Avrupa Bilim ve Teknoloji Dergisi
Özel Sayı 24, S. 385-391, Nisan 2021

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
Special Issue 24, pp. 385-391, April 2021

Research Article

Yaygın Kullanılan Çevik Yöntemlerin Küçük Ölçekli Bir Uygulamanın Geliştirilmesi Sürecinde Değerlendirilmesi Üzerine Bir Çalışma

Ali Murat Tiryaki[1*]

[1*] Çanakkale Onsekiz Mart Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Çanakkale, Türkiye, (ORCID: 0000-0001-8224-6319), tiryaki@comu.edu.tr

(2nd International Conference on Access to Recent Advances in Engineering and Digitalization (ARACONF)-10–12 March 2021)

(**DOI:** 10.31590/ejosat.902178)

ATIF/REFERENCE: Tiryaki, A.M. (2021). Yaygın Kullanılan Çevik Yöntemlerin Küçük Ölçekli Bir Uygulamanın Geliştirilmesi Sürecinde Değerlendirilmesi Üzerine Bir Çalışma. *Avrupa Bilim ve Teknoloji Dergisi*, (24), 385-391.

Öz

Yazılım yaşam döngüsünde dış etkenlerden gelecek değişikliklerin kaçınılmaz olduğunu kabul ederek bu değişikliklere daha hızlı ve daha kolay cevap verilebilmesini hedefleyen çevik yazılım geliştirme modelinin ticari sektördeki kullanımı giderek artmaktadır. Çevik geliştirme yaklaşımının tanıtılmasından sonra bu modeli temel alarak farklı bakış açıları ile çözümler sunan pek çok çevik yöntem önerilmiştir. Yöntemlerin belirli bir proje için uygunluğu projenin tipi, büyüklüğü, geliştirme ekibinin büyüklüğü ve deneyimi, organizasyonel etkenler gibi parametreler değerlendirilerek belirlenmelidir. Çevik yöntemlerin sayısının artması ile, projeler için bu yöntemlerin uygunluğunun belirlenerek uygun yöntemin seçimi önemli bir konu haline gelmiştir. Bu çalışmada çevik yöntemler arasında en yaygın olarak kullanılmakta olan Aşırı Programlama (Extreme Programming), Scrum, Rasyonel Birleştirilmiş Süreç (Rational Unified Process – RUP) ve Kanban yöntemlerinin küçük ölçekli projelerde kullanımının değerlendirilmesi hedeflenmiştir. Bu hedef doğrultusunda bu dört yöntem bir yükseklisans dersi bünyesinde proje olarak geliştirilen küçük çaplı bir uygulamanın geliştirilmesinde farklı ekipler tarafından deneyimlenerek belirli parametrelere göre karşılaştırılmıştır.

Anahtar Kelimeler: Yazılım mühendisliği, Yazılım geliştirme yöntemleri, Evrimsel geliştirim, Çevik süreçler.

A Study on the Evaluation of Commonly Used Agile Methodologies in the Development Process of a Small-Scale Application

Abstract

The use of the agile software development model which aims to respond faster and easier to these changes, accepting that changes in the software life cycle are inevitable due to external factors, is gradually increasing in the sector. After the introduction of the agile development approach, many agile methods that offer solutions with different perspectives based on this model have been proposed. The suitability of the methods for a specific project should be determined by evaluating parameters such as the project's type, size, size and experience of the development team, and organizational factors. With the increasing number of agile methods, determining the suitability of these methods for projects and choosing the appropriate method has become an important issue. In this study, it is aimed to evaluate the use of the most common four agile methods in small-scale projects. These methods are Extreme Programming, Scrum, Rational Unified Process (RUP) and Kanban. In line with this goal, these four methods were experienced by different teams in the development of a small-scale application developed as a project within a graduate course and compared according to certain parameters.

Keywords: Software engineering, Software development methodologies, Evolutionary development, Agile processes.

[*] Sorumlu Yazar: tiryaki@comu.edu.tr

1. Giriş

Çevik yazılım geliştirme yaklaşımı [1] yazılım geliştirme süreci içerisindeki aktivitelerin yinelemeli (iterative) ve artışlı (incremental) [2] bir yol içerisinde yerine getirilmesi ile yazılımın evrime açık bir şekilde geliştirilmesini hedefler. Bu yaklaşımda yazılım Çevik Birliği (Agile Alliance) tarafından tanımlanmış olan temel değerler ve bu değerler üzerine tanımlanmış çevik pratikler kullanılarak geliştirilmektedir.

Çevik yazılım geliştirme modelinin kökeni 1957 yıllarındaki IBM'deki yazılım geliştirme çalışmalarına dayanmaktadır. Bu çalışmalar daha sonra E. A. Edmonds [3]'de artırımlı yazılım geliştirme yaklaşımını tanımlamıştır. 2001 yılında Kent Beck öncülüğündeki bir ekip tarafından çevik yazılım geliştirme yaklaşımının temel prensiplerinin tanımlandığı Çevik Manifesto [4] yayınlanarak yazılım geliştirmede çevik süreçler resmi olarak tanıtılmıştır.

Çevik Manifesto yazılım geliştirme yaşam döngüsü içerisinde bir plana sıkıca bağlı kalmak yarine değişime karşılık vermenin daha önemli ve öncelikli olduğu vurgulamaktadır. Çevik pratikler gereksinimlerdeki değişikliklere hızlı karşılık verebilmenin yanı sıra, döngüsel geliştirme yaklaşımı sayesinde risklerin geliştirme sürecinin erken safhalarında tespit edilebilmesine imkân sağlamakta, pazara çıkış süresini kısaltmakta ve yazılım kalitesini arttırmaktadır [5].

Çevik Manifesto ile, şelale modelde değişiklik maliyetinin yüksekliğinin ortaya çıkardığı olumsuz etkilerden etkilenmekte olan yazılım şirketleri yazılım geliştirme projelerinde çevik yaklaşımı tercih etmeye başlamış ve çevik süreçlerin ticari hayattaki kullanımı giderek yaygınlaşmıştır. Günümüzde çevik yaklaşım yazılım geliştirme alanı içerisinde en çok tercih edilen model haline gelmiştir.

Literatürde çevik yaklaşımı temel alan Aşırı Programlama (Extreme Programming) [6],[7], SCRUM [8], Rasyonel Birleştirilmiş Süreç (Rational Unified Process - RUP) [9], Kanban [10] gibi pek çok çevik yöntem tanıtılmıştır. Her bir yöntem desteklediği çevik pratikler ile çevik yaklaşımın 4 temel ilkesini tanımladıkları sistematik bir yol içerisinde kullanarak geliştirme sürecini kolaylaştırmayı ve hızlandırmayı amaçlamaktadır. Aşırı Programlama gibi bazı yöntemler kodlama faaliyetleri üzerinde odaklanarak kodlama tarzına etki ederken SCRUM gibi bazı yöntemler ise yönetimsel faaliyetler üzerine odaklanmaktadır. Yöntemlerin farklı perspektifleri ve destekledikleri farklı çevik pratikler şirketlerin yönetim tarzı, organizasyonel yapı, personel yapısı ve mevcut proje gereksinimlerine göre kendilerine uygun olan geliştirme yöntemine karar verme gereksinimini ortaya çıkarmaktadır.

Bu çalışmada yaygın olarak kullanılmakta olan RUP, Aşırı Programlama, SCRUM ve Kanban yöntemleri bir yüksek lisans dersinde oluşturulan ikili gruplar tarafından bir yazılım sisteminin geliştirilmesi için deneyimlenmiştir. Deneysel yöntem olarak; başlangıç yazılım gereksinimleri tanımlanmış ve dört adet proje grubundan aynı gereksinimleri kendilerine atanan geliştirme yöntemi olarak geliştirmeleri istenmiştir. Başlangıç gereksinimlerini sağlayacak ilk sürümün tamamlanmasına yakın sisteme yeni gereksinimler eklenerek yöntemlerin var olan tasarımı önemli derecede etkileyen yeni gereksinimlere cevap verebilirliği izlenmiştir. Çalışma sonucunda, ilgili dört yöntem elde edilen araştırma bulguları test sayısı, yönetimsel aktiviteler, gereksinim değişikliklerine cevap verebilme maliyeti gibi ölçütler kullanılarak karşılaştırılmıştır.

Bildirinin bir sonraki bölümünde çevik geliştirim süreci içerisinde yer alan temel aşamalar genel karakteristikleri ile incelenmektedir. Üçüncü bölümde bildirinin konusu olan dört yöntemin örnek uygulamanın geliştirimi sırasında farklı ekipler tarafından kullanılması ve temel aşamaların ilerleyişi ile ilgili detaylar verilmektedir. Dördüncü bölümde ise, elde edilen araştırma bulguları dört yöntemin belirli ölçütlere göre karşılaştırılması şeklinde tartışılmaktadır.

2. Çevik Yazılım Geliştirme Süreci Aşamaları

Çevik yazılım geliştirme süreci temel olarak araştırma, planlama, üretim, bakım ve son olmak üzere 5 aşamadan oluşmaktadır [11]. Çevik geliştirim yöntemlerinin her biri farklı faaliyetler ile bu temel aşamaları gerçekleştirmektedir. Bu bölümde bu aşamaların genel karakteristikleri incelenmektedir.

Araştırma: Araştırma aşaması geliştirim sürecinde üretimden önceki ilk aşamadır. Bu aşamada projenin ne olduğu, sınırlarının ne olacağı, nelerin yapılabilinip nelerin yapılamayacağı, müşterinin tama olarak ne istediği gibi sorulara müşteri ile sürekli iletişim kurularak cevap aranır.

Araştırma aşaması içerisinde ekip üyeleri sistemin üretimi içerisinde kullanılması düşünülen teknolojileri deneyimler. Aynı zamanda kullanılacak bu teknolojilerin performans limitleri hakkında deneyim sahibi olmalıdırlar. Eğer ekip üyeleri kullanılacak teknolojiler konusunda deneyimli ise araştırma aşaması birkaç hafta gibi kısa bir sürede tamamlanabilir. Takım kullanılacak teknolojilere veya ilgi alana yabancı ise bu aşama için birkaç ay devam edebilir.

Bu adım, müşterinin ilk sürümün yeteneklerinden tatmin olmadığı ve geliştirici takım üyelerinin sistemin gerçekleştirilmesi sırasında nelerin çıkabileceğini hesaplayamadığı sürece devam eder.

Planlama: Planlama aşamasının amacı müşteriler ve geliştiricilerin belirlenen senaryoların en küçük ve tam olarak bitmiş setlerinin hazır olacağı tarihler üzerinde anlaşmalarıdır. Planlama oyunu pratiği bunu yapmak için kullanılabilir. Bu aşama sonucunda sürümler, bu sürümlerin planlanan bitiş tarihleri ve bu sürümlerin gerçekleştirilmesi için planlanan iterasyonları içeren bir plan elde edilir.

İlk sürümün süresi iki ile altı ay arasında planlanmalıdır. Bundan daha kısa bir süre gerçekleştirim riskini arttırır ve önemli ticari problemlere sebep olabilir. Birkaç aydan daha uzun bir süre ise geliştirim risklerini artırır.

Üretim: Üretim aşamasında, planlanan sürümlere ulaşılabilmesi için kısa süreli iterasyonlar gerçekleştirilir. Bu iterasyonları gerçekleştirim süresi 1-4 hafta veya buna yakın bir süre olarak grup yöneticisi tarafından belirlenir. Her iterasyonda daha önceki adımda gerçekleştirilen veya tasarlanan bileşenler daha uygun bir biçime getirilmeye çalışılır. Bu iterasyonlar sonucunda ortaya çıkacak sistem üzerinde çalışılan sürümün gereksnimleri için en uygun çözümü sunan sistem olmalıdır.

Her iterasyon bu iterasyon için planlanmış senaryoların her biri için fonksiyonel test durumlarının bir setini üretir. İlk iterasyon geliştirilmekte olan yazılım için temel bir mimari ortaya koyar. Bu yüzden ilk iterasyonlarda tüm sistem için önemli ve kritik olan senaryolar seçilmelidir. Sonraki

iterasyonlar için senaryo seçiminde ise müşterinin eğilimi ön planda tutulmalıdır. Bu eğilimi anlayabilmek için müşteriye "Sizin için bu iterasyon içerisinde çalışacak en önemli şey nedir?" sorusu sorulur.

Tüm bu iterasyonlar daha önceden hazırlanmış olan plan doğrultusunda gerçekleştirilir. Eğer planda bir sapma olacağı ortaya çıkarsa planda senaryoların eklenmesi, çıkarılması veya alanlarının değiştirilmesi gibi değişiklikler yapılabilir. Üretimin son iterasyonlarında proje başlangıcında 2-3 hafta olan iterasyon süreleri 1 hafta gibi sürelere kısaltılır. Geliştirim ekibinde kimin ne üzerinde çalıştığının herkes tarafından bilinmesi için günlük ayak üstü toplantılar düzenlenir.

Üretimin son aşamalarında yazılımın değişmesine neden olacak adımlar yavaşlar. Bu yazılımın gelişmesinin biteceği anlamına gelmez. Ama ortaya çıkan değişiklik geresinimlerinde yapılacak geliştirmenin bu sürüm içerisinde değişiklik yapılmasını gerektirecek kadar önemli olup olmadığı düşünülmelidir. Bu aşamada gerçekleştirilen değişikliğin ortaya çıkaracağı riskler projenin başlangıç aşamlarındakine göre çok daha fazladır.

Bakım: Bakım çevik yöntemler kullanılan bir yazılım geliştirme projesi için en önemli aşamadır. Çevik geliştirmede projeye geliştirim boyunca sürekli olarak var olan sistemin çalışması korunurken sisteme yeni işlevsellikler eklenerek devam edilir. Bakım ve evrim tüm proje geliştirim süresince baştan sona kadar devam eder. Bu yüzden dolayı çalışan sistemin güvenliğinin sürekli olarak garanti altına alınması ve korunması gerekmektedir. Bunun için test güdümlü geliştirme ve yeniden yapılandırma gibi teknolojiler kullanılmalıdır.

Proje boyunca ortaya çıkabilecek değişiklikler sadece müşteriden gelen gereksinim değişiklikleri değildir. Sürümler arasında geliştirim ekibinde veya kullanılan teknolojilerde değişiklikler meydana gelebilir. Daha önceki sürümde oluşturulan yapı içerisinde yapılması gerektiği sonradan görülen değişiklikler için büyük yeniden yapılandırmalar yapılması gerekebilir. Yeni mimarisel fikirler ortaya atılabilir.

Değişiklikler proje planında değişiklik yapılmasına neden olabilir. Dolayısıyla çevik geliştirimde yazılım sürekli olarak evrilirken aynı şekilde proje planı da geliştirim süreci boyunca evrilmektedir.

Son: Müşterinin yeni senaryolar ile gelmemesi projede son aşamaya gelindiğinin göstergesidir. Gelecekte sistem içerisinde değişiklik yapılmasıi gerektiğinde kullanılmak üzere programcı kataloğu, tasarım modelleri gibi dokümanlar hazırlanır.

Bir çevik yazılım geliştirme projesi olumlu veya olumsuz nedenlerde dolayı sonuçlandırılabilir. Olumlu neden müşterinin sistemden mutlu olması ve yakın gelecek için sisteme eklenebilecek bir şey bulamamasıdır. Olumsuz nedenler ise; müşterinin istediği yeni işlevselliklerin ekonomik olarak uygun olmaması, projenin zamanında tamamlanamaması ve tamamlanacağı sürenin kestirilememesi gibi nedenlerdir.

3. Yöntemlerin İlerleyişi

Bu bölümde CityCard uygulamasının geliştirilmesi sırasında çevik geliştirim sürecinin temel aşamalarında karşılaştırılmakta olan dört yöntemin ilerleyişi incelenmiştir.

Bu bölümde CityCard uygulamasının geliştirilmesi için kurulan ve herbiri bu çalışmada karılaştırılmak istenen yöntemlerden birini bu sistemin geliştirimi için kullanan proje ekiplerinin çevik geliştirim sürecinin temel aşamalarındaki ilerleyişleri incelenmektedir. Bu 5 temel aşamadan araştırma, planlama ve son aşamaları tüm proje grupları tarafından ortak gerçekleştirilmiştir. Diğer aşamalar ise her bir proje grubu için kullanılan yönteme özgü faaliyetler yerine getirilerek gerçekleştirilmiştir.

3.1. Araştırma

CityCard projesinin araştırma aşaması tüm yöntemler için ortak olarak yapılmıştır. Bu aşama müşteri ile güçlü bir iletişim gerektirmektedir. Projede bir müşteri olmadığından yükseklisans dersini veren akademisyen müşteri rolünü üstlenmiştir. Bu adımda müşteri istekleri belirlenerek sistem için bir vizyon oluşturulmuştur. Uygulamanın kapsamı ve sınırları belirlenmiş, fonksiyonel gereksinimler üzerinde fonksiyonel olmayan gereksinimler tanımlanmıştır. Teknik fizibilite çalışması yapılarak özellikle donanımsal parçalar içeren kart okuyucu konsolounun simule edilmesi için çözümler değerlendirilmiştir.

Geliştirilmek istenen projenin boyutunun küçük olması, ekip üyelerinin de kullanılacak teknolojilerde deneyim ve bilgi sahibi olması nedeniyle projede araştırma aşaması bir hafta gibi kısa bir sürede tamamlanmıştır.

Araştırma aşaması sonucunda CityCard uygulaması için kullanıcı gereksinimlerini, rolleri ve kısıtlamaları içeren bir vizyon elde edilmiştir.

3.1.1 CityCard: Şehir İçi Ulaşım Kartı Otomasyonu Vizyonu

CityCard şehir içi toplu taşıma araçlarında kullanılan bir otomasyon sistemidir. Toplu taşıma araçlarını kullanmak isteyen yolcular şehrin belli yerlerine yerleştirilmiş olan kart dolum merkezlerinden kartlarına kontör yükleyerek bu kontörler ile araçlara binebilmektedirler. Yolcu araca binip araçtaki konsola kartını gösterdiğinde bu ulaşım aracı için belirlenmiş olan ücret kartın bakiyesinden düşülür. Sistem şehir yönetiminin belirlediği süredeki binişleri aktarmalı biniş olarak kabul edebilecek şekilde kart hareketlerini takip edebilmelidir.

Sistem 4 adet rol içermektedir. Bunlar; kart sahibi, şoför, yükleme noktası ve yöneticidir. Her bir rol için başlangıç olarak belirlenen gereksinimler aşağıda tanımlanmıştır.

Kart Sahibi rolü 4 farklı profil içerir. Bunlar; Tam, Öğrenci, Yaşlı ve Engelli profilleridir. Bu profillerin birbirinden farkı, kart okuma sürecinde düşülen bakiye miktarıdır. Kart sahipleri sistemi kart ile tolu taşıma araçlarına binmek ve kart numaraları ile bakiye sorgulamak için kullanırlar.

Şoförler sistemi kendileri için geliştirilmiş panellere giriş yaparak kart okuma panelinden okutulan kartların bilgilerini takip etmek için kullanacaktır. Araçlardaki kart okuma panelleri bir şoförün aynı araçtaki şoför paneline giriş yapması ile aktif hale gelmelidir. Sistem şoför ve araç bazlı raporlar sunabilmelidir.

Yükleme noktaları Citycard kartlarına bakiye yüklenebilen noktalardır. Bu noktalar sistemden belirli bir bakiye satın alır ve sisteme bağlı yükleme panellerini kullanarak bu bakiyeleri kart sahiplerine satarlar. Bir yükleme noktası satın almış olduğu bakiye tükendiğinde herhangi bir karta bakiye yükleyemez. Tekrar yükleme yapabilmek için sistemden bakiye satın alması gerekir.

Yöneticiler sistemdeki tüm kullanıcıları tanımlayan, güzergâh ve ulaşım aracına göre her bir kart sahibi profili için ulaşım ücretleri gibi sistem parametrelerini belirleyen ve güncelleme yetkisine sahip olan kullanıcılardır. Kart oluşturma, güncelleme ve iptal etme Yönetici rolünün yetkisindedir. Ayrıca Yükleme Noktaları için bakiye yükleme yetkisine sahiptir.

Sistemin ana senaryosu olan karttan bakiye düşme senaryosu şu şekilde ilerlemektedir;

- Kart sahibi binmek istediği ulaşım aracındaki kart okuma paneline kartını yaklaştırdığından ulaşım aracı veya güzergaha göre kart sahibi profili için belirlenmiş olan ücret kartın bakiyesinden düşülür. Kartın bakiyesinin bu ücret için yetersiz olması durumunda karttan bakiye düşmez, "Yetersiz Bakiye" uyarısı verilir ve şoför panelinde uyarı görüntülenir.
- Aynı kart 5 dakika içerisinde aynı otobüsteki okuma panelinden tekrar okutulmaya çalışılması durumunda sistem kartın kart sahibi olmayan başka bir kişi tarafından kullanıldığını kabul eder ve "gösterilmiş kart" uyarısı verir ve bakiye düşmez. Bu durumda, şoförün kendi panelinden seçeceği kart tipi ile bakiye düşürme işlemi gerçekleştirilebilir.
- Bir kart okuma panelinden okutulan bir citycard sonraki 5-45 dakika arasında başka bir ulaşım aracında tekrar okutulursa bu biniş veya binişler aktarma olarak kabul edilecektir. Aktarmalı binişler kart sahibi profilinin indirimli ücretine göre fiyatlandırılmalıdır.

3.2 Planlama

CityCard uygulaması geliştirim sürecinde planlama aşaması tüm ekipler ile ortak olarak yapılmış ve ortak bir iterasyon planı oluşturulmuştur. Proje için oluşturulan bu genel iterasyon planında 2 ana iterasyon yer almaktadır. Süresi 2 ay olarak belirlenmiş olan ilk iterasyon, aşağıda listelenen gereknimlerin tamamının donanımsal aygıtlar simule edilerek geliştirilmesini içermektedir. İkinci ana iterasyon ise geliştirilen işlevselliklerin donanımsal aygıtlara entegre edilmesi ve arayüzlerin geliştirilmesi çalışmalarını içerecek şekilde planlanmıştır. İkinci iterasyonun süresi 1 ay olarak belirlenmiştir. Proje başlangıcında tüm ekipler ile ortak oluşturulan genel iterasyon planı Tablo 1'de gösterilmektedir.

Tablo 1. Genel İterasyon planı

İterasyon No	Görevler	Süre
1	Kart tanımlama,	2 ay
	Karta bakiye yükleme,	
	Karttan bakiye düşme,	
	Yükleme noktası ekleme,	
	Kart yükleme noktasına bakiye yükleme,	
	Şoför ekleme,	
	Ulaşım aracı ekleme,	
	Kart bakiyesi sorgulama.	
2	Donanımsal aygıtlara entegrasyon,	1 ay
	Arayüzlerin geliştirilmesi.	

3.3. Üretim

SCRUM'da iterasyon geliştirme süreci Sprint planlama toplantısı ile başlanmıştır. Sprint'te yapılacak iş sprint planlama toplantısında planlanmıştır. Tüm SCRUM takımı planı birlikte oluşturur. SCRUM'da toplantıların gerçekleştirilmesi ve bu toplantılar için belirlenen zaman sınırına uyulması zorunludur. Bu toplantıda başlayan sprintte ürün parçası olarak ne teslim edilebilir ve bu ürün parçasını teslim etmek için gerekli iş nasıl başarılacak sorularına cevaplar arandı.

Aşırı programlamada; iterasyonların geliştirme sürecine, müşteri hikayeleri ve iterasyon planında belirlenen sınıflar için sınıf diyagramı oluşturularak başlandı. Test güdümlü geliştirim – TGG (Test Driven Development) [12],[13] pratiğine uygun olarak, kodda risk oluşturabilecek önemli noktalar belirlenerek test durumları oluşturuldu. Kullanıcı hikayelerine göre ilerleme sağlandığından her iterasyonda riskler ve gereksinimler planlama oyunu pratiği [14] çerçevesinde yapılan tartışmalar ile belirlenmiştir.

RUP metodolojisi iterasyon geliştirme sürecinde iterasyona kod yazmak ya da test senaryoları çıkarmak gibi kodlanmış yöntemler kullanmaz. RUP'un temel felsefesi her aşamada modelleme yapılması üzerine kuruludur. Bu modellerin oluşturduğu dokümantasyon genellikle sistemin müşteri ile birlikte oluşturulmuş vizyonundan sistemi kimlerin kullanacağını ve sistemde neler yapabileceğini belirten aktör-hedef diyagramı oluşturmak ile başlar. Daha sonra aktörlerin yapabileceği işleri daha açıklayıcı bir şekilde ifade eden kullanım durumu diyagramları oluşturmaya başlanır.

Kanban kartları ile ilerleyen Kanban metodolojisi görselleştirmeye önem verir. Her kart geliştirme aşamasında küçük toplantılar yapılabilir. Toplantılar gerektiğinde düzenlenir zorunluluğu yoktur. Sistem geliştirilirken karşılaşılan zorluklar takım ile paylaşılır. Kanban kartları renklerine göre önemliliğini gösterir. Kartın bir aşamada tamamlanmasından sonra diğer bölüme aktarılır. Takım çalışmasına göre bir kişinin yeterlilikleri göz önüne alınarak kart atanır.

Aşırı programlama kullanan ekipte bu yöntemin 12 temel prensibinden bir tanesi olan test güdümlü geliştirim prensibi ile kod geliştirilmesinden önce testlerin geliştirilmesi ilk aşamada bir problem oluşturdu. Proje ekibinin daha önceden test güdümlü geliştirim tekniğini kullanmamış olması nedeni ile yeni bir yaklaşım olarak anlayışın benimsenmesinde zorlandığı görüldü. İlk iterasyonun geliştirilme sürecinde bu pratiği kullanarak geliştirme yapmak için normalden uzun bir çaba ve zaman harcandı. İlk iterasyonun tamamlanması ile ikinci iterasyonda bu pratiğe daha iyi uyum sağlanarak pratiğe uygun bir şekilde geliştirme aşamalarına devam edildi. Her aşama sonucunda pratiğin faydaları arasında en önemli avantaj olan sorunsuz sürümler ortaya çıkmış oldu.

Kanban metodolojisi öncelikle kanban kartının isteklerini inceleyip daha sonra kodlamaya geçmektedir. Her kart yapılacak olan işlemin en küçük parçacığı olarak nitelendirilir. Test aşamaları kodlama işlemlerinden sonra yapılır. Test aşamalarında çıkan sorunlar tekrar kodlama aşmasına döner testlerden geçen kart tamamlanmış olarak işaretlenir ve bir sonraki karta geçiş yapılır. Test aşamasında kartın stratejik rengine bakılarak süresi belirlenir. Müşteri tarafından yapılan geri dönüşler sonrasında değiştirilecek bölümler yeni kartlara bölünür. Her geliştirme sonucunda yeni test durumları oluşturulur.

Projenin başlangıcında SCRUM metodolojisini kullanan ekip de CityCard uygulamasının geliştiriminde kodlama yöntemi olaran test güdümlü geliştirim ile ilerlemeyi tercih etmiştir. Kodlama aşamasına geçmeden önce Sprint planlama toplantısında oluşturulan görevler doğrultusunda test durumları oluşturulmuştur. Bu test durumları kullanılarak öncelikli olarak testlerin geliştirilmesi ve ardından kodların geliştirilmesi sağlanmıştır. Bu yaklaşım için geliştirici takımının deneyim eksikliği olması nedeni ile sürecin başlangıcında bu prensibin uygulanmasında zorluklar yaşanmıştır. Ancak gereksinim analizi sonucunda ortaya çıkan kullanıcı hikayelerini göz önüne alarak testlerin geliştirilmesinde faydalanılmıştır.

Aynı şekilde, RUP metodolojisi kullanan ekip de test güdümlü geliştirim tekniğini temel RUP süreci içerisine entegre ederek kullanmıştır. Ekip UML kullanım durumu diyagramı ve kullanım durumu içeriklerinin [15] tanımlanmasından sonra üst seviye entegrasyon testleri yazarak geliştirime devam etmiştir. Her kullanım durumunun başlangıçta ana başarı senaryosu için testler tanımlanarak bu testlerin yönlendirmesi ile senaryoyu gerçekleştiren kodun gerçekleştirimi yapılmıştır. Ana başarı senaryosundaki tüm testleri başarılı olmasında sonra ise alternatf senaryolardaki her senaryo için aynı süreç işletilmiştir.

3.4. Bakım

Çalışmada konu edilen geliştirme yöntemlerinin proje geliştirimi sırasında ortaya çıkan kullanıcı gereksinimi değişikliklerine cevap verebilme açısından değerlendirilebilmesi amacıyla ilk iterasyonun tamamlanmasından sonra kullanıcı gereksinimlerinde şu değişiklik ve eklemeler yapılmıştır:

1. Şoförün sadece kart okumalarını takip edebildiği şoför paneline düğmeler eklenerek kart okutma işlemlerine müdahale edebilir hale getirilmesi gerekmektedir. Bu müdahaleler ile okutulacak karttan hangi kart tipine göre bakiye düşüleceğini belirleyebilirler. Örneğin, bir öğrenci kartından tam ücret alınmasını veya tam tersini sağlayabilmektedirler. Sistemin bu yeteneği sayesinde bir kartın kart sahibi dışındaki kişiler için de kullanılabilmesi sağlanacaktır.

2. Kart okutulan panelin ve şoför panelinin birbirinden bağımsız çalışan iki farklı süreç hale getirilmesi gerekmektedir.

RUP'da bu değişiklik talebinin karşılanabilmesi için ilk sürümde kart okuma senaryosu için oluşturulmuş olan analiz ve tasarım dokümanları gözden geçirilerek çözüm için gerekli düzenlemeler yapılmıştır. Değişikliklerin özellikle etkileşim diyagramlarını önemli derecede etkilediği görülmüştür. Daha sonra, tasarım modellerinde ortaya çıkan bu değişiklikler koda aktarılmıştır.

Projenin küçük çaplı olması ve geliştirilen yazılımın boyutunun küçük olması nedeniyle, RUP yöntemini kullanan ekipteki geliştiricilerin doğrudan koda müdahale etmek yerine analiz ve tasarım dokümanlarının yeniden düzenlenmesi konusunda motivasyonlarının düşük olduğu gözlemlenmiştir. Bununla birlikte, Yazılım tasarımının bu modellerde verilen kararlar ile yalın halde oluşturulmuş olması ve tasarım ile ilgili dokümanların detaylı olması nedeniyle değişiklikler sisteme kolayca entegre edilebilmiştir. Değişikler bu şekilde yaklaşık 10 gün süren bir iterasyon şeklinde projeye dahil edilmiş, analiz ve tasarım modelleri sistem ile uyumlu olarak güncel tutulmuştur.

SCRUM yönteminde yeni gelen kullanıcı gereksinimi için öncelikle sprint planlama toplantısı gerçekleştirilmiştir. Değişiklik talebinin gerektirdiği aktarma işlevinin gerçekleştirilmesi için yapılması gerekenler ve öncelikleri belirlenerek küçük görevlere bölünmüştür. Bu görevler öncelik sırasına göre ele alınarak kodda gerekli düzenlemeler yapılarak gerçekleştirilmiştir.

Sistemin geliştirilmesi sprintlere bölünerek ve müşteri ile etkileşim halinde bulunarak gerçekleştirildiği için müşteri tarafından istenen güncelleme ve değişiklikler planlama toplantısında detayları konuşulup tartışılarak zaten değişikliğe açık olan bir sisteme aktarılırken çok fazla efor gerektirmemiştir. Değişikliklerin sisteme aktarılması SCRUM yönteminde 1 haftada tamamlanmıştır.

Kanban'da yeni gelen istek karşısında yeni bir kanban kartı oluşturmuştur. Yapılan ufak toplantıda yeni iterasyonda yapılması gerekenlere karar verilmiştir. Kodda yeniden düzenlemesi gereken sınıflar güncellenmiş ve testlere yeni gereksinimleri kontrol edecek yeni test durumları dahil edilmiştir.

Kanban yönteminde değişiklikler 2 haftalık bir iterasyon ile sisteme dahil edilebilmiştir. Bununla birlikte, değişikliklerin diğer yöntemlere göre daha fazla birimi etkilediği gözlemlenmiştir. Bu durumun geliştirim süresi boyunca çok az sayıda modelleme yapılması ve tasarımın iyileştirme çalışması yapılmaması nedeniyle oluştuğu düşünülmektedir.

Aşırı programlamada gelen değişiklik talebi bir iterasyon toplantısı ile değerlendirilmiş ve iterasyon için küçük görevler belirlenmiştir. Geliştirim boyunca test güdümlü geliştirim tekniği kullanıldığından her görev için ilk olarak test durumları tanımlanmış ve kodlanmıştır. Başlangıçta başarısız olan bu testlerin yönlendirmesi ile kodda düzenlemeler yapılarak testlerin başarılı olmasını sağlayacak kodlar elde edilmiştir.

Kodun değişmesi sırasında daha önceki iterasyonlarda oluşturulmuş olan testlerin bazılarının başarısız olduğu görülmüştür. Başarısız olan bu testler incelendiğinde bazılarının algoritma mantığındaki değişimler nedeniyle başarısız olduğu fark edilmiş ve bu testler yeni oluşturulan mantığa göre yeniden düzenlenmiştir.

Aşırı programlama yöntemini kullanan ekip, değişiklikler 1 haftalık iterasyon süreci içerisinde sisteme dahil edebilmiştir. Bu 1 haftalık bir zaman diliminde yeni isteği sisteme entegre edecek uygun test senaryoları ve var olan test senaryolarının güncelleme gerektirenleri için gerekli güncellemeler yapılarak yeni istek sisteme entegre edilmiştir.

3.5 Son

Başlangıçtaki vizyona göre değişikliklerin gerçekleştirildiği üçüncü iterasyon projenin 13.-14. haftaları arasında tüm ekipler tarafından tamamlanmıştır. Bu iterasyonun tamamlanması ile proje için tanımlanmış olan tüm kullanıcı gereksinimleri yerine getirilerek sistem donanımsal aygıtlara entegre bir şekilde çalışır duruma getirilmiştir. Tüm çalışma ekiplerinde proje boyunca elde edilmiş olan doküman, model ve testler gerektiğinde yeniden kullanılabilecek şekilde depolanmıştır.

4. Değerlendirme

CityCard projesi geliştirilirken Aşırı Programlama, SCRUM, Kanban ve RUP yazılım geliştirme metodolojilerinde her bir iterasyonda geliştirilen test sayıları ve oluşturulan sınıfların sayısı Tablo 2'de karşılaştırmalı olarak görselleştirilmiştir.

Tabloyu incelediğimizde aşırı programlama yöntemini kullanan ekibin oluşturduğu testlerin sayısının diğer ekiplerinkine göre kayda değer şekilde yüksek olduğu göreülmektedir. Bunun sebebinin aşırı programlamanın test güdümlü geliştirim tekniğini geliştirimin odağına koyması ve model odaklı dokümantasyon yerine testlerin yazılım geliştirmenin her aşamasında oluşturulması üzerine kurulu olmasıdır.

Tablo 2. Test sayıları

Yazılım Geliştirme Yöntemi	1. İterasyon			2. İterasyon			3. İterasyon		
	Test Sayısı	Sınıf Sayısı	Süre (Gün)	Test Sayısı	Sınıf Sayısı	Süre (Gün)	Test Sayısı	Sınıf Sayısı	Süre (Gün)
Aşırı Programlama	32	8	40	42	9	20	55	11	7
SCRUM	23	12	52	32	14	24	41	16	7
RUP	19	9	61	28	12	30	31	13	10
KANBAN	10	4	46	19	5	25	32	8	14

RUP yöntemini kullanan ekibin oluşturduğu test sayısı daha az olmakla birlikte, daha önceki süreçlerde geliştirilmiş olan gereksinimler üzerinde değişiklikler gerektiren üçüncü iterasyonda test ve sınıf saysındaki değişimin az olmasının bu değişikliklerin sistem tasarımını ve testlerini çok fazla etkilememiş olduğu şeklinde yorumlamaktayız. Bu durum RUP yöntemi ile değişikliklerin sisteme yayılmasını önleyen, daha kaliteli bir tasarımın elde edildiğini göstermektedir.

Kanban yönteminin kullanan ekip hiçbir modelleme çalışması yapmadan kod gerçekleştirimine başlamıştır. Bu eğilimi tetikleyen unsurun projenin ölçeğinin oldukça küçük olması ve gereksinimlerin gerçekleştirim detaylarının çoğunun başlangıçta belirlenebiliyor olması olduğu düşünülmektedir. Doğrudan kod gerçekleştirmeye başlamak Kanban ekibinin ilk iterasyonu diğer ekiplere göre daha kısa bir sürede tamamlayabilmesini sağlamıştır. Bununla birlikte, kullanıcı gereksinimlerinin değiştiği üçüncü iterasyon sonunda test ve sınıf sayısının önemli derecede artış gösterdiği gözlemlenmiştir. Bu durumun nedeni, değişikliklerin modüler olmayan sistem tasarımında tüm bileşenlere yayılmış olmasıdır.

İterasyonların tamamlanma sürelerine baktığımızda RUP yönteminin diğer yöntemlere göre kayda değer şekilde yavaş kaldığını görülmektedir. RUP'un geliştirim süreci boyunca birbirinin girdisi olarak kullanılan analiz ve tasarım modellerinden oluşan detaylı modelleme gereksinimi küçük ölçekli projelerde zaman açısından bir deavanataj oluşturabilmektedir. Geliştirilen yazılımın büyüklüğü arttıkça, bu modelleme mantığı içerisinde elde edilen tasarım karalarının kaliteli tasarım oluşturmadaki avantajları görülebilmektedir. Üçüncü iterasyonun tamamlanma sürecine baktığımızda bu etkiyi az da olsa görebilmekteyiz. Orta ve büyük ölçekli projelerde bu etkinin çok daha fazla olacağı açıktır.

Aşırı programlama kullanan ekip iterasyonlar basit bir sınıf diyagramı oluşturarak başlamıştır. İterasyonlar bu diyagramlardaki sınıfların kodlanması ile devam ettirilmiştir. Aşırı programlamnın dokümantasyondan çok ekip içerisindeki iletişime önem veren felsefesi bu tür küçük ölçekli projelerde oluşturduğu çeviklik ve hızlı versiyon elde etme avantajları bu projede de gözlemlenmiştir. Geliştirim aktivitelerinin odağına test güdümlü geliştirimi koyan bu ekip hem ietrasyonlaeın tamamlanma sürelerinde en kısa süreleri yakalamış hem de değişikliklerin ortaya çıktığı üçüncü iterasyonda bu değişikliklere hızlı cevap verebilecek ve değişikliklerin sistem geneline yayılmasını engelleyecek tasarımı yakalamayı başarabilmiştir. Yazılım kodunun sürekli yeniden yapılandırmaya tabi tutulması sayesinde bu ekibin ikinci iterasyonun sonudna elde ettiği sade tasarımın değişikliklerin ortaya çıkardığı zaman ve efor maliyetini en aza indirebildiği gözlemlenmiştir.

SCRUM yöntemini kullanan ekip başlangıçta aşırı programalama ile benzer bir geliştirim yolu tercih ederek test güdümlü geliştirim ile gerçekleştirime başlasa da projenin ilerleyen aşamalarında bu tekniği devam ettiremediği görülmüştür. Bu durumun nedeninin ekibin test güdümlü geliştirim pratiğini yeniden yapılandırma, ikili programlama, metafor gibi diğer pratiklerle destekleyememiş olması olarak değerlendirilmiştir. Bununla birlikte SCRUM yöntemi önerdiği Sprint toplantıları ve günlük toplantılar ile ekibi oluşturan iki kişi arasındaki iletişimi sürekli güçlü tutabilmiştir. Bu sayede test güdümlü geliştirim yapılamamasının oluşturabileceği dezavantajlar önlenebilmiştir. Ekip güçlü iletişim ile üçüncü iterasyondaki değişikliklere hızlı cevap verebilmiştir.

Kanban yönteminde iterasyonlardaki görevler küçük görevlere bölünerek bu görevler için Kanban kartları oluşturulmuştur. Kartlar ile belirlene bu küçük görevler ekip üyeleri tarafından herhangi bir modelleme çalışması yapılmadan doğrudan kodlanarak gerçekleştirilmiştir. Bu geliştirim tarzı ilk iterasyonun hızlı bir şekilde tamamlanmasına imkân sağlasa da tasarımı iyileştirmek için çalışma yapılmamış olması modüler olmayan bir tasarımın ortaya çıkmasına neden olmuştur. Kalşiteli bir tasarım elde edilememesinin nedeninin ekip üyelerinin birbirleriyle iletişimi sürekli tutmadan birbirlerinden bağıımsız olarak kendilerine atanan görevlerin yerine getirmeye çalışmış olmaları olduğu gözlemelenmiştir. Bu durum, gereksinim değişiklikleri içeren üçüncü iterasyonda belirgin olarak ortaya çıkmış, değişiklikleri saplayabilmek için tasarımı ve kodu yeniden düzenlemek gerekmiştir. Bu nedenden dolayı üçüncü iterasyonu en geç tamamlayan ekip Kanban ekibi olmuştur. Yazılımın büyüklüğünün daha da artması ve yeni gereksinimler gelmesi durumunda bu dezavantajın daha büyük problemlere yol açmasının olasılığı yüksek görülmüştür.

Sonuç olarak, çalışma kapsmında gerçekleştirilen küçük çaplı uygulamanın geliştiriminde tasarımın kalitesi, geliştirim süresi, modelleme ve değişikliklere cevap verebilme parametreleri ile değerlendirildiğinde Aşırı Programlama ve SCRUM yöntemleri ile diğerlerine göre daha başarılı sonuçlar elde edilmiştir. Aşırı programlama ekibinin geliştirim süreci boyunca elde ettiği test durumları uygulama için sağlam bir dokümantasyon oluşturmaktadır. Bu durum iki yöntem arasında aşırı programlamayı bir adım öne çıkarmaktadır. RUP yöntemindeki yoğun modelleme gereksinimi küçük çaplı projelerin özellikle ilk iterasyonlarında süre açısından bir dezavantaj oluşturabilmektedir. Fakat, tasarımı sürekli düzgün planlaması ve bu sayede değişikliklere kabul edilebilir bir sürede cevap vermiş olması nedeniyle RUP yönteminin de tüm parametreleri dengeleyerek başarı yakaladığı değerlendirilebilir. Bu çalışmada Kanban yöntemi tasarım kalitesi ve değişikliklere evap verebilme parametrelerinde diğerlerine göre daha başarısız olmuştur. Bunun nedeninin Kanban ekibindeki iletişim kopukluğu olduğu gözlemlenmiştir.

Çalışma sırasında yapılan gözlemler ve elde edilen bulgular doğrultusunda küçük ölçekli yazılım projeleri için dokümantasyon ve modellemenin en aza indirgendiği, bunun yerine ekip içi iletişimin yüksek tutulduğu aşırı programlama ve SCRUM yöntemlerinin daha hızlı sonuçlar verdiği, çevik yol içerisine modellemeyi de entegre eden RUP yönteminin başlangıçta daha yavaş kalmasına rağmen detaylı tasarım çalışmaları ile yeni gereksinimlere hızlı cevap verebildiği gözlemlenmiştir. Kanban yönteminde kartlara ayrıştırılan görevlerin ekip üyelerine atanması sonrasında bireylerin birbirinden bağımsız olarak bu görevleri, yerine getirmeleri durumunda proje risklerinin öndemli derecede arttığı değerlendirilmiştir.

Kaynakça

[1] Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120-127.

[2] Gerald M. Weinberg, as quoted in Larman, Craig; Basili, Victor R. (June 2003). "Iterative and Incremental Development: A Brief History". *Computer* 36 (6): 47–56. doi:10.1109/MC.2003. 1204375.

[3] Edmonds, E. A. (1974). "A Process for the Development of Software for Nontechnical Users as an Adaptive System". *General Systems* **19**: 215–18.

[4] Beck, K.; et al. (2001). "Manifesto for agile software development", Agile Alliance

[5] Hunt, J. (2006). Agile methods and the agile manifesto. *Agile Software Construction*, 9-30.

[6] Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

[7] Jeffries, R., Anderson, A., & Hendrickson, C. (2001). *Extreme programming installed*. Addison-Wesley Professional.

[8] Sutherland, J., & Schwaber, K. (2013). The scrum guide. *The definitive guide to scrum: The rules of the game. Scrum. org*, *268*.

[9] Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

[10] Ahmad, M. O., Markkula, J., & Oivo, M. (2013, September). Kanban in software development: A systematic literature review. In *2013 39th Euromicro conference on software engineering and advanced applications* (pp. 9-16). IEEE.

[11] Moniruzzaman, A. B. M., & Hossain, D. S. A. (2013). Comparative Study on Agile software development methodologies. *arXiv preprint arXiv:1307.3356*.

[12] Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

[13] Astels, D. (2003). *Test driven development: A practical guide*. Prentice Hall Professional Technical Reference.

[14] Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70-77.

[15] Rosenberg, D., & Scott, K. (1999). *Use case driven object modeling with UML* (pp. 1-4). Reading: Addison-Wesley Professional.