Test Yönelimli Yazılım Geliştirme Metodlarının J2EE Platformu ve Bileşen Modellerine Uygulanması Üzerine Bir Çalışma

Fatih Algan[1], Tuğkan Tuğlular[2], Oğuz Dikenelli[3]

[1,2] İzmir Yüksek Teknoloji Enstitüsü, Bilgisayar Mühendisliği Bölümü, 35430, Urla, İzmir
[1] fatihalgan@iyte.edu.tr, [2] tugkantuglular@iyte.edu.tr
[3] Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova, İzmir
[3] oguzd@staff.ege.edu.tr

Özet. E*xtreme Programming* temelde yoğun iletişim ve basitlik ilkelerine dayanan, dinamik ve hızlı işleyen bir yazılım geliştirme metodolojisi olarak yorumlanmaktadır. Otomatik test, sürekli entegrasyon, program kodlarının ve tasarımın devamlı olarak yeniden ele alınıp iyileştirilmesi ise bu metodolojinin temel pratiklerindendir. Otomatize edilmiş testler ise bu süreçlerin önemli bir parçasıdır. Bu bildiride, bileşen modellerinin (EJB) kullanıldığı bir J2EE platformu yazılımında bileşen testinin ve fonksiyonel testlerin gerçekleştirilme yöntemleri ele alınmaktadır. Bildiride anılan çalışma, uygulama sunucusu üzerinde çalışan, oldukça karmaşık orta katman bileşenleri ile servisleri içeren ve uç tarafta da web teknolojilerini kullanan çok katmanlı bir yazılım mimarisi üzerinde geliştirilmiştir.

1 Giriş

Extreme Programming, testleri yazılım geliştirme etkinliklerinin merkezi olarak kabul eder. Yazılımı sürekli olarak test ederek onun hatasız çalıştığını ve gereksinimleri karşıladığını ispatlar. Otomatize edilmiş testler, olası test senaryolarının bir program haline getirilmesinden oluşur. Bu testler biriktirilir ve istenildiği anda çalıştırılarak çok sayıda olası senaryonun bir defada işletilerek sonuçların çok kısa bir sürede alınmasını sağlar. Otomatik test, testlerin sürekli olarak gerçekleştirilmesinin tek yoludur. *Extreme Programming*, yazılımın sürekli olarak yeniden ele alınıp iyileştirilmesi ilkesine dayanır. Dolayısıyla yeni geliştirilen ya da değiştirilen modüllerin doğru olarak çalıştığından emin olmak ve bunların yazılımın diğer modülleriyle entegrasyonu sonrasında ortaya çıkacak hataları yakalayabilmek için otomatize edilmiş testler zorunlu hale gelmektedir.

Otomatik testler; ünite testleri, entegrasyon testleri, fonksiyonel testler ve performans testleri olmak üzere dört grupta toplanabilir. Bu bildiriye konu olan çalışmada sadece entegrasyon testleri ve fonksiyonel testler kullanılmıştır. Entegrasyon testleri bir tek sınıfı veya küçük bir modülü değil de nesneler arasındaki etkileşimi sistem düzeyinde test edebilmeyi sağlar. Fonksiyonel testler kara kutu testleri olarak da adlandırılır. Bileşen testlerine oranla daha yüksek düzeyli testlerdir. Genellikle bir sistem senaryosunun tümünü veya bir parçasını test etmek için kullanılırlar.

Bu bildiride *J2EE* platformunda entegrasyon testleri ile fonksiyonel testlerin gerçekleştirilmesi üzerine örnek bir çalışma tanıtılmakta ve bu çalışma sırasında yaşanılan deneyimler açıklanmaktadır.

2 Test Ortamı ve Kullanılan Araçlar

Entegrasyon testleri ile fonksiyonel testlerin herbiri ile ilgili uygulamaların geliştirilebilmesi için özel araçlar bulunmaktadır. Ayrıca veritabanının testler için tasarlanmış örnek veri ile doldurulması, uygulamanın derlenmesi ve uygulama sunucusu üzerine kurulması ile testlerin başlatılması sürecinin tamamını otomatize edebilen çok sayıda araç bulunmaktadır. Bu bölümde yapılan çalışmada kullanılan araçlar; *Ant*, *Cactus*, *HttpUnit* tanıtılacaktır. Bu araçlar açık kaynak kodlu olup *GPL* lisansı [1] ile dağıtılmaktadır.

2.1 Ant

Ant [2] uygulamanın otomatik olarak derlenmesi ve uygulama sunucusu üzerine kurulumunun yapılabilmesini sağlayan bir araçtır. Ayrıca veritabanının örnek verilerle ilklenmesi işlemini de otomatize edebilir. *Ant* scriptleri XML formatındadır ve platform bağımsız olarak çalıştırılabilir. *Ant* scriptleri sayesinde uygulamaların değişik platformlarda değişik uygulama sunucuları üzerine otomatik kurulumu sağlanabilir.

<target></target> etiketleri, işletilebilir script modüllerinin sınırlarını göstermektedir. En son modül ise, varsayılan olarak işletilen ana modüldür ve **“createTables”** ile **“populateTables”** adlı modülleri sırayla çağırmaktadır. Bu işlem sonunda veritabanına test verileri aktarılmış olur. Testin bitişinden sonra ise, **“dropTables”** adlı modülü yine komut satırından işleterek veritabanı testten önceki haline getirilebilir.

2.2 Cactus

Cactus [3] entegrasyon testlerinin gerçekleştirilebilmesi için geliştirilmiş bir araçtır. *Cactus* testleri tasarım gereği, istemci ve sunucu olmak üzere iki parçadır. Bir yönlendirici aracılığı ile *Cactus*, istemciden gelen isteği sunucu içindeki test projesine aktarıp taşıyıcı ile konuşmaya başlayacaktır. İki ayrı projeye ihtiyaç varmış gibi görünse de istemci ve sunucu projeleri birbirinin tamamen aynısıdır ve aynı testleri içerirler. *Cactus* testlerinin sunucu bölümü, '*Enterprise Application (EAR)*' üzerine ayrı bir '*Web Application (WAR)*' modülü olarak eklenebilir. Testlerin istemci bölümü ise web sunucusunda bulunan yönlendiriciye (redirector) istekler gönderen bağımsız bir java uygulaması olacaktır. Konfigürasyon için izlenmesi gereken adımlar *Wosnick* tarafından açıklanmıştır [4].

2.3 HttpUnit

HttpUnit [5] fonksiyonel testlerin gerçekleştirilebilmesi için kullanılan bir araçtır. *HttpUnit*'te *Cactus*'un aksine test sınıfları taşıyıcıların içinden çalıştırılmak zorunda değildir. Web sunucusu dışarıdan sorgulanarak cevaplar alınır. Fakat isteklerin web sunucusuna gönderilmesi, kullanıcı oturumunun takibi, sunucudan dönen cevabın incelenebilmesi gibi gereksinimler için *HttpUnit* de bazı özel taşıyıcı nesnelerini taklit eden sınıflar sağlar.

HttpUnit web sunucusuna tamamen dışarıdan ve HTTP protokolü üzerinden istek gönderip cevapları ise HTTP protokolü üzerinden düz metin olarak aldığı için konfigürasyonu *Cactus*'e göre oldukça kolaydır. Yapılması gereken, bir java programı yazarak web sunucusunun URL adresine HTTP üzerinden istekler göndermektir. Konfigürasyon için yapılması gereken ise, yaratılan bir java projesinin classpath'ine *HttpUnit* ile gelen şu dosyaları eklemektir: js.jar, junit.jar, servlet.jar, Tidy.jar, httpunit.jar, xmlParserAPIs.jar.

3 Projenin Tanıtımı

Geliştirilmekte olan Öğrenci İşleri Bilgi Sistemi Projesi'nin amacı; öğrencilerin, öğretim üyelerinin ve ilgili idari birimlerin öğrenci ile ilgili bütün işlemleri İnternet üzerinden yürütebileceği bir sistem geliştirmektir. Uygulamanın üzerinde çalışacağı platform olarak IBM'in J2EE çözümü olan *WebSphere* ailesi seçilmiştir. *WebSphere* uygulama sunucusu olarak güçlülüğünü ve tutarlılığını kanıtlamış bir ürün olmasının yanında tüm tasarım, geliştirme ve test sürecini tek bir IDE içerisinden gerçekleştirebilme olanağı sunmaktadır. Halen devam etmekte olan projenin analizi, tasarımı ve geliştirilmesi süresince karşılaşılan çeşitli sorunlar aşamalı olarak *Extreme Programming* yöntemlerine yakınlaşılmasına yol açmıştır.

Öğrenci İşleri Bilgi Sistemi başlangıçta sadece Web tabanlı istemciler için hizmet verecek şekilde planlanmıştır. Projenin web katmanında J2EE uygulamalarında yaygın olarak kullanılan bir *Model-View-Controller* (MVC) çerçevesi olan *Struts* [6] kullanılmıştır. *Struts*, tasarım olarak uygulama mimarisine getirdiği disiplinin yanında sağladığı çok sayıda JSP Tag kütüphaneleriyle JSP sayfalarının yazılmasını ve sunumun oluşturulmasını kolaylaştırmaktadır.

Yazılım orta katmanında EJB teknolojisi kullanılmaktadır. J2EE ve EJB teknolojileri bazı tasarım kalıpları ve standartlaşmış yöntemler kullanılmadığı sürece performans ve yönetilebilirlik açısından ciddi sorunlar yaratabilir. *Struts*, programcıyı web katmanında bu kalıpların çoğunu kullanmaya yönlendirse de, EJB teknolojisi için böyle bir durum söz konusu değildir. Uygulamanın orta katmanında *Business Delegate, Session Facade, Data Transfer Object, Service Locator, Data Transfer Object Factory, Data Access Object* gibi standartlaşmış EJB tasarım kalıpları kullanılmıştır. Bunun dışında yoklama (audit trail) işlemleri için geliştirilmiş başka bir çerçeve olan Log4Java [7] da yazılım orta katmanına entegre edilmiştir.

Bunların yanında WebSphere platformunun EJB teknolojisine getirdiği bir ek olan EJB inheritance modeli, polimorfizmi bileşen modelleri üzerinde uygulayabilme ve daha esnek bir nesne modeli oluşturabilmesine olanak sağlamıştır.

4 Test Yönelimli Yazılım Geliştirme Kapsamında Ele Alınan Senaryo ve Gerçekleştirimi

Test yönelimli yazılım geliştirmeye yönelik araçların çalışmasını izleyebilmek için uygulamanın basit ve küçük bir modülü seçildi ve bu modül için test senaryoları oluşturuldu. Bu modül üç adet EJB'den oluşmaktaydı; fakülteler, bir fakülteye bağlı bölümler ve bir bölüme bağlı programlar. Bileşen testi için ilk senaryo bir fakültenin yaratılması, daha sonra yaratılan fakülteye ait bilgilerin okunması ve yaratılan fakültenin silinmesi idi. Testlerin tamamlanmasından sonra ise veritabanında kalan verileri bir *Ant* scripti yardımıyla temizleme yolu seçildi.

Cactus testleri basit java sınıflarından oluşur. Test sınıfımızın uygulama sunucusunun içerisinde bir Servlet gibi davranmasını sağlamak için *Cactus* API'sinin sağladığı *ServletTestCase* sınıfını extend etmesi gerekir. Testimizde kullandığımız diğer bileşen ve sınıflar ise şunlardır:

- **BirimlerController:** Test edilecek EJB Session Facade[8]'inin remote interface'i..
- **BirimlerControllerHome:** Test edilecek EJB Session Facade'inin home interface'i..
- **TestUtil:** EJB Session Facade'inin Home interface'ini JNDI namespace'den alıp döndüren bir singleton sınıf. (*Service Locator* [8]).
- **FakulteDTO:** Fakülteyi temsil eden bir DTO nesnesi (*ValueObject ya da Data Transfer Object*[8]).

Cactus, bir ünite testi aracı olan *JUnit* API'lerini kullanır. Bir JUnit testlerinin açıklanmasından önce aşağıdaki kavramların anlaşılması gerekmektedir [9]:

- **TestCase:** Birbirleriyle ilintili bir küme testin işletilmesi için bir ***fixture*** tanımlar.
- **Test Fixture:** Çeşitli kaynaklar (primitif değişkenler, veritabanı bağlantılatrı, IO Stream'leri, Singleton nesneler, testlerin çalışabilmesi için gerekli çeşitli başka nesneler, vb.) sağlar.
- **Test Suite:** Birbirleriyle ilgili bir küme TestCase'i bir TestSuite'ini oluşturur.

Bir TestCase'ini yazmak için şunlar yapılmalıdır:

1. **junit.framework.TestCase** sınıfını extend edin.
2. Eğer bazı ***fixture*** nesneleri gerekiyorsa **setup()** metodunu override edin.
3. Test metodlarını yazın. Test metodları **testXXX()** biçiminde tanımlanmalı ve geriye bir değer döndürmemelidir.
4. Eğer **fixture** nin parçası olan çeşitli kaynaklar serbest bırakılmak isteniyorsa **tearDown()** metodunu override edin.
5. Değişik **TestCase**'leri gruplayıp bir arada çalıştırmak istiyorsanız testlerden oluşan bir **TestSuite** i tanımlayın.

```
public class FakulteEJBTestCase extends ServletTestCase {
   //EJB remote interface'lerinin referansları
   private BirimlerController birimlerController;
   //Constructor…Junit için test case ismini tanımla
   public FakulteEJBTestCase(String theName) {
      super(theName);   }
   //JUnit TestRunner kullanarak testlere başla
   public static void main(String[] args) {
      junit.swingui.TestRunner.main(new String[]
      { FakulteEJBTestCase.class.getName()});   }
   //testXXX şeklinde başlayan bir TestSuite döndür
   public static Test suite() {
      return new TestSuite(FakulteEJBTestCase.class); }
   //InitialContext & JNDInamespace hazır, HomeInterface al
   public void setUp() {
      try {
      this.birimlerController = TestUtil.
      getBirimlerControllerHome().create();
      } catch (Exception e) { e.printStackTrace();    }}
   //JUnit testleri aşağıdan yukarıya doğru çalıştırır.
   //Bu yüzden test metodları sondan başa doğru yazıldı.
   public void removeChanges() throws Exception {
      testRemoveFakulte();    }
   //Bölümleri test edebilmek için yeniden fakülteyi yarat
   public void testSetForNextComponent() throws Exception {
      testCreateFakulte();    }
   //Yarattığın fakülteyi sil
   public void testRemoveFakulte() throws Exception {
      birimlerController.deleteFakulte((short)1);}
   //Bütün fakültelerin listesini al
   public void testGetAllFakulteler() throws Exception {
      Collection fakulteler =
```

```
      birimlerController.getAllFakulteler();
      assertFalse("Fakulteler_Size",fakulteler.size()==0);}
   //Yarattığın fakülteyi sorgula ve kontrol et.
   public void testGetFakulte1() throws Exception {
         FakulteDTO dto =  birimlerController.getFakulte((short)1);
      assertNotNull(dto);
         assertEquals("Fakulte_Kod: ", "01", dto.getFakulteKod());
      assertEquals("Fakulte_Ad: ", "FEN FAKÜLTESİ",
      dto.getFakulteAd().trim());
         assertEquals("Ing_Fakulte_Ad: ", "FACULTY OF SCIENCE",
      dto.getIngFakulteAd()); }
   //Önce Bir Fakülte Yarat
   public void testCreateFakulte() throws Exception {
         FakulteDTO dto = new FakulteDTO("01", "FEN FAKÜLTESİ", "FACULTY OF
      SCIENCE");
      birimlerController.createFakulte(dto);   }
}
```

Fakülteler, bölümler ve programlarla ilgili bileşen testlerinin yazılmasından sonra aynı modül için fonksiyonel testler yazıldı. Fonksiyonel testlerde, kullanıcının menü seçimleri de dahil olmak üzere bilgi sistemi ile olan tüm etkileşimini test eden senaryolar düşünüldü. Aşağıda bir fakültenin seçilerek bilgilerinin gösterilmesi senaryosunu test eden bir test sınıfı örnek olarak verilmiştir. İçeriği basit tutmak amacıyla örnekte kullanıcının hatalı veri girişi yapmadığı varsayılmıştır.

Fonksiyonel testlerin yazılmasında kullanılan HttpUnit temel olarak 3 adet sınıf sağlar. **WebConversation** sınıfı bir web browser'ını taklit eder ve istemcinin oturumunu yönetir. **WebRequest** sınıfı browserdan yapılan bir Http request'ini, **WebResponse** sınıfı ise bir Http response'ını taklit eder.

```
public class FakulteGosterTest extends TestCase {
   //Constructor for FakulteGosterTest
   public FakulteGosterTest(String arg0) {
      super(arg0);      }
   //Bir Test Suit'i yarat.
   public static Test suite() {
      return new TestSuite(FakulteGosterTest.class); }
   //Test senaryosunu başlat
   public static void main(String[] args) {
      junit.textui.TestRunner.run(suite());    }
   //Senaryo başlangıcı. Ana menüden bir seçim yapılır.
   public void testFakulteGoster() {
      try {
         System.out.println("Starting to test StartPage:");
         WebConversation wc = new WebConversation();
         WebResponse resp = wc.getResponse(
         "http://localhost:8080/OgrenciIsleriWeb");
         WebLink link = resp.getLinkWith("Birimler");
         System.out.println("Forwarding to: " +
         link.getURLString());
         sideMenuSelect(link);
      } catch(Exception e) {
         System.out.println(e.getMessage());   }     }
   // Detaylı menüden Fakülte ile ilgili işlemler seçilir.
   // Cevap olarak tüm fakültelerin bir listesi döner.
   private void sideMenuSelect(WebLink link) {
      try {
         System.out.println("Starting to test SideMenu selection:");
      WebConversation wc = new WebConversation();
      WebResponse resp = wc.getResponse(link.getRequest());
      link = resp.getLinkWith("Fakülte İşlemleri");
      secGoster(link);
         System.out.println("Forwardingto: " +
      link.getURLString());
```

```
   } catch(Exception e) {
      System.out.println(e.getMessage());   }     }
//Fakülte listesinden ilk fakültenin Göster linki //seçilir.
private void secGoster(WebLink link) {
   try {
   System.out.println("Starting Listele Action");
   WebConversation wc = new WebConversation();
   WebResponse resp = wc.getResponse(link.getRequest());
   link = resp.getLinkWith("Göster");
   showResult(link);
      System.out.println("Forwarding to: ” + link.getURLString());
   } catch(Throwable t) {
      System.out.println(t.getMessage());   }     }
//Seçilen fakülte bilgileri ile sonuç sayfa yazdırılır.
private void showResult(WebLink link) {
   try {
   System.out.println("Starting to render result page");
   WebConversation wc = new WebConversation();
   WebResponse resp = wc.getResponse(link.getRequest());
   System.out.println(resp.getText());
   } catch(Throwable t) {
   System.out.println(t.getMessage());}     }}
```

Bileşen testlerini çalıştırmak için öncelikle proje uygulama sunucusu üzerine kurulmalıdır. Daha sonra ise istemci *Cactus* modülünde (java projesi) ilgili test uygulamasını ya da bütün testleri birden çalıştıran uygulamayı başlatıp testlerin sonuçları alınabilir. Fonksiyonel testleri çalıştırmak için yazılan test programlarını (java programlarını) başlatmak yeterlidir[10].

5 Sonuç

Üzerinde çalışmakta olduğumuz proje bize otomatize edilmiş testlerin özellikle çok katmanlı mimarilerde ve web tabanlı sistemlerde daha da büyük önem kazandığını gösterdi. Bunun temel sebepleri, sunucu bileşenlerinin ve çok katmanlı yapının karmaşıklığının yanısıra bu bileşenlerin yalnızca uygulama sunucularının içerisinde barınıp yalnız başına çalıştırılamamaları ve *http* protokolünün yapısı gereği her testte senaryonun en başına dönülmesi gereği idi. Bu nedenle bileşen testleri ile fonksiyonel testlerin yazılması ve biriktirilmesinin uzun vadede uygulamanın bakımında büyük kolaylıklar sağlayacağına inandık.

Projenin ilk modülünün devreye alınmasından önce uyguladığımız “Yükleme Testleri” ve “Performans Testleri” ise çok kritik olan bazı problemleri zamanında farkedip sorunları giderebilmemizi sağladı. Bu problemlerin birincisi, çok sayıda kullanıcının paylaşılan kaynaklara aynı anda erişimi sonucu ortaya çıkabilecek ve veritabanında da kalıcı bozulmalara yol açacak kadar ciddi neticeler doğurabilecek olan koşutzamanlılık problemiydi. İkincisi ise, sistemi değişik sayılarda kullanıcıyı simüle ederek test etmenin hem uygulama sunucusu hem de veritabanı konfigürasyonumuzu optimize etmemize imkan vermesiydi. Başlangıç aşamasında aynı test senaryosunu 30 eşzamanlı iş parçacığı (thread) tarafından çalıştırdık ve uygulama test ortamında eşik değerine ulaştı. Testleri değişik konfigürasyonlarda çalıştırarak uygulama eşik değerini 100 eşzamanlı iş parçacığına kadar çıkarmayı başarabildik. Otomatize edilmiş performans testlerinin geliştirilen yazılımın sınırlarının ve yeteneklerinin tanınması, ihtiyaç duyulacak olan donanım konfigürasyonunun önceden belirlenebilmesi ve yazılıma duyulan güvenin artması açısından kritik olduğunu doğrudan yaşayarak gördük.

Testler yazılırken üzerinde durulması gereken önemli bir nokta da, testlerin önceden belirlenmiş bir sırada ve tümünün birarada çalıştırılabilecek şekilde tasarlanması gerekliliği oldu. Her testin bitişinden sonra testin veritabanında yaptığı değişiklikleri temizleyip geriye alarak, uygulamayı bir sonraki test için hazır hale getirmek gerekmektedir. Ayrıca iyi bir tasarım izleyerek, fonksiyonel testlerin bileşen testlerindeki test modüllerini yeniden kullanmasını sağlamak testlerin bakımını kolaylaştırmaktadır. Ayrıca test senaryolarını çalıştırmak için rastgele test verileri üreten ve bunların takibini yaparak sonuçları karşılaştıran örnekleyici sınıfların yazılması, performans testlerinin uygulanmasında izlediğimiz yoldur.

Bu bildiriye konu olan çalışmada, test yönelimli bir yazılım geliştirme metodu izlenmesi iki konuda avantaj sağladı. Yazılan testlerin tümünü günün sonunda çalıştırmanın özellikle modüllerin entegrasyonu aşamasında bir modülde yapılan değişikliklerin sistemin diğer bileşenlerinde başka bir hataya yol açıp açmadığını anlamak ve hatayı düzeltmek açısından oldukça kullanışlı bir yöntem olduğu anlaşıldı. Ayrıca, yazılan fonksiyonel testler sistem senaryolarının beklenen biçimde çalışıp çalışmadığını net bir şekilde gözler önüne serdi. Sonuç olarak, programcıların yazdıkları her kod parçasının etkileri konusunda endişelenmelerine gerek olmayacağından, daha etkin bir şekilde kod geliştirmelerinin mümkün olacağı düşünülmektedir.

Kaynakça

1. General Public License(GPL), http://www.opensource.org/licenses/gpl-license.php
2. Apache Software Foundation, The Apache Ant Project, http://ant.apache.org/.
3. Apache Software Foundation, The Apache Jakarta Cactus Project, http://jakarta.apache.org/ cactus/index.html.
4. Wosnick, S., Developing and Unit Testing With Open Source Apache Cactus Framework Tools In WebSphere Studio Application Developer, IBM WebSphere Developer Technical Journal, http://www.software.ibm.com/wsdd/techjournal/0206_wosnick/wosnick.html
5. Gold, R., HttpUnit, http://httpunit.sourceforge.net/.
6. Apache Software Foundation, The Apache Jakarta Struts Project, http://jakarta.apache.org/ struts/index.html.
7. Apache Software Foundation, The Apache Jakarta Project, http://jakarta.apache.org/ log4j/docs/index.html.
8. Alur, D., Crupi, J. ve Malks, D., Core J2EE Patterns, Sun Microsystems Press, ISBN: 0-13-064884-1.
9. HighTower, R., ve Lesiecki, N., Java Tools For Extreme Programming, Wiley Computer Publishing, ISBN: 0 – 471 – 20708 – X.
10. Bhogal, K. S., HttpUnit: A Civilized Way to Test Web Applications in WebSphere Studio, IBM WebSphere Developer Technical Article, http://www.software.ibm.com/wsdd/library/ techarticles/0303_bhogal/bhogal.html.