**DRY (Don’t Repeat Yourself)**

**Hazırlayan: Emre Can Tuluk**

Don’t repeat yourself (DRY), yazılım kalıplarının tekrarlanmasını ve buna bağlı olarak ortaya çıkabilecek fazlalıkların önlenmesini amaçlayan bir yazılım geliştirme ilkesidir. DRY prensibi, her bilginin bir sistem içinde tek, kesin ve yetkili bir temsili ile var olması gerektiğini vurgular. Prensip Andy Hunt ve Dave Thomas tarafından “Pragmatic Programmer” kitabında formüle edilmiştir.

Dave Thomas, “Çoğu kişi DRY’ı kodun tekrarından kaçınmak ile sınırlandırsa da prensibin ardındaki fikir bunun çok daha ötesindedir.” der. Tüm bu bilgiler göz önüne alındığında, neden her özelliği temsil eden bir yol bulmaya çalışmayalım? İlk akla gelen cevap, eğer aynı amaca hizmet eden birden fazla yol söz konusu ise büyük ihtimalle her yol bir diğeri ile çelişecektir. Çelişmeseler dahi, bir değişiklik gerçekleştiğinde yine bir sorun yaşanmayacağının garantisi yoktur.

DRY prensibinin başarıyla uygulanması, sistemde herhangi bir değişiklik söz konusu olduğunda diğer mantıksal elemanların bu değişiklikten etkileyip etkilemeyeceğini düşünmememiz anlamına gelir. Ek olarak, ilişkili mantıksal elemanlarda da etkiler tahmin edilebilir olacağı için düzgün bir şekilde süreç ele alınacak ve değişiklikler güncel olarak yönetilebileceklerdir.

DRY Prensibinin Avantajları

Sürdürülebilirlik (Maintainability):

DRY ile öne çıkan en önemli fayda tekrarlanan kodlar sebebiyle söz konusu olabilecek çakışmalar, aksamalar ve hataların önüne geçilebilmesidir. Bu sayede ilişkili bir logic işlemde ortaya çıkabilecek bir geliştirme ihtiyacı veya sorun kolaylıkla tek bir alan üzerinden çözülür.

Okunaklılık (Readability):

DRY prensibi ile yazılan bir kod çok daha kolay okunur, analiz edilir ve anlaşılır. Elbette bu doğrudan DRY sayesinde değil, DRY prensibi ile kodu ele alan geliştirici sayesinde ortaya çıkan bir durumdur. Çünkü, geliştirici bu prensip doğrultusunda daha fazla çaba göstererek daha detaylı bir çözüm yolu ortaya koymalıdır.

Tekrar Kullanılabilirlik (Reuse):

DRY vurguladığı bakış açısı gereği, kod tekrarı yerine kodun tekrar kullanımını teşvik eder. Yeniden kullanılabilir kod, geliştirme süresini hızlandırdığı için maliyet avantajı sağlar.

Test Edilebilirlik (Testing):

Ne kadar çok işlem söz konusu ise o kadar fazla test kodu yazılması gerekir. Eğer kod tekrarı söz konusu değilse bu daha az test işlemi demektir. Dolayısıyla DRY sayesinde daha etkin ve pratik test süreçleri (unit testing ve integration testing) söz konusu olacaktır.