# Test Güdümlü Geliştirme (TDD)

Test güdümlü geliştirme (TDD), kodu yazmadan önce bir testin yazıldığı bir yazılım geliştirme yaklaşımıdır.   
TDD, kaynak kodun tamamen birim test edilmesini sağlar ve **modüler**, **esnek** ve **genişletilebilir** koda yol açar. Sadece testleri geçmek için gereken kodu yazmaya odaklanarak tasarımı basit ve anlaşılır hale getirir.

## YAGNI Prensibi

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 özellik için harcanan analiz, programlama ve test etme süreleri boşa harcanmış olur.

## Takıma yeni katılan kişi testleri okuyarak sistem hakkında bilgi edinebilir.

* Yeni bir ekibe katıldığımızda bizden önce takımda bulunan kişiler yazılımı geliştirirken testleri yazdı ise sistem hakkında daha kısa sürede bilgi sahibi olabilirsiniz.

## Test güdümlü geliştirme yaptığımızda YAGNI prensibine ister istemez uyarız.

Meyve suyu sıkan bir yazılım geliştiriyoruz. Testimizde meyve verdik karşılığında meyve suyunu bekledik. Testimiz kırmızı ve çözülmeyi bekliyor. Geliştirme yaparken amacım sadece testi çözmek onun dışında herhangi bir kod bloğu eklemek değil. Sadece ihtiyacım olan kodu yazdım fazlasını değil.

Test olmadan yazdığımızı varsayalım. Geliştirme sırasında meyveyi verelim suyunu versin. Ya bir de ileride meyveyi suyunu versin ama şunlarıda yapsın derler en iyisi şunu da ekleyeyim dursun. **Bu özellik için harcanan analiz, programlama ve test etme süreleri boşa harcanmış olur.**

* **Test ekleyin:** TDD'deki her yeni özellik, herhangi bir özellik uygulanmadan önce yürürlüğe girdiği için başarısız olması gereken bir testle başlar. Özellik uygulanmadan önce bir test yazmak için temel gereksinim, geliştiricinin gereksinimini açıkça anlamaktır.
* **Tüm testleri çalıştırın ve yeni kodun başarısız olup olmadığını kontrol edin:** Bu, test kayışının doğru çalışmasını ve yeni testin yeni kod olmadan yanlışlıkla geçmediğini garanti eder. Bu adım ayrıca testi test eder ve yeni testin her zaman geçme olasılığını ortadan kaldırır.
* **Kod yazma:** Sonraki adım, sınamayı temizleyen kod yazmaktır. Yeni kod mükemmel değil ancak gereksinimlere göre değiştirildi. Sadece test için tasarlanmıştır ve diğer işlevleri kapsamaz.
* **Otomatik testler gerçekleştirin:** Üretilen her test senaryosu testi kolayca geçerse, kodun gerekli tüm özellikleri karşıladığı anlamına gelir. Böylece döngünün son adımı başlatılabilir.
* **Refactor kodu:** Bu, çoğaltmanın kaldırılmasına benzer. Yeniden düzenleme, mevcut işlevlere zarar vermez ve üretim ile test kodları arasındaki çoğaltmanın kaldırılmasına yardımcı olur. Kod artık gerektiği gibi temizlenir.
* **Tekrarla:** Döngü, önceki durumlarda olduğu gibi yeni bir testle tekrarlanır. Temel gereklilik, adımların boyutunun küçük olması ve her test çalışması arasında yaklaşık 1 ila 10 düzenleme yapılmasıdır. Sürekli entegrasyon geri döndürülebilir kontrol noktaları sağlar.

