Yazılımda Test İşgücü Kestirimi: Türkiye'deki Güncel Durum

Onat Ege Adalı[1], Alpay Karagöz[1], Zeynep Gürel[2], Touseef Tahir[3], Çiğdem Gencel[4]

[1]PROVEN Bilişim Teknolojileri Ltd., Ankara, Türkiye
eadali@proven.com.tr, akaragoz@proven.com.tr
[2]OnePIN, Inc., Ankara, Türkiye
gurellzeynep@gmail.com
[3]COMSATS Institute of IT, Lahore, Pakistan
touseeftahir@ciitlahore.edu.pk
[4]Free University of Bozen- Bolzano, İtalya
cigdem.gencel@gmail.com

Özet. Yazılım kalitesinin önemi pazarda rekabete dahil olmak isteyen yazılım kurumları için giderek daha fazla artmaktadır. Yazılım ürünlerinin kalitesini sağlamak için kullanılan en yaygın yöntemlerden biri de Bağımsız Yazılım Doğrulama ve Geçerlemedir. Bu yöntem yazılım geliştiren kurumlardan bağımsız olan kurumlar tarafından gerçekleştirildiği için de kaynakların etkin kullanımı ve buna ilişkin planlar oluşturmak, hem proje hem de test yöneticileri için büyük önem taşımaktadır. Bu çalışma, test işgücü kestiriminin güncel durumunu incelemek üzere sistematik bir literatür taraması ile birlikte Türk yazılım endüstrisinde gerçekleştirilmiş bir endüstriyel anketi sunmaktadır.
Anahtar Kelimeler: Yazılım Testi, Yazılım Kalitesi, İşgücü Kestirimi, Bağımsız Yazılım Doğrulama ve Geçerleme

Software Test Effort Estimation: Current Practices In Turkey

Onat Ege Adalı[1], Alpay Karagöz[1], Zeynep Gürel[2], Touseef Tahir[3], Çiğdem Gencel[4]

[1]PROVEN Information Technologies Ltd. Ankara, Turkey
eadali@proven.com.tr, akaragoz@proven.com.tr
[2]OnePIN, Inc., Ankara, Turkey
gurellzeynep@gmail.com
[3]COMSATS Institute of IT, Lahore, Pakistan
touseeftahir@ciitlahore.edu.pk
[4]Free University of Bozen- Bolzano, Italy
cigdem.gencel@gmail.com

Bu çalışma TÜBİTAK (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu) tarafından Proje 7151491 kapsamında desteklenmiştir.

Abstract. Software quality is continuously gaining importance for software organizations to stay competitive on the market. One of the most widely used techniques by the organizations to ensure the quality of the software products is Independent Software Verification and Validation. Since, the activities for this technique are conducted by an independent organization, creating plans to better utilize resources became crucial for both project and independent test team managers. This study presents the results of a systematic literature review and an industrial survey conducted in Turkish software industry, which we conducted to investigate the state of the art on test effort estimation.
Keywords: Software Test, Software Quality, Effort Estimation, Independent Verification and Validation

1 Giriş

Bağımsız Yazılım Doğrulama ve Geçerleme (BYD&G), kaliteyi arttırmak, maliyetleri ve riski düşürmek için bağımsız bir kurumun yazılım geliştirme ekibinin gerçekleştirdiği aktiviteler üzerinde doğrulama ve geçerleme yapmasına karşılık gelen bir yazılım mühendisliği sürecidir [1]. Avusturalya'da gerçekleştirilen güncel bir araştırma [2], araştırma kapsamına dahil olan 65 kurumun yaklaşık %68'inin Bağımsız Test Takımları' nın (BTT) olduğunu ortaya koymuştur.

Test döngü ve kaynaklarını planlamak ve yönetebilmek için, bir BTT yöneticisinin BYD&G aktiviteleri için gereken işgücünü kestirebilmesi gerekmektedir. Ancak, yazılım geliştirme işgücü, yazılım mühendisliğinde en kötü şekilde tahmin edilen unsur olmaya devam etmektedir. Bu konuda yapılan araştırmalarda projelerin yalnızca çeyreğinden azının doğru tahmin edildiği belirtilmiştir [3]. Birçok çalışma bütün geliştirme yaşam döngüsü için gereken işgücünün kestirimine odaklansa da test işgücünün kestirimine yoğunlaşan çalışma sayısı azdır.

Bu çalışmada, test işgücü kestiriminin güncel durumunu ve Türkiye yazılım endüstrisindeki güncel uygulamasını inceledik. İlgili araştırma sorularımız aşağıda sıralanmıştır:

AS1: Yazılım test işgücü kestirimi için hangi yöntemler/araçlar geliştirilmiştir?

AS2: Türk yazılım endüstrisinde yazılım test işgücü kestirimi için hangi yöntemler/araçlar kullanılmaktadır?

AS3: Test işgücü kestiriminde hangi unsurlar Türk yazılım endüstrisi tarafından önemli olarak değerlendirilmektedir?

Araştırma sorularımızı cevaplamak adına, öncelikle literatürü sistematik bir literatür araştırması ile gözden geçirdik ve sonrasında literatür ve kurumların uygulamaları arasında karşılaştırma yapmak için endüstriyel bir anket çalışması gerçekleştirdik.

Makalenin devamı şu şekilde düzenlenmiştir: Bölüm 2'de yazılım işgücü kestirimine dair genel bir geçmiş sunulmuştur. Daha sonra Bölüm 3'te sistematik literatür taramasının detayları verilmiştir. Bölüm 4'te endüstriyel anketin sonuçları sıralanmış, Bölüm 5'te ise literatür ve anket sonuçları kıyaslanmış ve tartışılmıştır. Son olarak da Bölüm 6'da sonuçlara yer verilmiştir.

2 Literatür Taraması

Bütün işgücü kestirim yöntemleri, geçmiş bilgi birikimlerinin ya da geçmiş proje verisinin bulunma durumuna dayanmaktadır. Böylece, her bir yöntem ana verinin kullanım şekline bağlı olarak (parametrik ya da analog gibi) farklı tekniklere dayanır. Bu çalışmada, test işgücü kestirimini konu alan çalışmaların incelenmesine odaklandık.

Bu kapsamda, çalışmalarda [4][5] sağlanan yordamları kullanarak Sistematik Literatür Taraması (SLT) gerçekleştirdik. SLT, literatür üzerinde iyi tanımlanmış aşamaların izlenmesi ile sistematik bir tarama gerçekleştirmek için geliştirilmiş bir yöntemdir [5][6]. Bir SLT' nin Planlama, Yürütme ve Raporlama olmak üzere 3 temel aşaması vardır. Literatür taramasını takiben sonraki bölümlerde, arama protokolü, ana çalışmaların seçilmesi, ana çalışmalardan çıkartılan veri ve çıkartılan verinin incelenmesini açıkladık.

2.1 Arama ve Seçme Süreci

Öncelikle yazılım işgücü kestirimi alanındaki ana kavram ve terminolojiyi inceledik ve temel anahtar kelimeleri belirledik. Sonrasında, her bir anahtar kelime için eş anlamlı, alternatif ve kapsayıcı kelimeleri kontrol ettik. Son olarak da mantıksal operatörler ('VE/AND' ve 'VEYA/OR') ve eşleme karakterini ('*') kullanarak arama sorgusunu oluşturduk. Arama sorgusu aşağıda yer almaktadır:

"(Software AND Test*) AND (Effort OR Cost) AND (Estimat* OR Prediction)" ,

Şekil 1'de görüldüğü gibi, ana çalışmaların seçilmesi birbirini takip eden üç arama ile gerçekleştirilmiştir. Bunlar: birincil arama, ikincil arama ve takriben kartopu izlemesidir.

Aramalar Ağustos 2016'ya kadar olan zaman dilimini kapsayacak şekilde yapılmıştır. Aramalarda çevrimiçi veritabanları ve arama motorları (IEEE Xplore, ACM, Springer Link, One Search, Science Direct, Scopus ve Web of Knowledge gibi), elektronik bültenler, konferans makaleleri ve gri yayınlar taranmıştır. Bu arama toplamda 3942 çalışma ile sonuçlanmıştır. Başlık, öz ve anahtar kelime gözden geçirmesi sonrasında bu çalışmaların 95'inin konu ile ilgili olduğu saptanmıştır (Tablo 1'i inceleyiniz).

Tablo 1. Belirlenen Potansiyel Çalışma Sayısı

Kullanılan Arama Motoru	**Birincil arama sonrası elde edilen çalışma sayısı**	**İkincil arama sonrası belirlenen çalışmalar**
IEEE Explore	48	10
ACM	1106	10
Springer Link	1706	27
One Search	170	11
Science Direct	147	13
Scopus	104	3
Web of Knowledge	661	21
Toplam	**3942**	**95**

Çalışmalar Mendeley aracına aktarıldıktan sonra 9 çalışmanın aynı olduğu görülmüş ve toplamda 86 çalışma kalmıştır. Genel olarak yazılım efor kestirimine odaklanan ve test efor kestirimine odaklanmayan çalışmalar dışındaki tüm çalışmaların tam metinleri gözden geçirilmiştir. Bu kapsamda, analiz aşamasına dahil edilecek toplam 33 tane ana çalışma kalmıştır. Son olarak da, arama protokolü neticesinde gözden kaçabilecek konu ile ilgili çalışmaların sayısını en aza indirgemek için kartopu izleme (sona kalan ana çalışmaların referans listelerinin ele alınması) ile 11 tane daha çalışma saptanmıştır. Sonuç olarak, literatür taramasına toplamda 44 tane ana çalışma dahil edilmiştir.

2.2 Veri Çıkartma

Veriyi tek biçimli, yapısal ve tutarlı bir biçimde çıkartabilmek için, veri çıkartma formlarını MS Excel üzerinden tanımladık. Her bir çalışmadan çıkartılan genel ve spesifik veri Tablo 2'de gösterilmiştir.

Tablo 2. Veri Çıkartma Formu

Amaç	**Tanımlayıcı Veri**
Genel Veri	Makale adı, yazar ad(lar)ı, yayım mecrası, yayım tarihi
Spesifik Veri	Yöntemin adı, Kestirim kapsamı, Modele girdi olarak kullanılan yazılım varlık ve özellikleri, Geliştirme yaşam döngüsünde uygulanabilirliği, Kullanılan ana kestirim tekniği, Deneysel çalışma detayları, Kestirim metodunun güçlü yanları ve kısıtları

2.3 Veri Analizi ve Sonuçlar

Veriyi analiz etmek için, hem nicel (temel istatistik) hem de nitel analiz (anlantısal analiz) yöntemlerini kullandık. Aşağıda çalışmaları temel aldıkları yöntemlere göre üç kategori altında sınıflandırdık. Birtakım çalışmalar, UML diyagramları ve tanımları ve Test Durumları gibi yazılım geliştirme yaşam döngüsünde erken aşamalarda ortaya çıkan varlıklar üzerinden efor kestirim yöntemleri ortaya koymuştur. Bazı çalışmalar, Yazılım Geliştirme Yaşam Döngüsü (YGYD)'deki eforların dağılımının analizi üzerine odaklanarak birbirini takip eden aşamaların eforlarını bir önceki aşamayı baz alarak kestirim yapmaktadır. Son kategoride ise kestirimlerin hataya meyilli modül/sınıf ve ilişkisel hata tahmini üzerine test işgücü kestirimini dağıtmak üzere yöntemler sunulmaktadır.

a) *Temel yazılım özellikleri üzerinden test eforu kestirimine odaklanan çalışmalar*

Arumugam ve diğerleri [8] nesneye yönelik yazılımların test büyüklüğünü ölçmek için, Sistem Testi Büyüklük Puanları (STBP) yöntemini önermişlerdir. Badri ve diğerleri [9] tanımladıkları Kalite Güvence İndikatörü (Qi) ile test durumları yazmaya dahil olan birim test eforunu arasındaki ilişkiyi araştırmışlardır. Bu yöntemin bir kısıtlaması ise kestirim kapsamının birim test eforu ile kısıtlı kalmasıdır. E Silva ve diğerleri [14] yapay sinirsel ağları, destek vektör makineleri ve doğrusal regresyonu analiz edip karşılaştırmıştır. Ancak yöntem daha önceden uygulanmış kestirim yöntemlerini ta-

mamlayıcı nitelikte, test tasarım aşamasından sonra kullanılabilir. Harrison ve Samaraweera [17] test durumu metriklerinin çeşitli diğer geliştirme ve tasarım metrikleri ile ilişkili olup olmadığına bakmıştır. Bulgularından biri, test eforu kestiriminde kullanışlı olduğunu savundukları, test durumu sayısı ile test zamanı arasındaki yüksek korelasyondur. Kushwaha ve Misra [23] var olan yazılım karmaşıklık metriği CICM - Complexity Measure with Cyclomatic Complexity'yi kıyaslamışlardır. CICM'nin bir yazılımı kavramak için gereken zaman ile ilişkili olduğu görülmüştür. Sheta ve diğerleri [28] yazılım test sürecine dahil edilmesi gereken testçiler ile ilgili bir tahmin etme yöntemi öne sürmüşlerdir. E Silva ve diğerleri [29] manuel test koşum eforunu kestirmek için zamana bağımlı *Accumulated Efficiency in Execution* değişkenini baz alan bir yöntem önermişlerdir. Özellikle birkaç döngü halinde test edilen, tekrarlı ve/veya belirgin yazılım sistemlerini test eden bağımsız test takımları için geliştirilmiştir. Haliyle, yöntemin uygulanabilirliği manuel olarak test yapan bağımsız küçük test takımlarıyla kısıtlıdır. Srivastava ve diğerleri [30] test eforunu kestirmek için meta-heuristic bat algoritmasını kullanan bir model öne sürmüşlerdir. Öne sürülen model çözümleri artırımlı olarak iyileştirerek eforu optimize etmek için kullanılmıştır. Srivastava ve diğerleri [31] cuckoo araması adı verilen meta-heuristic tekniğini baz alan yeni bir test eforu kestirim modeli geliştirmişlerdir. Zhu ve diğerleri [36] test durumu paketi seviyesinde test koşum eforu kestirimi için Durum Tabanlı Nedenleme'yi kullanan deneyim-tabanlı bir yöntem ortaya koymuşlardır. Bir test paketini test durumu sayısını, test koşum karmaşıklığını ve testçiyi dikkate alan 3 boyutlu bir vektörle bir deneyim veritabanı kurmuşlardır. Sonrasında veri setini eğitip, verilen test paketi vektörlerinin eforlarını kestiren bir model geliştirmişlerdir. Yine [37]'te erken aşamalarda efor kestirimi yapmak için, kullanım durumu senaryolarından test durum sayısını tahmin eden farklı bir yöntem önermişlerdir. Aranha ve Borba [38] "test durumu karmaşıklığı" konseptini tanımlayan metrik tabanlı bir yöntem öne sürmüşlerdir. Van Veenendaal ve Dekkers [39] sistem ve kabul testleri için gereken eforu kestirmek için Test Puan Analizi (TPA) isimli bir yöntem sunmuşlardır. Puanı ve üretkenlikten (verilen bir hacim test işini gerçekleştirmek için gereken zaman) test eforu hesaplanır. Felipe ve diğerleri [15] TPA ve Kullanım Durumu Puanları (KDP) yöntemlerini gerçek proje verisi kullanarak deneysel bir çalışma ile kıyaslamışlardır. Nageswaran [40] kabul testi eforunu kestirmek için KDP'yi kullanmıştır. Bu yöntemin bir dezavantajı aşamalarına ayırmadan bütün bir test sürecinin eforunu kestirmesi ve kestirimi yapmak için öznel yargıya ihtiyaç duymasıdır. Nageswaran'ın metodundaki bu sorunlardan bazılarını gidermek için Almeida ve diğerleri [44] tarafından modifikasyonlar önerilmiştir. Xiaochun ve diğerleri [41] test paketi koşturma vektörü ve test durum sayısı tahmin etme yöntemlerini içeren bir test eforu kestirim yöntemi öne sürmüşlerdir. Bu kestirimin doğruluk payı uzman görüşüne bağlıdır. Ashish ve Dharmender [42] ve [43] gereksinim bazlı test efor kestirimi karmaşıklığı için bir metrik tanımlamıştır. Baudry ve diğerleri [45] test durumları tarafından ele alınan sınıf etkileşimlerinin sayısı ve karmaşıklığını baz alan bir tasarımın test edilebilirliği ile ilgili bir ölçüm ortaya koymuşlardır. Önerilen ölçüm ile test eforu arasındaki ilişkinin doğası daha fazla araştırılmamıştır. Zhou ve Liu [46] kullanım durumlarını baz alarak test paketi büyüklüğü kestirimi önerisinde bulunmuşlardır. Test paketi büyüklüğü test durumu sayısı tahmin edilerek bulunmuştur.

Aranha ve diğerleri [48] testlerin büyüklüğü ve koşturma karmaşıklığını ölçmeyi hedefleyen, koşturma puanı isimli yeni bir metrik tanımlamışlardır.

b) Yazılım Geliştirme Yaşam Döngüsünde (YGYD) iş dağılımına dayanan test eforu kestirimi odaklı çalışmalar

Ferrucci ve diğerleri [16] birbirini takip eden aşamaların eforlarının tahmini için YGYD'deki her bir aşamaya harcanan eforu analiz etmişlerdir. Bir önceki aşamanın efor verisini kullanmayı, Fonksiyon Nokta (FN) ile test efor kestirimi ile kıyaslamışlardır. 25 adet uygulamanın verisini kullanarak, bir önceki aşama eforu ve FN'nin bir arada kullanımının iyi bir kestirim sağladığını deneysel olarak göstermişlerdir.

Primandaria ve Sholiq [26] küçük ve orta ölçekteki şirketlerden toplanan 15 projenin verisi üzerinden YGDY efor dağılımlarını araştırmışlardır. Bu dağılım Kullanım Durumu Puanı (KDP) yöntemi ile maliyet kestirimi hesaplamasında bir referans olarak kullanılmıştır.

c) Hata ve Güvenilirlik tahmini üzerine test efor dağılımına odaklanan çalışmalar

Bareja ve Singhal [10] hataya meyilli ve test için öncelik verilecek modülleri tahmin ederek test eforunun azaltılması için kullanılan teknikleri tartışmışlardır.

Bluemke [11] hataya meyilli sınıfların belirlenmesi üzerine odaklanmışlardır. LMS metrikleri makalede bir betikle birlikte sunulmuştur ancak çalışmanın geçerliliği ve etkililiğine dair bir kanıt sağlanmamıştır. Calzolari ve diğerleri [12] bakım ve test eforunun gelişimini avcı-av dinamik modeli kapsamında, lineer ve lineer olmayan matematik varyasyonları analiz ederek gösteren bir model geliştirmişlerdir. Stikkel [47] efor takibi ve sistem testi sürecinin tamamlanma kestirimi için Calzolari'nin lineer modeline bir ek geliştirmiştir. Son olarak test eforu problemini ele almak için yazılım geliştirme süreci efor kestirimi için olağan bir yöntem ele alınmıştır. Dhanyamraju ve diğerleri [13] test durumlarının risk değerine göre tüm hata senaryolarının ele alınması için, test durumu altkümesi seçimi ve farklı ortamlarda koşturmak için bu test durumlarının ilgili şekilde planlanması için bir yöntem önermişlerdir. [32], [33] ve [34] hata tahmini ve test kodu değişimi tahminine odaklanmıştır. Yamada ve diğerleri [35] Weibull eğrisi tarafından tanımlandığı gibi yazılım test aşaması için harcanan test eforunu bünyesine alan bir yazılım-güvenilirliği büyüme modeli geliştirmişlerdir.Huang ve Lyu [20] en uygun yayınlanma süresini içerecek şekilde, yazılım test eforu ve etkinliğinin yazılım güvenilirliğinin modellenmesine olan etkisini çalışmışlardır. Bu çalışma [19]'deki çalışmalarına dayanmaktadır. Huang ve Kuo [21] lojistik test eforu fonksiyonunu barındıran homojen olmayan bir Poisson sürecine dayanan yazılım-güvenilirliği büyüme modelini daha detaylı olarak ele almıştır. Huang [18] daha sonra homojen olmayan bir Poisson sürecine dayanan yazılım-güvenilirliği büyüme modelini kurmak için bir şema tanımlamıştır. Jin ve Jin [22] YGBM için kuantum parçacık yığın optimizasyonu isimli algoritması isimli, yığın akıllı optimizasyon algoritmasının iyileştirilmesi ve uygulanmasını incelemişlerdir. Lin ve Huang [24] birçok YGBM'nin test eforu tüketim oranlarındaki muhtemel değişimleri dikkate almadıklarını ve bu durumun test kaynak ataması yaklaşımlarını etkileyebileceğini öne sürmüştür. Bu sebeple, Weibull tipi test eforu fonksiyonlarına yeni

bir çoklu değişim noktası konsepti dahil etmişlerdir. Pham ve Wang [25] yayınlanacak yazılımın güvenilirliği ve toplam yazılım maliyetine dayalı olarak en uygun yazılım yayın süresi için quasi-renewal sürecinin bazı özelliklerinin kullanılmasını araştırmışlardır. Geleneksel yazılım test ölçümlerine ek olarak hatasız yazılım süresi bilgisi de önerilen modeller içerisinde sağlanmıştır. Qian ve diğerleri [27] YGBM'nin uyumunu ve tahmin etme gücünü daha da iyileştirmek için test eforu ve güvenilirlik modellemede kusurlu hata ayıklamayı birleştirmişlerdir.

3 Endüstriyel Anket

Araştırma Soruları 2 ve 3'ü cevaplandırabilmek için bir anket tasarladık ve bu anketi Türkiye'deki yazılım sektörü çalışanlarına sunduk. Aşağıda, kullanılan örneklendirme tekniği, veri toplama aracı ve analiz prosedürü ile ilgili detaylar yer almaktadır.

3.1 Veri Toplama

Anket soru formu 3 ana bölümden oluşmaktadır, bunlar a) Katılımcı ve kurumuyla ilgili demografik bilgiler, b) Katılımcının kurumunda kullanılan belirli test işgücü kestirimi yöntemi ile ilgili bilgiler, c) Katılımcının öznel görüşlerinin alındığı fikir sorularıdır.

Anketi, sonucunda anlamlı analizler yapabilmek adına katılımcıların tüm gerekli soruları doldurmalarını gözeterek tasarlanmıştır. Sorular kapalı uçlu yanıtlardan oluşmaktadır. Soruların çoğu, araştırmacıların cevapları yanlış yorumlamasını engellemek için katılımcıların önceden tanımlı seçenekleri seçeceği şekilde tasarlanmıştır. Örneğin; uygulama tiplerinin sınıflandırması için ISO 12182 [7] yazılım sınıflandırmaları ve iş sektörü sınıflandırması için de Uluslararası Yazılım Karşılaştırmalı Değerlendirme Standartları Grubu'nun (YKDSG/ISBSG) sınıflandırması kullanılmıştır.

Fikir beyan edilen soruların cevaplanması için 5 ölçülü Likert Ölçeği kullanılmıştır. Likert ölçeği psikometrik bir ölçek olup, anket ve soru formu tasarımlarında en çok kullanılan ölçektir. Ölçekte a) Her zaman (5), Genellikle (4), Bazen (3), Ara Sıra (2), Asla (1), ve b) Kesinlikle katılmıyorum (1), Katılmıyorum (2), Ne katılıyorum ne katılmıyorum (3), Katılıyorum (4), ve Kesinlikle katılıyorum (5) ve c) Kritik (1), Çok Önemli (2), Önemli (3), Az Önemli (4), Önemsiz (5) cevapları yer almaktadır.

Anketi tasarlamak için Google Forms anket altyapısı kullanılmış ve cevaplar Google Sheets üzerinden toplanmıştır. Anket 2017'de 1 ay süreyle cevap toplamak için erişilebilir kalmış ve bu süreçte ankete toplamda 99 yanıt alınmıştır.

3.2 Veri Analizi ve Sonuçlar

Veri toplandıktan sonra öncelikle verinin bütünlüğünü kontrol edilmiş ve tamamlanmamış veri sebebiyle toplamda 2 yanıtı analiz dışı bırakılmıştır. Veri analizi için MS Excel programı kullanılmıştır. Sorular arasındaki mantıksal bağımlılıklar her bir soru

için alınan yanıtların toplam sayısında farklılıklar oluşturmuştur. Bu sebeple de her bir soru için toplam yanıt sayısını açık bir şekilde aşağıda verilen sorularda yer almaktadır. AS2 için, anket katılımcıların test işgücü kestirimi için kullandığı yöntem ve araçları içeren sorular içermektedir. Yöntemler için Tablo 4'te görüldüğü üzere 33 yanıt toplanmış bunun 28 tanesi bir işgücü kestirim yöntemi kullandığını belirtmiştir. Anket sonucunda Test Durumu Puanı ve test durumlarının sayısı tabanlı yöntemler endüstride en çok kullanılan yöntemler olarak ortaya çıkmıştır.

Tablo 3. Test İşgücü Kestirimi için Kullanılan Yöntemler

Yöntem	**Katılımcı Sayısı**	**%**
Test Durumu Noktaları	13	46,4
YGYD'deki iş gücü dağılımına dayalı	16	57,1
Metrik tabanlı: Kod satır sayısı	8	28,6
Metrik tabanlı: Cyclomatic Karmaşıklık	2	7,1
Metrik tabanlı: Test durumlarının sayısı	9	32,1
Metrik tabanlı: Fonksiyon Nokta	4	14,3
Diğer:	2	7,1

*Diğer için verilen yanıtlar: "Bilmiyorum" ve "Uzman görüşü"

Araçlar için ise Tablo 5'te görüldüğü üzere toplamda 88 katılımcıdan yanıt alınmıştır. Sonuçlar MS Office'in (%49) ile test işgücü kestirimi için en çok kullanılan araç olduğunu göstermektedir. Ancak katılımcıların neredeyse yarısı (%46) herhangi bir araç kullanmadıklarını söylemişlerdir.

Tablo 4. Test İşgücü Kestirimi İçin Kullanılan Araçlar

Araç	**Katılımcı Sayısı**	**%**
MS Office (Excel, Project, etc) /Apache OpenOffice etc.	43	48,9
SLIM (Software Lifecycle Model)	5	5,7
COCOMO Family	4	4,5
Comparative Estimating Tool	2	2,3
CostXpert	1	1,1
SEER-SEM	1	1,1
Function Point Workbench	5	5,7
PRICE-S	1	1,1
SMRe	1	1,1
Orange Effort Estimator	1	1,1
PQM Plus	1	1,1
KnowledgePlan	3	3,4
Reality Checker	1	1,1
Herhangi bir araç kullanmıyoruz	40	45,5
Diğer:	6	6,9

* Diğer için verilen yanıtlar: "Bilmiyorum" ve "Şirket Aracı"

AS3 için ankette katılımcıların temel yazılım kalite karakteristiklerini test işgücü üzerindeki sahip oldukları etkinin önemine göre puanladıkları bir soru yer almaktadır. Tablo 6'da görüldüğü üzere bu karakteristikler için toplamda 88 katılımcıdan yanıt alınmıştır. Her bir karakteristik katılımcıların %83'ü tarafından "Kritik" ve "Önemli" olarak puanlanmıştır. Test edilebilirlik (%89), yeniden kullanılabilirlik (%90), Veri bütünlüğü (%90) ve kullanım kolaylığı (%92) oranlarıyla "Kritik" ve "Önemli" olarak en fazla puanlanan karakteristikler olmuşlardır.

Tablo 5. Kalite Karakteristiklerin Test İşgücüne Olan Etkisi

Karakteristik	**Kritik (%)**	**Çok Önemli (%)**	**Önemli (%)**	**Az Önemli (%)**	**Önemsiz (%)**
Yeniden Kullanılabilirlik	19,3	37,5	34,1	9,1	0
Bakım Yapılabilirlik	20,5	37,5	29,5	12,5	1,1
Test Edilebilirlik	35,2	19,3	35,2	8,0	2,3
Güvenilebilirlik	23,9	21,6	42,0	11,4	1,1
Erişebilirlik	31,8	22,7	31,8	11,4	2,3
Performans	9,1	37,5	38,6	13,6	1,1
Güvenlik	21,6	29,5	33,0	14,8	1,1
Gizlilik	11,4	33,0	42,0	11,4	2,3
Kullanım Kolaylığı	17,0	36,4	38,6	8,0	0
Veri Bütünlüğü	23,9	38,6	28,4	5,7	0

Ankette aynı zamanda katılımcıların hangi metrikleri test işgücü kestirimi için önemli gördüklerine dair de bir soru bulunmaktadır. Tablo 7'de görüldüğü üzere bu soru için toplamda 28 katılımcıdan yanıt alınmıştır. Katılımcıların çoğu test durumu sayısını, test durumu adım sayısını, testçi sayısını ve testçilerin deneyimini kestirim yapmakta önemli etmenler olarak seçmişlerdir. Kullanım Durumlarının Sayısı be kodun büyüklüğü de önemli olarak görülen etmenler olmuşlardır.

Tablo 6. Test İşgücü Kestirimi için Önemli Sayılan Metrikler

Metrik	**Katılımcı Sayısı**	**%**
Kodun Büyüklüğü (Kod Satır Sayısı)	12	42,9
Fonksiyon Nokta	14	50,0
Kullanım Durumu Sayısı	18	64,3
Sınıfların Sayısı	6	21,4
Cyclomatic Karmaşıklık	5	17,9
Test Durumlarının Sayısı	22	78,6
Çalıştırılacak toplam test adımlarının sayısı	22	78,6
Görevli testçi sayısı	19	67,9
Testçilerin tecrübe düzeyi	22	78,6
Diğer:	0	-

4 Sonuçlar

Bu çalışmada yazılım test işgücü kestirimi method ve araçlarının güncel durumu (AS1) ve bu pratiğin Türk yazılım endüstrisindeki durumu incelenmiştir (AS2, AS3). Literatür taraması üç kategori altında sınıflandırılan çeşitli temel yöntemlerin saptanmasıyla sonuçlanmıştır. Bu yöntemler yazılım nitelikleri, YGYD işgücü dağılımı ve hata ve güvenilirlik tahminlemeye dayanmaktadır. Her bir kategori için birtakım yöntemler içermektedir.

Literatür bulguları yapılan anket sonuçları ile kıyaslandığında; 1)Türk yazılım endüstrisinde birçok şirketin yazılım test işgücü kestirimi yaptığı ve kullandığı bir yöntemin olduğu, 2) Literatürde var olan yöntemlerin yalnızca küçük bir kısmının kullanıldığı (Test Durum Noktaları (%47), YDYG işgücü dağılımına dayanan (%57) geleneksel yöntemlerin ve/ veya Test Durum Sayısı (%32))görülmüştür. Bunun yanı sıra anketi dolduran birçok katılımcı, mevcut yöntemleri geliştirmek adına ilave metrikler önermişlerdir. Endüstriyel anketin sonuçları ortaya çıkarmıştır ki katılımcıların %69'u kurumlarında hiçbir kestirim yönteminin kullanılmadığını söylemiştir. Geçmiş verisinin, güncel makine öğrenme tekniklerinin ve veritabanlı tahminleme için veri madenciliğinin etkin kullanımlarının Türk yazılım endüstrisinde yer almadığı görülmüştür. Bunun sebebi bilinmemekte olup, ileriki çalışmalarda incelenecektir.

Literatür taraması test işgücü kestirimine en sık konu olan girdilerin test durumu, kullanım durumu, kod satır ve test personeli sayılarının olduğunu göstermiştir. Bu metrikler daha sonra uzman görüşü, analoji, durum tabanlı sebeplendirme ve öğrenme gibi yöntemlere girdi olduklarını göstermiştir. Anketin sonuçları göstermiştir ki Türk yazılım endüstrisinde en sık kullanılan metriklerin %79 ile test durumu sayısı, çalıştırılacak test adımı sayısı ve test personelinin deneyimi olduğunu göstermiştir. Ancak, sınıf sayısı (%21,4) ve Cyclomatic Karmaşıklık (%17,9) girdilerinin uygulamada en az sıklıkla kullanılan metrikler olduğu görülmüştür.

Katılımcıların test işgücü kestiriminde önemli olduğunu düşündükleri karakteristiklere bakıldığında, katılımcıların çoğunun bu karakteristikleri önemli ve kritik olarak puanladığı görülmüştür (test edilebilirlik (%89), yeniden kullanılabilirlik (%90), veri bütünlüğü (%90) ve kullanım kolaylığı (%92)). Ancak bu karakteristikler literatür taramasında bulunan yöntemler tarafından önemli olarak atfedilmemişlerdir.

Sonuç olarak, çalışmamızda test işgücü kestirimi konusunda literatür ve Türk yazılım endüstrisindeki uygulama konusunda bir bilgi boşluğu olduğu görülmüştür. Gelecekteki çalışmalarımızda bu boşluğun daha detaylı olarak araştırmak ve bu boşluğun arkasındaki sebepleri araştırmayı hedefliyoruz.

Referanslar

1. ESA, Guide for Independent Software Verification and Validation, ESA, 2005.
2. S. P. Ng, T. Murnane, K. Reed, D. Grant, and T. Y. Chen, "A preliminary survey on software testing practices in Australia," in *Proceedings of 2004 Australian Software Engineering Conference*, pp. 116–125, 2004.
3. ISBSG, "Estimates - How accurate are they?", The International Software Benchmarking Standards Group, Special Analysis Report, 2012.
4. B. A. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering", in 26th International Conference on Software Engineering, pp. 273–281, 2004.

5. B. Kitchenham and S. Charters, “Guidelines for Performing Systematic Literature Reviews in Software Engineering”, Technical Report EBSE-2007-01, Software Eng. Group, Keele Univ. and Dept. of Computer Science, Univ. of Durham, UK, 2007.
6. D. I. K. Sjoberg, T. Dyba, and M. Jorgensen, “The Future of Empirical Methods in Software Engineering Research”, in Future of Software Engineering, pp. 358–378, 2007.
7. ISO/IEC TR 12182:1998: Information technology – Categorization of software.
8. C. Arumugam and C. Babu, “Test Size Estimation for Object Oriented Software Based on Analysis Model”, *Journal of Software*, vol. 10, no. 6, pp. 713–729, 2015.
9. M. Badri, F. Toure and L. Lamontagne, “Predicting unit testing effort levels of classes: An exploratory study based on Multinomial Logistic Regression modeling”, in *Procedia Computer Science* , vol. 62, pp. 529–538, 2015.
10. K. Bareja and A. Singhal, “A Review of estimation techniques to reduce testing efforts in software development”, in *International Conference on Advanced Computing and Communication Technologies*, vol. 2015–April, pp. 541–546, 2015.
11. I. Bluemke, “Object oriented metrics useful in the prediction of class testing complexity”, in *Conference Proceedings of the EUROMICRO* , pp. 130–136, 2001.
12. F. Calzolari, P. Tonella, and G. Antoniol, “Maintenance and Testing Effort Modelled by Linear and Non Linear Dynamic Systems”, *Information and Software Technology*, vol. 43, no. 8, pp. 477–486, 2001.
13. S.U.M.P. Dhanyamraju, S. Chacko, S. S. P. Kanakadandi and G. K. Durbhaka, “Automated Regression Test Suite Optimization based on Heuristics”, in *4th International Conference on Artificial Intelligence with Applications in Engineering and Technology*, pp. 48–53, 2014.
14. D. G. E. Silva, M. Jino, ve B. T. De Abreu, “Machine learning methods and asymmetric cost function to estimate execution effort of software testing”, in *3rd International Conference on Software Testing, Verification and Validation*, pp. 275–284, 2010.
15. N. F. Felipe, R. P. Cavalcanti, E. Habib, B. Maia, W. P. Amaral, A. C. Farnese, J. De. Faria, “A Comparative Study of Three Test Effort Estimation Methods”, *Revista Cubana de Ciencias Informaticas*, vol. 8, no. 8, pp. 1–13, 2014.
16. C. FerrucciGravino and F. Sarro, “Exploiting prior-phase effort data to estimate the effort for the subsequent phases”, in *Proceedings of the 10th International Conference on Predictive Models in Software Engineering - PROMISE ’14*, pp. 42–51, 2014.
17. R. Harrison and L.G. Samaraweera, “Using Test Case Metrics to Predict Code Quality and Effort”, *ACM SIGSOFT Software Engineering Notes*, vol. 21, no.5, pp. 78–88, 1996.
18. C.-Y. Huang, “Performance analysis of software reliability growth models with testing-effort and change-point”, *Journal of Systems and Software*, vol. 76, no. 2, pp. 181–194, 2005.
19. C.-Y. Huang, L. Jung-Hua Sy-Yen Kuo, and M.R. Lyu, “Software reliability modeling and cost estimation incorporating testing-effort and efficiency”, in *10th International Symposium on Software Reliability Engineering*, 1999.
20. C.Y. Huang and M.R. Lyu, “Optimal release time for software systems considering cost, testing-effort, and test efficiency”, *IEEE Transactions on Reliability*, vol. 54, no. 4, pp. 583–591, 2005.
21. C.-Y. Huang and S.-Y. Kuo, “Analysis of Incorporating Logistic Testing-Effort Function into Software Reliability Modeling”, *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 261–270, 2002.
22. C. Jin and S.W. Jin, “Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization” in *Applied Soft Computing Journal*, vol. 40, pp. 283–291, 2016.
23. D.S. Kushwaha and A.K. Misra, “Software test effort estimation”, *ACM SIGSOFT Software Engineering Notes*, vol.*33*, no. 3, pp. 1, 2008.
24. C.T. Lin and C.Y. Huang, “Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models”, *Journal of Systems and Software*, vol. 81, no. 6, pp. 1025–1038, 2008.
25. H. Pham and H.Z. Wang, “A quasi-renewal process for software reliability and testing costs”, *IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS PART A-SYSTEMS AND HUMANS*, vol. 31, no.6, pp. 623–631, 2001.
26. P.L. Primandari and Sholiq, “Effort Distribution to Estimate Cost in Small to Medium Software Development Project with Use Case Points”, in *Procedia Computer Science,* vol. 72, pp. 78–8, 2015.
27. Z. Qian, Z. Jun and L.I. Jing, “Software reliability modeling with testing-effort function and imperfect debugging”, *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 10, no.8, pp. 1992–1998, 2012.

28. A.F. Shet, S. Kassaymeh and D. Rine, “Estimating the Number of Test Workers Necessary for a Software Testing Process Using Artificial Neural Networks”, in *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 7, pp. 186–192, 2014.
29. D. G. E. Silva, B.T. de Abreu and M. Jino, “A Simple Approach for Estimation of Execution Effort of Functional Test Cases”, in *International Conference on Software Testing Verification and Validation* , pp. 289–298, 2009.
30. P.R. Srivastava, A. Bidwai, A. Khan, K. Rathore, R. Sharma and X.-S. Yang, “An empirical study of test effort estimation based on bat algorithm”, *INTERNATIONAL JOURNAL OF BIO-INSPIRED COMPUTATION*, 2014.
31. P.R. Srivastava, A. Varshney, P. Nama and X. -S Yang, “Software test effort estimation: A model based on cuckoo search”, *International Journal of Bio-Inspired Computation*, vol. 4, no. 5, pp. 278–285, 2012.
32. S.N. Umar, “A Statistical model for estimating software testing defects”, *International Journal of Engineering Science & Technology*, vol. *5*, no. 7, pp. 1394–1396, 2013.
33. B. Van Rompaey and S. Demeyer, “Estimation of test code changes using historical release data”, in *Working Conference on Reverse Engineering,* pp. 269–278, 2008.
34. H.B. Yadav and D.K. Yadav, “A fuzzy logic based approach for phase-wise software defects prediction using software metrics”, *Information and Software Technology*, vol. 63, pp. 44–57, 2015.
35. S.Yamada, J. Hishitani and S. Osaki, “Software-Reliability Growth with a Weibull Test-Effort: A Model & Application”, *IEEE Transactions on Reliability*, vol.*42*, no.1, pp. 100–106, 1993.
36. X. Zhu, B. Zhou, L. Hou, J. Chen and L. Chen, “An experience-based approach for test execution effort estimation”, in *Proceedings of the 9th International Conference for Young Computer Scientists,* pp. 1193–1198, 2008.
37. X. Zhu, B. Zhou, F. Wang, Y. Qu and L. Chen, “Estimate test execution effort at an early stage: An empirical study”, in *International Conference on Cyberworlds,* pp. 195–200, 2008.
38. E. Aranha, and P. Borba, “An estimation model for test execution effort”, Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, pp. 107-116, 2007.
39. E.V. Veenendaal and T. Dekkers, T. “Test point Analysis: A Method for Test Estimation”, Project Control for Software Quality, pp. 47-61, 1999.
40. S. Nageswaran, “Test effort estimation using use case points”, Quality Week, pp. 1–6, 2001.
41. Z. Xiaochun, Z. Bo, W. Fan, Q. Yi and C. Lu, “Estimate test execution effort at an early stage: An empirical study”, *Proceedings of the IEEE International Conference on Cyberworlds* , pp. 195–200, 2008.
42. S. Ashish and S.K. Dharmender, “A metric suite for early estimation of software testing effort using requirement document and its validation”, *Proceedings of the IEEE Second International Conference on Computer and Communication Technology*, pp. 373-378, 2011.
43. S. Ashish and S.K. Dharmender, “Applying requirement based complexity for the estimation of software development and testing effort”, ACM SIGSOFT Software Engineering Notes, vol. 37, no. 1, pp. 393-415, 2012.
44. E.R.C. Almeida, B. T. D. Abreu and R. Moraes, “An alternative approach to test effort estimation based on use cases*”, Proceedings of the International Conference on Software Testing Verification and Validation,* pp. 279 – 288, 2009.
45. B. Baudry and Y.L. Traon, “Measuring design testability of a UML class diagram”, Journal of Information and Software Technology, pp. 859–879, 2005.
46. W. Zhou and Q. Liu, “Extended class point approach of size estimation for OO product”, in *2nd International Conference on Computer Engineering and Technology (ICCET)*, vol. 4, pp.117-122.
47. G. Stikkel, “Dynamic model for the system testing process”, *Journal of Information and Software Technology*, vol. 48, no. 7, pp. 578– 585, 2006.
48. E. Aranha and P. Borba, “Estimating manual test execution effort and capacity based on execution points”, *International Journal of Computer & Applications*, vol. 31, no. 3, pp.167-172, 2009.