Metin Tabanlı Test Durumlarının Kullanacağı Yazılım Modüllerinin Tahmin Edilmesi

M. Yunus Dönmez[1], Özlem Demir[1], Erdinç Özkan[1], Eren Samaner[1]

Ömer Mert Candan[2], Beste Seymen[2], Cemal Yılmaz[2]

[*] Mühendislik ve Doğa Bilimleri Fakültesi, Sabancı Üniversitesi, 34956 İstanbul
[+] NETAŞ Telekomünikasyon A.Ş., Yenişehir Mah. Osmanlı Bulvarı No: 11, 34912 İstanbul
[1] {ydonmez, emiroglu, eozkan, esamaner}@netas.com.tr
[2] {mcandan, besteseymen, cyilmaz}@sabanciuniv.edu

Özet. Yazılım sistemlerinin kalitelerini artırmak için gerek sistemlerin kaynak kodlarının (statik) gerekse sistem yürütmelerinin (dinamik) analiz edilmesi uzun yıllar boyunca uygulanan tekniklerdir. Son yıllarda bu analiz tekniklerine ek olarak; yazılım gereksinim dokümanları ve yazılım kullanım kılavuzları gibi, yazılımlara ait metin tabanlı eserlerin de (software artifacts) otomatik analizlerinin yapılması hususunda çok çeşitli çalışmalar yapılmıştır. Bu bildiri ise otomatik metin analiz yöntemleriyle geleneksel program analiz yöntemlerini birleştirmek suretiyle özgün bir melez analiz yöntemi sunmaktadır. Önerilen analiz yönteminin amacı metin tabanlı test durumlarının doğal dil işleme yöntemleriyle analizlerinin yapılıp, bu test durumlarının sistem yürütmeleri esnasında kullanacakları modülleri tahmin etmektir ki bu bilgi test durumlarının önceliklendirilmesi ve metin tabanlı test havuzlarından elde edilecek çeşitli yapısal kapsama değerlerinin tahmin edilmesi gibi çok çeşitli amaçlar için kullanılabilir. Büyük ve kompleks bir endüstriyel sistem üzerinde yapılan deneysel çalışmalarda önerilen yöntemin 0.741'lik bir F-skor'u ile modülleri tahmin ettiği ve rastgele yöntemlere göre de istatistiksel olarak daha anlamlı sonuçlar verdiği gözlemlenmiştir.

Anahtar Kelimeler: Program analizi, doğal dil işleme, yazılım sınama

Predicting the Software Modules to be Exercised by Test Cases Given in Natural Language

Abstract. Both static and dynamic program analysis techniques have been extensively used for improving software quality. Recently, other types of analysis approaches have also been developed to analyze the software artifacts written in natural language, such as requirements specification documents and user manuals. In this work, we present a hybrid approach, which combines text analysis-based approaches and traditional program analysis approaches to further improve software quality. In particular, we develop a natural language processing-

based approach to analyze textual test cases created manually by practitioners and predict the modules that these test cases would exercise at runtime. Note that this information can then be used in a wide spectrum of quality assurance tasks, such prioritizing the test cases and predicting different types of structural coverage that the test cases would achieve. In the empirical studies conducted on a large and complex industrial software system, the proposed approach successfully predicted the modules to be used with an average F-measure of 0.741 and it was significantly better than random prediction in the statistical manner.

Keywords: Program analysis, natural language processing, software testing

1 Giriş

Program analiz teknikleri kabaca iki başlık altında toplanabilir: static analiz ve dinamik analiz teknikleri. Statik analizlerde programın kaynak kodu, programın yürütülmesine ihtiyaç duyulmadan analiz edilir. Dinamik analizlerde ise program, verilen bir girdi (ya da girdi seti) ile çalıştırılır ve program işleyişi hakkında toplanan verilerin analizi yapılır. Diğer bir değişle; statik analiz, kaynak kodu, dinamik analiz ise program yürütmelerini inceler. Her iki analiz türünde de elde edilen sonuçlar yazılımın kalitesini artırmak için kullanılır. Son zamanlarda üzerinde çalışmalar yapılan diğer bir analiz türü de yazılım gereksinim dokümanları ve yazılım kullanım kılavuzları gibi yazılımlara ait metin tabanlı eserlerin (software artifacts), doğal dil işleme yöntemleri yardımıyla analizlerinin yapılmasıdır. Bu tür çalışmalara örnek olarak son kullanıcılara ait sosyal paylaşımlarının otomatik analiz edilmesi sonucunda kullanıcıların yazılım sistemleri hakkındaki görüşlerinin analiz edilmesi verilebilir [4, 5, 6, 7, 8, 9].

Bu bildiri ise otomatik metin analiz yöntemleriyle geleneksel program analiz yöntemlerini (daha da özelleştirilecek olursak; dinamik analiz yöntemlerini) birleştirmek suretiyle özgün bir melez analiz yöntemi sunmaktadır. Önerilen analiz yönteminin amacı; manuel olarak hazırlanmış metin tabanlı test durumlarının doğal dil işleme yöntemleriyle analizlerinin yapılıp, bu test durumlarının koşturulmaları esnasında kullanacakları modülleri tahmin etmektir. Bu tür tahminlerin geniş bir kullanım alanı bulunmaktadır. Örneğin; elde edilen modül kullanım bilgisi, modüllerin hataya yatkınlığı (error proneness) bilgisi ile bileştirilmek suretiyle test durumlarının önceliklendirilmesi gerçekleştirilebilir ya da test durumlarından elde edilecek olan yapısal kapsama (structural coverage) değerlerinin tahmin edilmesinde kullanılabilir ki bu da test havuzlarının yeterince kapsama sağlamadığı sistem kısımlarının bulunmasına olanak sağlamak suretiyle test havuzlarındaki eksikliklerin giderilmesine yardımcı olabilir.

Bildiriye konu olan bu problem gerçek bir problemdir. Bu probleme konu olan ürün NETAŞ Telekomünikasyon A.Ş.'nin yazılım ürün sahipli yaptığı gerçek zamanlı çalışan, farklı protokol tiplerini (H.248, H.323, SIP, MGCP, NCS, vb.) destekleyen, çok sayıda hat (1.25 milyon) ve trank (400 bin) desteği veren C20 VoIP çağrı sunucusudur. Bu sistem görev-kritik (mission-critical) bir sistem olduğu için sistem üzerinde yapılan değişikliklerden sonra sistemin detaylı bir şekilde doğrulanması ve sağlanma-

sı (system verification and validation) önem arz etmektedir. Bu noktada NETAŞ tarafından sıklıkla kullanılan yöntemlerden biri de uçtan uca manuel testlerin geliştirilmesi ve koşturulmasıdır. Bu amaç doğrultusunda (ilgili süreç basitleştirilerek anlatılacak olursa) sistem üzerinde yapılan değişiklerin analizleri yapıldıktan sonra test planlaması yapılmakta, bu planlar doğrultusunda metin tabanlı test durumları geliştirilmekte ve sonrasında ise oluşturulan her test durumu manuel olarak koşturulmaktadır. Test durumlarının yazı dili İngilizce'dir. Şekil 1'de koşturulan test durumları hakkında fikir verebilmek amacıyla bir test durumu örnek olarak verilmiştir. Kullanılan test durumu bilinçli olarak basit test durumları arasından seçilmiştir; çok daha karmaşık test durumları mevcuttur. Ayrıca; verilen bu test durumu her ne kadar test edilen sistem hakkında bilgisi olmayan birisi için çok da bir anlam ifade etmese de sistemin test mühendisleri bu türdeki metinleri kolaylıkla anlamlandırabilmekte ve gerekli aksiyonları alabilmektedir.

SETUP:
1. Access Node/G6 BRI lines A is INSV and idle on ATCA lab
2. SIP SSL line B is INSV and idle on ATCA lab
3. SST looparound trunk are INSV and idle on ATCA lab
4. Line BRI A is provisioned in Point-To-Point (Static TEI) with only one LTID assigned

PROCEDURE:
1. Make a call from A to B via SST trunk
2. B rings and answers with 2-way speech path
3. A goes onhook to release the call

EXPECTED RESULTS:
1. The call is release completely
2. A and B are back to idle
3. CDR is generated on core
4. New call can be made between A and B

Şekil 1. Örnek bir test durumu.

NETAŞ bünyesinde konuyla ilgili izlenen süreçlerde bütün test durumları geliştirildikten sonra testlerin manuel koşturulmasına geçilmektedir. Test durumlarının koşturulması zahmetli ve zaman alan bir süreçtir. Sistem üzerinde yapılan değişikliklere bağlı olarak koşturulması gereken test durumlarının sayısı binleri bulabilmekte ve koşturma süresi ise birkaç takvim ayını rahatlıkla alabilmektedir.

Harcanan eforun büyüklüğü ve süresi düşünüldüğünde, test mühendisleri, izledikleri süreçleri iyileştirmek istemektedir. Bu amaç için de bazı yöntemlerin geliştirilmesi öngörülmektedir. Fakat, bu yöntemlerin birçoğu doğal dilde yazılmış test durumlarının koşturulmaları esnasında kullanacakları modüllerin tahmin edilmesine dayanmaktadır. Test mühendisleri bu tür tahminleri, gerek test durumlarının gerekse test edilen sistemin karmaşıklığından dolayı yapamamaktadırlar. Bu bildiride ise; bahse konu türlerde tahminler yapan bir yöntem geliştirilmiş ve geliştirilen bu yöntem

NETAŞ tarafından gerçek sistemler üzerinde gerçek test durumlarının manuel olarak koşturulmasından elde edilen veriler kullanılarak değerlendirilmiştir.

Bildirinin geri kalan kısmı şu şekilde organize edilmiştir: Bölüm 2, geliştirilen yöntemi sunmaktadır; Bölüm 3, yapılan deneyleri ve elde edilen sonuçları tartışmaktadır; Bölüm 4, deneysel çalışmadaki dış tehditleri irdelemektedir; Bölüm 5, ilgili bazı çalışmaları özetlemektedir; Bölüm 6 ise elde edilen sonuçların bir özetini sunmakla birlikte gelecekteki olası çalışma konuları hakkında bilgi vermektedir.

2 Yöntem

Önerilen yöntemin amacı; doğal dilde yazılmış bir test durumunun koşturulması esnasında kullanacağı modüllerin tahmin edilmesidir. Dikkat edilirse burada modül kavramı genel anlamıyla kullanılmış olup yürütmeler esnasında kullanılabilecek olan herhangi bir yapıyı ifade etmektedir. Dolayısıyla tahmin edilen modüller bir test durumu tarafından kullanılan kaynak kod parçaları, fonksiyonlar ve sınıflar olabileceği gibi kütüphaneler, bileşenler (component) ve alt sistemler de olabilir. Diğer bir değişle; bir modülün neyi ifade ettiği uygulamaya bağlıdır. Bu çalışmada ise her bir modül bir yazılım bileşenini ifade etmektedir.

Önerilen yöntem iki aşamadan oluşmaktadır: *eğitim* ve *tahmin* aşamaları. Eğitim aşaması girdi olarak doğal dilde yazılmış bir dizi test durumunu ve bu test durumlarının (hepsinin ya da bir kısmının) koşturulmasında elde edilmiş ve koşturulmalar esnasında kullanılan modülleri ifade eden izleri (trake) alır. Çıktı olarak ise istatistiksel açıdan bir anlam ifade eden (*terim*, *modül*) çiftleri hesaplar ki bu çiftler eğitim setinde sıklıkla gözlemlenen terim (test durumlarını ifade eden metinlerden çıkarılmış terimler) ve modül (test durumlarının koşturulması ile elde edilen izlerde gözlemlenen modüller) ikililerini ifade eder. Tahmin aşamasında ise önceden bilinmeyen bir test durumu girdi olarak alınır ve bu test durumunun kullanacağı modüller tahmin edilir.

Bu bölümün geri kalan kısmında eğitim ve tahmin aşamaları detaylı bir şekilde işlenmektedir.

2.1 Eğitim Aşaması

Eğitim aşamasında girdi olarak alınan test durumları kümesi ve bu küme için toplanmış izler şu şekilde analiz edilir. Öncelikle, her bir test durumunun içeresinde test durumu için istatistiksel olarak anlam ifade eden terimler bulunur. Bu amaç için; test durumundaki gereksiz kelimler (stop words) ayıklanır [1], geri kalan kelimelerin kökleri bulunur (stemming) [2] ve bu kökler kullanılarak n-gram'lar hesaplanır (bu bildiri için n = 4). Buradaki 4-gram'lar art arda gelen 1 li, 2 li, 3 lü ve 4 lü kelime dizinleri kümesidir. Bu işlem ile elde edilen her bir kelime veya kelime dizinine *terim* denir.

Sonrasında, elde edilen her bir terimin ilgili test durumu için ne kadar anlam ifade ettiğinin anlaşılması için tfidf endeksi [3], hesaplanır; *tf* terim frekansını (term frequency), *idf* ise ters doküman frekansını (inverse document frequency) ifade eder. Verilen bir test durumları havuzu T, bu havuzda yer alan bir test durumu t ve terim s için bu metrikler şu şekilde hesaplanır:

$$tf(s,t) = \frac{s\ terimin\ t\ test\ durumunda\ görünme\ sayısı}{t\ test\ durumundaki\ toplam\ terim\ sayısı}$$

$$idf(s,T) = \log\left(\frac{T\ havuzundaki\ toplam\ test\ durumu\ sayısı}{s\ terimini\ içeren\ test\ durumunlarının\ sayısı}\right)$$

$$tfidf(s,t,T) = tf(s,t)\ x\ idf(s,T)$$

Dolayısıyla bir terim bir test durumunda ne kadar sıklıkla gözüküyorsa (yüksek *tf* değeri) ve aynı terim farklı test durumlarında ne kadar az gözüküyorsa (yüksek *idf* değeri), o terim o test durumu için o kadar önemlidir (yüksek *tf* ve *idf* çarpım değeri).

Bu çalışmada anlam ifade eden terimlerin bulunması için *tfidf* skorlarına dayalı bir eşik değeri (threshold) kullanılmaktadır. Dokümanın geri kalanında bu eşik değeri *tfidf eşik değeri* olarak adlandırılmaktadır (bu çalışmada *tfidf eşik değeri = 0,05*). Verilen bir *tfidf* eşik değeri ve test durumu için; *tfidf* skorları eşik değerinden büyük olan terimler o test durumu ile ilişkilendirilir. Diğer terimler ise eldeki test durumu için istatistiksel bir anlam ifade etmediği kanaati oluştuğu için ayıklanır ve analiz dışında tutulur.

Bir test durumu için bu test durumuyla ilişkilendirilen terimler yukarıda anlatıldığı gibi bulunduktan sonra bu terimler ile test durumundan elde edilen izlerin içinde yer alan modüller ilişkilendirilir. Bu amaç için her bir terim ve modül çifti için aşağıdaki formül kullanılarak bir skor hesaplanır:

$$ms(s,m,T) = \frac{Modül\ m'i\ çalıştıran\ ve\ s\ terimini\ içeren\ test\ sayısı}{Modül\ m'in\ çalıştırıldığı\ test\ sayısı}$$

Dolayısıyla, hiç beraber gözlemlenmemiş terim ve modül çiftlerinin skoru 0 olacaktır. Diğer durumlar içinse; bir terim, bir modülün çalıştırıldığı test durumlarının ne kadar çoğunda gözüküyorsa bu terim ve modül çifti için skor o kadar fazla olacaktır.

Anlamlı terim ve modül çiftlerinin bulunabilmesi içinse bir eşik değeri kullanılmaktadır. Dokümanın geri kalanında bu eşik değeri *terim-modül eşik değeri* olarak adlandırılmaktadır (bu çalışmada *terim-modül eşik değeri = 0,90*). Bu eşik değerinin üstünde olan terim modül çiftleri kullanılmak suretiyle her bir terim, içinde geçtiği test durumları tarafından kullanılması muhtemel modülleri ifade eden bir modül kümesi ile ilişkilendirilir.

2.2 Tahmin Aşaması

Yukarıda da anlatıldığı üzere eğitim aşamasının çıktısı, terimler ve her bir terimle ilişkilendirilmiş bir modül kümesidir. Tahmin aşamasında ise verilen bir test durumu için ilk olarak aynen eğitim aşamasında olduğu gibi anlamlı terimler bulunur. Diğer bir değişle; test durumundaki gereksiz kelimler ayıklanır, geri kalan kelimelerin kökleri bulunur, bu kökler kullanılarak n-gram'lar hesaplanır, elde edilen her terim için *tfidf* skoru hesaplanır ve skorları *tfidf* eşik değerinin üzerinde olan terimler belirlenir.

Sonrasında, seçilen her bir terim için eğitim aşamasında bu terimle ilişkilendirilmiş modül kümesi bulunur ve bu kümelerin birleşimi, verilen test durumu tarafından kullanılması muhtemel modül kümesi olarak rapor edilir.

3 Deneyler

Önerilen yöntemi değerlendirmek için bir dizi deney gerçekleştirildi. Bu deneylerde NETAŞ tarafından C20 çağrı sunucusunun çekirdek biriminde (hat servis ve çağrı yönlendirmelerinin yapıldığı ana birim) yapılan yazılım geliştirme ve yazılım bakım çalışmalarının doğrulanması ve sağlanması için geliştirilen 303 tane gerçek test durumu ve bu test durumları arasından seçilmiş 32 test durumunun çalıştırılmasından elde edilen gerçek izler kullanıldı. İzler bu çalışma için toplandığından daha önceki zamanlarda geliştirilen bahse konu test durumlarının tekrardan (ve sadece bu çalışma için) koşturulması gerekmiştir. NETAŞ çalışmanın yapıldığı zaman dilimi içinde 32 test durumu için iz toplayabilmiştir. Daha fazla iz toplama ve analiz etme çalışmaları devam etmektedir.

3.1 C20 Çağrı Sunucusu

C20 çağrı sunucusu, ses ve verinin IP tabanlı iletimi için bir altyapı sağlayan ve farklı operatörler için kullanılan, çağrı kontrolü, ağ bağlantısı, anahtarlama, sinyalleşme ve protokol etkileşimi sağlayan bir sistemdir. Bu sunucu, dünya Telekom marketinde lider olup 243 farklı müşteriye destek veren bir IP santraldir. 41 milyon satir kod, 1290 servisi, 38 bin kod modülü, 92 trank ve hat desteği ile beraber PROTEL, C, C++, Java, XML, Perl, XSL ve SQL yazılımları üzerine inşa edilmiştir.

C20 çağrı sunucusunda, Çekirdek (CORE), Ağ Geçidi Denetleyicisi (GWC) ve Ağ Geçidi (GW) ana bileşenleri yer almaktadır. Çekirdek, sistemin ve çağrıların kontrolünü sağlayan merkezi birimdir. Çağrıların yönetilmesi, servislerin çalıştırılması, faturalama gibi işlemler bu ana bileşende gerçekleşir. IP Ağ Geçitleri, ses ve sinyal verilerini taşır ve IP olmayan sinyalleşmeleri IP tabanlı sinyalleşmelere dönüştürür. Bu bileşenler sayesinde, Zaman Bölmeli Çoğullama (TDM) kullanan aboneler IP dünyasına erişimi sağlanır. Ağ Geçidi Denetleyicisi ise ağ geçitleri ile çekirdek arasındaki bağlantıları kurar ve sinyalleşmeleri denetler. Çekirdekten aldığı yönergeler ile farklı protokoller ile desteklenen ağ geçitlerinin yönetimden sorumludur.

3.2 İzlerin Toplanması

İzlerin toplanması test altındaki sistem tarafından sağlanan araçlar yardımıyla her bir test durumu için aşağıdaki şekilde gerçekleştirilmiştir:

1. Sisteme iz toplamayı başlat komutu verilmiştir
2. Test durumu koşturulmuştur
3. Sisteme iz toplamayı durdur komut verilmiştir

Elde edilen iz, gözle okunabilir bir formatta olup kavramsal olarak sistem içerisinde yer alan çok çeşitli prosesler (ki test altındaki sistem dağıtık bir sistem olarak düşünülebilir) tarafından kullanılan modül isimlerini içermektedir. Her bir proses için ayrı bir iz rapor edilmiştir. Dolayısıyla, verilen bir test durumu için modül listesi bu test durumu tarafından koşturulan bütün proseslerde kullanılan modüllerin birleştirilmesi ile elde edilmiştir.

3.3 Kullanılan Veri Seti Hakkında İstatistikler

Tablo 1 deneylerde kullanılan veri seti hakkında istatistiksel bilgiler sunmaktadır. Özetlenecek olursa, gözlemlenen toplam eşsiz terim sayısı 9423; toplam modül sayısı 13197; toplam eşsiz modül sayısı 1171; test başına ortalama modül sayısı 412,40 ve test başına ortalama eşsiz modül sayısı 36,60 olmuştur.

Tablo 1. Deneylerde kullanılan veri seti hakkında birtakım istatiksel bilgiler.

Veri seti özelliği	Adet
Toplam eşsiz terim sayısı	9423
Toplam modül sayısı	13197
Toplam eşsiz modül sayısı	1171
Test başına ortalama modül sayısı	412,40
Test başına ortalama eşsiz modül sayısı	36,60

Bu istatistikler, kullanılan veri setinin önerilen yöntemin değerlendirilmesi için uygun olduğunu da göstermektedir zira 1171 tane eşsiz modül arasından ortalama 36,60 tane modülü şans eseri tahmin etmek (deney sonuçları tartışılırken de görüleceği üzere) zor bir iştir.

3.4 Operasyonel Model

Deneylerde, izleri toplanmış test durumlarının sayısı 32'dir. Değerlendirmenin ise 32-katlı (32-fold) bir şekilde yapılması kararlaştırılmıştır. Diğer bir değişle her bir deney döngüsünde bir tane test durumu tahmin seti olarak seçilip geri kalan 31 tane test durumu eğitim için kullanılmıştır. Her döngüde de farklı bir test durumunun tahmin amaçlı kullanıldığı düşünülecek olursa toplamda 32 tane model eğitilmiş ve her bir model daha önceden görülmemiş bir test durumu kullanılarak değerlendirilmiştir. Bu yöntem "birini dışarıda bırak" (leave-one-out) yöntemi olarak adlandırılıp bu çalışmada da olduğu gibi öğrenme ve test için kullanılacak olan kayıt sayısının az olduğu durumlarda sıklıkla kullanılmaktadır.

Ayrıca deneylerde *tfidf* eşik değeri olarak *0,05,* terim-modül eşik değeri olarak ise de *0,90* kullanılmıştır. Bu eşik değerlerine ilgili histogramlar incelenerek ve gerekli görüldüğü durumlarda ise birtakım deneyler gerçekleştirilerek karar verilmiştir.

3.5 Değerlendirme Çerçevesi

Dikkat edilecek olursa tahmin aşamasının girdi olarak bir test durumu aldığı ve çıktı olarak ise bir modül kümesi hesapladığı anlaşılabilir. Bu test durumunun gerçekten kullandığı ve izlerden elde edilmiş modül kümesi M ve tahmin edilen modül kümesi ise M' olarak alındığında önerilen yöntemin başarısı aşağıdaki metrikler kullanılarak hesaplanmıştır:

$$kesinlik = \frac{|M \cap M'|}{|M'|}$$

$$duyarlılık = \frac{|M \cap M'|}{|M|}$$

$$F - skor = \frac{2\ x\ kesinlik\ x\ duyarlılık}{kesinlik + duyarlılık}$$

Bu metriklerden kesinlik (precision), tahmin edilen modüllerin hangi oranda doğru olduğunu; duyarlılık (recall), tahmin edilmesi gereken modüllerin hangi oranda doğru olarak tahmin edildiğini ve F-skor (F-measure) ise kesinlik ve duyarlılığa eşit ağırlık verilerek hesaplanmış bir başarı ölçüm birimini ifade etmektedir. Her bir metrik, 0 ile 1 arasında (0 ve 1 dahil) değer almakta olup, bu değer ne kadar büyükse önerilen yöntem de o oranda başarılıdır.

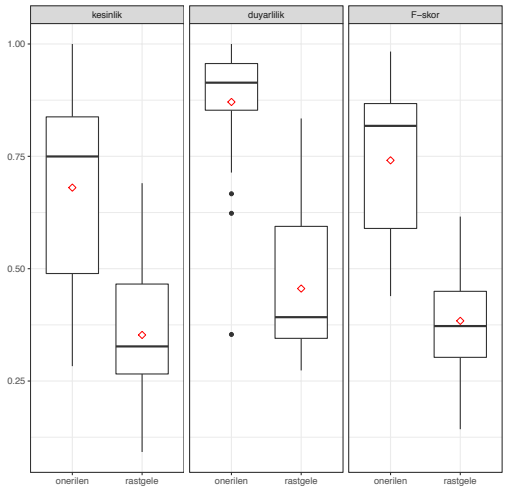
Deneylerde elde edilen sonuçların şans eseri elde edilmediğini göstermek için de önerilen yöntemle elde edilmiş sonuçlar, rastgele modül seçimi yapan bir yöntemle karşılaştırılmıştır. Bu amaç için modülleri tahmin edilen her bir test durumu için önerilen yöntemin tahminde bulunduğu kadar modül, bütün eşsiz modüller setinden rastgele bir şekilde seçilmiş, bu işlem 20 kere tekrarlanmış ve her seferinde tahmin edilen modül seti için kesinlik, duyarlılık ve F-skor metrikleri hesaplanmıştır. Bildiride *rastgele yöntem* adı altında rapor edilen bu değerler, 20 deneyin ortalaması alınarak elde edilmiş değerlerdir.

Tablo 2. Deneylerden elde edilen sonuçlar.

yöntem	min	ortalama	max	stddev
		F-skor		
önerilen	0,439	0,741	0,983	0,164
rastgele	0,143	0,384	0,616	0,111
		Kesinlik		
önerilen	0,283	0,680	1	0,213
rastgele	0,092	0,353	0,690	0,130
		Duyarlılık		
önerilen	0,354	0,871	1	0,136
rastgele	0,274	0,456	0,834	0,140

3.6 Deney Sonuçları

Tablo 2 deneylerden elde edilen sonuçları özetlerken Şekil 2 bu sonuçları bir Box-Whisker grafiği kullanmak suretiyle görselleştirmektedir. Bu grafikte yer alan her bir kutucuğun alt çizgisi ilgili veri setindeki ilk çeyrek bölümü, üst çizgisi üçüncü çeyrek bölümü, kutucuk içindeki çizgi orta (median) değeri ve baklava şekli ise ortalama değeri ifade etmektedir. Diğer bir değişle görselleştirilen verinin %25'i alt çizginin altına, %25'i üst çizginin üstüne ve dolayısıyla da %50'si kutucuğun içine tekabül etmektedir.



Şekil 2. Deneylerden elde edilen sonuçlar.

Önerilen yöntem test durumları tarafından kullanılacak modülleri ortalama *0.741*'lik bir F-skor'u ile doğru tahmin ederken rastgele modül seçiminde elde edilen ortalama F-skor değeri *0,384* olmuştur. Ayrıca, parametrik olmayan Kruskal-Wallis testi, önerilen yöntemin rastgele seçim yapan yöntemden istatistiksel olarak daha anlamlı (statistically significant) sonuçlar elde ettiğini göstermiştir (*p-value < 2.2e-16*).

4 Geçerliliğe Tehditler

Deneysel bir çalışma olması hasebiyle bu bildiride sunulan çalışma, her deneysel çalışmada olduğu gibi, elde edilen sonuçların genelleştirilmesi hususunda bir takım dış tehditler (external threats to validity) içermektedir. Bu tehditler, deneylerde kullanılan kobay yazılım sisteminin, bu sistem için yazılmış test durumları havuzunun ve bu test durumlarından elde edilen izlerin geneli ne kadar temsil kabiliyetine sahip olduğu ile ilgilidir. Bu türdeki tehditleri azaltabilmek amacıyla deneylerde endüstriyel

boyutlarda büyük ve karmaşık bir yazılım sistemi, bu sistem için yazılmış gerçek test durumları ve bu test durumlarının bahse konu sistem üzerinde koşturulmasından elde edilmiş gerçek izler kullanılmıştır. Bu tehditleri büyük ölçüde ortadan kaldırmak içinse deneylerin, çok çeşitli alanlarda çok çeşitli endüstriyel yazılım sistemleri üzerinde tekrarlanması gerekmektedir.

5 İlgili Çalışmalar

Son zamanlarda, doğal dil işlemeye dayalı yöntemler kullanarak yazılımlara ait metin tabanlı eserlerin analizlerini yapan ve analizden elde edilen sonuçları yazılımların kalitelerini artırmak için kullanan çok çeşitli çalışmalar gerçekleştirilmiştir. Bu çalışmalar, 1) son kullanıcıların sosyal medyada yaptığı paylaşımların analizlerinin yapılması suretiyle yazılım isterlerinin toplanmasına ya da bu isterlerin güncellenmesine yardımcı olmak [4, 5, 6, 7, 8, 9], 2) özellikle mobil platformlardaki uygulamaların ihtiyaç duyduğu ve genellikle gizlilik ve güvenlik açığı oluşturabilme riskine sahip izinlerin bu uygulamaların uygulama dükkanlarındaki açıklamalarıyla uygun olup olmadığının analizini yapmak [10], 3) uygulamalar arasındaki benzerlikleri belirlemek [11], 4) mobil uygulamaların market isimlerine, indirilme sayılarına ve son kullanıcı değerlendirmelerine bakılarak uygulamaların güvenilirlik seviyelerini belirlemek [12] gibi alanlarda yoğunlaşmıştır.

6 Sonuç

Bu bildiride doğal dillerde yazılmış test durumlarının koşturulmaları esnasında kullanacakları modülleri tahmin eden ve doğal dil işleme teknikleri ile dinamik program analiz tekniklerini birleştiren bir yöntem önerilmiştir ve bu yöntem gerçek sistemler üzerinde gerçek test durumları kullanılarak deneysel olarak değerlendirilmiştir. Deneylerin sonucunda, önerilen yöntemin 0.741'lik bir F-skor'u ile modülleri tahmin ettiği ve rastgele yöntemlere göre de istatistiksel olarak daha anlamlı sonuçlar çıkardığı gözlemlenmiştir. Bu sonuçların daha da iyileştirmesi noktasında farklı makine öğrenmesi algoritmalarının kullanılabileceği ve/veya sadece test durumlarında yer alan terimler yerine bu terimlerle eş anlama sahip terimler üzerinden de analizlerin yapılabileceği değerlendirilmektedir.

Bu bildiri, yazarların konu hakkında yayımladığı ilk bildiri olup bundan sonra da bu alanda çalışılması planlanmaktadır. Bu kapsamda yapılması planlar işler şu şekilde sıralanabilir: 1) modül izlerine sahip test durumlarının sayısını artırmak ve yeni kobay sistemler üzerinde de çalışmak, 2) önerilen yöntemi çevrimiçi bir hale getirmek; test durumları koşturuldukça daha koşturulmamış olan test durumları için olası modül listesini güncellemek ve 3) elde edilen tahminler kullanılarak test durumu önceliklendirilmesi ve test durumlarının sağlayacağı yapısal kapsamanın tahmin edilmesi.

Onaylar

Bu araştırma Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) tarafından desteklenmiştir (3150586 nolu TEYDEB projesi).

Kaynaklar

1. PyPi Stop-words, https://pypi.python.org/pypi/stop-words, last accessed 2017/06/03.
2. PyPi Snowball-stemmer, https://pypi.python.org/pypi/snowballstemmer, last accessed 2017/06/03.
3. Tf-idf: A single page tutorial, http://www.tfidf.com, last accessed 2017/06/03.
4. E. Kowalczyk, A. M. Memon and M. B. Cohen: Piecing together app behavior from multiple artifacts: A case study. In: International Symposium on Software Reliability Engineering (ISSRE), pp. 438-449, Gaithersbury, MD (2015).
5. S. Rabiger, A. Girisken, and C. Yilmaz: How to Provide Developers Only with Relevant Information? In: International Workshop on Empirical Software Engineering in Practice (IWESEP'16), pp. 12-17, Osaka, Japan, (2016).
6. Rahim Dehkharghani and Cemal Yilmaz: Automatically Identifying a Software Product's Quality Attributes through Sentiment Analysis of Tweets. In: International Workshop on Natural Language Analysis in Software Engineering, pp. 25-30, San Francisco, CA, (2013).
7. L. V. Galvis Carreño and K. Winbladh: Analysis of user comments: an approach for software requirements evolution. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 582–591, San Francisco, CA, (2013).
8. B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh: Why people hate your app: Making sense of user feedback in a mobile app store. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1276–1284, Chicago, IL, (2013).
9. E. Malmi: Quality matters: Usage-based app popularity prediction. In: Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, ser. UbiComp'14 Adjunct, pp. 391–396, New York, NY (2014).
10. R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, Whyper: Towards automating risk assessment of mobile applications. In: USENIX Security Symposium (USENIX Security 13), vol. 13, pp. 527-542 (2013).
11. Mehmet Cagri Calpur, Sevgi Arca, Tansu Cagla Calpur, and Cemal Yılmaz: Model Dressing for Automated Exploratory Testing. In: Proceedings of the 2017 IEEE International Conference on Software Quality, Reliability, and Security Companion, pp. 577-578, Prague, Czech Republic (2017).
12. G. Dini, F. Martinelli, I. Matteucci, M. Petrocchi, A. Saracino, and D. Sgandurra: A multi-criteria-based evaluation of android applications. In: International Conference on Trusted Systems in Trusted Systems (in Trust 2012), pp. 67–82, London, UK (2012).