Yazılım Hata Tahmininin Web Uygulamalarında Kullanılabilirliği

Serdar Biçer ve Banu Diri

Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü, İstanbul, Türkiye
mehmet.serdar.bicer@std.yildiz.edu.tr
banu@ce.yildiz.edu.tr

Özet Yazılım testinde uygulanabilecek en basit yaklaşım verilen bir kod parçasındaki bütün olasılıkları test etmektir. Bu durum zaman ve bütçe kısıtları nedeniyle pratikte imkansızdır. Yazılım hata tahmini yöntemleri proje yöneticileri tarafından, test aşamasında, kısıtlı olan kaynakları efektif bir şekilde dağıtmak için kullanılmaktadır. Bu alandaki çalışmalar özellikle 2005 yılından itibaren artarak devam etmektedir. Bu çalışmada literatürde var olan metriklerin web uygulamaları için yeterli olup olmadığı sorgulanmıştır. Web uygulamaları üzerinde yaptığımız deneyler hata tahmininin web uygulamaları üzerinde optimum sonuçlar vermekten uzakta olduğunu göstermektedir. Bu tip uygulamaları geliştirmede kullanılan yaşam döngüsü, diğer uygulamalar için kullanılanlarla aynı olsa da teknik bakımdan ayrıştıkları bazı noktalar bulunmaktadır. Bu nedenle yazılım hata tahmini alanında web uygulamalarına özel metrikler oluşturulmasını önermekteyiz.

1 Giriş

Bir yazılım projesinin başarısını belirleyen ana faktör kalitesidir [33]. Yazılım kalitesi için birden fazla tanım bulunmakla birlikte bunlar içinde öne çıkanı "yazılımın ne kadar iyi tasarlandığı ve çıkan ürünün bu tasarıma ne kadar uyduğu"dur [32]. Yazılımın kalitesi geliştirme sürecinin test aşamasıyla çok yakından ilişkilidir. Bu ilişki projenin zaman ve bütçe kısıtlarını da çok yakından etkiler. Örneğin 2002'deki IEEE Metrik Paneli'nde [5] araştırmacılar harcanan eforun yarısının aslında önlenebileceğini, bunların %80'inin de hataların küçük bir kısmından (yaklaşık %20) kaynaklandığını öne sürmüşlerdir. Bu tip önlenebilir eforlar daha önceden keşfedilip daha az masrafla çözülebilecek veya tamamen önlenebilecek hatalardan kaynaklanmaktadır [7]. Dikkatli tasarlanmış test aktiviteleri başarılı ürünler doğururken kaotik, rastgele veya doğru yapılmayan test aktiviteleri kısıtları aşmış veya iptal edilmiş ürünlere yol açar. Yazılım testinde uygulanabilecek en basit yaklaşım verilen bir kod parçasındaki bütün olasılıkları test etmektir. Bu durum zaman ve bütçe kısıtları nedeniyle pratikte imkansızdır. Bu nedenle yazılım proje yöneticileri ürünlerindeki hataya yatkınlığı ölçmek için çoğunlukla öğrenme tabanlı tahmin yöntemleri kullanmaktadır.

Yazılım hata tahmini yöntemleri proje yöneticileri tarafından, test aşamasında, kısıtlı olan kaynakları efektif bir şekilde dağıtmak için kullanılmaktadır. Bu

yöntemler yazılım testinde görev yapan kişilere test senaryolarının ne şekilde üretetileceğine ve organize edileceğine karar vermelerine yardımcı olmaktadır. Hatalı modüllerin doğru tahmin edilmesi yazılım testinin masrafını azaltır ve proje yöneticileri kısıtlı kaynaklarını işlere atama konusunda daha rahat hareket edebilirler [34]. İdealde bir hata tahmini modeli bütün hataları doğru tahmin ederken hatasız modülleri hatalı olarak işaretlememelidir. Ancak pratikte bu duruma çok az rastlanır [2]. En yeni tahmin modelleri bile bu noktaya erişmekten çok uzaktadır [15, 23]. Yüksek tahmin oranına sahip modeller yüksek yanlış alarm oranına sahiptir. Yüksek yanlış alarm oranları hatasız kodların boş yere test edilmesine yol açar. Bu durum yüksek güvenlik gerektiren uygulamalar için bir soruna yol açmaz çünkü bu tip uygulamalarda karşılaşılacak bir hatanın bedeli çok yüksektir. Ama bu durum kaynak açısından kritik projeler için ciddi bir problemdir [13, 14, 20]. Kodun gereksiz yere gözden geçirilmesi test aşamasını uzattığından bütçe ve zaman kısıtlarını aşma riskini arttırır. Bu nedenle mühendisler doğru ve yanlış tahmin oranlarını dengeleme yoluna gitmelidir [20].

Bu alanda çalışan araştırmacılar şimdiye kadar hata tahmini modellerini kurarken statik kod metrikleri, kod değişim metrikleri, geliştirici ve modül ağları gibi farklı metrik kümelerinden yararlandılar. Bunlar arasında statik kod metrikleri 1970'lerden beri kullanılmaktadır [1, 4, 19]. Otomatik araçlar yardımıyla da projelerden metrikleri çıkarmak çok daha kolay hale gelmiştir. Geçen yıllarda araştırmacılar kullanılan metrik setlerinin tavan performansa ulaştığını göstermiştir [23]. Bu tavan etkisini ortadan kaldırmanın 2 yolu vardır:

- Var olan metrik setlerine yeni veri madenciliği teknikleri uygulamak
- Var olan veri madenciliği tekniklerini yeni metriklere uygulamak

Araştırmalarda hata tahmini modellerinin performansını arttırmak için yeni veri madenciliği teknikleri bulmaya çalışmanın harcanan emeğe değmeyeceği gösterilmiştir [23]. Bundan dolayı eğitim verisinin kalitesini arttırmak veya kullanılan metrik setlerinde yenilikçi davranmak tahmin modellerinin performansını arttırmak için daha efektif bir yöntem olacaktır.

2012 yılı verilerine göre Kuzey Amerika'nın %78'i, Avrupa'nın %63'ü Internet kullanmaktadır [40]. Dünya çapında yapılan Internet tabanlı işlemlerin yıllık tutarı trilyon dolarlarla ölçülmektedir [35]. E-ticaret dışında her gün milyonlarca kullanıcının arama motorları (örn: Google), sosyal paylaşım platformları (örn: Facebook, Twitter), bilgi paylaşımı (örn: Wikipedia) gibi farklı amaçlarla farklı web sitelerini kullandıkları bilinmektedir. Bu kadar büyük bir Internet kullanımı karşısında firmalar açısından erişilebilir olmak günümüzde büyük bir ihtiyaç halini almıştır. Son yıllarda kızışan tarayıcı savaşları ve buna paralel gelişen teknoloji ve performans artışı geliştiricilerin bu alanda ilerlemesine imkan sağlamıştır. Bu alanda geliştirme yaparken kullanılan teknolojilerin de ilerlemesiyle web geliştiricileri artık daha özgürce daha iyi uygulamalar çıkarabilmektedir. Mobil cihaz kullanımındaki artışla birlikte web uygulamalarına artık çok daha farklı tipte ekranlardan erişilebilmek gibi gereksinimler eklenmeye başlanmıştır.

Ancak web uygulamalarındaki hatalar firmalara milyonlarca dolar kaybettirmeye devam etmektedir. Web uygulamalarının masaüstü uygulamalardan farklı

olarak yüksek erişilebilirliğe sahip olması gerekmektedir. Uygulamada yaşanacak en ufak sıkıntıların firmalara faturası büyük olmaktadır. Örneğin 2001 yılı şükran günü tatilinde Amazon'un yaşadığı sıkıntılar 20 dakikada 500 bin dolar kaybetmesine neden olmuştur [3]. Hataların görünmeyen faturası ise daha büyüktür, her hata kullanıcı sadakatinin bozulmasına ve müşteri kaybına neden olmaktadır [30].

Web uygulamalarını geliştirmede kullanılan yaşam döngüsü, diğer uygulamalar için kullanılanlarla aynı olsa da teknik bakımdan ayrıştıkları bazı noktalar bulunmaktadır.

- Öncelikle web uygulamalarının geliştirilmesinde birden fazla programlama dili, tasarım özelliği, dışarıdan kullanılmakta olan kütüphane ve bileşenler bulunur. Bunlara örnek olarak geleneksel programlama dilleri, script dilleri, düz HTML sayfaları, XML tabanlı şablon dosyaları, veritabanları, resimler ve CSS kodları verilebilir.
- Geliştirilen uygulamalar tarayıcılara bağımlı halde çalışmaktadır. Aynı kod farklı tarayıcıda farklı şekilde çalışabilmektedir. Bunu önlemek için kodun tarayıcı bağımsız çalışacak şekilde yazılması ve uygulamanın farklı tarayıcılar için test edilmesi gerekmektedir.
- Güvenlik zafiyeti daha fazladır. Öncelikle kullanıcı tarafında çalışan kodlara erişip incelemek çok kolaydır. Ayrıca Internet aracılığıyla daha geniş bir kullanıcı kitlesine hitap ettiğinden daha fazla tehdite maruz kalmaktadır.
- Dış dünya değişimlerinden daha çok etkilenmektedir. Internet bağlantısının kaybolması veya yavaşlaması durumları geliştirme sırasında hesap edilmezse istenmeyen durumlarla karşılaşma şansı yüksektir.
- Uygulama bileşenleri gerçek ortamda ve hatta geliştirme sırasında farklı makinelere dağıtılmış halde bulunabilir ve bu halde birbirleriyle uyumlu ve bir bütün çalışmak durumundadırlar.

Bütün bunlar uygulamanın karmaşıklığını arttırıcı faktörlerdir [29]. Bu çalışmada web uygulamaları için hata tahmini yapılarak performans değerlendirmesi yapılmaktadır. Araştırma sorumuz **"Kullanılmakta olan yazılım hata tahmini yöntemleri web uygulamaları için ne kadar iyi sonuçlar vermektedir?"** şeklindedir.

Araştırma sorumuzu yanıtlayabilmek için açık kaynak 6 web uygulamasının hataya yatkınlıklarını dosya bazında inceledik. Bu işlem için yaygın olarak kullanılmakta olan metrikler ve sınıflandırma algoritmalarını kullandık. Aldığımız sonuçlar mevcut hata tahmini yöntemlerinin web uygulamaları için halen düşük performansla çalıştığını göstermektedir.

2 İlgili Çalışmalar

Yazılım hata tahmini alanındaki çalışmalar özellikle 2005 yılından itibaren artarak devam etmektedir [8]. Bu çalışmalarda farklı tipte metrikler kullanılmakla birlikte statik kod metrikleri yaygın olarak kullanılan metrik tiplerinin başında gelmektedir [1, 12, 15, 16, 18, 19, 21, 27, 33]. Literatürdeki ilk hata tahmini

çalışması satır sayısı kullanılarak yapılmıştır [1]. Daha sonra Halstead metrikleri [12] ve McCabe metrikleri [16] kullanılmaya başlanmıştır. Bu metrikler uygulamanın karmaşıklığı ve boyutu hakkında fikirler vermektedir. Günümüzde en yaygın kullanılan metrik tipleri bunlardır. Ancak bu çalışmalarda genel olarak masaüstü uygulamalarından çıkarılan metrikler kullanılmış olup herhangi bir web uygulaması için çıkarılmış bir metrik seti bulunmamaktadır.

Yazılım hata tahmininde kullanılan metrik setlerinden bir diğeri kod değişim (code churn) metrikleridir [9, 10, 26, 28]. Bu metrik setleri Subversion ve GIT gibi versiyon kontrol sistemlerinden çıkarılmaktadır. Geliştiricilerin kod üzerinde yaptığı değişiklikler kullanılarak, eklenen/silinen satır sayısı, yapılan değişiklik sayısı, değişiklik yapan geliştirici sayısı gibi özellikler çıkarılmaktadır. Kod değişim metriği ilk olarak Munson tarafından [26] ortaya atılmıştır. Yapılan çalışmalarda statik kod metriklerinden daha iyi sonuç verdiği gözlenmiştir.

Bunlar dışında diğerlerine göre nispeten daha yeni bir metrik tipi olarak sosyal ağ metrikleri de yazılım hata tahmininde kullanılmaktadır [6,17,31,39,41]. Bu çalışmalarda kullanılan metrikler koddan bağımsız olup, sosyal ağlar geliştirici veya dosyalardan oluşturulmaktadır. Bu alanda çalışanlar birbirine bağımlılığı olan dosyalar veya aynı dosya üzerinde çalışmış olan geliştiricileri birbirleriyle bağlayarak sosyal ağlar kurmuş bu ağlardan sosyal ağ analizi yöntemleri ile metrikler çıkarmışlardır.

3 Yöntem

Bu bölümde çalışmada kullanılan veri kümeleri ve araştırma yöntemleri açıklanmaktadır.

3.1 Veri Kümeleri

Araştırma sürecinde ilk olarak literatürde var olan metriklerin web uygulamaları için yeterli olup olmadığı sorgulanmıştır. Bu metrikler web uygulamalarına özel ortaya atılmış olmasa bile programlama dillerinin genel yapısından dolayı uygunluk göstermeleri olasıdır. PHP tabanlı 6 uygulama incelenerek, statik kod metrikleri ve kod değişim metrikleri kullanılarak uygulamalardaki hatalar tahmin edilmeye çalışılmıştır. Uygulamalar hakkında bazı istatistikler Tablo 1'den görülebilir. Yapılan denemelerde her proje için 2 farklı tipte metrik seti için farklı algoritmalar kullanılmıştır. Sonuçların değerlendirilmesi için 10 katlı çapraz geçerleme kullanılmıştır.

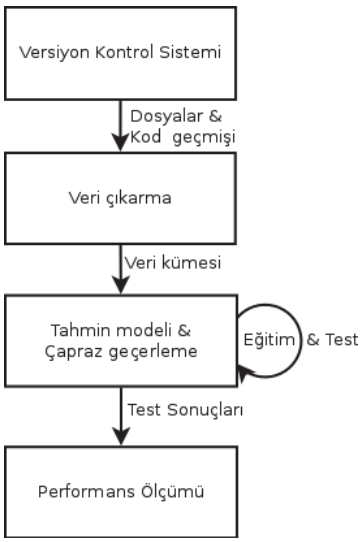
3.2 Hata Tahmin Modeli

Bu çalışmada makine öğrenmesi yöntemlerine dayanan bir hata tahmin yöntemi uygulanmıştır. Kullanılan yöntemin görsel temsili Şekil 1'de görülebilir. Versiyon kontrol sistemleri kodlara ve kod geçmişlerine ulaşmak, buralardan metrikler çıkarmak için kullanılmıştır. Uygulamalarda yer alan dosyaların hataya

Tablo 1: İncelenen Uygulamalar

Uygulama Adı	Sürüm	Geliştirici Sayısı	Satır Sayısı	Dosya Sayısı	Commit Sayısı	Hatalı Dosya Oranı
Laravel	3.0	54	51448	308	2559	%34
Symfony	2.2	753	285875	4048	13144	%49
phpMyAdmin	3.5	330	1140741	1142	70113	%24
Guzzle	3.0	29	48052	413	632	%15
Wordpress	3.0	53	382600	1246	25712	%44
Joomla	3.1	239	581606	5573	15726	%30

meyilli olup olmadıkları farklı tipte metrikler ve sınıflandırma algoritmaları kullanılarak tahmin edilmeye çalışılmıştır. Sınıflandırma için Naive Bayes, Bayes Net ve Random Forest algoritmaları kullanılmıştır. Bu algoritmalar yazılım hata tahmini alanında yaygın olarak kullanıldıkları ve genelde iyi sonuç verdikleri gözlendiği için tercih edilmiştir [8, 15, 19, 23]. Girdi olarak statik kod metrikleri ve kod değişim metrikleri kullanılmıştır. Örnekleme sapmasını engellemek için 10 katlı çapraz geçerleme kullanılmıştır. Deneylerin gerçeklenmesi için Weka uygulaması [11] kullanılmıştır. Veri setlerine eğitim ve test işlemlerinin uygulanması ile hata tahmini sonuçları elde edilmiştir. Bu sonuçlar performans ölçümü aşamasına girdi olarak kullanılmıştır.



Şekil 1: Öğrenme tabanlı hata tahmini sistemi mimarisi

Veri Çıkarma Veri çıkarma işlemi her proje için benzer şekilde ilerlemiştir. Öncelikle proje kodları Tablo 1'de belirtilen sürümler için Github sayfalarından indirilmiştir. Hatalı modüllerin işaretlenmesi için indirilen sürümler temel

alınarak 1 sene içinde hata olarak işaretlenmiş kod değişimleri çıkarılmıştır ve değiştirilmiş dosyalar hatalı olarak işaretlenmiştir. Bir kod değişimini hata olarak işaretleyebilmek için kod teslim mesajında (bug, error, fix, fail) gibi anahtar kelimeler aranmıştır.

Statik kod metriklerini çıkarmak için Understand [38] adlı uygulama kullanılmıştır. Bu metrik tipleri sadece programlama dilleri için kullanılabildiğinden, veri setine sadece PHP ve JavaScript dosyaları dahil edilmiştir. Kod değişim metriklerini çıkarmak için basit bir script yazılmıştır. Bu metrik tipi için PHP ve JavaScript dosyalarının yanında HTML, CSS ve XML dosyaları da veri setine dahil edilebilmiştir. Sadece temel alınan sürümden 1 sene öncesine kadar üzerinde değişiklik yapılmış dosyalar veri setine dahil edilmiştir.

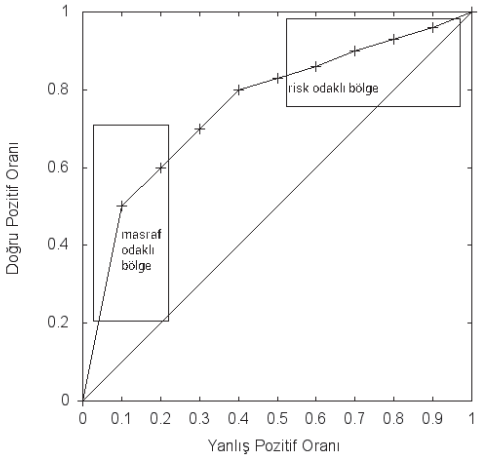
Metrik Tipleri Çalışmada kullanılmak üzere statik kod metrikleri ve kod değişim metrikleri seçilmiştir. Bu metrikler araştırmalarda en yaygın kullanılan metrik tipleri oldukları ve genelde iyi sonuç verdikleri gözlendiği için seçilmiştir. Kullanılan metrikler Tablo 2'de listelenmiştir.

Tablo 2: Kullanılan Metrikler

Statik Kod Metrikleri	**Kod Değişim Metrikleri**
Satır sayısı	Kod teslimi sayısı
Kod satır sayısı	Kod teslim eden kişi sayısı
Boş satır sayısı	Eklenen satır sayısı
Yorum satır sayısı	Silinen satır sayısı
Yorum/kod oranı	Son sürümde kod teslimi sayısı
İfade sayısı	Son sürümde kod teslim eden kişi sayısı
Döngüsel karmaşıklık	Son sürümde eklenen satır sayısı
Tasarımsal karmaşıklık	Son sürümde silinen satır sayısı
Temel karmaşıklık	Popüler kod teslim eden kişi yüzdesi
Yol sayısı	
Kod blok seviyesi	

Performans Ölçümü Çalışmada tahmin modellerinin performansı hata tahmini çalışmalarında yaygın olarak kullanılan doğru pozitif oranı (DPO) ve yanlış pozitif oranı (YPO) ölçümleri kullanılmaktadır [6, 13, 15, 19, 36]. Bu ölçümler tahmin algoritmalarının veri setleri kullanılarak eğitilmesi ve oluşan tahmin modellerinin test edilmesiyle elde edilmektedir. DPO modelin gerçekten hataya yatkın olan modülleri bulmadaki başarısını gösterirken YPO aslında hatasız olan modülleri hatalı işaretlediğini belirtir. Hata tahmininde DPO oranını yükseltip YPO oranını düşüren yöntemler daha değerli bulunmaktadır. Bu nedenle mümkün olduğunca (DPO, YPO) çiftini (1,0) ideal noktasına yaklaştıran tahmin yöntemlerine ulaşmaya ihtiyaç vardır. Maalesef bu ideal durum pratikte

çok nadir görülmektedir. Ölçümlerin ideal duruma yakınlığını ölçmek için denge adı verilen performans ölçütü kullanılmaktadır. Belirtilen ölçütler (1), (2) ve (3) kullanılarak Tablo 3'deki karışıklık matrisi yardımıyla hesaplanmaktadır.



Şekil 2: ROC eğrisinde bölgeler

Doğru tahmin bir modelin başarısını belirlemek için önemli bir etkendir ancak yanlış tahmin de oldukça önemlidir. Bu durum Şekil 2'de gösterilmiştir. Risk odaklı bölgedeki tahmin modelleri yüksek DPO'ya sahip olmakla beraber YPO'ları da oldukça yüksektir. Bu durum hata içermeyen çok sayıda dosyanın hatalı olarak işaretlenmesi anlamına gelip, gereğinden fazla dosyanın incelenmesi sonucunu doğurur. Bu da test aşamasının masrafının artmasına neden olmaktadır. Hatasızlığın çok önemli olduğu projeler için bu kabul edilebilir bir durum olmakla beraber projelerin çoğu bu kategoride yer almamaktadır. Masraf odaklı bölge orta-düşük DPO'ya ve çok düşük YPO'ya sahiptir. Bu bölgeye düşen tahmin modelleri sınırlı kaynaklara sahip projeler için daha kullanışlıdır [13].

Veri dağılımının normal dağılıma uyacağını doğrudan farz edemeyeceğimiz için uygulanacak farklı yöntemlerle bulunan sonuçların birbirinden farklı olup olmadığının kontrolü Mann-Whitney U testi kullanarak yapıldı.

$$DPO = \frac{DP}{DP + YN} \tag{1}$$

$$YPO = \frac{YP}{YP + DN} \tag{2}$$

$$Denge = 1 - \frac{\sqrt{YPO^2 + (1 - DPO)^2}}{\sqrt{2}} \tag{3}$$

Tablo 3: Karmaşıklık matrisi

		Gerçek Durum	
		Hatalı	Hatasız
Tahmin Edilen	Hatalı	DP	YP
	Hatasız	YN	DN

4 Sonuçlar

Araştırma sorumuzu cevaplayabilmek için 6 veri seti üzerinde 10 katlı çapraz geçerleme ile 3 farklı sınıflandırma algoritması kullanılmıştır. Sonuçlar Tablo 4 ve 5'de görülebilir. Tahmin modellerinin başarılarına denge ölçümü kullanılarak karar verilmiştir. Sonuçlar karşılaştırılırken Mann-Whitney U testi kullanılmıştır. Bu sonuçlardan bazı çıkarımlar yapmak mümkündür. Uygulama bazında kullanılan farklı algoritmalar arasında başarısı daha yüksek olanlar koyu yazılmıştır. Sınıflandırma algoritmaları arası performans karşılaştırması yapıldığında Random Forest ve Bayes Net algoritmalarının Naive Bayes'e göre daha iyi sonuçlar verdiği görülebilir.

Metrik setleri açısından baktığımızda kod değişim metriklerinin statik kod metriklerine göre daha iyi sonuçlar verdiği görülebilir. Ortalama denge sonuçları arasındaki farklar istatistiksel açıdan anlamlı bulunmuştur. Bu sonuçlar önceki çalışmaları [9, 15, 24, 25] doğrulamaktadır. Ancak en başarılı olan skorların büyük çoğunluğunda, tahmin oranı çoğu projede yüksek çıkmasına rağmen hatalı tahmin oranı da oldukça yüksektir. Bu durumun kaynak açısından kısıtlı projeler için pratikte sağladığı bir yarar bulunmamaktadır. Çünkü bu durum hata içermeyen çok sayıda modülün de hatalı olarak işaretlenmesine neden olacağı için test aşamasında yüksek efor harcanmasına sebep olup, hata tahmininin kullanılma amacıyla örtüşmemektedir. Ortalama değerlere bakıldığı zaman DPO, YPO ve denge değerlerinin bu alanda benzer performans kriterleri kullanılarak yapılmış diğer çalışmalarda bulunan ölçümlerden [9,19,22,24,25,37] daha düşük olduğu görülebilir. Bu sonuçlar web uygulamalarına özel bir hata tahmini çalışması yapılmasının gerekli olduğu yönündeki düşüncemizi kuvvetlendirmiştir.

5 Tartışma

Bu araştırmada yazılım hata tahmininde uygulanmakta olan tekniklerin web uygulamalarında ne kadar uygulanabilir olduğu araştırılmıştır. Web paradigması yükselişini 2000'li yılların başında yapmış olsa da günümüzde halen gayet revaçta olan bir alandır. Bu tip uygulamalarda yapılan hatalar firmalara çok daha pahalıya mal olmaktadır. Doğaları gereği barındırdıkları teknik detaylar nedeniyle web uygulamalarınının ayrı bir yere konması gerekmektedir. Yazılım hata tahmini alanında bugüne kadar yapılmış çok sayıda çalışma var olsa

Tablo 4: Statik Kod Metrikleri

	Naive Bayes			**Bayes Net**			**Random Forest**		
	DPO	**YPO**	**Denge**	**DPO**	**YPO**	**Denge**	**DPO**	**YPO**	**Denge**
Laravel	0.39	0.21	0.54	0.78	0.44	**0.65**	0.88	0.48	0.65
Symfony	0.90	0.65	0.53	0.72	0.32	**0.70**	0.88	0.65	0.53
phpMyAdmin	0.40	0.16	0.56	0.42	0.20	**0.57**	0.42	0.14	**0.58**
Guzzle	0.90	0.53	0.62	0.78	0.31	**0.73**	0.94	0.70	0.50
Wordpress	0.89	0.72	0.48	0.76	0.60	0.54	0.73	0.42	**0.65**
Joomla	0.12	0.03	0.38	0.75	0.19	**0.78**	0.89	0.30	**0.77**
Ortalama	**0.6**	**0.38**	**0.52**	**0.70**	**0.34**	**0.66**	**0.79**	**0.45**	**0.61**

Tablo 5: Kod Değişim Metrikleri

	Naive Bayes			**Bayes Net**			**Random Forest**		
	DPO	**YPO**	**Denge**	**DPO**	**YPO**	**Denge**	**DPO**	**YPO**	**Denge**
Laravel	0.91	0.55	0.61	0.71	0.13	**0.78**	0.83	0.36	0.72
Symfony	0.93	0.73	0.48	0.87	0.57	**0.59**	0.87	0.66	0.52
phpMyAdmin	0.25	0.06	0.47	0.51	0.16	**0.64**	0.47	0.15	0.61
Guzzle	0.91	0.61	**0.56**	0.88	0.58	**0.58**	0.93	0.77	0.45
Wordpress	0.96	0.41	0.71	0.80	0.03	**0.86**	0.84	0.12	**0.86**
Joomla	0.93	0.45	0.68	0.86	0.32	0.75	0.86	0.29	**0.77**
Ortalama	**0.82**	**0.47**	**0.59**	**0.77**	**0.30**	**0.70**	**0.80**	**0.39**	**0.66**

bile bu çalışmalar araştırmanın ana fikrinden farklı nitelikler ortaya koymaktadır. Var olan yöntemlerin farklı uygulamalar için kullanılmasından çıkarılan sonuç, yazılım hata tahmini yöntemlerinden bu alanda yeterince faydalanamadığı kanısı doğurmuş ve bu alana özel bir çalışma yapılması gerektiği yönündeki fikrimizi güçlendirmiştir. İlgili çalışmalarda ortaya çıkarılmış olan veri setinin zenginleştirilmesi fikrine paralel olarak ilerisi için web uygulamalarında hata tahmini yapılması için özel bir metrik seti çıkarılması, bu sayede hata tahmininde kullanılan veri setlerinin iyileştirilip bu tip uygulamalarda daha iyi sonuçlar alınması tavsiye edilmektedir. Özellikle kozmetik hataların öne çıktığı bu tip uygulamalarda HTML/CSS için metrik seti çıkarılması düşünülebilir.

Kaynaklar

[1] Akiyama, F.: An example of software system debugging. In: IFIP Congress (1). pp. 353–359 (1971), http://dblp.uni-trier.de/db/conf/ifip/ifip71-1.html#Akiyama71

[2] Alpaydın, E.: Introduction to Machine Learning. The MIT Press, 2nd edn. (2010)

[3] California power outages suspended–for now. http://news.cnet.com/2100-1017-251167.html, accessed: 2014-04-12

[4] Basili, V.R., Perricone, B.T.: Software errors and complexity: An empirical investigation. Commun. ACM 27(1), 42–52 (1984), http://doi.acm.org/10.1145/69605.2085

[5] Basili, V., McGarry, F., Pajerski, R., Zelkowitz, M.: Lessons learned from 25 years of process improvement: the rise and fall of the nasa software engineering laboratory. In: Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on. pp. 69–79 (2002)

[6] Biçer, S., Bener, A.B., Çağlayan, B.: Defect prediction using social network analysis on issue repositories. In: Proceedings of the 2011 International Conference on Software and Systems Process. pp. 63–71. ICSSP '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1987875.1987888

[7] Boehm, B., Basili, V.R.: Software defect reduction top 10 list. Computer 34(1), 135–137 (2001), http://dx.doi.org/10.1109/2.962984

[8] Çatal, C., Diri, B.: Review: A systematic review of software fault prediction studies. Expert Syst. Appl. 36(4), 7346–7354 (May 2009), http://dx.doi.org/10.1016/j.eswa.2008.10.027

[9] Çağlayan, B., Bener, A., Koch, S.: Merits of using repository metrics in defect prediction for open source projects. In: Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS '09. ICSE Workshop on. pp. 31–36 (May 2009)

[10] Graves, T.L., Karr, A.F., Marron, J.S., Siy, H.: Predicting fault incidence using software change history. IEEE Trans. Softw. Eng. 26(7), 653–661 (Jul 2000), http://dx.doi.org/10.1109/32.859533

[11] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. 11(1), 10–18 (Nov 2009), http://doi.acm.org/10.1145/1656274.1656278

[12] Halstead, M.H.: Elements of Software Science (Operating and Programming Systems Series). Elsevier Science Inc., New York, NY, USA (1977)

[13] Jiang, Y., Cukic, B., Menzies, T.: Fault prediction using early lifecycle data. In: Software Reliability, 2007. ISSRE '07. The 18th IEEE International Symposium on. pp. 237–246 (2007)
[14] Jiang, Y., Cukic, B., Menzies, T.: Cost curve evaluation of fault prediction models. In: Software Reliability Engineering, 2008. ISSRE 2008. 19th International Symposium on. pp. 197–206 (2008)
[15] Lessmann, S., Baesens, B., Mues, C., Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. Softw. Eng. 34(4), 485–496 (2008), http://dx.doi.org/10.1109/TSE.2008.35
[16] McCabe, T.: A complexity measure. Software Engineering, IEEE Transactions on SE-2(4), 308–320 (Dec 1976)
[17] Meneely, A., Williams, L., Snipes, W., Osborne, J.: Predicting failures with developer networks and social network analysis. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 13–23. SIGSOFT '08/FSE-16, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1453101.1453106
[18] Menzies, T., Di Stefano, J., Chapman, M., McGill, K.: Metrics that matter. In: Software Engineering Workshop, 2002. Proceedings. 27th Annual NASA Goddard/IEEE. pp. 51–57 (Dec 2002)
[19] Menzies, T., Greenwald, J., Frank, A.: Data mining static code attributes to learn defect predictors. Software Engineering, IEEE Transactions on 33(1), 2–13 (2007)
[20] Menzies, T., Stefano, J., Ammar, K., McGill, K., Callis, P., Davis, J., Chapman, R.: When can we test less? In: Software Metrics Symposium, 2003. Proceedings. Ninth International. pp. 98–110 (2003)
[21] Menzies, T., Distefano, J., S, A.O., (mike Chapman, R.: Assessing predictors of software defects. In: in Proceedings, workshop on Predictive Software Models (2004)
[22] Menzies, T., Milton, Z., Turhan, B., Cukic, B., Jiang, Y., Bener, A.: Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering 17(4), 375–407 (2010)
[23] Menzies, T., Turhan, B., Bener, A., Gay, G., Cukic, B., Jiang, Y.: Implications of ceiling effects in defect predictors. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering. pp. 47–54. PROMISE '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1370788.1370801
[24] Mısırlı, A.T., Çağlayan, B., Miranskyy, A.V., Bener, A., Ruffolo, N.: Different strokes for different folks: A case study on software metrics for different defect categories. In: Proceedings of the 2Nd International Workshop on Emerging Trends in Software Metrics. pp. 45–51. WETSoM '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1985374.1985386
[25] Moser, R., Pedrycz, W., Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th International Conference on Software Engineering. pp. 181–190. ICSE '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1368088.1368114
[26] Munson, J.C., Elbaum, S.G.: Code churn: A measure for estimating the impact of code change. In: Proceedings of the International Conference on Software Maintenance. pp. 24–. ICSM '98, IEEE Computer Society, Washington, DC, USA (1998), http://dl.acm.org/citation.cfm?id=850947.853326

[27] Nagappan, N., Ball, T.: Static analysis tools as early indicators of pre-release defect density. In: Proceedings of the 27th International Conference on Software Engineering. pp. 580–586. ICSE '05, ACM, New York, NY, USA (2005), http://doi.acm.org/10.1145/1062455.1062558
[28] Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th International Conference on Software Engineering. pp. 284–292. ICSE '05, ACM, New York, NY, USA (2005), http://doi.acm.org/10.1145/1062455.1062514
[29] Offutt, J.: Quality attributes of web software applications. IEEE Softw. 19(2), 25–32 (2002), http://dx.doi.org/10.1109/52.991329
[30] Pertet, S., Narasimhan, P.: Causes of failure in web applications. Tech. Rep. CMU-PDL-05-109, Parallel Data Laboratory, Carnegie Mellon University (2005)
[31] Pinzger, M., Nagappan, N., Murphy, B.: Can developer-module networks predict failures? In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 2–12. SIGSOFT '08/FSE-16, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1453101.1453105
[32] Pressman, R.S.: Software Engineering: A Practitioner's Approach. McGraw-Hill Higher Education, 6th edn. (2005)
[33] Shull, F., Basili, V., Boehm, B., Brown, A.W., Costa, P., Lindvall, M., Port, D., Rus, I., Tesoriero, R., Zelkowitz, M.: What we have learned about fighting defects. In: Proceedings of the 8th International Symposium on Software Metrics. pp. 249–. METRICS '02, IEEE Computer Society, Washington, DC, USA (2002), http://dl.acm.org/citation.cfm?id=823457.824031
[34] Song, Q., Shepperd, M., Cartwright, M., Mair, C.: Software defect association mining and defect correction effort prediction. IEEE Trans. Softw. Eng. 32(2), 69–82 (2006), http://dx.doi.org/10.1109/TSE.2006.1599417
[35] Sprenkle, S.E.: Strategies for Automatically Exposing Faults in Web Applications. Ph.D. thesis, University of Delaware, Newark, DE, USA (2007)
[36] Tosun, A., Turhan, B., Bener, A.: Practical considerations in deploying ai for defect prediction: A case study within the turkish telecommunication industry. In: Proceedings of the 5th International Conference on Predictor Models in Software Engineering. pp. 11:1–11:9. PROMISE '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1540438.1540453
[37] Turhan, B., Menzies, T., Bener, A.B., Di Stefano, J.: On the relative value of cross-company and within-company data for defect prediction. Empirical Softw. Engg. 14(5), 540–578 (Oct 2009), http://dx.doi.org/10.1007/s10664-008-9103-7
[38] Understand - source code analysis & metrics. http://scitools.com, accessed: 2014-05-03
[39] Wolf, T., Schroter, A., Damian, D., Nguyen, T.: Predicting build failures using social network analysis on developer communication. In: Proceedings of the 31st International Conference on Software Engineering. pp. 1–11. ICSE '09, IEEE Computer Society, Washington, DC, USA (2009), http://dx.doi.org/10.1109/ICSE.2009.5070503
[40] World internet users statistics usage and population stats. http://www.internetworldstats.com/stats.htm, accessed: 2014-04-12
[41] Zimmermann, T., Nagappan, N.: Predicting defects using network analysis on dependency graphs. In: Proceedings of the 30th International Conference on Software Engineering. pp. 531–540. ICSE '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1368088.1368161