Yürütülebilir Doğal-Dil Test Gereksinimler: Bir Test-Otomasyon Deneyim Raporu

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140

Yürütülebilir Doğal-Dil Test Gereksinimler: Bir Test-Otomasyon Deneyim Raporu

Vahid Garousi
Queen's Üniversitesi Belfast,
Belfast, Birleşik Krallık
v.garousi@qub.ac.uk

Alper Buğra Keleş, Zeynep Özdemir Güler, Yunus Balaman
Saha BT A.Ş., İstanbul, Türkiye
{alper.keles, zeynep.ozdemir, yunus.balaman}
@sahabt.com

Özet. Test otomasyonu, endüstride sürekli büyüyen yeni metotların ve araçların kullanıldığı bir alan olmakla beraber son zamanlarda geliştirilen yeni yaklaşımlardan Behavior-Driven Development (BDD) sayesinde testlerin doğal bir dille geliştirilebilir ve çalıştırılabilir nitelikte olması sağlanmıştır. BDD yaklaşımı, test mühendislerinin doğal ve yalın bir dille test senaryoları oluşturabilmesi, iş birimleri ve yazılım ekipleri tarafından net ve anlaşılır olan bu senaryolara iş birimleri tarafından katkı sağlanması bu sayede geliştirme süreçlerinde yaşanabilen karmaşıklıkların ortadan kaldırılmasına yarar sağlamıştır. Bu bildiride, *Gauge* adlı bir test aracı kullanarak, yeni nesil otomasyon çerçevesinde yürütülebilir test spesifikasyonları geliştirerek, çalışmaların sonuçlarını bir deneyim raporu olarak sunuyoruz. Bu çalışma kapsamında, birçok ülkedeki firmalara test çözümleri sunan SAHA BT A.Ş. firmasının endüstriyel deneyimlerine yer verilmiştir.

Anahtar kelimeler: Test otomasyon, Test gereksinimler, Deneyim raporu

Executable Natural-Language Test Specifications: A Test-Automation Experience Report

Vahid Garousi
Queen's University Belfast
Belfast, UK
v.garousi@qub.ac.uk

Alper Buğra Keleş, Zeynep Özdemir Güler, Yunus Balaman
Saha BT A.Ş., Istanbul, Turkey
{alper.keles, zeynep.ozdemir, yunus.balaman}
@sahabt.com

Abstract. Test automation technologies are rapidly evolving and new methods and tools contantly appear the in industry. One of the new technologies is a follow-up to the Behavior-Driven Development (BDD) approach in which executable test specifications are written in the form of regular natural-language sentences and are then directly executed for the purpose of testing. Development of test suites in such a manner provides various benefits (e.g., enabling testers to write test cases in natural language) and, at the same time, exposes many research challenges which have to be addressed. We report in this paper an exploratory case-study, in the form of an experience report, with developing executable test specifications in a next generation automation framework named *Gauge*. The reported industrial experience is in the context of a large software testing company named Saha BT A.Ş. which provides automated testing tools and services to a large number of clients in several countries.

Keywords: Test automation, Test specifications, Experience report

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140

1 Giriş

Yazılım testi maliyetli ve çaba gerektiren bir iş sürecidir. 2013'de Cambridge Üniversitesi tarafından hazırlanan bir rapora göre [1], 2013 itibari ile küresel ölçekte yazılım hatalarının bulunması ve çözümlenmesi toplamda 312 milyar dolarlık bir masrafa mal olmaktadır.

Yazılım test işlemleri genellikle manuel (el yordamı) ve otomatik (test otomasyon) olmak üzere iki şekilde gerçekleştirilmektedir. Manuel test yaklaşımında, testçi (test mühendisi) son kullanıcının rolünü alarak, Test-Edilen Yazılımı (TEY) ile kılavuz ile birlikte klavye ve fare kullanarak etkileşim yapmakta ve TEY' in her test durumunu (test case) doğru yapıp yapmadığını kontrol etmektedir. Bir diğer yöntem olan, otomatik test yaklaşımında ise test mühendisi özel bir test aracı (yazılım) kullanarak (örneğin: JUnit test çerçevesi), test kodu geliştirip böylece geliştirilen test kodlarını TEY üzerinde çalıştırırlar. TEY ile etkileşim, insan yerine test aracı ile yapılarak test işleminin hızı, verimliliği ve dikkati artırılabilmektedir. [2-4]. Bu konu üzerine örnek bir endüstriyel vaka-çalışması (case-study) raporunda [5], uluslararası büyük bir yazılım firmasının toplam test masraflarında, test otomasyonunun bir yılda 3.2 milyon dolar tasarruf sağladığı belirtilmiştir.

Test otomasyonu teknolojileri dünyada [6]ve Türkiye'de [7] eş zamanlı olarak hızlı bir şekilde gelişmektedirler. Bu gelişen teknolojilerden biri de Davranışa-Yönelik Geliştirme (DYG), *İngilizce: Behaviour-Driven Development (BDD)*'dir. BDD' de test durumları doğal dilde (örneğin: Türkçe veya İngilizcede) yazılıp ve koşulabilmektedirler. Bu yöntemin İngilizce metodolojisinde üç anahtar kelimesi mevcuttur bunlar; "*Given, When, Then"* şeklindedir. BDD' nin Türkçeye de aktarılmasıyla [8] yukarı da belirtilen üç anahtar kelime sırasıyla "*Diyelim ki, Eğer ki, O zaman"* şeklinde kullanılmaktadır.

BDD yaklaşımı faydalı olmasına rağmen, birçok test mühendisinin yorumlarına göre [9, 10], BDD'nin üç-kelime yapısı bazı durumlarda sınırlayıcı olabilmekte ve her test durumunu geliştirmek için uygun olmayabilmektedir, buna istinaden Gauge adlı test aracının ortaya çıkarılma sebebi bu şekilde anlatılmıştır: "*Unlike BDD tools, Gauge does not prescribe the process with a strict syntax*" [9], tercüme: "*BDD araçlarından farklı olarak, Gauge, test işleminin sınırlayıcı bir yöntem ile reçete etmez*".

Diğer bir yandan, farklı çalışma[11] ve makalelerde de belirtildiği üzere [12], yazılım gereksinimleri çoğu zaman doğal dilde yazılmaktadırlar ve çoğu firma test durumlarını doğal dilde yazılmış olan gereksinimlerden manuel şekilde tasarlamaktadır. Bu işlem genelde çok maliyetli, çaba-gerektiren ve hata-içerebilen bir süreçtir. Bu sebeplerden dolayı, doğal-dil işleme (*İngilizce:* Natural Language Processing (NLP)) kullanılarak bu alanda yayınlanmış çok sayıda çalışma ve yöntem bulunmaktadır [13]. Ancak, doğal dilde yazılmış gereksinimlerin özgür yazı tarzlarını kullanması için, yöntemlerin kapsamı genelde 100%'e ulaşamamaktadır. Daha da kötüsü, eğer gereksinimler beklenilen gramer ve formda yazılmamışlarsa, doğal-dil işleme yanlış test durumları üretebilmektedir [13].

Yukarıda belirtilen tüm sıkıntıları çözmek adına, 2018 yılında, dünyanın ünlü yazılım firmalarından, ThoughtWorks, *Gauge* adlı bir test çerçevesi tasarlamış ve açık-kaynak kodlu olarak kullanıma sunmuştur. Gauge yazılım gereksinimlerinin doğal dil şeklinde yazılmalarını sağlayarak bu gereksinimlerin direkt bir şekilde yürütülmelerini sağlamaktadır. Böylece, Gauge test otomasyonu bir sonraki seviyeye yükselmektedir. Ayrıca Gauge'e benzer mevcut hiçbir test aracı bulunmadığı için yeni nesil test otomasyon aracı olarak adlandırılabilir.

Türkiye'nin lider konumunda bulunan yazılım ve test servis firmasında (Saha BT A.Ş.) gerçekleştirilen sistematik bir analiz sonucunda [14], büyük web ve mobil uygulamalarının testi için, Gauge test çerçevesi seçilmiş ve birçok sayıda test modülü geliştirilmiştir.

Bu bildiride sunulan çalışma, bir endüstri-akademi iş birliği [15-17] olmakla birlikte "TESTOMAT – The Next Level of Test Automation" başlıklı EUREKA ITEA3 projesi çerçevesinde gerçekleştirilmiştir. Amacımız, Gauge'i Saha BT'nin seçilmiş belirli birkaç test projesinde sistematik şekilde kullanmak, test otomasyonun etkisi ve faydalarını değerlendirmek ve deneyim raporu şeklinde

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140

keşifsel bir vaka-çalışması (*İnglizce:* exploratory case-study) [13] sunmaktır. Test otomasyonun etkisi ve faydalarının analizi amaçlı literatürde birçok makale yayınlanmıştır [18-21].

2 İlgili çalışmalar

Test otomasyonu hem endüstri [22] hem de bilimsel literatürde [3] aktif olarak rağbet gören bir konudur. Bu çalışma kapsamında da belirli seçilmiş örnek makaleler üzerinden ilerlemeyi planladık [6, 20, 23-26].

Test otomasyonu alanında birçok endüstriyel vaka-çalışmasının toplandığı ve sunulduğu bir kitap literatürde mevcuttur [6]. Diğer yandan, BDD test yaklaşımı kullanılarak, birçok çalışma yapılıp makalelerde sunulmuştur [23, 24].

Bu bildirinin ilk yazarının da bu alanda birçok çalışması bulunmaktadır ve "Türkiye de gömülü yazılımlar için test otomasyon deneyimi" [25], havacılık yazılımların test otomasyon deneyimi" [26], ve "hukuk yönetim yazılımların test otomasyon deneyimi"[20] adlı çalışmalar bunlardan bazılarıdır.

Kanıta-dayalı yazılım mühendisliği (*İnglizce:* Evidence-based software engineering) [27] alanında yer alan bu makale, direkt yürütülebilir doğal-dil test gereksinimleri ile test otomasyon deneyimi ve sonuçlarını sunmaktadır.

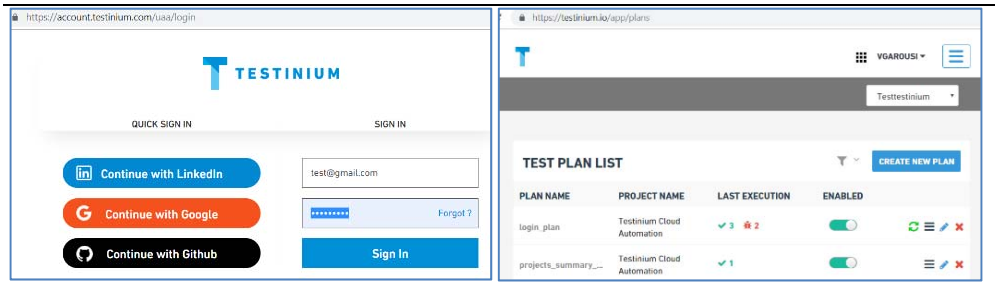
3 Endüstriyel bağlam ve Ar-Ge projesinin amacı

Saha Bilgi Teknolojileri (BT) A.Ş. 2010 yılında yazılım projelerine danışmanlık, eğitim ve yazılım geliştirme amacı ile kurulmuştur. Kısa sürede Java konusunda deneyimli bir ekip kurularak Türkiye'nin önde gelen teknoloji firmalarına hizmet vermeye başlamıştır. Zaman içinde Yazılım Test Otomasyonu alanında uzmanlaşarak endüstri tarafından kabul edilen Selenium ve Appium açık kod platformlar üzerinde çalışan bir test otomasyon ürünü geliştirmeyi iş planlarına dahil etmiştir. 2014 yılında bir TEYDEB projesi çerçevesinde, kendi Ar-Ge ekibini kurarak global pazarda rekabet edebilecek bir test otomasyon ürünü üzerinde çalışmaya başlamıştır, şu anda firma bünyesinde 183 personel bulunmaktadır. Ar-Ge projeleri sonucu geliştirdiği ürünleri: Testinium, Loadium, S-Box, QA Dashboard ve Analytics'dir.

Test otomasyon hizmetleri ve test araçları sunan bir firma olmasından dolayı, Saha BT her zaman daha etkin ve etkili yeni test yaklaşımlarını aramak ve geliştirmeyi hedeflemektedir. Çoğu test aracı ve çerçeveleri endüstride mevcut durumda bulunduğu için, Saha BT'nin test mühendisleri belirli her bir test otomasyon ihtiyacı için, literatürde bulunan[14] mevcut sistematik yöntemleri kullanarak, en doğu test aracını seçmektedirler.

2018'de endüstride pazara çıktığından bu yana, test araçları çeşitli projelerde pilot şekilde kullanılıp başarılı sonuçlar alındıktan sonra, 2019 yılı ikinci dönemi itibari ile daha fazla projede kullanılmaya başlanacaktır. Bu test araçlarının kullanıldığı projelerin birisi de firmanın kendi test aracı olan Testinium'un kendisini test etmesi diğer bir deyişle, test aracını test etmektir. Bu makalenin sonraki kısımlarında (Başlık 4.2) bu test-edilen yazılım (TEY) dan bahsedeceğimiz için, kısa şekilde Testinium'u açıklarsak; Testinium web-tabanlı bir araç olarak, Gauge, Seleinum gibi birçok test aracında yazılan test kodları üzerine test yönetim işlemlerini sağlamaktadır. Şekil 1'de Testinium ara-yüzüne ait bir ekran görüntüsü verilmektedir.

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140



Şekil 1. Testinium test aracını Gauge'de yazılan test kodları ile test etmek. İlk ekran görseli login sayfasını göstermektedir.

Makalenin bütünlüğünü korumak ve anlaşılabilirliğini artırmak adına, yürütülebilir test gereksinimleri konusu üzerinden bir örnek ile ilerleyebiliriz. Testinium'un login sayfasına yazılan örnek bir test Tablo 1'de gösterilmiştir. Bu test kodlarında önemli olan noktalardan biri soyutlama seviyesidir (*İnglizce: level of abstraction*). Görüldüğü üzere, Gauge'de yazılan test gereksinimleri yüksek soyutlama seviyesine sahiptirler ve kodlar arası fonksiyon çağrıları devam ederek Java'da yazılmış Seleinum test kodları çağırılmakta ve son olarak test-edilen yazılımı (TEY) çağırmaktadır. Basit gibi görünen bu faydalı mantık ilişkisi Gauge'in en güçlü özelliğidir ve akademik literatürde mükemmel seviyede çözülemeyen bir soruyu açıkça çözümlemiş olmaktadır [13]. Tablo 1'de verilen terimleri (test gereksinimleri vb.) bir sonraki başlık altında açıklayacağız.

Bu form kapsamında test kodu yazmak, soyutlama seviyesini adım adım değiştirmektedir ve makalenin sonraki kısmında tartıştığımız birçok faydayı test mühendislerine sağlamaktadır. Bu çalışmanın gerçekleştirildiği Ar-Ge projesinin amacı Gauge'i Saha BT'nin çeşitli test projelerinde sistematik şekilde kullanmak ve test otomasyonun etki ve faydalarını değerlendirebilmektir.

Tablo 1. Testinium'un login sayfasına yazılan örnek bir test

<table>
<tr><td rowspan="2">Soyutlama seviyesi (level of abstraction): yüksek</td><td>1 test gereksinimi (specification). 3 test senaryosu</td><td>Tags:Login_InputAlanKontrolleri
*Login ekranındaki Email alanının kontrollerini yap
*Login ekranındaki Password alanının kontrollerini yap
*Login ekranındaki LinkedIn butonunun kontrollerini yap</td></tr>
<tr><td rowspan="2">2 test kavramı (concept). 3+4 test adımı (test steps)</td><td># Login ekranındaki Email alanının kontrollerini yap
*"https://***.testinium.com/***/***" adresine git
*"tbEmailOfLogin" elementinin görünürlüğü "true" mu kontrol et
*"testinium***" ve "***" ile giris yap</td></tr>
<tr><td rowspan="2">Soyutlama seviyesi: düşük</td><td># <email> ve <password> ile giris yap
* "tbEmailOfLogin" elementine <email> değerini yaz
* "tbPasswordOfLogin" elementine <password> değerini yaz
* "btnSignInOfLogin" elementi var mı
* "btnSignInOfLogin" elementine tıkla</td></tr>
<tr><td>Bir test adımın implementasyonu (step implementation)</td><td>@Step("<key> elementine <text> değerini yaz")
public void sendKey(String key, String text){
ElementInfo elementInfo = StorHelper.INSTANCE.findElementInfoByKey(key);
sendKeyBy(ElementHelper.getElementInfoToBy(elementInfo),
elementInfo.getIndex(), text);
}</td></tr>
</table>

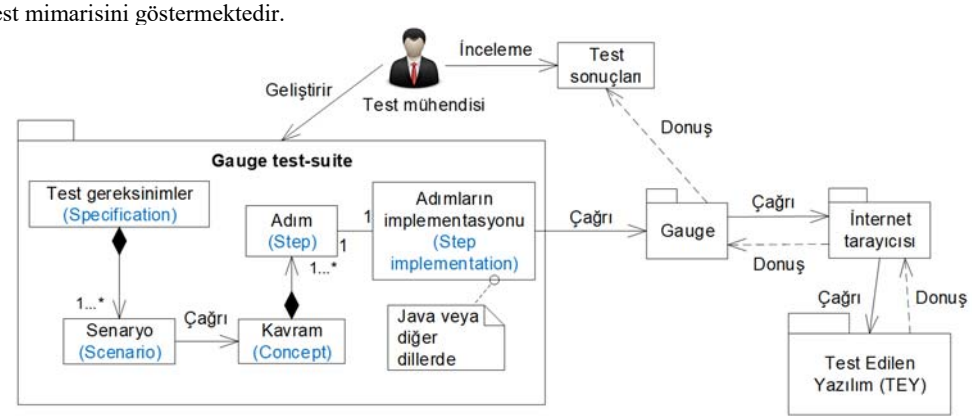
4 Test otomasyon yaklaşımı ve uygulanması

Bu ana başlık altında öncelikle test otomasyon strateji ve mimarisinden, daha sonra test kodlarının geliştirilme sürecinden ve son olarak da test kalıplarının kullanımı ve test kokularının önlenmesinden bahsedilecektir.

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140

4.1 Test otomasyon strateji ve mimarisi

Test otomasyonun başarılı olması için, uygun ve etkin bir test strateji tasarlamak gerekmektedir [28]. Bu çalışmada siyah-kutu (black-box) ve arayüz-bazlı (GUI-based) test yaklaşımı ile ilerlemeyi planladığımız için, test aracı olarak Gauge seçilmiştir. Şekil 2' de Gauge ile belirlenen test otomasyon test mimarisini göstermektedir.



Şekil 2. Test otomasyon mimarisi

Test mühendisi tarafından Gauge için test kümesi geliştirilmektedir. Etkin test-kod geliştirilmesi için [4] (örneğin: testlerin kullanılabilirliği), Gauge uygun test tasarım özellikleri sağlamaktadır. Tablo 1'de verilen örnek kod göz önüne alınarak, bu kümede ilk önce test gereksinimleri (specification) geliştirilmektedir.

Test gereksiniminde, siyah-kutu (black-box) bir test durumudur ve TEY'in özelliklerinin dokümantasyonu da algılanabilmektedir [29]. Bir test gereksinimin içinde bir veya daha fazla test senaryoları yer almalıdır. Her senaryo ile eşlenen bir test kavramı (concept), normal kod fonksiyonlarına benzer olarak geliştirilmelidir (Tablo 1'deki örneğe bakınız). Her test kavramı içinde bir veya daha çok test adımı yerleşebilir. Son olarak, bu adımları yürütülebilir hale getirmek için, adımların implementasyonunun Java veya diğer dillerde geliştirilmesi gerekmektedir. Örneğin, Tablo 1'de `"<key> elementine <text> değerini yaz"` test adımı için, Java dilinde Selenium'u kullanarak TEY'de `<key>` elemanı bulup ve onun içinde `<text>` değerini yazıyoruz.

Görüldüğü üzere, test kavramlarının (concept) yeniden kullanılabilirliği amacıyla adımları mantıksal gruplandırarak tek bir ünitede birleştirme olanağını sağlamaktadır. Yukarıda bahsedildiği gibi farklı soyutlama seviyeleri bir araya gelerek yüksek soyutlama seviyesinde test gereksinimleri yazarak, direkt yürütmeyi (koşmak) mümkün kılmaktadır.

4.2 Test-durum tasarımı ve test-kod geliştirilmesi

Test-durum tasarımı, yazılım test alanında önemli bir yere sahiptir [30]. Siyah-kutu ve beyaz-kutu test-durum tasarım tekniği gibi birçok yöntem mevcuttur. Keşifsel vaka-çalışmamızda, yazılım gereksinimleri kullanılarak, denklik-sınıfı bölümleme (equivalence-class partitioning) tekniğini kullanarak, test-durumları tasarlanmış ve Gauge kodları geliştirilmiştir. Vaka-çalışmamız firmanın mevcut üç test projesine odaklanmıştır. Tablo 1'de belirtilen üç TEY listesini ve Gauge kodlarının ve boyut ölçütlerini göstermektedir. Bilgi gizliliği ve korunumundan dolayı, Testinium hariç diğer sistemlerin isimleri makale kapsamında verilmemektedir, yazılım tipi/ sektör sütunu gibi anonim adlar (TEY2 ve TEY3) gösterilmektedir ve bunlar Türkiye de büyük kullanıcıya sahip büyük ölçekli e-ticaret ve telekomünikasyon sistemleridirler.

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140

Test kodlarının geliştirilme sürecinde en kaliteli test-kodlarını elde etmek için endüstri ve akademinin en iyi pratikleri (best practices) oldukça kullanılmıştır [4]. Bir sonraki başlık altında bu konu detaylı sunulacaktır.

Tablo 1. Farklı test-edilen yazılımlar (TEY) için test ve kod istatistikleri

Test-edilen yazılım (TEY)	Yazılım tipi/ sektörü	Ekran (özellik) sayısı	Gauge test gereksinim sayısı	Test senaryo sayısı	Test kavram sayısı	Test adım sayısı	İmplementasyon Java kod satır sayısı	Efor tahmini (adam ay)
TEY1-Testinium	Test otomasyon aracı	7	162	1,667	518	2,658	1,123	4
TEY2	E-ticaret	14	114	539	67	449	3,415	12
TEY3	Telekomünikasyon	8	14	181	5,411	508	1,348	5

4.3 Test kalıplarının kullanımı ve test kokularının önlenmesi

Tasarım kalıpları (design patterns) tanımından ilham almış olan test kalıpları (test patterns) [31, 32] üst-seviye test otomasyon pratiklerin biridir. Test kalıpları yazılım test projelerindeki genel problemleri en iyi yolla çözmeyi planlamaktadır. Birçok test kalıbı literatürde de mevcuttur [31], örneğin: dört-adım (four-phase) test kalıbı bir test metodun dört-adıma bölünmesini önermektedir: kurulum / hazırlama (setup), egzersiz (exercise): TEY veya birimi çalıştırmak, doğrulama (verify), ve test-sonu temizleme (teardown) şeklindedir. Daha önceki projelerimize benzer şekilde [20, 26] bu projede de test kalıpları kullanılmıştır. Uyguladığımız iyi pratiklerin bir diğeri ise test kodlarının modüler ve kolayca anlaşılabilirliği ile kolay bakımıdır.

Ayrıca, test otomasyon dünyasında iyi pratiklerin tersi anlamına gelen anti patern veya test kokuları (test smells) terimleridir [33], örneğin: test-kod duplike olması test-kokusu ve karmaşık test mantığı test-kokusu gibi. Projemizde test-kodları geliştirilirken, bu ve diğer test kokularını mümkün oldukça önledik ve kodda bulduğumuzda da düzeltme yaptık (*refactoring*).

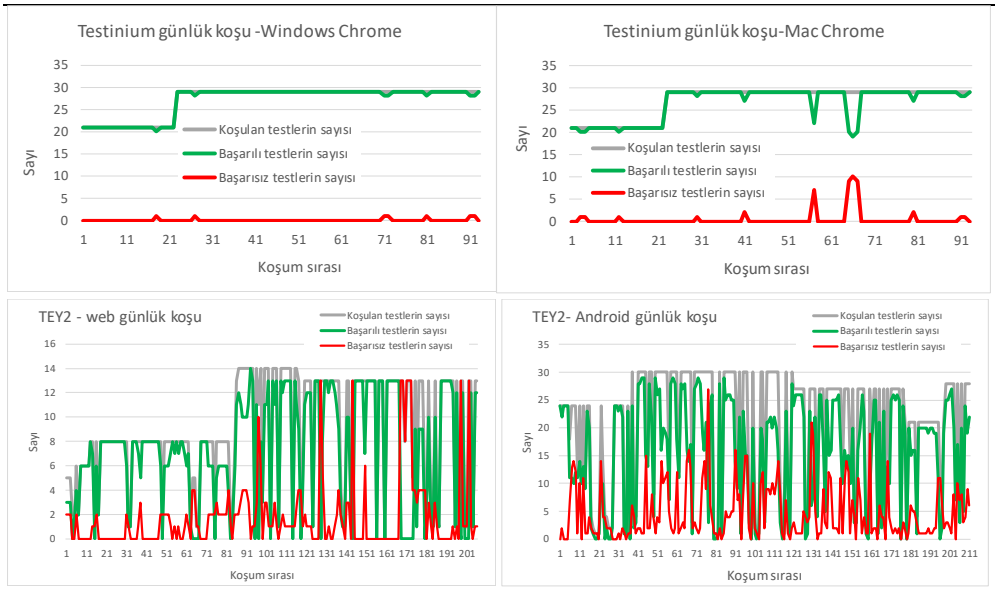
Özet olarak, test kalıplarının kullanımı ve test kokularının önlenmesi kaliteli test-kodları geliştirebilmek adına yardımcı oldular. Sonraki makalelerimizde, bu konularda daha detaylı örnek ve bilgileri paylaşıyor olacağız.

5 Test otomasyonun etkisi ve faydaları

Test ve test otomasyonun amacı TEY' de mevcut olan hataların hepsini bulmak veya sisteme olan güveni arttırmaktır [30]. Test otomasyonun etkisi ve faydalarını değerlendirmek için, birçok yöntem kullanılmaktadır bunlar: (1) testlerin TEY' in geçen sürümlerinde bulduğu hataların sayısı ve trendi, (2) TEY kod ve gereksinimlerin kapsama oranı (test coverage) ve (3) mutasyon testi (TEY' e bilerek hata enjekte etmek) şeklindedir. Başlangıç adımı olarak ilk yöntem bu projede uygulanmıştır. Seçtiğimiz TEY'lerin (Tablo 1) ikisi için (Testinium ve TEY2) ve ayrıca her bir sistem için iki farklı test konfigürasyonu, Şekil 3'de de Gauge testlerin belirli bir zaman döneminde hata bulmak trendi görselleştirilmiştir. DevOps yöntemini kullanarak, testler her gün sabah saat 6.00 ve akşam 18.00 da otomatik koşulmaktadırlar. Testinium için 93 koşum sayısı ve dolaysıyla 46 gün verisi Şekil 3'de verilmektedir.

Şekil 3'de gösterilen hata verileri ve eğilimlerinden anlaşıldığı üzere Testinium için hata oranı genel olarak daha azdır, ama TEY2 sistemi için çoğu günde yüksek hata oranı görülmektedir. TEY2 datasının bilgi güvenliği dolayısı ile bu gözlemin kök ve nedenlerini bu makalede detaylı olarak vermemekte ve tartışmamaktayız. Bununla birlikte Testinium i*le karşılaştırmalı olarak TEY2 datası incelendiğinde başarılı testlerin sayısının oranlarındaki fark net bir şekilde görülmektedir. Buda test süreçlerindeki iyileştirmenin arttırılmasına paralel olarak TEY' in kalitesinin de artmasına olanak sağlamaktadır.

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140



Şekil 3. Gauge testlerin belirli bir zaman döneminde hata bulma trendi.

6 Sonuç ve gelecek çalışmalar

Bu keşifsel vaka-çalışmasında (*İnglizce:* exploratory case-study), *Gauge* adlı test aracı kullanılarak, yeni nesil otomasyon çerçevesinde yürütülebilir test spesifikasyonları geliştirerek, çalışmaların sonuçları bir deneyim raporu olarak sunuldu. Projede şu ana kadar, akademik yöntemleri kullanarak, başarılı bir şekilde test işlemleri yürütülmüştür. Keşifsel vaka-çalışmamız kapsamında, projemizde bir sonraki adım olarak belirli araştırılması gereken konular önümüzde mevcuttur: (1) geliştirilen testlerin hata bulma gücünü (*İngilizce:* fault detection effectiveness), kod ve gereksinimlerin kapsama oranını (coverage) ölçmek; (2) Test-edilen yazılım (TEY) bakım ve evrimi ile beraber test kodlarının bakımı [34].

Teşekkür: Bu çalışma, "TESTOMAT – The Next Level of Test Automation" başlıklı AB ITEA3 projesi ve Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK) 9180076 sayılı projesi tarafından desteklenmiştir.

Kaynakça

1. Britton, T., et al., *Reversible Debugging Software.* University of Cambridge, Judge Business School, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.370.9611, 2013.
2. Amannejad, Y., et al. *A search-based approach for cost-effective software test automation decision support and an industrial case study*. in *Proc. of International Workshop on Regression Testing, co-located with the IEEE International Conference on Software Testing, Verification, and Validation*. 2014.
3. Garousi, V. and F. Elberzhager, *Test automation: not just for test execution.* IEEE Software, 2017. **34**(2): p. 90-96.
4. Garousi, V. and M. Felderer, *Developing, verifying and maintaining high-quality automated test scripts.* IEEE Software, 2016. **33**(3): p. 68-75.
5. Infosys Co., *Case Study-Automating testing for a leading insurance company in Europe and the United States.* https://www.infosys.com/industries/insurance/case-studies/Pages/automated-testing.aspx, Last accessed: Apr. 2017.
6. Graham, D. and M. Fewster, *Experiences of Test Automation: Case Studies of Software Test Automation*. 2012: Addison-Wesley Professional; 1 edition.
7. Garousi, V., et al. *Türkiye'deki yazılım test uygulamaları anketi*. in *Ulusal Yazılım Mühendisliği Sempozyumu (UYMS)*. 2013.

This is the pre-print of a paper that has been published in the proceedings of the 13th Turkish National Software Engineering Symposium (UYMS), September 2019
https://openaccess.iyte.edu.tr/bitstream/handle/11147/7537/?sequence=140

8. Altıntaş, A.B., *Cucumber ve Java ile Davranışa yönelik geliştirme nasıl yapılır (BDD – Behavioral Driven Development) ?* https://kodcu.com/2016/07/cucumber-ve-java-ile-davranisa-yonelik-gelistirme-nasil-yapilir-behavioral-driven-development-bdd/, Last accessed: May 2019.
9. Maliackal, Z., *Why we built Gauge.* https://blog.getgauge.io/why-we-built-gauge-6e31bb4848cd, Last accessed: May 2019.
10. Matts, C., *The tragedy of Given-When-Then.* https://theitriskmanager.com/2019/04/06/the-tragedy-of-given-when-then/, Last accessed: May 2019.
11. Garousi, V., et al., *A survey of software engineering practices in Turkey.* Journal of Systems and Software, 2015. **108**: p. 148-177.
12. KIZILCA, H. and A.E. YILMAZ, *Doğal Dilde Yazılmış Gereksinimlerin Analiz Yöntemleri ve bu Yöntemlerin Türkçe için Uygulanabilirliği.* Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, 2008: p. 69-77.
13. Garousi, V., S. Bauer, and M. Felderer, *NLP-assisted software testing: a systematic review.* arXiv preprint arXiv:1806.00696, 2018.
14. Raulamo, P., M.V. Mäntylä, and V. Garousi. *Choosing the right test automation tool: a grey literature review*. in *International Conference on Evaluation and Assessment in Software Engineering*. 2017. Karlskrona, Sweden.
15. Garousi, V., et al. *What industry wants from academia in software testing? Hearing practitioners' opinions*. in *International Conference on Evaluation and Assessment in Software Engineering*. 2017. Karlskrona, Sweden.
16. Garousi, V., K. Petersen, and B. Özkan, *Challenges and best practices in industry-academia collaborations in software engineering: a systematic literature review.* Information and Software Technology, 2016. **79**: p. 106–127.
17. Garousi, V. and K. Herkiloğlu. *Selecting the right topics for industry-academia collaborations in software testing: an experience report*. in *IEEE International Conference on Software Testing, Verification, and Validation*. 2016.
18. Rafi, D.M., et al. *Benefits and limitations of automated software testing: Systematic literature review and practitioner survey*. in *Proceedings of the International Workshop on Automation of Software Test*. 2012.
19. Ramler, R. and K. Wolfmaier. *Economic perspectives in test automation: balancing automated and manual testing with opportunity cost*. in *Proceedings of international workshop on Automation of software test*. 2006. ACM.
20. Garousi, V. and E. Yıldırım. *Introducing automated GUI testing and observing its benefits: an industrial case study in the context of law-practice management software*. in *Proceedings of IEEE Workshop on NEXt level of Test Automation (NEXTA)*. 2018.
21. Jolly, S.A., V. Garousi, and M.M. Eskandar. *Automated Unit Testing of a SCADA Control Software: An Industrial Case Study based on Action Research*. in *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 2012.
22. CKC Seminars, *www.testautomationday.com.* Last accessed: May 2019.
23. Li, N., A. Escalona, and T. Kamal. *Skyfire: Model-based testing with cucumber*. in *IEEE International Conference on Software Testing, Verification and Validation*. 2016.
24. Sivanandan, S. *Agile development cycle: Approach to design an effective Model Based Testing with Behaviour driven automation framework*. in *Annual International Conference on Advanced Computing and Communications*. 2014.
25. Urul, G., V. Garousi, and G. Urul. *Gerçek Zamanlı Gömülü Yazılımlar için Test Otomasyonu: Türkiye Endüstrisinden Bir Yaklaşım ve Deneyim Raporu*. in *Ulusal Yazılım Mühendisliği Sempozyumu (UYMS)*. 2014.
26. Garousi, V., et al., *Experience in automated testing of simulation software in the aviation industry.* IEEE Software, July/August 2019.
27. Kitchenham, B.A., T. Dyba, and M. Jorgensen. *Evidence-based software engineering*. in *Proceedings of the international conference on software engineering*. 2004.
28. Starreveld, A., *How to create an effective test automation strategy.* https://medium.com/@abstarreveld/considerations-for-an-effective-test-automation-strategy-a5bd027b3fa3, Last accessed: May 2019.
29. Gauge team, *Writing Specifications in Gauge.* https://docs.gauge.org/latest/writing-specifications.html, Last accessed: May 2019.
30. Ammann, P. and J. Offutt, *Introduction to Software Testing*. 1st ed. 2008: Cambridge University Press.
31. Meszaros, G., *xUnit Test Patterns*. 2007: Pearson Education.
32. Abrahms, J., *Selenium's Page Object Pattern: The Key to Maintainable Tests.* https://justin.abrah.ms/python/selenium-page-object-pattern--the-key-to-maintainable-tests.html, Last accessed: Apr. 2017.
33. Garousi, V. and B. Küçük, *Smells in software test code: A survey of knowledge in industry and academia.* Journal of Systems and Software, 2018. **138**: p. 52-81.
34. Shewchuk, Y. and V. Garousi. *Experience with Maintenance of a Functional GUI Test Suite using IBM Rational Functional Tester*. in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*. 2010.