TEST GÜDÜMLÜ YAZILIM GELİŞTİRME SÜRECİ VE KULLANILAN ÇERÇEVELER

Ecir Uğur KÜÇÜKSİLLE , Nurullah ÖZTÜRK, İbrahim Arda ÇANKAYA, Asım Sinan YÜKSEL
Süleyman Demirel Üniversitesi, Bilgisayar Mühendisliği, Türkiye
ecirkucuksille@sdu.edu.tr, n.ozturk66@gmail.com, ardacankaya@sdu.edu.tr, asimyuksel@sdu.edu.tr

Özet. Test Güdümlü Yazılım Geliştirme modeli önce test koşullarının yazılmasını, sonrasında da yazılan testleri geçecek ve kendinden beklenen işlevi yerine getirecek kodun yazılarak bir yazılımın geliştirilmesini öngören yazılım geliştirme modelidir. Başarılı test sürecinin gerçekleştirilmesi ile en az hataya sahip yüksek doğrulukta yazılımlar üretilebilmektedir. Günümüzde test güdümlü yazılım geliştirme sürecinde kullanılmak üzere hazırlanmış birçok test çerçevesi bulunmaktadır. Bu çalışmada test güdümlü yazılım geliştirme süreci ve var olan test çerçeveleri hakkında bilgi verilerek, bu çerçevelerin kısa bir karşılaştırılması yapılmıştır.

Anahtar Kelimeler. Test Güdümlü Yazılım Geliştirme, Test Çerçeveleri, Test Çeşitleri

1 Giriş

Yazılım projelerinin en önemli bölümlerinden biri test yazım sürecidir. Başarılı bir test süreci sonunda en az hataya sahip yüksek doğrulukta yazılımlar üretilir. Türkiye'de yazılım kalitesi hakkında 2013 yılında yapılan araştırma sonuçlarına göre test işiyle görevli olan test uzmanlarının oranı geçen sene %54 iken bu yıl oran %74 olmuştur. Test otomasyonu ile ilgili "performans testleri ve simülasyonu" %43'lük oran ile birinci sıradayken, "test yönetimi, test işletimi ve birim testi otomasyonu" %30-35 oranla ikinci sırada olmuştur [1].

Ülkemizde artık test güdümlü yazılım geliştirme sürecine ve test araçlarına olan ihtiyaçların arttığı gözlenmiştir. Bu nedenle bu çalışmada genel olarak test güdümlü yazılım süreçleri, birim testi ve yaygın olarak kullanılan dillerdeki en yaygın birim testi çerçeveleri hakkında bilgi verilerek kıyaslanması yapılmıştır.

2 TEST GÜDÜMLÜ YAZILIM GELİŞTİRME SÜRECİ

Test Güdümlü Programlama yöntemi, Test Öncelikli Geliştirme (TÖG) ve Gözden Geçirme (Refactoring) yöntemlerinin bir araya getirilmesi ile oluşturulur [2]. Test

güdümlü yazılım geliştirmekteki hedef gerekli işi yapacak kodu yazmadan önce, bu koda ait olan testin yazılmasıdır. Test yazıldıktan sonra yapılacak olan iş ise test kodunu başarıyla geçecek kodun yazılmasıdır. Başarılı bir test sonrasında tekrar başa dönüp, kod incelenerek problem varsa yeninden düzenleme yapılmalıdır.

Test güdümlü programlamanın algoritma adımları şunlardır [3]:

- Testin yazılması
- Testin derlenmesi: Bu aşamada test başarısız olacaktır. Çünkü teste tabi tutulacak sınıflar ve metodlar henüz yazılmamıştır.
- Sadece derlenmeye yetecek kadar kodun yazılması ve gerekirse yeniden gözden geçirilmesi
- Testin çalıştırıp başarısız olunduğunun gözlenmesi
- Testi çalıştırıp, testin geçildiğinin görülmesi
- Testlerin tekrar çalıştırılıp, başarılı olduğunun gözlenmesi
- Kodun gözden geçirilerek yeniden düzenlenmesi
- İlk adıma dönülerek yeniden başlanılması

Test güdümlü yazılım geliştirme ile birlikte oluşturulan tasarım, kendiliğinden var olan bir tasarım modeli değildir. Yazdığımız testler şekillendikçe oluşan bir tasarım modelidir. Testleri oluşturmamızın bize sağladığı pek çok yarar vardır. Bunlardan birincisi, oluşturduğumuz testler sayesinde uygulamamızın beklendiği gibi çalışıp çalışmadığının önceden fark edilmesidir. İkincisi ise, bu testler sayesinde doğal olarak dokümantasyon oluşturulmuş olur.

Değişik türlerde testler oluşturmak mümkündür. Bunlar [4]:

- Birim Testleri
- İşlevsel Testler
- Entegrasyon Testleri
- Regresyon Testleri
- Kabul Testleri
- Sistem Testleri
- Stres Testleri
- Performans Testleri

Bu çalışmada birim testi ve işlevsel testler üzerinde durulmuştur. Ayrıca güncel programlama dillerinde yaygın olarak kullanılan test çerçeveleri hakkında bilgi verilerek karşılaştırmaları yapılmıştır.

2.1 BİRİM TESTİ (Unit Test)

Birim Testi, yazılım projesindeki her bir birimin (sınıf ya da metot) hatasız bir şekilde çalıştığını kanıtlamak için oluşturulan testtir. Birim testi yazılım geliştirme sürecini kolaylaştırılır, hızlandırır ve her bir sınıfın, metodun doğru çalıştığını garantilememizi sağlar. Bir testin birim testi olup olmadığını [3, 5]:

- Veritabanı etkileşimi varsa,

- Ağ iletişimi varsa,
- Dosya sistemiyle etkileşimi varsa,
- Testin başarılı olması için gerekli ayarların değişmesi gerekiyorsa,

birim testi değildir diyebiliriz.

Geliştiriciler zamanlarının sadece %10'dan %50 ye kadar olan bölümünde üretken bir şekilde kod yazar. Geri kalan kısım düşünme, tasarlama, tartışma gibi aktivitelerle geçer. Eğer TDD yaklaşımı kullanılmıyorsa manuel olarak hata ayıklama, yakalama ve düzeltme aşamasında çokça vakit kaybedilecektir. Birim Testlerini yazmamızı ve kontrol etmemizi kolaylaştıracak birçok çerçeve geliştirilmiştir. Neredeyse her dil için bir test çerçevesi bulunmaktadır.

2.2 İşlevsel Test(Functional Test)

İşlevsel testler uygulamanın bileşenleri arasındaki etkileşimi, bileşenlerin uyumunu test etmek amacı ile yapılır. Bu testler, durum senaryolarından elde edilir ve her durum senaryosu bir işlevsel teste dönüşür. İşlevsel testler çok önemlidir. Bu testlerde bir işlevde yaşanan sıkıntı birbirine bağlı tüm işlevleri etkilemektedir. Düzeltilen hata da o işleve bağlı tüm işlevlerde düzeldiği anlamına gelir. Ancak birim testinde durum böyle değildir. Hatanın ilgili işlevi başka işlevlerle bağlantılı olmadığı için alınan hata da diğer işlevleri etkilememektedir.

Birim testi ile işlevsel test arasında farklılıklar vardır. Birim testinde kod yazmadan önce test kodu yazılırken, işlevsel test yazılımın müşteriye tesliminden önce yapılır. Oluşturulan birim testleri bir kod bloğunun niçin var olduğunu anlatırken, işlevsel testler uygulamamızı oluştururken hangi özelliklerin gerçekleştiğini dokümantasyonunu sağlar.

3 Kullanılan Çerçeveler

Bu bölümde masaüstü programlamada en yaygın olan (C++, Java, C#) dillerde, web programlamada (JSF,ASP.net, PHP) ve mobil programlamada (IOS ve Android) kullanılan birim testi çerçeveleri incelenmektedir.

3.1 Masaüstü Programlama Dillerinde (C++, Java, C#, Scala, Python) Kullanılan Çerçeveler

C++ dilinde kullanılan birçok çerçeve mevcuttur. Bunların bir kaçından kısaca bahsetmek gerekirse:

Boost Test: Birim Test kütüphanesi Boost C++ kaynak kütüphanesinin bir parçasıdır. Ticari olmayan kullanımı teşvik eden Boost Lisansı ile lisanslanmıştır. Mimari, üyelik fonksiyonlarını test fonksiyonu gibi kullanan test sınıflarına sahip xUnit mimarisinden farklıdır. Çerçeve, testlerin çalışması için gerekli kodu üreten makrolar kullanır. Esas testleri gerçekleştirmek için ise "Assertion" lar kullanılır. Bu çerçeve

test araçlarının kullanımını ve bunları hiyerarşik bir yapıda organize etmeyi kolaylaştıran araçlar sunar. Kullanıcıları hata denetimi, raporlama, parametre işleme gibi karmaşık işlemlerden kurtarır. Çerçevenin başlatılması için bir main() fonksiyonu sağlar, komut satırı parametrelerini ayarlar, kullanıcının hazırlamış olduğu init_unit_test_suite(argc, argv) fonksiyonunu çağırır ve kullanıcının testlerini çalıştırır. Test sonucunda başarılı ve başarısız olan test durumları sürekli izlenir ve o ana kadar yapılan testlerin, toplam yapılması gereken test miktarına göre ilerleme durumu çeşitli formatlarda raporlanır. Birim Test Çerçevesi hem basit hem de karmaşık testler için uygundur. Buna rağmen, üretim ortamındaki kodu test etmek amacıyla kullanılmaz. Birim Test Çerçevesi işbirliği içinde olan çeşitli bileşenlerden oluşur. Tüm bileşenler boost::unit_test ismi altında yer almaktadır.

Cppunit Test: İlk olarak uygulamanın görev testleri yazılıp sonra bu testlerde başarılı/başarısız olma durumu kontrol edilir. Doğru şekilde uygulanması halinde tüm birim testlerden geçmelidir. Başarısızlığın önceden fark edilmesinde, hataların önceden ayıklanmasında bize yardımcı olan bir test çerçevesidir.

Bu iki birim testi çerçevesi birçok ortak özelliğe sahip olsa da birbirine göre üstün oldukları alanlar da bulunmaktadır. Bunu bir tablo üzerinde göstermek gerekirse [6]:

Tablo 1. Boost Test / CppUnit Test Çerçevelerinin Kıyaslaması

Özellik	Boost Test	Cppunit Test
Profesyonel geliştirme	Var	Var
Sürdürülebilirlik	Var	Yok
Xml çıktısı	Var	Var
Son test	-	Yok
Fikstürler	Var	Var
Otomatik test	Var	Var
Devre dışı iken test	NCBI uzantı desteği var	Yok
Dahili zengin assert fonksiyonları	-	Var
Özelleştirilmiş assert	Var	Var
Ölümcül olmayan assert	Var	Yok
Hata yakalama	?	Yok
Arayüz çalıştırma durumu	?	Var

Java platformlarında en çok bilinen temel iki test çerçevesi JUnit ve TestNG dir. İkisi de karmaşık test senaryolarında tam olarak istenilen kod parçacıklarında test oluşturmaya izin verebilecek kadar güçlü çerçevelerdir.

JUNIT: Junit tekrarlanabilir testleri yazmak ve çalıştırmak için kullanılan açık kaynak kodlu bir çerçevedir. Birim test çerçeveleri için XUnit mimarisinin bir örneğidir. JUnit özellikleri, programdan beklenen sonuçları test etmek için beklentileri, yaygın test verilerini paylaşmak için test fikstürlerini içerir.

TestNG: TestNG işlevselliği sayesinde kullanımı kolaydır ve güçlü bir test aracıdır. TestNG birçok özelliği desteklemektedir. Örneğin; esnek test konfigüra-

syonu, veritabanı testi için destek, aynı sınıfın birden çok örneği için destek, güçlü çalıştırma modeli, gömülü betik dili (beanshell) için daha fazla esneklik, çalışma zamanı ve günlüğü için varsayılan JDK fonksiyonları gibi birçok güçlü yönleri olan bir çerçevedir. [7].

Tablo 2. JUnit / TestNG çerçevelerinin Kıyaslanması

Özellik	JUnit	TestNG
Kullanıcı tanımlı yaşam döngüsü	Var	Var
Test organize edebilme (grup vb.)	Yok	Var
Dağıtılmış test uygulaması	Yok	Var
Paralel test uygulaması	Yok	Var
Veri güdümlü testler	Yok	Var
Bağımlı testler	Yok	Var
IDE bağlantısı	Var	Var
Ant bağlantısı	Var	Var
Maven bağlantısı	Var	Var
Alana göre uzantılar (veritabanı, HTTP vb.)	Var	Yok
Bilgi paylaşım platformu	Var	Var

Kullanıcı tanımlı yaşam döngüsü, test yapanlara hangi metotların hangi sırada olacağını belirten bir özelliktir. Junit ve TestNG nin ikisinde de bu özellik mevcuttur. JUnit ile gruplama işlemi sadece basit bir sınıfta gruplama işlemidir. Büyük projelerde testleri dağıtık ya da paralel modda çalıştırmak kesin olarak uygulamanın hızını artırır. Veri güdümlü testler sayesinde büyük miktarlardaki veriler basit bir test senaryosu oluşturularak bütün verilere uygulanabilir.

C# programlama dilinde kullanılan test çerçeve araçları tüm Microsoft programlama dillerinde ortak olarak kullanılmaktadır. En yaygın kullanılan test araçları NUnit te XUnit.net'tir. NUnit te XUnit .net e çok benzer şekilde test kısımları direk olarak projede kodun içine oluşturulur. Fakat mekanizması köklerde, durumlarda ve özelliklerde oldukça farklıdır. .Net için geliştirilmiş bir birim test çerçevesidir. XUnit ise XML programların karmaşık senaryolarını test eder. Yapı olarak JUnit e çok benzerdir.

Scala ölçeklenebilir programlama anlamına gelmektedir. Scala programlama dili ruby, java, c++ dillerinin birleşimi şeklinde bir dildir.

ScalaTest, Scala ve Java kodları test edilebilen bir test aracıdır. Junit, TestNG, ScalaCheck, Jmack gibi test araçları ile bağlantılıdır. Bundan dolayı daha hızlı ve başarılı bir yapıdır. ScalaTest projede bütün kısımlara uygulanabilir, en küçük proje parçacığından geniş çaplı projelere kadar destek sunar.

Python, test temelli geliştirme için kullanışlı bir dildir. Kendi içindeki bileşenler sayesinde başka eklentilere gerek duymadan testler yapılabilir. Standart kütüphanesinde test temelli geliştirmeye gerekli her şey mevcuttur.

Unittest, JUnit'in Python dili versiyonu olan bir test çerçevesidir. Python 2.1 versiyonu ve daha üst versiyonlarında temel olarak yüklü şekilde gelir. Diğer dillerdeki birim testler ile benzer bir yapıdadır. Unittest Xunit stilinde bir çerçeve

olduğundan Pyunit diye adlandırılmıştır. Xunit'in diğer stilleriyle neredeyse aynı şekilde çalışır.

3.2 Web Programlama Dillerinde (JSF,ASP.NET,PHP) Kullanılan Çerçeveler

Web çatısı altında incelediğimizde ise JSF dilinde en yaygın olarak bilinen test çerçevesi CACTUS iken ASP.Net te NUNIT, PHP ise PHPUNIT'tir.

Cactus, Jakarta projesinin [9] server taraflı Java kodlarının birim testleri için geliştirilmiş basit bir test çerçevesidir. Cactus'ün amacı server taraflı kodların test yazımının maliyetini düşürmektir. JUnit kullanmaktadır. Cactus, konteyner stratejisi uygular. Testler konteyner içerisinde çalıştırır. Konteyner stratejisinde konteyner bir nesnedir. Bu nesne diğer nesneleri içinde bulundurur ve bu nesnelere erişimi, nesnelerin oluşturulmasını, yok edilmesini yönetir. Bu test daha iyi kod yazmayı, kod incelemeyi ve evrensel bir kod oluşturmayı sağlar.

PPHUnit ile test oluşturmanın normal test oluşturmaktan farkı yoktur. Sadece testin yapılma şekli farklıdır. Yani programın beklendiği gibi çalışıp çalışmadığının kontrol edilme biçimi, bir dizi testin uygulanma şekli, yazılımın birimlerinin/bileşenlerinin (units) doğrulunu otomatik olarak test eden çalıştırılabilir kod parçalıkları farklılık oluşturmaktadır. Bu çalıştırılabilir kod parçacıklarına da birim test adı verilir.

Ruby on Rails, Ruby programlama dili üzerinde çalışan, açık kaynak kodlu bir web tabanlı uygulama geliştirme çatısıdır. Rails, testleri yazmayı çok basit hale getirmiştir. Rails'in internet tarayıcısına bağlı kalmadan gelen istekleri test edebilme şansı vardır.

Rails'in kendisine ait birim testi çerçevesi mevcuttur. Rspec [10], Ruby on Rails için test aracıdır. Esnek ve düzenlenebilir raporlar verebilir. İçerisinde taklit (mocking /stubbing) çerçevesi gömülüdür, komut zenginliği vardır.

3.3. Mobil Programlama Dillerinde (iOS, Android) Kullanılan Çerçeveler

Mobil platformda durumu ele aldığımızda ise iOS ve Android programlama dillerinde en çok kullanılan test çerçeveleri iOS için Ocunit/Ghunit [11, 12] iken Android te ise Robotium/Monkeyrunner'dir [15, 16].

*OCUnit'te hazır olarak yeni test sınıfı mevcuttur ve direk oluşturulabilir [13, 14]; fakat GHUnitte temelde hazır olarak böyle bir sınıf bulunmamaktadır, dışarıdan eklenti ile GHUnit te de hazır test sınıfı oluşturulabilir.

*OCUnitin testleri derlenmeden önce uygulamaya eklenir ki bu olayda bir problem olmaması için ekstra dikkat gerekir. GHUnitte ise mantıksal ve uygulama testleri arasında çok farklılıklar yoktur. Ayrıca GHUnitte setUpClass ve tearDownClass metotları mevcuttur [13, 14].

*OCUnit ve GHUnit te test metodunu debug yapma işlemi aynı şekildedir.

*Test sonuçlarını veya test kayıtlarını yorumlamak, değerlendirmek OCUnit'te problemlidir. Log kayıtları sadece konsolda görünmektedir. GHUnitte test sonuçları UITableView de görünmektedir ki bu tabloya filtre uygulanabilir, yönetilebilir.

Robotium, Android uygulamalarını test etmeye yarayan açık kaynak kodlu bir test çerçevesidir [15]. Bu çerçeveye ait tüm kaynak kodlar [17]'den indirilebilir. Test senaryosu geliştirenler bu çerçeveyi kullanarak fonksiyon, sistem ve kabul testleri

yazabilirler. Kullanıcının geliştirilen bir uygulamayı kullanırken izlediği adımları taklit ederek test senaryoları oluşturulabilir.

Monkeyrunner, Android uygulamalarını test etmeyi sağlayan diğer bir test çerçevesidir [16]. Bu aracı kullanarak uygulamalar dışarıdan kontrol edilebilir. Uygulamalar emulatör ya da gerçek bir araca yüklenerek çalıştırılabilir. Monkeyrunner tam olarak bir test etme aracı değildir. Çünkü Robotium'da olduğu gibi test senaryoları oluşturulup test yapılamaz. Bunun yerine uygulmalara girdi gönderilip (tıklama ya da tuşa basma yoluyla), bunun sonucunda oluşan çıktıların ekran görüntüleri kaydedilebilir. Tablo 3'de Robotium ve Monkeyrunner için kıyaslama yapılmıştır [15, 16].

Tablo 3. Robotium / Monkeyrunner çerçevelerinin Kıyaslanması

Kriterler	**Robotium**	**Monkeyrunner**
Kaydetme ve Tekrar Çalıştırma	Kayıt desteklenmiyor Tekrar çalıştırma başarılı	Kayıt ve yeniden çalıştırma başarılı
Nesne Tanımlama	Nesne tanıma tam olarak desteklenir	Tanıma ve kontrol noktaları iyi derecede desteklenir
Doğrulama / Kontrol Noktası	Doğrulama / kontrol noktası çok başarılı	Yüksek derecede doğrulama ve kontrol noktası desteği
Test Verisi Kullanımı	Başarılı	Başarılı
Raporlama	Başarılı	Başarılı
Komutlar	Genelde başarılı	Çok başarılı değil
Etkinliği	Çok etkin	Etkin
Genişletilebilme	Çok genişletilebilir değil	Çok genişletilebilir değil
Lisans/Destek	Açık kaynak kodlu, geleceği var	Açık kaynak kodlu, geleceği var

4 Sonuç

Yapılan araştırmalar gösteriyor ki hataların fark edilmesi ve çözümlenmesi şirketlere, kurumlara para, zaman ve itibar kaybettirmektedir. Bu nedenle de artık günümüzde test güdümlü yazılım geliştirme sürecine verilen önem artmıştır. Bu amaçla projelerin iyi kurgulanması ve doğru test araçlarının kullanılması projelerin başarıya ulaşmasında en önemli etkendir.

Sonuç olarak, test süreçleri yazılım geliştirme yaşam döngüsü içerisinde önemli bir yere sahiptir ve geliştirilecek yazılımın zamanında tamamlanmasını ve maliyetini ve doğrudan etkiler. Bu süreç için doğru test araçlarının kullanılması sürecin başarısını önemli ölçüde etkiler. Doğru testler, doğru planlama, doğru kişiler ve iyi test araçları ile başarı yakalanır. Test güdümlü modern yazılım geliştirme tekniklerinin temel yapı taşlarını testler oluşturmaktadır. Bu nedenle test araçlarının seçimi ayrı bir öneme sahiptir. Bu çalışma kapsamında test aracı seçiminde yazılımcılara faydalı olabilecek çeşitli çerçevelere değinildi, birbirlerine göre üstünlükleri, zayıflıkları ortaya konuldu. Bu tespitler, Süleyman Demirel Üniversitesi bünyesinde gerçekleştirilen projelerden

ve bu konuda yapılan literatür taramaları sonucunda elde edilen bilgilerden faydalanılarak oluşturulmuştur. Bu çalışmanın diğer testleri de kapsayacak şekilde genişletilmesi ve detaylandırılması mümkündür. Bu yönüyle çalışmamız ileride yapılacak benzer çalışmalara örnek teşkil edecektir.

Kaynaklar

1. Turkey Software Quality Report 2013, http://www.testrisk.com/2013/05/turkey-software-quality-report-2013.html
2. Ambler, S. W. (2006). Introduction to test driven development (TDD).
3. Yamuç A. ,Çakın A. (2012). Web Uygulamaları Geliştirilmesinde Test Güdümlü
4. Programlamanın Yeri: Bir Örnek Durum Çalışması, Akademik Bilişim Konferansı
5. Acar Özcan, Extreme Programming, Pusula Yayınevi, 2009
6. Feathers, M. (2004). *Working effectively with legacy code*. Prentice Hall Professional.
7. Llopis, N. (2004). Exploring the C++ unit testing framework jungle.
8. Louridas, P. (2005). JUnit: unit testing and coiling in tandem. *Software, IEEE*, *22*(4), 12-15.
9. http://jakarta.apache.org
10. http://rspec.info
11. http://www.sente.ch/software/ocunit/
12. https://github.com/gabriel/gh-unit
13. Horovitz, A., Kim, K., LaMarche, J., & Mark, D. (2013). Unit Testing, Debugging, and Instruments. In *More iOS6 Development* (pp. 481-510). Apress.
14. Lee, K. (2011). *Core Objective-C in 24 Hours*. Keith Lee.
15. Knott, D. (2011). Robotium@ XING. Automated regres-sion tests on mobile Android devices. *Book your training with Díaz & Hilterscheid!*, 26.
16. Rajath, H. (2012). Automated Testing Framework For Android Based Applications. *International Journal of Research in Robotics Applications-IJRRA*, *1*(1).
17. https://code.google.com/p/robotium/