu adıma en iyi örneklerden birisidir.

-Açıkçası bu kitabı okumadan önce hiç yorum satırı koyma alışkanlığım yoktu. Kitabı okuyup eskiden yazdığım kodları inceledikten sonra fark ettim ki anlaşılırlığı sıfıra yakın ve temiz kod mantığından bir hayli uzak. Bu bölüm oldukça işime yaradı diyebilirim.😊

Bölüm 5-Biçimlemek:

→En çok dikkat edilmesi gereken bölümler “top 3” yapılsa bence kesinlikle girmesi gereken bir bölüm. Tek sayfada 500-1000 satır kod yazdığım oluyor bazen acele ettiğim için bazen de dikkatsizlikten. Ve geri dönüp baktığımda kodun içinde kayboluyorum. Ancak bunları parçalar halinde yazarak ya da birden çok yazılmış aynı kod parçalarını birleştirerek daha anlaşılır hale getirebiliriz. Peki biçimlendirme nasıl olur ?

1) Dikey biçimlendirme:

→ Yakından ilişkili kavramlar birbirine dikey olarak yakın tutulmalıdır.

→ Çağrılan bir işlev, çağrıyı yapan bir işlevin altında olmalıdır. Bu, kaynak kodu modülünü yüksek seviyeden düşük seviyeye kadar güzel bir akış oluşturur.

2)Yatay biçimlendirme:

→ Satırlar içinde hizalama yapmak

→ İfade gövdelerini hizalamalı ve kendi satırlarında tutmalıyız. Örneğin bir noktalı virgülü kendi satırında girintiyle görünür hale getirmediğimiz sürece, görmek çok zor.

Bölüm 6-Nesneler ve Veri Yapıları:

→Kod yazarken değişkenlerimizi gizli tutmamızın bir nedeni var. Başka kimsenin onlara bağımlı olmasını istemiyoruz. Öyleyse neden bu kadar çok programcı, nesnelerine otomatik olarak alıcı ve ayarlayıcı ekliyor ve özel değişkenlerini herkese açıkmış gibi gösteriyor?

Nesneler davranışı açığa çıkarır ve verileri gizler. Bu, mevcut davranışları değiştirmeden yeni nesne türleri eklemeyi kolaylaştırır. Demeter Kuralına göre hiçbir modül bir diğer modülünü iç dünyasını bilmemelidir.

- Her birim diğer birimler hakkında kısıtlı bilgiye sahip olmalıdır. Sadece birbiriyle yakından ilişkili olanlar birbirini tanımalı.

- Her birim yalnızca kendi arkadaşlarıyla konuşmalı; yabancılarla konuşmamalı.
- Sadece yakın arkadaşlarıyla konuşmalı.

Bu sayede gevşekçe bağlı modüller üretilmiş olur. Bu da projenin esnekliğini, bakım yapılabilirliğini, anlaşılabilirliğini ve test edilebilirliğini artırır.

Bölüm 7-Hata İşleme:

→ Hata işleme, programlarken hepimizin yapması gereken şeylerden sadece biridir. Kullanıcıdan aldığımız veriler anormal olabilir. Kısacası, işler ters gidebilir ve yaptıklarında, programcılar olarak kodumuzun yapması gerekeni yaptığından emin olmaktan sorumluyuz.

Temiz kod okunabilir, ancak aynı zamanda sağlam olmalıdır. Sağlam olması için hata sınıflandırması yapmalıyız. Hataları sınıflandırmanın birçok yolu vardır. Onları kaynaklarına göre sınıflandırabiliriz: bir bileşenden mi yoksa başka bir bileşenden mi geldiler? Veya türlerine göre sınıflandırabiliriz: ağ arızaları veya programlama hataları mı? Hata sınıflandırmasının yanında fonksiyondan null döndürmemeli ve bir fonksiyona parametre olarak null verilmemelidir.

Bölüm 8-Sınırlar:

→Sistemlerimizdeki tüm yazılımları nadiren kontrol ediyoruz. Bazen üçüncü taraf paket satın alır veya açık kaynak kullanırız. Diğer zamanlarda, bizim için bileşenler veya alt sistemler üretmek için kendi şirketimizdeki ekiplere güveniyoruz. Her nasılsa, bu yabancı kodu kendimizinkiyle temiz bir şekilde bütünleştirmeliyiz. Bu bölümde, yazılımın sınırlarını temiz tutmak için gerekli teknikleri anlatıyor. Peki bunlar nedir?

1)Üçüncü Taraf Kodunu Kullanma: Bu sağlayıcılar geniş uygulanabilirlik için çaba gösterir, böylece birçok ortamda çalışabilir.

2)Genellikle alınan hazır yazılımlar kullanım şeklinin anlatıldığı şekilde kopyalanır ve adım adım ilerlenir. Evet kod çalışır ama nasıl çalıştığını bile bilmezsin. Fakat doğrusu alınan yazılımın test ortamından geçirilerek, araştırılarak ve sindirilerek enjekte edilmesi gerekmektedir.

Bölüm 9-Birim Testler:

→ Testler, bir projenin sağlığı için üretim kodu kadar önemlidir. Çünkü testler üretim kodunun esnekliğini, sürdürülebilirliğini ve yeniden kullanılabilirliğini korur ve geliştirir. Test kodları üretim kodu değiştikçe güncellenmelidir ve üretim kodu kadar önemlidir. Üretim kodunun esnek, bakım yapılabilir ve yeniden kullanılabilir olmasını birim testleri sağlar.

Testler olmadan her değişiklik, potansiyel bir hatadır.

Temiz bir testi ne sağlar? Özetle okunabilirlik diyebiliriz. İkinci bir dikkat edilmesi gereken hususta yazılmış hata testinin totalinden fazlasını oluşturan bir kod parçası yazmamak gerekiyor.

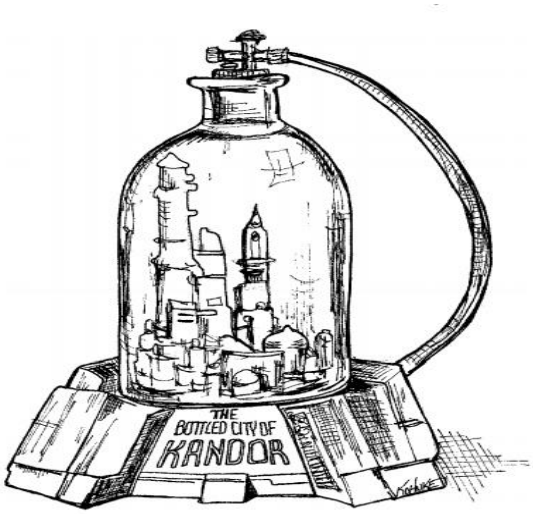
Bölüm 10-Sınıflar:

→Her sınıfın en üstünde değişkenler bulunmalıdır. Bunlar da öncesinde public sonrasında private değişkenler gelmelidir. Sınıfları oluştururken dikkat edilmesi gereken ilk ve en önemli kural kısa olmasıdır. Çünkü kısa sınıf ve fonksiyonlar kolay isimlendirilir, kolay yazılır, kolay anlaşılır. Sınıfların sorumlulukları 70 adet *public* fonksiyonu aşmamalıdır.

Bölüm 11-Sistemler:

"Complexity kills. It sucks the life out of developers, it makes products difficult to plan, build, and test."

—Ray Ozzie, CTO, Microsoft Corporation



→Sistemler kısaca kurulan yapının temelidir. Bu sebeple sistemler yazılırken her şey planlanarak sıfırdan başlanarak ve kollara ayrılmadan tek tek adım adım işleri complex yapılara çevirmeden ilerlenmelidir. Örneğin bir proje oluşturulduğu zaman öncelikle ekranlar arasındaki bağlantılar kurulmalı bir inşaat gibi temelleri atılmalı ve ondan sonra içleri doldurulmalı. İçleri dolarken de biran da değil tane tane yani sistematik bir şekilde ilerlenmeli.

Bölüm 12-‘Emergence’:

→Kitapta bahsedilen Kent Beck’in 4 basit tasarım kuralı ile temiz bir tasarım ortaya çıkarabiliriz. Eğer bu kurallara uyarsak, kodun yapısı ve tasarımı hakkında bilgi edinir , SRP (Single Responsibility Principle) ve DIP(Dependency Inversion Principle) gibi ilkelerin uygulanmasını kolaylaştırırız. Peki bu 4 kural nedir?

1) Tüm testleri çalıştırmak.
2) Tekrarlanmış kod yazmaktan kaçınmak.
3) Açıklayıcı olmak.
4) Sınıf, metot ve fonksiyon sayısını en aza indirmek.

SON

Hazırlayan: Esat Yener