Web Uygulamaları Geliştirilmesinde Test Güdümlü Programlamanın Yeri: Bir Örnek Durum Çalışması

Ali Yamuç[1], Ayhan Çakın[2]

[1] Yıldız Teknik Üniversitesi, Enformatik Bölümü, İstanbul

[2] Yıldız Teknik Üniversitesi, Enformatik Bölümü, İstanbul

ayamuc@yildiz.edu.tr, acakin@yildiz.edu.tr

Özet: Genel olarak bilinen ismiyle "Test Driven Development"(TDD) ya da diğer adıyla "Test First Programming" olarak bilinen test güdümlü programlama yöntemi, aslında ilk olarak Çevik(Agile) yazılım geliştirme metodunun bir parçası olarak, bu sürecin oluşturucusu Kent Beck tarafından tanıtılmıştır. Şuan günümüzde ise bu yöntem, gelişmiş olan pek çok çevik yazılım geliştirme sürecinin çok önemli kısımlarını oluşturmaktadır. Bu yöntemi önemli kılan özelliği, Test Güdümlü Programlamanın geleneksel geliştirme yöntemlerin tamamen değiştirmesi ve bu sayede yazılım kalitesine yaptığı katkılardır.

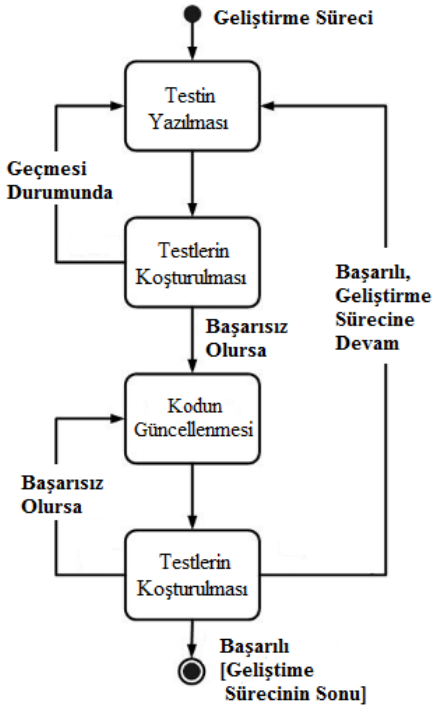
Web tabanlı uygulamalar günümüzde artık her alanda kullanılmakta ve gün geçtikçe gerek erişilebilirlik gerekse ortak kullanım alanlarının artmasından dolayı uygulamalar web ortamına taşınmaktadır. Web uygulamaları geliştirilmesinde TDD bazı çatılar (framework) tarafından desteklenmekte anca diğer klasik metotlara göre zahmetli gözüktüğünden dolayı pek de uygulanmamaktadır. Özellikle karmaşık web uygulamalarının geliştirilmesinde TDD'nin kullanımı birçok hatayı çok önceden önlemekte ve servislerin hizmet dışı kalması ya da yanlış sonuçlar üretmesi kod yazımı aşamasında engellenebilmektedir. Biz bu çalışmamızda; öncelikle Test Güdümlü Programlamanın günümüzdeki yazılım süreçlerine katkı ve faydalarını ve özellikle web tabanlı uygulamaların geliştirilmesindeki kullanım yöntemlerini ele aldık. Bunların ardından ise test güdümlü programlamayı destekleyen ve şuan hızla gelişmekte olan güncel nesne tabanlı PHP çatılarından(framework) biri olan "Yii" çatısını TDD yönünden, örneklerle inceledik.

Anahtar Sözcükler: Test Güdümlü Programlama, Test Öncelikli Programlama, Çevik Yazılım Geliştirme, Web Tabanlı Uygulama, Yii.

1. Giriş

Son dönemlerde yazılım piyasasında yapılan araştırmalarda, bug'ların analizinin ve çözümlerinin şirketlere büyük miktarlarda hem zaman hem para ve en önemlisi de prestij kaybettirmesi, artık günümüzde testler yazmanın ekstra bir yük değil, bir gereklilik olduğunu ve TDD'nin yazılım geliştirme sürecindeki önemini göstermektedir.

Test Güdümlü Programlama yöntemi, evrimsel bir yaklaşıma sahip olup, iki temel birimin bir araya getirilmesi ile oluşturulduğunu söyleyebiliriz: Test Öncelikli Geliştirme(TFD) ve Gözden Geçirme(Refactoring). Test öncelikli geliştirmenin temelinde yatan ise kodu yazmaya başlamadan önce ona ait bir test yazmak ve daha sonra bu testi karşılayacak kadar yeterli kodu yazmaktır[1]. Bu yöntemdeki ilk adım kodun başarısız olmasına yetecek kadar kısa bir test yazmaktır. Daha sonra bu testleri koşturarak başarısız olduğundan emin olduktan sonra, ikinci adım kodu güncelleyerek bu testi geçmesini sağlamaktır. Testleri tekrar koştuktan sonra eğer kodumuz halen başarısız ise kodu güncelleyip yeniden test etmek gerekir. Testlerin tamamını geçtiğimiz anda bu döngünün en başına dönüp, yeniden başlamak gerekir. Tabiki en başa dönmeden, kodu gözden geçirip dizayn ve gereksinimlere uymayan kısımları düzeltilebilir ve böylece uygulanan süreci Test Öncelikli Geliştirmeden, Test Güdümlü Programlamaya dönüştürmüş oluruz.



Şekil 1 - Test Öncelikli Geliştirme

Test güdümlü programlama geleneksel geliştirme yöntemlerini özellikle yazılım kalitesini geliştirmek açısından tamamen değiştirmektedir. Örneğin yeni bir modül geliştirmek istediğimizde, bu sürece başlamadan önceki ilk adımımız, elimizdeki tasarımın bu modülü geliştirmek ve entegre etmek için en iyi tasarım olup olmadığını sorgulamaktır. Eğer öyle ise, hemen test öncelikli geliştirme sürecine başlanabilir ancak tasarımın uygun olmadığını ya da yetersiz olduğunu düşünüyorsak, önceliğimiz geliştirme sürecinden çok; varolan tasarımın modülün ekleneceği kısmının gözden geçirilmesi(refactoring) ve bu kısmın yeni eklenecek özelliğe uygun hale getirilmesidir. Aslında bu sayede, adım adım elimizdeki tasarım gelişmekte ve gelecekteki çalışmalarımız için çok daha kaliteli ve uyumlu hale gelmektedir.

TDD bir yazılım sürecini küçük parçalardan oluşan bir tasarım ve geliştirme sürecine dönüştürmektedir. Çünkü baktığımızda TDD'nin temel amacı, kodlama sürecini küçük parçalara bölmektir ve Kent Beck bu süreçleri tanıttığı "Test-Driven Development: By Example"[2] adlı kitabında şu iki konsepte dikkat çekmektedir:

- Elinizde başarısız bir test senaryosu olmadan asla tek satır kod bile yazmayın,
- Kod tekrarı yapmayın.

Wiiliam Wake ise "Extreme Programming Explored "[3] adlı kitabında, bu süreci maddeler halinde şu şekilde anlatmıştır:

1. Test kodununun yazılması
2. Test kodunun derlenmesi
3. Test kodunun derlenmesine yetecek kadar kodun yazılması ve derlenmesi

4. Testleri çalıştırıp başarısız olduğunun gözlenmesi
5. Kodun sadece testi geçecek kadar güncellenmesi
6. Testlerin tekrar çalıştırılıp, başarılı olduğunun gözlenmesi
7. Kodun gözden geçirilmesi ve gerekli değişikliklerin yapılması(Refactoring)
8. Sonraki bölüm için başa dönülmesi

Bu adımlar tüm geliştirme süreci boyunca tekrarlanır.

Kod tamamlandıktan sonra test yazmayı beklerseniz, baştaki tüm bu yöndeki niyetinize rağmen muhtemelen test yazmayacaksınız. Testinizi önce yazarak ve kodlama sürecine direkt olarak test yazma sürecini enjekte ederek en kapsamlı testi sağlamış olursunuz. Test süreçlerinin bu derinliği ile karmaşık yazılım uygulamalarıyla birlikte gelen korku ve stresi minimize edeceğiz ve değişiklikler ve eklemeler yaptıkça gelen pozitif geribildirimlerin sürekli olarak gelmesi ile geliştiricilerde ve paydaşlarda ciddi bir güven duygusu oluşacaktır.

2. TDD: Neden ve Nasıl?

Test güdümlü programlamanın bir üretim kodu yazmadan önce testlerin yazıldığı ve uygulandığı bir yazılım geliştirme yöntemi olduğundan kısaca bahsettik. Bu konudaki ilk ve tanımsal kitap, özellikle birim testler üzerinde yoğunlaşan, Kent Beck'e ait "Test Driven Development" kitabıdır. Bu kitapta özellikle Birim Test kavramı üzerinde durulmuş olsa da daha sonraki yapılan bazı çalışmalarda, TDD'nin bütün seviyelerde uygulanabilirliği tartışılmıştır.

Michael Feathers "Working Effectively With Legacy Code" kitabında[4], TDD'nin gerekliğinden şöyle bahsetmektedir:

> "*Test olmadan yazılan kod kötü koddur. Ne kadar iyi yazılmış, ne kadar güzel, ne kadar nesne tabanlı ya da ne kadar iyi kapsüllenmiş(encapsulated) olursa olsun, bunu değiştirmez. Testlerle kodun davranış biçimi çabuk ve denetlenebilir şekilde değişir. Testler olmadan kodumuzun iyiye ya da daha kötüye mi gideceğini gerçekten bilemeyiz.* "

TDD'nin neden kullanılması gerektiği konusunda birçok sebep bulunabilir ancak burada bahsedilen, bunların en önemlilerinden biri gibi gözükmektedir. Bunu daha somut bir örnekle kafamızda canlandırabiliriz[5]. Test olmadan yazılım geliştirme sürecini bir odanın tabanını karanlıkta çivilemeye benzetebiliriz. Odanın bir ucundaki kısmının tabanını çivilemeyi bitiriyoruz ve diğer köşeye doğru gidiyoruz. Bu kısmı çivilemeye başladığımızda, karanlık olduğundan dolayı diğer kısımdaki tabanın çivilerinin yerinden çıktığını veya tabanın yerinden kalktığını farkedemeyebiliriz. İşte test olmadan kod yazmak aynen buna benzetilebilir, yani bir değişiklik yaptığımızda ya da mevcut sisteme yeni bir modül eklediğimizde, eldeki sistemin ne şekilde, neresinden etkileneceğini bilemeyiz. Test güdümlü programlama, örneğin devamı olarak söylemek gerekirse, odadaki ışığı açmak ve bir arkadaşımızın bizim için diğer ucu izleyip haber vermesine benzetilebilir.

Birim ve fonksiyonel testleri yazmanın pek çok yararları vardır. Bunlardan biri, bu testler dokümantasyon oluşturmanın muhteşem bir yoludurlar. Birim testler bize bir kod bloğunun niçin var olduğunun tam hikâyesini anlatırlar. Benzer bir şekilde, fonksiyonel testler bir uygulama içerisinde

hangi özelliklerin gerçeklendiğini dokümente ederler. Bu testleri yazmayı özenle ve sabırla sürdürürseniz, uyguluma geliştikçe doğal olarak dokümantasyon da gelişecektir.

Bu testler aynı zamanda kodunuzun ve uygulamanızın beklendiği gibi çalışıp çalışmadığını geliştiricilere ve diğer paydaşlara sabit bir şekilde tekrar tekrar güvence vererek paha biçilemez bir geribildirim mekanizması sunmaktadır. Kodunuzda değişiklik yaptığınız her vakit testlerinizi çalıştırırsınız ve sistem davranışının beklenmedik bir şekilde değişip değişmediğine dair hemen geribildirim alırsınız. Bu yaklaşım uygulamanın davranışında geliştiriciye inanılmaz bir güven verir ve daha az hatalı ve daha başarılı projeler elde edilmesini sağlar. Bu anında geribildirim kod temelinin (code base) tasarımını değiştirmede ve geliştirmede büyük kolaylık sağlar. Bir geliştirici var olan kodda geliştirmeler yaparak ve akabinde bir takım testleri çalıştırarak uygulama davranışının değişip değişmediğini doğrular. Fonksiyonel ve birim testler tarafından sağlanan bu güven ile geliştirici daha iyi, daha kararlı uygulamalar geliştirebilir.

2.1. Otomatikleştirilmiş Testler (Automated Tests)

Geri-bildirim toplamak çevik geliştirmenin[6] en önemli yapıtaşlarındandır. Geri-bildirimler uygulamanın kullanıcılarından, proje paydaşlarından, geliştirme takım üyelerinden ve yazılımın kendisinden gelebilir. Bu şekilde yazılım geliştirerek uygulamaların hizmete sokulması ve entegrasyonu gibi konularda olabilecek birtakım sorunlardan geliştiricilerin hemen haberlerinin olması sağlanır. Birim (unit) ve fonksiyonel (functional) testler yazarak ve onları sık sık çalıştırarak uygulamayı bu geribildirim mekanizmaları ile güçlendirmiş oluruz.

2.2. Birim Testler (Unit Tests)

Birim testler, adından da çıkarım yapılabileceği gibi, otomatikleştirilmiş olarak sadece bir birimi test eden testlerdir, örneğin bir sınıfı ya da metodu. Genel olarak test edilen bir sınıf ise, bu sınıfın bağlı olduğu tüm değişkenler yapay(mock) nesneler ile ilklendirilir ve bu şekilde testlerin koşması sağlanır. Buradan bir birim testinin ne olduğunu çıkarmak için, Micheal Feathers'ın bir birim testinin "ne olmadığı" ile ilgili özetine bakabiliriz.

Bir test eğer aşağıdaki koşulları sağlıyor ise "birim test" değildir:

- Eğer bir veritabanı ile etkileşimi varsa
- Eğer ağ üzerinden iletişimi varsa
- Eğer dosya sistemine müdahalede bulunuyorsa
- Eğer başka bir birim testi ile aynı anda koşturulmaya çalışılıyorsa
- Eğer bu testin koşması için ortam değişkenleri ya da yapılandırılma dosyalarının değişmesi gerekiyorsa

2.3. Fonksiyonel Testler (Functional Tests)

Fonksiyonel testler elimizdeki sistemin önceden belirlemiş olduğumuz tasarım kriterlerine tam olarak uyup, gereksinimleri karşılayıp karşılamadığını ölçmek amacı ile yapılan testlerdir. Başka bir deyişle, fonksiyonel testler fonksiyonel gereksinimlerin sistemde eksiksiz ve doğru biçimde uygulanıp uygulanmadığını ölçen testlerdir. Genel olarak kara kutu testi(black box test) grubunda anılmakta olup, yani bu testler için sistemin alt seviyelerinde çalışan mekanizmalar hakkında bilgiye sahip olmaya gerek yoktur.

Fonksiyonel testler, gereksinim kullanım senaryolarından(requirement use cases) çıkartılır ve her senaryo bir fonksiyonel test haline gelir. Bu senaryoya ait modül yada modüller yazılıp, birim testleri yapıldıktan sonra karşılık gelen fonksiyonel test uygulanır.

3. Bir Örnek Durum: Akademik Katalog Sistemi

3.1. Giriş

Test Güdümlü Geliştirme bir örnek durum çalışması olarak göstermek için çevik metodoloji ve TDD kullanarak geliştirmiş olduğumuz web-tabanlı Akademik Katalog Sistemi uygulamasından faydalanacağız. Bu örnek durum çalışmamızda Akademik Katalog Sistemi uygulamasının sadece ufak bir kesitini ele alacağız. Bu kesit üzerinde gereksinim analizi, modelleme ve tasarım yaparak ve TDD kullanarak nasıl geliştirildiğini örneklerle anlatacağız. TDD kullanarak yapacağımız uygulama geliştirmeyi anlamak için önce Akademik Katalog Sistemi'nin amacı ve kapsamı hakkında kısa bir bilgilendirme yapalım.

Akademik Katalog Sistemi yüksek öğretim kurumlarının Bologna Süreci [7] kapsamında online olarak sunulması zorunlu olan bilgilerin belli ara yüzlerle sunulması ve yönetilmesi için web tabanlı bir bilgi sistemidir. Bu sistemin temel amacı yükseköğrenim kurumlarının öğrenim süreçlerinde kalite güvenceyi, şeffaflığı, izlenebilirliği ve ölçülebilirliği sağlamaktır. Bunun için öğrencilerin kayıtlı oldukları derslerde, dersin öğrenim çıktılarını ne ölçüde başarabildiğini tespit etmek için değişik anketler ve yöntemler uygulanır. Dersin öğrenim çıktılarını karşılamak için öğrenciye yüklenen iş yükünün ağırlığına bağlı olarak derse ECTS kredisi dersin koordinatörü tarafından atanır. Bu ECTS kredileri de her akademik dönem sonunda eldeki sayısal veriler ve öğretim elemanlarının bir miktar inisiyatifi ile düzenli olarak güncellenir. Akademik Birimler tarafından sunulan programların yeterliliklerini sağlamak için dönemlik ve program sonunda toplanacak ECTS kredileri standartlaştırılır. Bu ECTS kredisi kısıtlarına uyarak program direktörü ders havuzundan dersleri seçerek bu müfredata yerleştirir. Programa kayıtlı olan öğrencilerin program yeterliliklerini karşılaması ve mezun olabilmesi için program müfredatında olan dersleri başarıyla tamamlaması gerekmektedir. Programlardaki derslerin, program yeterliliklerini ile ne ölçüde karşıladığı program direktörünün sorumluluğundadır. Ders koordinatörleri de programın yeterliliklerine, amacına ve hedefine uyulması için müfredatlardaki derslerin ECTS kredilendirmesini Akademik Katalog Sistemi'nin sağladığı geri-dönüş mekanizmaları ile güncellemekle sorumludurlar. Böylelikle, nicel olarak izlenebilir, ölçülebilinir, sorumlulukların ayrıştırıldığı ve şeffaf böyle bir sistemle daha başarılı bir öğretim imkânı sağlanır.

Temel olarak bu amaç ve kapsamda olan bu uygulamamızın örnek durum olarak üzerinde duracağımız belli bir kesitin kullanıcı hikâyelerini ve gereksinim analizlerini yapacağız. Daha sonra bu gereksinimleri karşılamak üzere TDD ile geliştirme yapacağız. Bu kapsamda birim ve fonksiyonel testler yazarak, "TDD ile nasıl uygulama geliştirilir" in bir örnek durum çalışmasını yapmış olacağız.

3.2. Kullanıcı Hikâyeleri (User Stories)

Kullanıcı hikâyeleri yazılımın istenen özelliklerini tanımlamak için çok iyi bir yoldur. Kullanıcı hikâyeleri bir kullanıcının yazılımı üzerinde neler yapabileceğinin belirtilmesini sağlar. Bu hikâyeler önce basit başlamalı ve sonra her özelliği derinlemesine detaylandırarak zamanla karmaşıklaştırılmalıdır.

Örnek durum çalışması olarak ele aldığımız Akademik Katalog Sistemi; kullanıcının hesap oluşturduğu, kimlik doğrulaması (authentication) yaptığı ve kullanıcının sahip olduğu rollerine göre uygulamanın değişik özelliklerine eriştiği, çok-rollü kullanıcı tabanlı bir sistemdir. Kullanıcılara

sistemde yapabilirliklerine göre uygun roller atanır. Her rolün sistemde belli izinleri (permissions) vardır ve kullanıcılar bu izinlere bağlı olarak tanımlanmış aksiyonlar(actions) gerçekleştirirler.

Sistemde en temel 2 tane kullanıcı olacaktır: Ziyaretçiler (Guest) ve Yetkililer (Authenticated). Ziyaretçi login süreciyle sistemden yetkilendirilip Yetkili kullanıcı olabilir. Ziyaretçiler sistemin online olarak herkese sunduğu ders, program, öğretim elemanı, akademik birim gibi bilgileri sistemden edinebilirler. Yetkili kullanıcılar sahip oldukları rollere göre sistemde değişik işlevlere sahiptirler. Yetkili kullanıcılar şu gibi rollere sahip olabilirler: Öğretim Elemanı (Instructor), Ders Koordinatörü (Course Coordinator), Program Direktörü (Program Director), Akademik Birim Başkanı (Academic Unit Head), Admin

Öğretim Elemanı rolüne sahip kullanıcılar dersi vermekle sorumludurlar. Bu kullanıcılar sistemdeki kendilerine Ders Koordinatörü tarafından atanan dersleri görürler. Ders Koordinatörleri dersleri koordine etmekle sorumludurlar. Bu kapsamda dersleri oluştururlar ve dersin detaylı bilgilerini diğer bir ifadeyle izlencesini (syllabus) ve kimler tarafından verileceği bilgilerini girerler. Program Direktörü olan kullanıcı program hakkındaki bilgileri girerler ve dönem dönem hangi derslerin okutulacağını gösteren müfredat tasarımını (curriculam design) yapar. Akademik Birim Başkanı ise akademik birim tarafından sunulan programları oluşturur, programlara direktör atar ve akademik biriminin öğretim elemanlarını oluşturur ve bilgilerinin yönetilmesini sağlar. Admin ise akademik birimleri oluşturur ve yönetir ve onlara akademik birim başkanı atar.

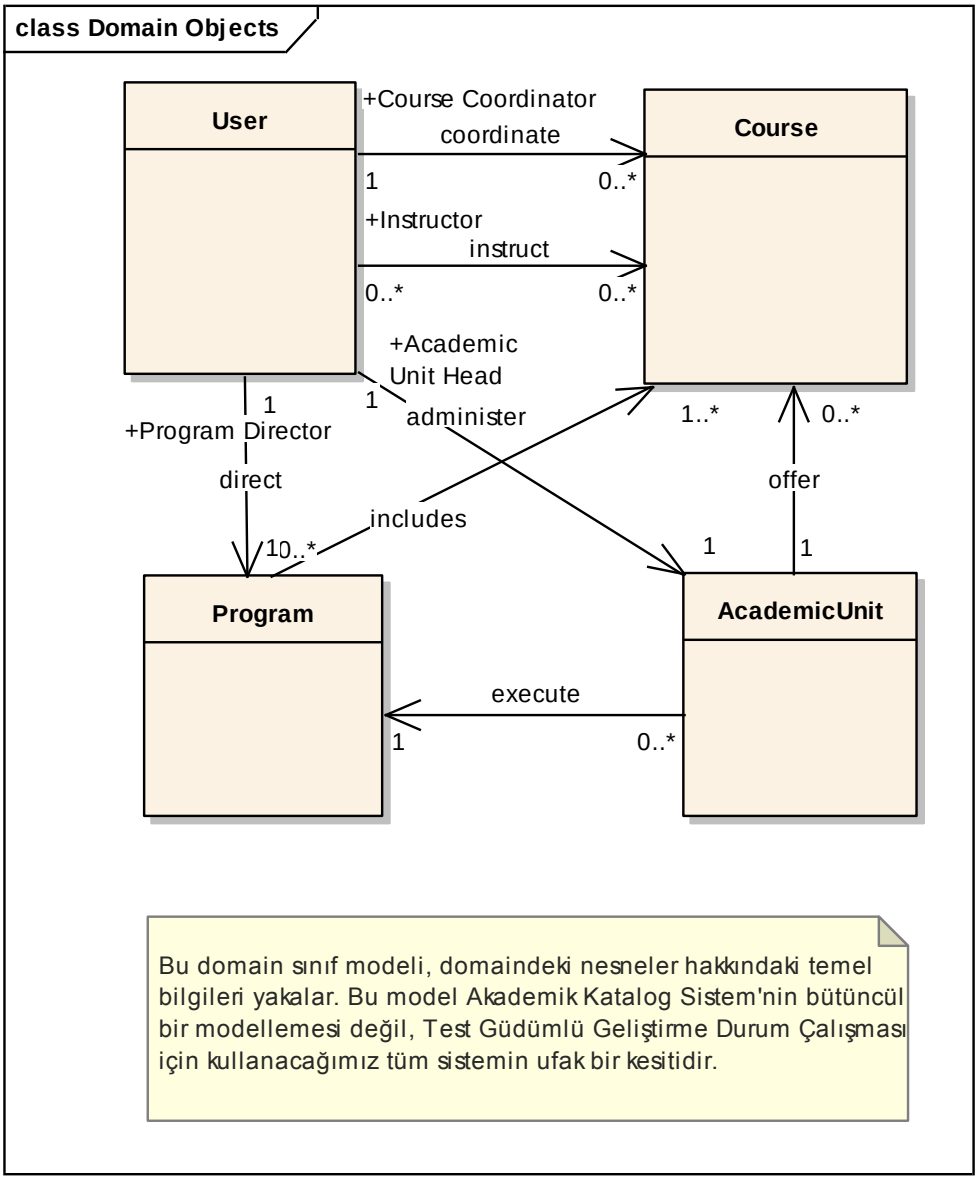
Bu kullanıcı hikâyelerinden uygulamamızda ana domain nesneleri olarak 4 temel varlık (enitity): kullanıcı (user), ders (course), program, akademik_birim (academic_unit) elde edilmektedir.

Kayıt, yetkilendirme veya ders bilgileri oluşturulması gibi sistemin diğer pek çok özelliklerinde daha fazla detaya girebilirdik fakat biz bu temel özellikler üzerinden başlamak için gereken spesifikasyonların temel hatlarını ortaya çıkardık. Gerçekleştirim (implementation) süreci boyunca daha da atomik detayları ortaya çıkaracaktır elbette fakat bu çalışmanın amacı komple bir yazılım gereksinim spesifikasyonları çıkarmak veya modelleme yapmak değil; uygulama hakkında genel bir çerçeve vererek birim ve fonksiyonel test örnekleriyle uygulumanın belli bir kesitinin gereksinim spesifikasyonlarını karşılamaktır. Böylelikle TDD'nin yazılım geliştirme sürecindeki yerini ve önemi belirten örnek bir uygulamayı göstermiş olacağız.

3.3. Veri Modelleme

Gereksinimler ve spesifikasyonlara göre analiz ve tasarımı düşünürken aynı zamanda veri modellemesi hakkında da düşünmeye ihtiyaç vardır. Daha önce ana hatları ortaya konulan kullanıcı hikâyeleri ve gereksinim spesifikasyonlarından tüm ana isimleri (main nouns) çıkartarak, *Kullanıcı, Ders, Program* ve *Akademik Birim* domain nesnelerini belirlemiştik. Bu domain nesnelerinin ilişkisel modeli Şekil-2'dedir fakat modeli basitleştirmek için öznitelikleri (attributes) belirtilmemiştir. Bu modelde kullanıcının Öğretim Elemanı, Akademik Birim Başkanı, Program Direktörü ve Ders Koordinatörü rolleri ve bu rollerle ilgili sınıfların ilişkisi gösterilmiştir. İlişkileri çokluk (multiplicity) yönünden incelersek, Öğretim Elemanı rolündeki bir kullanıcı 0 veya çok ders verebilir ve bir ders 0 veya çok öğretim üyesi tarafından verilebilinir. Program Direktörü rolündeki bir kullanıcı bir programa direktör olabilir ve bir program bir direktör tarafından yönetilir. Akademik Birim Başkanı rolündeki kullanıcı bir akademik birimi yönetebilir ve bir Akademik Birim bir Akademik Birim Başkanı tarafından yönetilir. Bir Program'da birden çok ders olurken, bir ders 0 veya çok programda yer alabilir. Bir Akademik Birim 0 veya çok Program'ı yürütürken, bir program bir Akademik Birim tarafından yürütülür. Bir Akademik Birim 0 veya çok ders sunarken, bir ders bir Akademik Birim

tarafından sunulur. Bir kullanıcı 0 veya çok ders koordine edebilirken. Bir ders 1 kullanıcı tarafından koordine edilmek zorundadır.



Şekil 2 - Sınıf Domain Nesneleri

Domain nesneleri arasındaki ilişkileri belirterek gerçekleştirime doğru ilerlerken, fikir düzeyindeki soyut gereksinim spesifikasyonlarından somut veriyi modellemeye doğru bir adım daha yaklaştık.

Uygulamamızın inşasının daha derinlerine inmeden önce, geliştirme yaklaşımımız üzerinde durmamız lazım. Biz bazı spesifik geliştirme geliştirme metodolojilerinden ve prensiplerinden faydalanacağız. Kodlamaya başlamadan önce metodoloji ve prensipler üzerinde durmak yerinde olacaktır.

3.4. Geliştirme Metodolojisi

Biz çevik metodolojiden (agile methodology) esinlenilmiş döngüsel (iterative) ve artımlı (incremental) bir geliştirme metodolojisi benimsedik. 'Çevik'(Agile) kavramı modern yazılım geliştirmede üzerine pek çok anlamlar yüklenmiş bir terimdir ve geliştiriciler arasından değişen pek

çok anlama sahip olabilir. Bizim sürecimizde çevik metodoloji şeffaf ve açık işbirliği, sabit geri-bildirim döngüleri ve değişen gereksinimlere hızlı cevap verebilme yetisini kapsamaktadır.

Kodlamaya başlamadan önce uygulama hakkında tüm detayların belirtilmesini bekleyemeyeceğimiz için biz artımsal (incrementally) olarak çalışacağız. Belli bir özelliğin detayları netleşip sonlandığında diğer özellikler veya uygulama detayları hala tasarım/planlama aşamasında olsa bile, bu özelliğin gerçekleştirimine başlayabiliriz. Bu özelliğin gerçekleştirimini sürecinde döngüsel bir model takip edeceğiz. Öncelikle döngü planlaması yapacağız ve her döngüde TDD yaklaşımı ile test ve bu testleri başarıyla geçen kodları yazarak süreci tamamlayacağız. Herkes mutlu olduğunda, yeni özelliği ile birlikte uygulamayı hizmete sokacağız ve bundan sonra diğer döngüde gerçeklenecek diğer özellik veya özelliklerin spesifikasyonlarını toplamaya başlayacağız.

3.5. Yii'de TDD

Uygulamamızda TDD sürecini gerçeklemek için yaptığımız unit testler için PHPUnit [8] ve fonksiyonel testler için Selenium Server [9] kullanılmaktadır. Selenium Server'ın kurulması ve ayarlarıyla ilgili detaylar bu makalenin kapsamı dışındadır. Yii PHPUnit ve Selenium Server test çatıları ile oldukça sıkı bir şekilde entegre durumdadır. TDD belli bir test çatısını önermez fakat bize en az bir tanesinin kullanılmasını şiddetle tavsiye eder. Yii'nin PHPUnit ve Selenium Server ile entegrasyonundan istifade etmek için bu test çatılarını tercih ettik.

3.5.1. Yii'de Birim Testler

Yii'de (Yii Dokümantasyon) bir birim testi Yii çatısının CTestCase sınıfından türetilerek (extend) elde edilen bir PHP sınıfı ile yapılır. Yii'nin isimlendirmedeki düzeni şöyledir: Abc sınıfı test ediliyorsa eğer test sınıfına AbcTest adı verilir ve AbcTest.php dosyası olarak protected/tests/unit altına kaydedilir. Test sınıfı, adı testXyz gibi olan test metodları içerir. Xyz genellikle test edilen sınıftaki test edilen metodun adıdır.

3.5.2. Yii'de Fonksiyonel Testler

Birim testler gibi fonksiyonel testler de PHP sınıfları olarak yazılırlar. Fakat, CTestCase yerine CWebTestCase sınıfından türerler (extends). İsimlendirme düzeni (conventions) birim testteki gibidir fakat AbcTest.php dosyası protected/tests/functional klasörü altında kaydedilir.

3.6. Örnekler: Birim Test

TDD yaklaşımı, örnek durum çalışması olarak ele aldığımız Akademik Katalog Sistemi uygulamasının geliştirilmesi boyunca uygulanacaktır. Bu geliştirme kapsamında yapılan bazı birim testler ve fonksiyonel testler örnek olarak gösterilecektir. Örnek olarak şu kullanıcı hikâyesin ele alıyoruz:

- "*Ders Koordinatörleri dersleri koordine etmekle sorumludurlar. Bu kapsamda dersleri oluştururlar ve dersin detaylı bilgilerini diğer bir ifadeyle izlencesini (syllabus) ve kimler tarafından verileceği bilgilerini girerler.*"

Bu hikâyeyi ele alıyoruz şimdi. Bu hikâyedeki gereksinimlerin gerçeklemesine başlamadan önce testlerimizi yazmalıyız. Bunun için ilk olarak birim testleri oluşturuyoruz. Bu gereksinim temel olarak ders modeli ile ilgili olduğundan Yii çatısının CTestCase sınıfından türeyen "CourseTest" adlı bir sınıf oluşturuyoruz ve "test/units" altında CourseTest.php olarak kaydediyoruz. Komut satırından test dizinin altına gelip phpunit unit/CourseTest.php yazıyoruz.

```
class CourseTest extends CTestCase
{
	//nothing
}
```

Ve ilk testimiz yapıyoruz ve CourseTest sınıfı bulunamadığına dair bir uyarı 1 tane başarısızlık(failure) veriyor.

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit unit/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

F

Time: 1 second, Memory: 4.75Mb

There was 1 failure:

1) Warning
No tests found in class "CourseTest".


FAILURES!
Tests: 1, Assertions: 0, Failures: 1.
```

Testi geçmek için CourseTest sınıfı içerisinde bir test metodu yazmamız lazım. Gerçeklemeye çalıştığımız kullanıcı hikâyesinden ders koordinatörünün ders oluşturmasını test ediyoruz şimdi. Bunun için öncelikle ders koordinatörü rolüne sahip bir kullanıcının ait olduğu akademik birim'in sunuyor olacağı bir ders eklemesi senaryosunu test etmeliyiz. Bunun için adım adım önce başarısız bir test, sonra testi başarıyla geçmek için gereken kodları yazacağız. Öncelikle, testCRUD adlı bir test metodu yazıyoruz ve test metodumuzun çalışıp çalışmadığını anlamak için şunu yazıyoruz:

```
public function testCRUD()  {

	$this->assertTrue(true);

}
```

Ve test metodumuzun çalışıp çalışmadığını anlamak için "phpunit unit/CourseTest.php" yazarak test ediyoruz ve aşağıdaki sonucu alıyoruz.

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit unit/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 4.75Mb

OK (1 test, 1 assertion)
```

Uygulamamızda dersi açmak demek Course tablosunda yeni bir kayıt girmek demek. Bunun için öncelikle ders oluşturmayı incelememiz lazım. Bunun için Course.php dosyasını oluşturup, ActiveRecord [10] sınıfından türeyen Course sınıfını oluşturuyoruz. ActiveRecord sınıfı CRUD işlemler için Yii'nin kullandığı verimli defacto bir yapısal paterndir. Course sınıfı veritabanındaki course tablosu ile eşleşmektedir [11]. Bu model sınıf ile tüm veritabanı işlemlerini pratik bir şekilde yönetebiliriz. Şimdi yeni bir Course nesnesi oluşturup veritabanına kaydedelim.

```
…
$newCourse=new Course();
$this->assertTrue($newCourse->save());
```

…

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit unit/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

F

Time: 1 second, Memory: 6.75Mb

There was 1 failure:

1) CourseTest::testCRUD
Failed asserting that <boolean:false> is true.

C:\xampp\htdocs\ais_academitv\protected\tests\unit\CourseTest.php:28

FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
```

Testimiz başarısız oldu. Model nesnesini save() metodu ile veritabanına kaydetmesinde bir problem oldu; çünkü Course model sınıfında ve bununla eşleşen veritabanındaki course tablosunda boş olarak kayıt yapılamayacak kısıtlar var (Foreign Key Constraints). Bunun için şu kodu yazıyoruz:

```
//In order to simplify testing process, ASSUME THAT user id is equal to 1
//and username is equal to 'ayamuc' who is coordinator of the course which will be newly created.
//And 'ayamuc' is a member of Computer Engineering Department
$course_coordinator_id     = 1;
$course_coordinator_name    = 'ayamuc';
$course_title            = 'Test Driven Development';
$course_code             = 'SE 404';

$courseCoordinator=User::model()->findByPk( (int) $course_coordinator_id );
$this->assertEquals($courseCoordinator->username,$course_coordinator_name);

$coordinator_acdmc_unit_id =  $courseCoordinator->academic_unit_id;

//CREATE the course
$newCourse=new Course();
$newCourse->setAttributes(array(
  'course_coordinator_id'  => $course_coordinator_id,
   'title'              => $course_title,
   'code'               => $course_code,
   'owner_academic_unit_id'=> $coordinator_acdmc_unit_id,
));
$this->assertTrue($newCourse->save());
```

Bu kod ile ders eklemek için bir ders koordinatörü rolüne sahip bir kullanıcı olmalı ve kullanıcının hangi akademik birimde görev yaptığı bilinmelidir. Bu varsayım ve bu bilgi ile testi sadece ders varlığına odaklıyoruz, basitleştiriyoruz problemi. Şimdi tekrar test yapıyoruz.

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit unit/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

.

Time: 1 second, Memory: 7.00Mb

OK (1 test, 2 assertions)
```

Bu aşamada pek çok defa başarısız test yazılıp sonra testi başarıyla geçmek için gereken kodlar yazılabilinir. Fakat biz genel resmi vermek için başarı durumunu veriyoruz şimdi. Bu noktaya gelince

kadar User ve AcademicUnit model sınıfları oluşturulup, bunların her birinin kapsamlı testleri yapılmış olması lazım tabii. Biz şimdilik problemi daraltıyoruz. Şimdi de ders varlığının temel CRUD işlemlerinden READ, UPDATE ve DELETE işlemlerini test edelim.

```
…
//READ the course
$retreivedCourse=Course::model()->findByPk($newCourse->id);
$this->assertTrue($retreivedCourse instanceof Course);
$this->assertEquals($course_title,$retreivedCourse->title);
$this->assertEquals($course_code,$retreivedCourse->code);
$this->assertEquals($coordinator_acdmc_unit_id,$retreivedCourse->owner_academic_unit_id);
…
```

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit unit/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

.

Time: 0 seconds, Memory: 7.00Mb

OK (1 test, 6 assertions)
```

Test 6 tane toplam sorguyu da başarıyla tamamlayarak pozitif bir şekilde sonlandı. Veritabanına kaydedilen ders nesnesinin title, code, academic_unit alanlarının doğru bir şekilde kaydedilip kaydedilmediğini anlamak için veritabanından findByPk metodu ile ders nesnemizi alıp teker teker sorguluyoruz. Şimdi sınıf nesnemiz üzerinde güncelleme(update) işleme yapalım.

```
//UPDATE the newly created project
$course_title_updated      = 'Test Driven Development on Agile Web Development';
$newCourse->title =  $course_title_updated;
$this->assertTrue($newCourse->save(false));
$retreivedCourse=Course::model()->findByPk($newCourse->id);
$this->assertNotEquals($course_title_updated ,$course_title);
$this->assertEquals($course_title_updated ,$retreivedCourse->title);
```

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit unit/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

.

Time: 1 second, Memory: 7.00Mb

OK (1 test, 9 assertions)
```

Yeni sorgulamalarla birlikte testimiz başarılı oldu. Başlığı “Test Driven Development” olan dersin başlığını “Test Driven Development on Agile Web Development “olarak değiştirdik ve doğru olup olmadığını test ettik. Şimdi de ders silme işlemini test edelim.

```
//DELETE the course
$newCourseId =  $newCourse->id;
$this->assertTrue($newCourse->delete());
$deletedCourse=Course::model()->findByPk($newCourseId);
$this->assertEquals(NULL,$deletedCourse);
```

Ders nesnemiz silindi ve buna bağlı olarak "course" tablosundaki ilgili kayıt da silinmiş oldu. Bunu test etmek için de nesnemizin NULL olup olmadığını sorguladık ve testimiz başarıyla sonlandı.

Bu test işlemleri ile ders nesnemizin ilgili koordinatör kullanıcı tarafından gereken alanları doldurularak oluşturulması, veritabanı'ndan geri çekilmesi, güncellenmesi ve silinmesi işlemlerini test etmiş oldu. Bu tüm işlemleri CourseTest sınıfı içerisindeki testCRUD metodu içerisinde gerçekleştirdik.

Şimdi çıkış noktamız kullanıcı hikâyesine dönersek, tüm bu testleri başarıyla geçmek için gereken tüm sınıfları ve yöntemleri gerçeklemek gerekmekteydi. Test-dayalı düşünerek kodlama mantığımızı kökten değiştirmiş oluyoruz aslında. Önce testi düşünüp onu yazıyoruz sonra gereken kodlamayı yapıyoruz.

3.7. Örnekler: Fonksiyonel Test

Yii'de fonksiyonel test yapmak için tests/functional dizini altına CourseTest.php adı altında bir dosya açıyoruz ve içerisine CourseTest adından CWebTestCase sınıfından türeyen bir sınıf tanımlıyoruz.

- "*Ders Koordinatörleri dersleri koordine etmekle sorumludurlar. Bu kapsamda dersleri oluştururlar ve dersin detaylı bilgilerini diğer bir ifadeyle izlencesini (syllabus) ve kimler tarafından verileceği bilgilerini girerler.*"

Kullanıcı hikâyesine dönersek. Yaptığımız birim testler daha da çoğaltılabilir de ama biz örnek olsun diye temel CRUD işlemlerini test ettik. Birim testler sadece geliştirici tarafında yapılırken, fonksiyonel testler hem geliştirici hem de paydaşlar tarafında yapılabilinir. Seleninum Server herhangi bir tarayıcıyı kullanarak testi görsel olarak da gösterebilir. Fonksiyonel testler verilen kullanıcı hikâyesinin ve bununla ilgili gereksinim spesifikasyonlarının ne ölçüde karşılanıp karşılanmadığını tespit etmek için çok verimli, otomatize bir yöntemdir. Şimdi fonksiyonel testi kodlayalım.

```
class CourseTest extends CWebTestCase
{
	public function testIndex()
	{
		$this->open('');
		$this->waitForPageToLoad("30000");
		 $this->assertTextPresent('Academic');

	}
}
```

CWebTestCase setUp fonksiyonunun içerisinde gereken konfigurasyoları yapıyoruz. Bu örneğimizde tarayıcı olarak Firefox kullandık ve başlangıç url'ini belirttik. Gözle de takip edebilmek için her aksiyon arasında 1 sn gecikme için setUp fonksiyonunda konfigürasyon yaptık. Kodu basitleştirmek için bu yukardaki kodumuzda belirtmedik.

Fonksiyonel testimizde ilk test metodumuz testIndex. Ana sayfamızı açıp 3 sn bekleyip Academic kelimesinin olup olmadığını sorguluyoruz. Şimdi phpunit functional/CourseTest.php yazarak konsol testimizi başlatalım.

```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit functional/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

.

Time: 13 seconds, Memory: 5.00Mb

OK (1 test, 1 assertion)
```

Testimiz başarıyla sonlandı. Aynı zamanda açılan tarayıcıda da sonucu gördük. Şimdi testimizi daha da karmaşıklaştıralım. Kullanıcımız sisteme giriş yapsın. Akabinde görev yaptığı bölümün sayfasına girsin ve bir ders oluştursun. Gereken temel bilgileri girsin. Sonra kaydetsin ve bölümün sunduğu derslerden kontrol etsin dersi var olup olmadığını. Ders bilgilerini tekrar görsün. Daha sonra güncelleme linkine tıklasın. Dersi verenler alanına bazı hocalar eklesin ve kaydetsin. Sonra kaydımızı silsin ve silmek için gereken doğrulama mesajının doğru bir şekilde gelip gelmediğini sorgulasın. Son olarak sistemden çıkış yapsın. Şimdi bu testi yapmak için gereken kodu yazalım.

```
public function testCourseCreate()
{
	$this->open('');
	 $this->waitForPageToLoad("3000");
	// ensure the user is logged out
	if($this->isTextPresent('Logout'))
		$this->clickAndWait('link=Logout');

	//test login process, including validation
	$this->clickAndWait('link=Login');
	$this->assertElementPresent('name=LoginForm[username]');
	$this->type('name=LoginForm[username]','ayamuc');
	$this->clickAndWait("//input[@value='Login']");
	$this->assertTextPresent('Password cannot be blank.');
	$this->type('name=LoginForm[username]','ayamuc');
	 $this->type('name=LoginForm[password]','1');
	$this->clickAndWait("//input[@value='Login']");


	$this->open('?r=academicunit/view&id=3');
	$this->waitForPageToLoad("3000");
	$this->assertTextPresent('Computer Engineering');
	$this->clickAndWait('link=Create Course');


	$this->type('name=Course[title]','Test Driven Development');
	$this->type('name=Course[code]','SE 444');
	$this->clickAndWait("//input[@value='Create']");
	$this->assertTextPresent('Test Driven Development');

	 $this->open('?r=course/courselist&aid=3');
	$this->assertTextPresent('SE 444');
	$this->assertTextPresent('Test Driven Development');

	$this->clickAndWait('link=SE 444');
	$this->assertTextPresent('SE 444');
	$this->assertTextPresent('Test Driven Development');

	 $this->clickAndWait('link=Update Course');
	 $this->assertElementPresent('name=Course[title]');
	$this->assertElementPresent('name=Course[code]');

	 $this->type('name=Course[instructor]','Semih Dinç, İsmet YiğitBaşı');

	 $this->clickAndWait("//input[@value='Save']");
	 $this->assertTextPresent('Semih Dinç, İsmet YiğitBaşı');
```

```
        $this->clickAndWait('link=Delete Course');
        $this->assertEquals('Are you sure you want to delete this item?',$this->getConfirmation());

        $this->assertTextNotPresent('Login');
        $this->clickAndWait('link=Logout (ayamuc)');
        $this->assertTextPresent('Login');
}
```

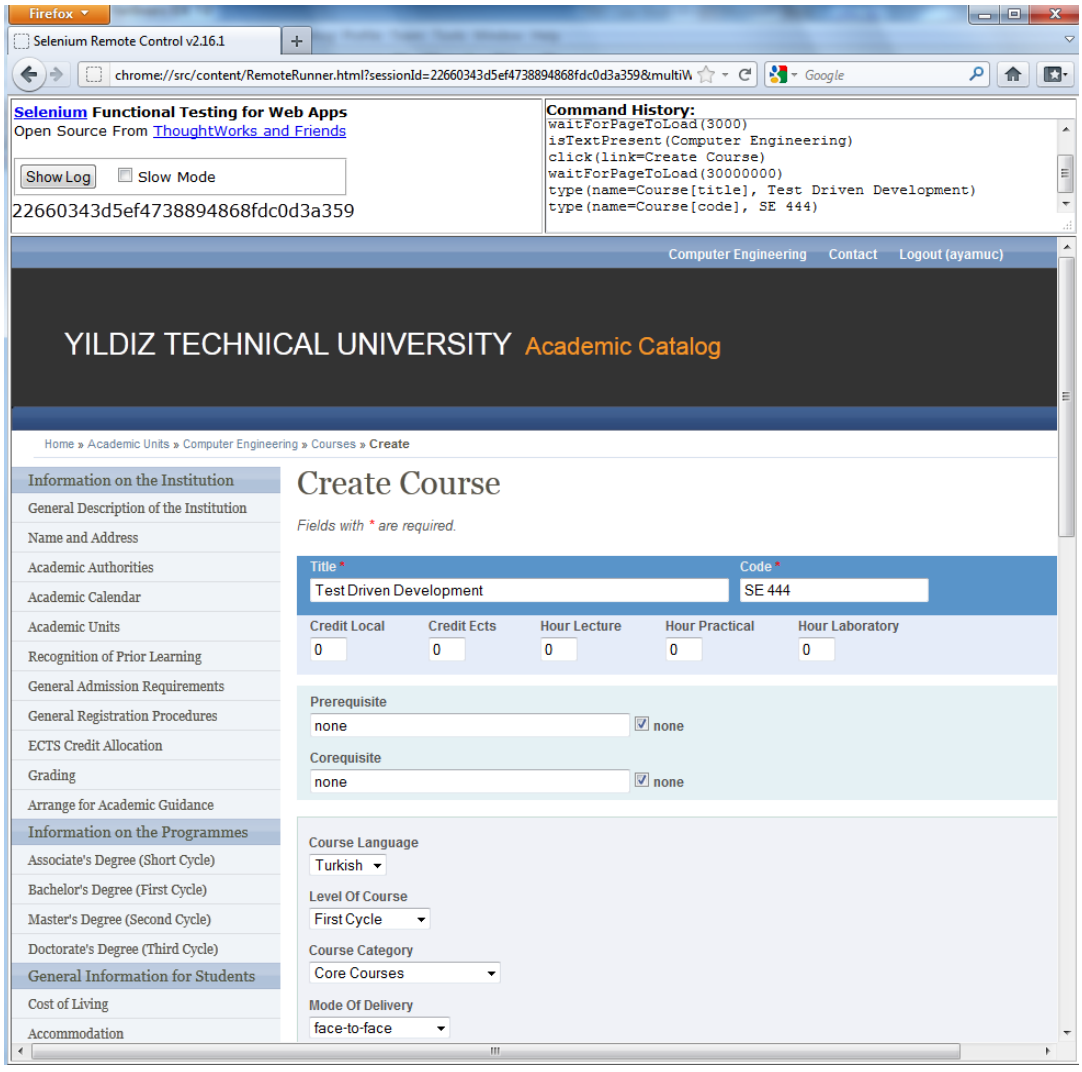
```
C:\xampp\htdocs\ais_academitv\protected\tests>phpunit functional/CourseTest.php
PHPUnit 3.5.10 by Sebastian Bergmann.

.

Time: 10 seconds, Memory: 5.25Mb

OK (1 test, 14 assertions)
```

Testimiz 14 tane sorgulamayla birlikte başarıyla sonlandı. Böylelikle kullanıcı hikâyelerinin ne ölçüde karşılanıp karşılanmadığını, artımsal olarak süren geliştirmelerde bu otomatik testi her seferinde çalıştırarak tespit ederiz ve böylelikle sistemin kararlılığı ve bütünlüğü sağlamış oluruz.



Şekil 3 - Fonsiyenl Test sırasında ders bilgilerini girme ve sonra kayıtın başarıyla yapılıp yapılmadığına dair sorgulamaları gösteren bir ekran resmi.

4. Sonuç

Bu çalışmamızda Yii çatısı kullanarak geliştirmiş olduğumuz web-tabanlı Akademik Katalog Sistemi uygulamasında uyguladığımız metodolojik yaklaşımları ele aldık. Bu kapsamda Test-Güdümlü Geliştirme'yi artımsal çevik metodoloji ile birlikte kullanarak yaptığımız testler ve sonuçlarını gösterdik. Böylelikle özellikle web-tabanlı uygulamalarda ihmal edilen bu yaklaşım tarzları ile güvenli, daha az hatalı ve daha başarılı uygulamalar geliştirilebileceğiniz gözlemledik. Bu yaklaşım ile bir geliştirici var olan kodda geliştirmeler yaptığında ve akabinde bir takım testleri çalıştırdığında uygulama davranışının değişip değişmediğini hemen doğrulayabilir. Bu kapsamda uygulanan fonksiyonel ve birim testler ile sağlanan bu güven ile geliştirici daha iyi, daha kararlı uygulamalar geliştirebilir. Sonuç olarak, bu metot ile özellikle karmaşık web-uygulamalarında, paydaşlarda ve geliştiricilerde sağlanan güven duygusu ile oldukça kaliteli uygulamalar geliştirilebileceğini gözlemledik.

Kaynaklar

[1]. Introduction to Test Driven Development (TDD) www.agiledata.org: Techniques for Successful Evolutionary/Agile Database Development.

[2]. Beck K., Test Driven Development by Kent Beck. Addison Wesley. ISBN: 978-0321146533

[3]. Wake W.C., Extreme Programming Explored, 2000.

[4]. Feathers M., Working Effectively with Legacy Code by Michael Feathers. Prentice Hall. ISBN: 9780131177055

[5]. Grenyer P., An Introduction To Test Driven Development, June 2011

[6]. Agile Modeling Web Sitesi, http://www.agilemodeling.com/artifacts/userStory.htm

[7]. ECTS Resmi Web Sitesi, http://ec.europa.eu/education/lifelong-learning-policy/doc48_en.htm

[8] PHPUnit, http://www.phpunit.de/manual/current/en/index.html

[9] Selenium Server, http://seleniumhq.org/projects/remote-control/

[10] Yii Active Record, http://www.yiiframework.com/doc/guide/1.1/en/database.ar

[11] Yii ORM, http://www.yiiframework.com/doc/guide/1.1/en/database.overview