

## TEST DRIVEN DEVELOPMENT EĞİTİMİ

### NEDEN TDD?

TDD, kod yazılmadan önce test senaryolarının yazılması, bu senaryolara bağlı olarak kodun yazılması ve refactor edilmesi tekniğidir. TDD yaklaşımıyla, yazılması planlanan kodun test senaryoları, sürekli olarak koşturulabilir ve bu sayede programın daha az hata ile geliştirilmesi sağlanabilir. Ayrıca, waterfall yöntemlerdeki yazılım tasarımı problemi de, isteği karşılayacak sınıfların yazılması zorunluluğu ile büyük ölçüde aşılmaktadır. Bu da geliştirilen yazılımın kalitesini arttırmakla kalmaz, değişim yönetimini ve tutarlılığı maksimize ederek yazılım maliyetlerini düşünülmediğinden çok daha aşağıya çeker.

Bilindiği gibi birim testleri, tam olarak yazılan metotların işlevlerini düzgün bir şekilde yerine getirip getirmediğini test etmektedir. Ancak birim testleri düzgün bir şekilde yapabilmek görüldüğü kadar kolay değildir. Öncelikle kullanılacak olan araçların belirlenmesi gerekmektedir. Önceleri kalite kontrol ekipleri tarafından kullanılan, çeşitli script dilleri ile konfigüre edilen büyük test motorlarının birim testleri için yeterince güçlü ve kolay kullanılabilir olmamaları TDD'nin oluşmasına ön ayak olmuştur. TDD bu noktada sadece birim testlerin yapılmasını kolaylaştırmakla kalmaz, kalite kontrol testleri, kullanıcı kabul testleri, yazılım tasarımı ve değişiklik yönetiminin en az maliyetle gerçekleştirilmesine olanak sağlar. Günümüzde geleneksel yöntemlerle geliştirilen çoğu proje, yukarıdaki problemlerden dolayı başarısız olmakta ya da kalitesiz ve demode olduklarından iş ihtiyaçlarını karşılayamamaktadırlar. Ancak, doğru ve bilinçli uygulanan TDD sayesinde, projelerin başarı oranlarının ciddi artışlar gösterdiği ispat edilmiştir. TDD nin getirdiği iş yükü, sağladığı kazancın yanında önemsenmeyecek kadar küçüktür.

Kısaca TDD'nin getirileri şöyle sıralanabilir;

- Sadece ilgili birim testlerin yapılarak kodun güvenli hale getirilmesi.
- Kod tasarımından kaynaklanabilecek problemlerin ortan kaldırılması.
- Testlerin bir bütün haline getirilerek, geriye dönük testlerin sürecin önemli bir parçası haline getirilmesi.
- Yeni eklenen kodlar, ya da değiştirilen kodlarda mevcut kodların işlevlerinin bozulmaması.
- Kodların güvenli ortamda nesneye yönelik tasarı mimarisine uygun üretilmesini azami ölçüde kolaylaştırması.
- Kodların dokümantasyon yerine, test senaryolarından daha kolay anlaşılabilir olması.
- Sağlanan güvenli ortam sayesinde, refactoring işlemlerinin güvenli hale getirilmesi.
- Bug oluşması ihtimalinin azaltılması.
- Her an live ortama geçilebilecek kodların çıkarılması.
- Gereksiz kod kalabalığının ortadan kaldırılması.
- Test ekibinin gerçek test işlemlerine odaklanmasının sağlanması.

- Proaktif çalışma sayesinde, sıkıcı bir geliştirme ortamı yerine, eğlenceli ve güvenli bir ortamda motivasyonu yüksek ekiplerin oluşturulabilmesi.

Not: Eğitimde Java dilinde yazılmış kod örnekleri kullanılacaktır ancak isteğe göre tercih edilen dil değiştirilebilecektir.

## EĞİTİMİN İÇERİĞİ:

### 1. Gün

- TDD'ye genel bakış: TDD nedir, TDD'nin getirileri nelerdir, TDD nasıl uygulanır?
- JUnit: JUnit nedir, nasıl kullanılır?
- Uygulama: Varolan kod için JUnit testlerinin yazılması.
- Yazılım dizaynında ZEN Yaklaşımı: Basit dizayn nedir, nasıl yapılır?
- Demo: Basit bir uygulamada TDD aşamaları.
- Uygulama: Basit bir class için TDD'nin uygulanması.
- TDD'nin etkileri: TDD verimli bir şekilde nasıl uygulanır, TDD'nin günlük işlerimize olan etkileri nelerdir,
- TDD'nin diğer yazılım geliştirme pratikleriyle (debugging, modular design, component reuse vs.) etkileşimleri.

### 2. Gün

- Demo: TDD ile yazılım mimarisi.
- Uygulama: Çoklu class ortamında TDD uygulaması, dizayn ve mimarinin gelişimi.
- Demo: Sıfır entegrasyon hatası.
- Uygulama: Mock objelerin kullanımı.

### 3. Gün

- Demo: Örnek bir web sayfasında TDD uygulaması.
- Story ve Story Testing: User story, TDD ve Story testing ile müşteri için önemli fonksiyonların önceliklendirilmesi.
- Exercise: Story test dizaynı.
- Demo: Rails ve TDD
- Özet

**KİMLER KATILMALI:**

Eđitim, TDD'yi uygulamayı dūřūnen, yazılım proje deneyimi olan, herhangi bir Object Oriented programlama dili ile yazılım geliřtiren uzmanlara, yazılım mimarlarına, test uzmanlarına, proje yōneticilerine, bilgi iřlem yōneticilerine ve akademisyenlere yōneliktir.

Katılımcıların eđitime gelmeden ōnce ařađıdaki kaynakları incelemeleri ōnerilmektedir:

- [www.junit.org](http://www.junit.org)

- [www.jmock.org](http://www.jmock.org)

- Kent Beck, Test Driven Development: By Example

- J. B. Rainsberger with Scott Stirling, JUnit Recipes: Practical Methods for Programmer Testing