Çevik Yazılım Geliştirmede BDD/TDD Yöntemlerinin ve Yazılım Kalite Araçlarının Kullanılması: Bir Yazılım Mühendisliği Dersindeki Tecrübe

Gökhan Akyol, Haluk Gümüşkaya
İzmir Yüksek Teknoloji Enstitüsü, Urla, İzmir
gokhanakyol@iyte.edu.tr, haluk@gumuskaya.com

Özet. Çevik yazılım geliştirme modeli iyi uygulandığında, yazılım proje yönetimini, yazılım takım elemanları arasındaki iletişimi ve yazılım geliştirme süreçlerini ve süresini iyileştirmektedir. Çevik yazılım geliştirme modelinde, son kullanıcının yazılım geliştirme sürecinin içinde olduğu bir yazılım ürünü, değişen gereksinimlere daha hızlı cevap verir. Bu çalışmada Davranış Tabanlı Tasarım (BDD-Behavior Driven Design) ve Test Tabanlı Geliştirme (TDD-Test Driven Development) tekniklerinin SaaS (Software as-a Service) uygulamaları geliştirmede beraber kullanıldığı bir bütünleşik çevik yazılım geliştirme süreci ve çerçevesinde, günümüzde kullanılabilecek yazılım test ve kalite ölçme araçları karşılaştırılmaktadır. Sunulan çevik yazılım geliştirme süreci ve çerçevesi bir son sınıf Yazılım Mühendisliği dersinde üç yıl öğrencilerin geliştirdikleri projelerde uygulanmıştır. Bu çalışma, dört farklı proje gerçekleştirme teknolojileri ve platformları (Ruby/Rails, Java EE/Spring, C#/ASP.NET ve PHP/(Zend, Codeigniter, Laravel) kullanılarak, Microsoft Azure ve Heroku bulut servis sağlayıcılarında çalışan SaaS uygulamaları geliştirmede kullanılan BDD ve TDD test ve kalite araçlarıyla elde edilen tecrübeyi sunmaktadır.

Anahtar Kelimeler: Çevik Yazılım Geliştirme, BDD, TDD, Yazılım Test ve Kalite Araçları

1 Giriş

Yazılım mühendisliği, proje ve süreç yönetimi teknolojik gelişmelere bağlı olarak değişmektedir. Günümüzde yazılımlar gittikçe bulut servis sağlayıcıları üzerinden son kullanıcılara sunulmaktadır ve bulut teknolojileri ve servisleri yazılım geliştirme fazlarını değiştirmektedir. Yazılım endüstrisinde çoğunlukla kullanılan yazılım geliştirme metotları şelale ve çevik yazılım geliştirme yöntemleridir. Şelale modelinde yazılım geliştirme safhaları sırasıyla yerine getirilir. Gereksinimlerin toplanması ve analiz safhasında, müşteri ve ürün özellikleri toplanır ve analiz edilir. Tasarım ve gerçekleştirme fazlarında gereksinim değişiklikleri genelde göz ardı edilerek yazılımın esnekliği kısıtlanmış olur [1]. Şelale modelinin getirmiş olduğu dezavantajlar ve günümüz hızlı ürün teslimi, iş baskısı, yazılım geliştiren firmaları çevik yazılım geliştirme yöntemlerini kullanmaya yönlendirmiştir. Çevik yazılım geliştirme, müşteriyi projeye dahil etme, değişen gereksinimlere ayak uydurma, iş analisti, yazılımcı, test edici gibi kişilerin projede beraber görev almaları, haftalık

yapılan yüz yüze iletişim, sık aralıklarla ürün testi ve teslimi gibi özelliklerinden dolayı şirketlerce çok kullanılan bir yazılım geliştirme süreci olmuştur [2].

Çevik yazılım geliştirmede kullanılan yaygın yazılım proje yönetim tekniği Scrum modelidir. Scrum modelinin en önemli özellikleri, Scrum takım modeli, kullanıcı hikayeleri (user stories) ve koşudur (sprint). Scrum modelinde ürün sahibi (product owner), scrum ustası (scrum master) ve geliştirme takımı bulunur. Müşteri gereksinimleri kullanıcı hikayeleri aracılığı ile tanımlanmaktadır. Her bir koşu sonunda ortaya çıkan ara ürünler sayesinde müşteri proje gelişmesini takip eder ve projenin bir parçası olur. Çevik yöntemler ile proje küçük parçalara ayrılarak karmaşıklık azaltılır, projenin riski en aza indirilerek para ve zaman kaybının önüne geçilmiş olur [3].

Bu çalışmada çevik yazılım geliştirmede yaygın olan Scrum ve BDD (Behavior Driven Design) ve TDD (Test Driven Development) tekniklerinin bir arada kullanıldığı ve yazılım kalite araçları ile desteklenen bulut yazılım servisleri (SaaS uygulamaları) geliştirme tecrübesi sunulacaktır. Bu yazılım projeleri, gerçek müşteri problemlerini çözmeye yönelik ve müşterilerin projelere dahil edildiği gerçek uygulamalardır. Sunulan bu yazılım geliştirme yöntemi ve teknikleri Gediz Üniversitesi Bilgisayar Mühendisliği Bölümü'nde verilen COM 401 kodlu Yazılım Mühendisliği dersinde 2013-2015 yılları arasında üç sene uygulanmıştır. Bildiride sunulan model, ilk iki sene sadece Ruby/Rails ile geliştirilen projelerde, son 2015 senesinde dört farklı programlama dili ve ortamı (Ruby/Rails, Java/Spring, C#/ASP.NET ve PHP/(Zend, Codeigniter, Laravel) kullanılarak geliştirilen projelerde kullanılmıştır. BDD/TDD çatıları ve farklı programlama dilleriyle geliştirilen projeleri destekleyen bulut servis sağlayıcıları olan Microsoft Azure ve Heroku bulut ortamlarına projeler taşınarak SaaS uygulamaları çalıştırılmıştır.

Bu bildirinin yapısı şu şekildedir: İkinci kısımda çevik yazılım geliştirme, bulut bilişimin yazılım geliştirmeyi nasıl etkilediği ve BDD/TDD ile ilgili çalışmalar sunulmuştur. Üçüncü kısımda bulut servisleri geliştirmeye yönelik geliştirdiğimiz acil yazılım geliştirme modelimiz verilmektedir. Dördüncü kısımda Yazılım Mühendisliği dersimizde üç yıl bu model temelli geliştirilen proje uyugulamaları ve kullanılan teknolojiler ve araçlar karşılaştırmalı olarak verilmektedir. Son kısımda bu uygulamalarla elde edilen eğitim ve uygulama tecrübeleri sunulmaktadır.

2 İlgili Çalışmalar

Bulut için geliştirdiğimiz acil yazılım geliştirme modelimizde, ilgili projeyle ilgili problem tanımlanıp ilk kaba gereksinim analizi yapıldıktan sonra, yazılım geliştirme süreci BDD/TDD yöntemleri kullanılarak devam eder ve ürün geliştirilir. BDD/TTD yöntemleri ve prensipleri, alan tabanlı tasarım (domain driven design) ve nesneye yönelik analiz ve tasarımdan alınan fikirlerle geliştirilmiştir [4].

Son on yılda gereksinim mühendisliği alanında aşağıdaki önemli gelişmeler olmuştur. Bu gelişmelerden biri acil süreçlerin yaygınlaşması ve BDD/TDD'nin yazılım tasarım ve geliştirmede kullanımının artmasıdır [5].

- İş analistliğinin profesyonel bir meslek olması ve bu alanda Uluslararası İş Analistliği Enstitüsü ve Uluslarası Gereksinim Mühendisliği Heyeti gibi organizasyonların kurulması.
- Gereksinim analizi ve yönetimi ile ilgili araçların olgunlaşması ve bu araçların prototip geliştirme, modelleme ve benzetim gibi alanlarda kullanılarak gereksinim analizine yardımcı olması.
- Çevik yazılım geliştirme yöntemlerinin kullanımının artması ve acil projelerde gereksinimlerin ele alınma yöntemlerinin gelişmesi (BDD/TDD, kullanıcı hikayeleri, …)

Yukarıda yer alan gereksinim mühendisliği ilerlemelerine ek olarak, son yıllarda bulut bilişimin Yazılım Geliştirme Yaşam Döngüsünü (SDLC) çok etkilemeye başladığı, yazılım analiz, tasarım ve gerçekleştirme yöntemlerini değiştirdiğini görmekteyiz [6].

Çevik yazılım geliştirme süreçleri, yazılım endüstrisi ve eğitiminde çok kullanılmaktadır. Şirketlerdeki kullanımının yanında üniversite müfredatlarında yer alan Yazılım Mühendisliği derslerinde çevik yazılım geliştirme yöntemleri yaygın olarak kullanımaktadır [7-9]. Scrum bir çevik yazılım geliştirme proje yönetim modeli olup yazılım mühendisliği projelerinde kullanımı kolay bir yöntemdir. Scrum proje yönetim modeli bir kaç haftalık kısa zaman dilimleriyle geliştirilen yazılımın, tekrar tekrar gözden geçirilip müşteriye en iyi ve kısa zamanda tamamlarak teslim edilmesini sağlamayı amaçlamaktadır. Scrum modelinin kullanıldığı bulut uygulamaları geliştirme, acil yazılım geliştirme felsefesine çok uygundur [10]. Bulut servis sağlayıcılar ve platformlar, ayrıca sundukları alt yapı servisleri ile çevik yazılım geliştirmeye yardımcı olmakta ve hızlandırmaktadır [11].

Geleneksel yazılım mühendisliği yaşam döngüsünün veya acil yazılım süreçlerinin anlatıldığı birçok kitap mevcuttur. A. Fox ve D. Patterson tarafından yazılan kitap [12] çevik yazılım geliştirme ve bulut bilişim tabanlı yazılım mühendisliğini sunan ilk kitap olmuştur. Ruby/Rails kullanarak BDD/TDD yöntemleri ve Scrum acil proje yönetimi ile SaaS uygulamaları geliştirip bir bulut servis sağlayıcıda çalıştırmayı pratik olarak sunarak yazılım mühendisliğine yeni bir bakış açısı getiren bu kitap üç yıl ana kaynağımız oldu.

TDD 2000'li yıllardan günümüze özellikle çevik yazılım geliştirmede uygulanamtadır. TDD'de test kodlarının fonksiyonel esas koddan önce geliştirilmesi gerekir. Son yıllarda TDD iki aşamada ele alınmaktadır. Bunlar, Birim Test Tabanlı Geliştirme (UTDD) ve Kabul Test Tabanlı Geliştirme (ATDD) şeklindedir. UDDD birim testleri yaparak sistemin fonksiyonel olarak çalışmasını sağlayan geliştirme yöntemidir. ATDD ise müşteri odaklı iş yapma seviyesinde yapılan bir geliştirme

yöntemidir [13]. TDD'nin test çatısı (test framework) modelleriyle birlikte kullanılması sonucu yazılım hataları düşmekte ve yazılımın kalitesi artmaktadır [14]. TDD uzun yıllardır birçok yazılım firmasında yaygın kullanıldığı ve yazılım hatalarını %50 oranında düşürdüğü görülmüştür [15]. BDD ürün sahibinin kullanıcı hikayelerini kullandığı ve Scrum ustasına aktardığı bir tasarım modelidir. Kullanıcı hikayeleri BDD'nin temel taşlarından bir tanesidir. Ürün sahibi kullanıcı hikâyelerini belirli bir öncelik sırasına göre ürün içeriğine (Product Backlog) yerleştirir [16].

3. Çevik Yazılım Geliştirme Modeli Yaklaşımımız

Çevik yazılım geliştirme modeli yaklaşımımız, geleneksel yazılım geliştirme yaşam döngüsü safhalarından farklı şekilde tanımlanmıştır. İki ana kısımdan oluşan çevik yazılım geliştirme modelimiz aşağıda verilmektedir:

Problemin tanımlanması ve analizi:
- İş analizi ve önemli gereklerin toplanması ve analizi

Çözüm:
- Sistem (fonksiyonel) tasarımı ve fonksiyonel testler (BDD)
- Test ve kodlama (sınıfların ve metodların test sürümlü geliştirilmesi) (TDD)
- Yazılım kalite ölçümü ve iyileştirmeler (refactoring)
- Bulut servis sağlayıcıya aktarım

Birinci safha iş analizi ve gereksinimlerin toplanmasıdır. MoSCoW metoduna göre toplanan gereksinimler sınıflandırılır. İkinci safha olan çözüm alanında sistem tasarımı BDD ile yapılır. Kodlama ve test kısmı TDD ile geliştirilirken, yazılımın kalitesi ve olgunluğu çeşitli ölçütler sayesinde yazılım kalite araçları ile son kullanıcının taleplerine uygun hale getirilir. Her iterasyon sonunda yazılım bir bulut sağlayıcıya (Heroku veya Azure) aktarılarak çevik yazılım geliştirme süreci tamamlanmış olur.

Çevik yazılım geliştirme yaşam döngüsü, gereksinimlerin uyarlandığı ve test edilebilir hale geldikten sonra son kullanıcı ile beraber çalışılarak bir veya iki haftalık koşularla sistemin oluşturulmasıdır. Son kullanıcı, her bir koşuda aktif olarak görev alır ve gereksinimlerin uygunluğunu denetler. Eğer son kullanıcının talepleri yerine getirilir ise, yeni gereksinimlerin ele alınması ile yazılım geliştirme devam eder. Geliştirilen yazılım, modüler ya da bir bütün olarak tüm test senaryolarından geçip kullanıcının onayı ile bulut servis sağlayıcıya aktarılarak yazılım geliştirme son bulur.

Son yıldaki yazılım geliştirme modelimiz, iş analizi ve gereksinim mühendisliği fazıyla başlatıldı. Çevik yazılım geliştirmede doküman üretme yok denecek kadar az olmasına rağmen, yeni modelimizde projelerin en önemli kısımlarının yazılması ve bir proje yönetim planının üretilmesi istenmiştir. READ (Requirement Elicitation and Analysis Document) adını verdiğimiz ilk doküman, problemin tam olarak ne olduğu, önemli ilk çalışmaların yapıldığı, sistemin fonksiyonel (functional) ve fonksiyonel olmayan (non-functional) gereksinimlerinin öncelik sırasına göre sınıflandırıldığı ve

en önemli kullanım senaryolarının (Use Case) belirlendiği ve kullanıcı ekranlarının bulunduğu bir dokümandır.

Modelimiz yazılım geliştirmeye klasik gerekler mühendisliği uygulamaları ile başladıktan sonra BDD ile devam etmektedir. BDD, uygulamanın davranışına yönelik sorular sorarak, yazılım ekibi ile son kullanıcı ilk ana fonksiyonları beraber tasarlar, geliştirir ve test eder. BDD yazılımın ana fonksiyonlarının önce doğru ele alınıp alınmadığını ve geliştirme ekibi tarafından gerçekleştirip gerçekleştirilmediğini test eder. BDD ile tasarlanan kullanıcı hikayeleri test senaryolarına dönüştürülür. Kabul testleri ile kullanıcıların ana ihtiyaçlarının karşılanması hedeflenir. Geliştirilen projelerde kullanıcı hikâyeleri Pivotal Tracker [17] proje yönetim aracı ile yönetilmiştir. Bu uygulama ile tanımlanan kullanıcı hikayeleri öncelik sırasına göre test ve geliştirme ekibine yollanır. Test ve geliştirme ekibi tarafından gerçekleştirilen fonksiyonlar, ürün sahibi tarafından kontrol edilir. Eğer kullanıcının isteği tam olarak yerine getirilmişse, Pivotal Tracker'daki "Kabul (Accept)" tuşu ile işlem sonlandırılır. Aksi durumda ise "Kabul Etmeme (Reject)" tuşu ile işlem tekrar test ve yazılım geliştirme ekibine geri gönderilir.

Proje boyunca Pivotal Tracker programı ile takım performasını, verimliliğini ve projenin hızını kullanıcı hikayeleri ile ölçebiliriz. Kullanıcı hikayeleri geliştirilirken bir yandan da kullanıcı ara yüzleri çeşitli programlar vasıtasıyla oluşturulmaktadır. BDD ile otomatik kabul testini sağlamak için çeşitli programlar kullanılmaktadır. BDD için kullanılan mevcut araçlar genellikle Cucumber [18] programı temel alınarak geliştirilmiştir. Cucumber ve benzeri programlarda, öznitelikler (features) , senaryolar (scenarios), adımlar (steps) ve adım tanımlamaları (step definitions) belirli bir yapı ve programlama dili içerisinde tanımlanır. Adım tanımlamaları, test metotlarını oluşturarak BDD için kabul testleri yerine getirilir. BDD safhasından sonra kod gerçekleştirme safhasının yer aldığı TDD safhası gelir. Bu safhada önce birim testleri yapılır ve sonra ana kodlar geliştirilir. BDD ve TDD'nin bir arada kullanıldığı ve çeşitli programlama dillerine uygun olan yazılım altyapıları (frameworks) ve tasarım şablonları mevcuttur. Yazılım kalite araçlarına da uyumlu olan yazılım altyapıları bulut servis sağlayıcıya aktarılarak çevik yazılım geliştirme modeli başarılı bir şekilde uygulanmış olur.

4. Çevik Yazılım Geliştirme Modeli Yaklaşımımızın Uygulamalı Örnekleri

Çevik yazılım geliştirme modelimiz üç yıl son sınıf Yazılım Mühendisliği dersinde uygulanmıştır [19]. İlk iki yıl çevik yazılım geliştirmede Ruby/Rails kullanıldı. BDD/TDD yöntemleri ile yazılım geliştirme, test ve kalite araçları, bulut servis sağlayıcı Heroku, Ruby dili ve Rails yazılım altyapısının sağlamış olduğu araçlar kullanıldı. Tablo 1'de çevik yazılım geliştirme sürecinde kullanmış olduğumuz araçlar gösterilmektedir. İlk iki sene temel geliştirme süreci çevik yazılım geliştirme olup önemli bir dokümantasyon işi yapılmamış ve UML kullanılmamıştır.

Tablo 1. SaaS uygulamaları için çevik yazılım geliştirme araçları (2013 ve 2014 dönemleri).

Programlama Dili ve Alt Yapısı	Ruby / Rails
BDD ve TDD Araçları	Cucumber (BDD) ve RSpec (TDD)
Kullanıcı Arayüzü Aracı	Indigo Studio
Proje Yönetim Aracı	Pivotal Tracker
Yazılım Kalite Aracı	Metric_fu
Kod Versiyonlama Aracı	GitHub
Bulut Servis Sağlayıcı	Heroku
Çevik Yazılım Geliştirme Yöntemi	Scrum

Modelimizin son uygulandığı 2015 yılında çevik yazılım geliştirme sürecimizi koruyarak öğrencilere iş analizi ve gereksinim mühendisliği uygulamalarını yazılım projelerine dahil ettik. Geçen iki seneden bir diğer önemli farkımız ise programlama dili ve altyapısı kısmını serbest bırakarak öğrencilerin kendi seçimine sunduk.

Son sene yapılan projeler, kullanılan diller ve alt yapılar, yazılım test ve kalite ölçme araçları ve kullanılan bulut sağlayıcılar Tablo 2'de gösterilmektedir. Yazılım Mühendisliği dersi 3 saat teorik 2 saat laboratuvar olarak yapılmıştır. Çevik yazılım geliştirme sürecinin uygulamalı kısmı dersin asistanları tarafından yürütülmüştür. Projeler bir dönemlik olup her bir proje 4 ya da 5 kişilik bir proje ekibinden oluşmuştur. Belirlenen zamanlarda proje ekipleri her bir koşu sonunda projelerini sunmuşlardır. Proje geliştirmede 3 adet koşu yer almıştır.

Tablo 2. SaaS uygulamaları için çevik yazılım geliştirme araçları (2015 dönemi).

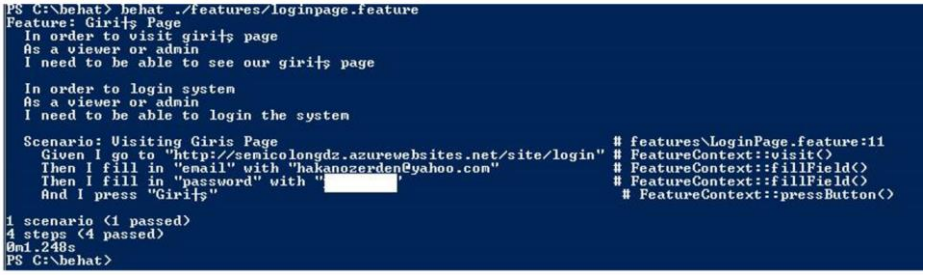
#	Project İsmi	Dil / Alt Yapı	Test Ortamları ve Yazılım Kalite-Metrik Araçları			Bulut Servis Sağlayıcı
			Kabul Testi (BDD)	Birim Testi (TDD)	Kalite Metrikleri	
1	Food Recipes	PHP/Zend	Behat	PHPUnit	PHP_Depend	Heroku
2	Nutra System	C#ASP.NET MVC / .NET	SpecFlow	NUnit	NDepend	Azure
3	Gediz Fast Food	C#ASP.NET MVC / .NET	SpecFlow	NUnit	ReSharper	Azure
4	Financial Analysis System for Restaurants	Ruby/Rails	Cucumber	RSpec	Metric_fu	Heroku
5	Club Management System	PHP/CodeIgniter	Behat	PHPUnit	PHP_Depend	Heroku
6	Customer Tracking System	PHP/Laravel	Codeception	PHPUnit	PHP_Depend	Heroku
7	Subel Business	PHP/CodeIgniter	Behat	PHPUnit	PHP_Depend, PHP_CodeSniffer	Heroku
8	Rent a Car	PHP/CodeIgniter	Behat	PHPUnit	PHP_CodeSniffer	Heroku
9	Food Pre-Order System	Java/Spring	JBehave	JUnit	SonarQube	Heroku

Tablo 2'de görüldüğü gibi, 5 proje PHP ile 2 proje ise C#/ASP.NET ile ve diğer kalan iki projeden biri Java EE, diğeri ise Ruby/Rails ile geliştirilmiştir. Bütün projeler, kullanılan dile bağlı olarak BDD/TDD yöntemleri ile ve yazılımın bulut

servisleri ve uygulamaları olarak, sunucudaki yazılım MVC tasarım şablonu temel alınarak RESTful tarzı web servisleri şeklinde geliştirilmiştir.

4.1. PHP Tabanlı Çevik Yazılım Geliştirme

PHP tabanlı çevik yazılım geliştirme, beş takım tarafından üç farklı yazılım alt yapısı kullanılarak gerçekleştirilmiştir. PHP ile yazılım geliştirmede kullanılan fonksiyonel gereksinimlerin test ortamı Behat'tır. Behat ortamı bir web tarayıcıda ya da bir uygulama ara yüzü yazılımında kullanılabilmektedir [20]. Behat ortamında Gherkin dili [21] kullanılarak **`Given-When-Then`** kelimeleriyle ana fonksiyonlar tanımlanır ve fonksiyonel test senaryoları oluşturularak kabul testleri geliştirilir. Şekil 1'de Behat ile yapılmış olan bir giriş sayfasının kullanıcının isteği doğrultusunda testi yapılmaktadır. Öznitelik (feature) tanımlaması kullanıcı hikayesinden yapılır. Ardından senaryo açıklaması ile kullanıcı hikayeleri Gherkin dili ile fonksiyonel olarak dönüştürülür. Bu dönüşüm kullanıcı girişi modülünde yer alan metotlarla eşlenerek testin başarılı olup olmadığı görülür.



Şekil 1. Behat ile yapılan bir BDD fonksiyonel test örneği.

Bir diğer BDD aracı Codeception'dır. PHP uygulamalarında kullanılan bu araç kullanımı kolay, tarayıcı ve yazılım altyapısı (framework) desteği olan ve modüller arası entegre çalışan bir ortamdır [22]. Test tabanlı geliştirmede (TDD) ise bütün projelerde PHPUnit kullanılmıştır. PHPUnit birim testleri yapmayı sağlayan ortamdır [23]. PHP yazılım çatıları ile entegre çalışan bu ortam MVC mimarisinde de kullanılmaktadır. Birim testlerini sağlayan ortamlarda ortak olarak bulunan ve test etmeyi sağlayan anahtar kelime "**`Assert`**" deyimidir. Bu deyim beklenen ve gerçek değerlerin kontrolünü yaparak modül testinde kullanılır.

BDD ve TDD'nin bir arada kullanılarak geliştirilen PHP tabanlı çevik yazılım projelerinin, yazılım kalite metrikleri ile yazılım kalitesini ölçen kalite araçları ile kod kalitesi ölçülmektedir. Programcılar ve son kullanıcı tarafından görülen problemler, sistemin içyapısı yeniden düzenlenerek, iyileştirme (refactoring) teknikleri ile kodlar düzeltilerek programın hatasız bir biçimde kullanıcıya teslim edilmesi sağlanır. PHP tabanlı çevik yazılım geliştirmede kullanılan yazılım kalite araçları PHP_CodeSniffer ve PHP_Depend'dir. PHP_CodeSniffer [24] belirli bir veri setinde tanımlanmış olan kod standartlarına göre programın analiz edilmesini sağlamaktadır. PHP_Depend [25] ise daha gelişmiş olan ve yazılım kalite ölçütlerine göre programı statik olarak analiz

eden araçtır. Şekil 2'de verilen örnek, bir PHP programının yazılım ölçütlerine göre istatistiğini gösteren bir XML örnek şemasıdır. Bu şemada, projede bulunan paket sayısı (nop), sınıf sayısı (noc), metot sayısı (nom), toplam kod satırı (loc) gibi metrikler gösterilmektedir.

Şekil 2. PHP_Depend ile yazılım ölçütleri istatistiğinin çıkartılması.

PHP_Depend programı yazılım kalite ölçütlerini derecelendiren bir matris yapısı da sunmaktadır. Derecelendirmenin çeşidine göre hangi yazılım kalite ölçütünün iyileştirilmesi ya da hangi yazılım kalite ölçütünün iyi durumda olduğunu gösteren bir modeldir.

Bütün bu işlemlerin ardından hatasız ve kullanıcı ihtiyacını karşılayan yazılım projeleri Heroku [26] bulut sağlayıcısına aktarılarak yazılımın bulut servisleri olarak çalışması sağlanır. PHP tabanlı çevik yazılım geliştirmede kullanılan araçlar Tablo 3'de listelenmektedir.

Tablo 3. PHP çevik yazılım geliştirme araçları.

Programlama Dili ve Alt Yapısı	PHP (Zend, Codeigniter, Laravel)
BDD ve TDD Araçları	Behat, Codeception (BDD) ve PHPUnit
Proje Yönetim Aracı	Pivotal Tracker
Yazılım Kalite Aracı	PHP_CodeShiffer ve PHP_Depend
Kod Versiyonlama Aracı	GitHub
Bulut Servis Sağlayıcı	Heroku
Çevik Yazılım Geliştirme Yöntemi	Scrum

4.2. C#ASP.NET Tabanlı Çevik Yazılım Geliştirme

C#ASP.NET teknolojisini kullanan iki proje takımı vardı. Bu projelerdeki BDD aracı SpecFlow'dur [27]. SpecFlow Cucumber alt yapısını kullanmaktadır ve yine Gherkin dili ile öznitelikler belirtilmektedir. Adımlar ve adım tanımlamaları SpecFlow ile otomatik oluşturularak metotlar geliştirilir.

C#ASP.NET MVC ile geliştirilen projelerde test tabanlı geliştirme aracı olarak NUnit [28] kullanılmıştır. NUnit .NET ortamında test alt yapısı sağlamaktadır. Tıpkı PHPUnit gibi "`Assert`" deyimi NUnit tarafından da kullanılır. .NET ortamında birim testini tanımlayan anahtar kelimeler bulunur. Örneğin, sınıfları test etmek için

test edilen sınıf tanımlamasının başına "**`TestClass`**", metotları test etmek için ise "**`TestMethod`**" anahtar kelimeleri yazılır.

C#ASP.NET MVC ile geliştirilen projelerin yazılım kalite aracı statik kod analiz aracı NDepend'dir [29]. NDepend yazılım kalite ölçütlerine göre kodları ayrıntılı bir biçimde analiz edip durumunu gösteren ve isim uzaylarının (namespaces) bağımlılık grafiğine aktarıldığı bir araçtır.

Bu projelerde kullanılan bir diğer yazılım kalite aracı ReSharper'dır [30]. ReSharper kod kalite analizi yapma, hata ayıklama, kod kokularını (code smells) önleme ve kod standartlarına göre kod yazmayı denetleme gibi fonksiyonları bulunur. BDD, TDD ve yazılım kalite ölçme araçlarıyla geliştirilen .NET projeleri Microsoft Azure bulut sağlayıcısına aktarılır ve RESTful bulut servisleri olarak çalışır. C# ASP.NET çevik yazılım geliştirme araçları Tablo 4'te verilmiştir.

Tablo 4. C# ASP.NET çevik yazılım geliştirme araçları.

Programlama Dili ve Alt Yapısı	C# ASP.NET MVC (.NET Framework)
BDD ve TDD Araçları	SpecFlow (BDD) ve NUnit (TDD)
Proje Yönetim Aracı	Pivotal Tracker
Yazılım Kalite Aracı	NDepend ve ReSparper
Kod Versiyonlama Aracı	GitHub
Bulut Servis Sağlayıcı	Heroku
Çevik Yazılım Geliştirme Yöntemi	Microsoft Solutions Framework Team

4.3. Java EE Tabanlı Çevik Yazılım Geliştirme

Java EE tabanlı çevik yazılım geliştirme, sadece bir proje gurubu tarafından tercih edilmiştir. Spring yazılım çatısı ile geliştirilen çevik uygulama BDD, TDD ve yazılım kalite araçlarıyla desteklenmiştir. Çok gelişmiş bir mimariye sahip olan Spring çatısı, üçüncü parti araçlarla entegre bir biçimde çalışmakradır. BDD ile Spring çatısını birleştiren araç JBehave'dir [31]. JBehave açık kaynak tabanlı bir BDD aracıdır. JBehave ile beş adımda Spring çatısı altında BDD uygulanmaktadır:

- Kullanıcı hikayesi yazma. (Gherkin dili ile)
- Kullanıcı hikayelerini Java metotlarıyla eşleştirmek.
- Kullanıcı hikayelerini yapılandırmak.
- Kullanıcı hikayelerini çalıştırmak.
- Sonuçları raporlamak.

Java projesinin TDD aracı JUnit'tir [32]. JUnit diğer çevik yazılım modellerinde yer alan TDD araçlarının en gelişmiş olanıdır. Diğer TDD araçları JUnit modeli temel alınarak geliştirilmiştir. JUnit üçüncü parti araçlarla iç içedir, bu yüzden gelişmiş bir kütüphane desteği ve geri bildirim seçenekleri mevcuttur. Bu projede kullanılan yazılım kalite ölçme aracı SonarQube'dur [33]. SonarQube bir açık kaynak platform olup sürekli olarak kod kalitesini ölçmektedir. SonarQube'ün kendi sunucusu olup veri tabanındaki Java kodlarını inceleyerek yazılım ekibine geri bildirimde bulunur. Ayrıntılı bir rapor sunan SonarQube kullanımı kolay bir altyapı sağlamaktadır. Tablo 5'de Java EE çevik yazılım geliştirme araçları verilmiştir.

Tablo 5. Java EE çevik yazılım geliştirme araçları.

Programlama Dili ve Alt Yapısı	Java EE (Spring Framework)
BDD ve TDD Araçları	JBehave (BDD) ve JUnit (TDD)
Proje Yönetim Aracı	Pivotal Tracker
Yazılım Kalite Aracı	SonarQube
Kod Versiyonlama Aracı	GitHub
Bulut Servis Sağlayıcı	Heroku
Çevik Yazılım Geliştirme Yöntemi	Scrum

5. Sonuçlar

Bu çalışmada bir Yazılım Mühendisliği dersinde 3 yıl öğrenci projelerinde, çevik yazılım geliştirme modeli temel alınarak, BDD/TDD teknikleri kullanılarak ve çeşitli programlama dilleri ve altyapıları ile geliştirilen SaaS uygulamalarından kazanılan tecrübe sunulmuştur. Elde edilen en önemli sonuçlar özetle şu şekilde sıralanabilir:

Fonksionel testlerde kullanılan Cucumber, Ruby programlama dili ile geliştirildiği için Ruby/Rails ortamı BDD'yi güçlü olarak sunan ilk ortam olmuştur. Cucumber temel alınarak diğer programlama dilleri ve ortamları için BDD alt yapıları geliştirilmiştir. Son yılda Ruby kullanmayan 3 takımın projelerinde BDD'nin bu programlama dilleri ve ortamları tarafından da acil yazılım geliştiren takımlara sağlandığı görülmüştür.

İlk iki yıl sadece Ruby/Rails dili ve ortamı ile projelerin gerçekleştirilmesi istenmiş ve bu ortamdaki test ve kalite araçları ile bir SaaS uygulamasının çok hızlı geliştirilip Heroku ortamına taşındığı görülmüştür. Heroku ilk yıllar sadece Ruby/Rails için ücretsiz yazılım platformu sunuyordu. Son yıldaki projelerde PHP ve Java projeleri için yine ücretsiz destek verdi. Heroku bulut servis sağlayısının özellikle başlangıç düzeyinde ve öğrenci projeleri için iyi bir PaaS servis sağlayıcı olduğu görüldü.

Öğrencilerden istenen bir gereksinim de sunucu tarafında MVC şablonuna göre yazılımlarını yapılandırmalarıydı. MVC Rails tarafından çok güçlü sağlanmaktaydı. MVC desteğinin C#, Java ve PHP programlama dilleri için de değişik yazılım alt yapılarıyla sağlandığı görüldü.

Diğer bir teknolojik tasarım isteği, sunucuda ilk kısımda (Controller katmanı) HTTP isteklerini ele alan RESTful servislerin yazılmasıydı. Sınıfta Ruby örnekleri öğrencilere sunulda, onlar kendi RESTful servislerini seçmiş olduğu dillerde ve yazılım alt yapılarında geliştirdi.

İlk iki yılda sadece acil yazılım geliştirme temel alınmıştı ve öğrencilere klasik Yazılım Mühendisliği derslerinde sunulan ve günümüzde bir çok projede mühendislerden istenen iş analistliği, gerekler mühendisliği, dokümantasyon ve planlama gibi önemli konular verilmemişti. Bu konuların öğrencilere öğretilmemesinin eksikliğini daha öğrencilerimiz mezun olup iş hayatına atılmadan son sınıf bitirme projelerinde iki yıl hemen gördük. Yazdıkları tezler çok zayıf oldu. Yazılım Mühendisliğinin planlama ve dokümantasyon yönünü öğrenemediler. Yazılım Mühendisliğinin klasik konularının da derse dahil edilip en başta projelerin

klasik yöntemlerle başlayıp BDD ve TDD ile acil yazılım geliştirmeyle devam etmesiyle ortaya melez bir model çıktı. Kanaatimizce bu tür hem klasik Yazılım Mühendisliği konularının öğretilip bunların projelerde kullandırılması ve acil süreçlerle buluta taşınacak SaaS uygulamalarının geliştirilmesi, öğrencilere günümüz teknolojilerini kısa sürede öğrenmelerini sağlayacaktır.

İlk iki yıl derste öğrencilere 2-3 hafta Ruby/Rails öğretilmişti. Son yılki Ruby/Rails örnekleri yine derste verildi, ama uzun zaman ayrılarak ve laboratuvarda asistan desteğiyle Ruby/Rails öğretilmedi. Son yıl bir proje takımı kendileri Ruby/Rails'i öğrendi. Java, C# ve PHP'yi seçen takımlar bu programlama dillerini biliyorlardı. Bir Yazılım Mühendisliği dersindeki projeler için genellikle yeni bir programla dili öğretilmez; öğrenciler hakim oldukları ve rahat kullandıkları bir programla dilini ve ortamını kullanır. Bizim üç yıllık uygulamalarımızda, Ruby/Rails'in 3-4 haftada öğretilmesinin mümkün olduğu ve hatta son yılda öğrenciler tarafından kendilerince öğrenilebileceği ve SaaS uygulamaları geliştirmede kullanılabileceği görüldü.

Son yıl öğrenciler projelerin ortak teknik gereksinimleri olan MVC'ye göre yazılımlarını yapılandırma, RESTful servis geliştirme, BDD/TDD teknolojilerini kullanma, yazılım kalitesini ölçmeyi seçtikleri dillerde ve platformlarda kendileri öğrendi. Projeyi gerçekleştirme teknolojisinden bağımsız olarak, BDD/TDD acil yazılım geliştirme yöntemlerinin verilebileceği görüldü.

Öğrenciler dersimizde geliştirmiş oldukları projeler ile Yazılım Mühendisliğinin bir çok konusunu günümüz teknolojileri için uyguladıklarından çok önemli bilgi ve tecrübeye daha okul projeleriyle kazanmış oldular. Öğrencilerin bu şekilde verilen Yazılım Mühendisliği dersi uygulamalarıyle kazandıkları birikimlerinin mezun olduktan sonra çalışma hayatında çok faydalı olduğuna dair öğrencilerimizden birçok geri dönüşüm aldık. Ancak öğrencilerin bir kısmı diğer derslerinin yanında bu dersteki verilen bilgilerin çok ve uygulamaların ağır olduğunu belirtip şikayet ediyorlardı. Bu tek ders içeriğinin iki derse yayılarak bir yıl boyunca dersteki konuların verilmesi, öğrencileri rahatlatacağı için daha iyi sonuçlar alınabilir.

Kaynaklar

1. Başar A., Özkaya A., Kesgin F.: Yazılım Geliştirme Süreçlerinde Şelale Yönteminden Çevik Yaklaşıma Geçiş: Bir Teknoloji Şirketinde Uygulama, 9. Ulusal Yazılım Mühendisliği Sempozyumu (2015).
2. Beck, Kent ve diğerleri: Principles Behind the Agile Manifesto, Adres: http://www.agilemanifesto.org/principal.html, Erişim: 21.05.2016
3. Nalbant B., Bıçakçı M.: Savunma Projelerinde Çevik Metodolojiler, 9. Ulusal Yazılım Mühendisliği Sempozyumu (2015).
4. North D.: Introducing BDD, Better Software Magazine (2006).
5. Software Requirements, 3rd Edition, Wiegers K., Beatty J., Microsoft Press (2013)
6. Software Engineering Frameworks for the Cloud Computing Paradigm. Editors: Mahmood Z., Saeed S. (Eds.), Springer (2013).
7. What is Scrum? An Agile Framework for Completing Complex Projects. Scrum Alliance: http://www.scrumalliance.org/why-scrum, Erişim: 22.05.2016.
8. Wagh R.: Using Scrum for Software Engineering Class Projects. Agile India pp.68-71 (2012).

9. Zorzo S. D., Ponte L., Lucredio D., Using Scrum to Teach Software Engineering: A Case Study, IEEE Frontiers in Educatıın Conference, pp 455-461 (2013).
10. Krishna R., Jayakrishnan R.: Impact of Cloud Services on Software Development Life Cycle. Software Engineering Frameworks fort he Cloud Computing Paradigm. Editors: Mahmood Z., Saeed S. (Eds.) Springer, pp. 79-99 (2013).
11. Tsai W., Hang Y., Shao Q.: EasySaaS: A SaaS Development Framework, IEEE International Conference on Service Oriented Computing and Applications, pp 1-4 (2011).
12. Engineering Software as a Service: An Agile Approach Using Cloud Computing. First Edition, Fox A., Patterson D., Strawberry Canyon LLC (2014).
13. Latorre R.: Effects of Developer Experience on Learning and Applying Unit Test-Driven Development, IEEE Transactions on Software Engineering, Vol. 40, No: 4, April (2014).
14. Wilkerson J. W., Nunamaker Jr. J. F., Mercer R.: Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development, IEEE Transactions on Software Engineering, Vol. 38, No: 3, May/June (2012).
15. Nagappan N., Maximilien M. E., Bhat T., Williams L.: Realizing Quality Improvement through Test-Driven Development Results and Experiences of Four Industrial Teams, Emprical Software Engineering, Vol. 13, No. 3, pp. 289-302 (2008).
16. Üçel Ö.: Scrum ve Kullanıcı Hikayeleri, https://prezi.com/bqixbxztkb8i/scrum-ve-kullanc-hikayeleri, Erişim: 23.05.2016.
17. Pivotal Tracker: http://www.pivotaltracker.com/
18. Cucumber: https://cucumber.io/
19. http://www.gumuskaya.com/Teaching/Courses/COM401-2015/com401.htm
20. Behat: http://docs.behat.org/en/v3.0/
21. Gherkin: https://cucumber.io/docs/reference
22. Codeception: http://codeception.com/
23. PHPUnit: https://phpunit.de/
24. PHP_ CodeSniffer: http://pear.php.net/package/PHP_CodeSniffer/redirected
25. PHP_ Depend: https://pdepend.org/
26. Heroku: https://www.heroku.com/
27. SpecFlow: http://www.specflow.org/
28. NUnit: http://www.nunit.org/
29. NDepend: http://www.ndepend.com/
30. ReSharper: https://www.jetbrains.com/resharper/
31. JBehave: http://jbehave.org/
32. JUnit: http://junit.org/junit4/
33. SonarQube: http://www.sonarqube.org