Web Uygulamaları için Model Bazlı Test Süreci Otomasyonu

Uğur Özkan[1], Hasan Sözer[2]
Özyeğin Üniversitesi, İstanbul, Türkiye
[1]ugur.ozkan@ozu.edu.tr, [2]hasan.sozer@ozyegin.edu.tr

Özet. Yazılım sistemlerinin boyutları ve karmaşıklıkları arttıkça, bu sistemlerin test edilmesi daha maliyetli olmaktadır. Maliyeti düşürmek ve verimliliği artırmak için model bazlı test teknikleri geliştirilmiştir. Bu teknikler ile sistem kullanım modelinden otomatik olarak test senaryoları oluşturulabilmektedir. Fakat sistem modelinin manuel bir şekilde oluşturulması gerekmektedir. Ayrıca, model ile oluşturulan test senaryolarının sistem üzerinde otomatik çalıştırılmalarını sağlamak için betikler veya özel programlar geliştirilmesi gerekmektedir. Bu çalışmada, Web uygulamaları için model bazlı test sürecini otomasyon desteği ile iyileştirmekteyiz. Farklı araçları birleştirerek, sistem modelinin yarı-otomatik bir yöntem ile oluşturulmasını ve bu model ile oluşturulan test senaryolarının, ayrı bir program geliştirilmesine gerek olmaksızın, sistem üzerinde otomatik çalıştırılmalarını sağlamaktayız.

1 Giriş

Bugün insanlar hemen her alanda yazılım sistemlerine artan bir şekilde bağımlı hale gelmektedir. Bu sebeple yazılım güvenirliğini sağlamak için yazılım testleri büyük bir önem taşımaktadır. Diğer yandan, yazılım sistemlerinin boyutları ve karmaşıklığı artmaktadır. Sonuç olarak, test için harcanan süre artmakta ve ağırlıklı olarak manuel iş gerektiren test aktiviteleri atlanmaktadır.

Manuel iş miktarını azaltarak yazılım test süreçlerini daha verimli hale getirmek üzere Model Bazlı Test (MBT) teknikleri önerilmiştir [1] [2]. MBT, teste tâbi tutulan bir sistemin modellenmesine dayanan bir yazılım test tekniğidir. Bu teknikle, geliştirilen bir model veya modellerden bir dizi soyut test senaryoları oluşturulmakta ve oluşturulan bu senaryolar daha sonradan somut hale getirilerek teste tâbi tutulan sistemde uygulanmaktadır. Uygulanan test senaryolarından elde edilen sonuçlar daha sonra analiz edilmektedir [1]. MBT etkin bir yöntem olmakla birlikte, model oluşturmak ve test senaryolarını somut/çalıştırılabilir test durumlarına dönüştürmek için çok fazla zaman ve çaba gerektirmektedir.

Bu bildiride, Web uygulamalarına yönelik MBT için model oluşturma ve test senaryolarının çalıştırılabilir test durumlarına dönüştürülmesi işlemlerinin yarı-otomatik yapılabilmesini sağlayan bir yöntem ve araç seti öneriyoruz. Otomasyon seviyesi az veya hiç programlama bilgisine sahip olmayan kişilerin bile MBT uygulayabilmesini sağlayacak niteliktedir. Yaptığımız deneylerde, basit bir Web

sayfasının MBT ile sınanması için gereken yarım saati aşkın sürenin, üç dakika gibi kısa bir zamana indirilebildiği görülmüştür.

Bildirinin organizasyonu: bir sonraki bölümde MBT hakkında genel bilgiler sunulmakta ve bu çalışmada kullanılan araçlar tanıtılmaktadır. Üçüncü bölümde, ele alınan problem ve ihtiyaçlar listelenmektedir. Dördüncü bölümde, çözüm yaklaşımımız ve yaklaşımımızı somutlaştıran süreç detaylarıyla anlatılmaktadır. Beşinci bölümde elde ettiğimiz sonuçlar tartışılmaktadır. Altıncı bölümde, çalışmamızla ilgili yapılmış diğer çalışmalar özetlenmektedir. Son olarak, yedinci bölümde temel çıkarımlar özetlenmekte ve ileriye dönük yapılacak çalışmalar listelenmektedir.

2 Model Bazlı Test ve Yazılım Test Otomasyon Araçları

Bu bölümde öncelikle MBT yaklaşımı kısaca açıklanmaktadır. Devamında ise bu çalışmada kullanılan araçlar ve teknolojiler tanıtılmaktadır.

MBT, yazılım test senaryolarının otomatik olarak oluşturulduğu, yazılım testlerini gerçekleştirmekte kullanılan, model bazlı geliştirme yaklaşımının (model-based development) bir uygulamasıdır [1]. MBT, sistemin gereksinimlerinin ve görevlerinin tanımlandığı modelleri temel almaktadır. Bu modeller genellikle teste tabi tutulacak sistemin beklenilen davranışlarının kısmî gösterimidir ve test senaryoları oluşturabilmek için gerekli olan bilgileri içermektedir. MBT için kullanılan modeller genellikle sistem hayata geçirilmeden önce hazırlansalar da, sistem uygulamaya konulurken veya konulduktan sonra da hazırlanabilmektedir. Her hâlükârda, bu modeller ağırlıklı olarak manuel bir şekilde geliştirilir ve dolayısıyla ek çaba gerektirir. Buna rağmen, harcanan emek ile elde edilen verim, geleneksel yazılım testi yaklaşımlarına göre avantaj sağlamaktadır [3] çünkü sistem davranışlarının sadece bir defa tanımlanması yeterli olmaktadır. Her yeni bir özellik için manuel bir şekilde oluşturulan test senaryoları yerine, güncellenen model ile otomatik oluşturulan senaryolar kullanılmaktadır. Böylece test senaryolarını güncellemek ve idâme ettirmek daha kolay olmaktadır.

Biz bu çalışmada MBT aracı olarak, Java dili ile geliştirilmiş açık kaynak kodlu bir araç olan GraphWalker[1] aracını (sürümü 3.2.0) kullanmaktayız. GraphWalker, sonlu durum makinalarından (Finite State Machine – FSM) yararlanarak test senaryoları üreten bir MBT aracıdır. GraphWalker, XML tabanlı GraphML[2] biçimine uygun bir şekilde oluşturulmuş modelleri işleyip, muhtemel test dizinlerini oluşturmaktadır. Kullanılan model bir yönlendirilmiş çizgedir (directed graph) ve sistem durumlarını (state) temsil eden birden çok düğüm (node) ve aksiyonları gösteren birden çok ayrıtlara (edge) sahip olabilmektedir.[1]

[1] http://graphwalker.org/
[2] http://graphml.graphdrawing.org/

GraphML dosyası bir çizim öğesi, ve bu öğenin içinde sıralanmış düğüm ve ayrıt öğeleri içermektedir. Her düğümün kendine özgü bir kimliğinin (ID) olması gerekmektedir. Aynı şekilde her ayrıtın da kendisine mahsus bir kimliğinin olmasının yanı sıra, bağlı olduğu kaynak ve hedef düğümlerin kimliklerini içermektedir. GraphML formatında çizimlerin oluşturulmasını ve değiştirilmesini sağlayan çeşitli araçlar bulunmaktadır. Biz bu çalışmada, Java programlama dilinde yazılmış açık kaynaklı bir çizim programı olan yEd[3] aracını (sürüm 3.14.0) kullandık.

GraphWalker, düğüm kaplamı (vertex coverage), ayrıt kaplamı (edge coverage), düğüm erişimi (vertex reach) gibi durma koşulları ile 'Random', ve 'A*' gibi yol üreticileri (path generator) kullanarak GraphML formatında oluşturulan modeli dolaşmaktadır. Bu araç uzun ve tahmin edilmesi zor dizinler oluşturduğu için rastlantısal olarak daha kapsamlı test sonuçlarının elde edilmesini sağlamaktadır. Fakat oluşturulan dizinler sistem üzerinde çalıştırılabilecek test durumları değildir.

Sistem üzerinde otomatik çalıştırılabilecek test durumlarını elde etmek için GraphWalker tarafından oluşturulan test adımlarına tekâbül eden betikler (script) yazmak gerekmektedir. Çalışmamızda, bu betikler açık kaynak kodlu bir yazılım aracı olan Selenium[4] (sürüm 2.8.0) kullanılarak otomatik bir şekilde oluşturulmaktadır. Selenium, her platformda çalışabilen, web sitelerinin testlerinde kullanılan, Java programlama dilinde yazılmış bir yazılım test aracıdır ve temelde kaydet – oynat (capture – replay) işlevleri sunmaktadır. Ayrıca, Web uygulamaları için test adımlarının otomatik çalıştırılmasını sağlayan komutları bir kütüphane olarak programcılara sağlamaktadır.

Yukarıda tanıtılan araçlar bir sonraki bölümde değinilecek olan problem ve ihtiyaçları gidermek amacıyla bir araya getirilmiş ve geliştirdiğimiz yardımcı araçlar ile birlikte çalışmaları sağlanmıştır.

3 Problem ve İhtiyaçlar

MBT, test senaryolarının sistematik ve otomatik bir şekilde oluşturulması, bu senaryoların etkin bir şekilde idâme ettirilmesini sağlamakla birlikte bazı kısıtlamalar ve dezavantajlar da içermektedir. Literatürde yer alan bazı çalışmalarda [4], MBT kullanımına ilişkin zorluklar raporlanmıştır. Bu zorluklar, çoğunlukla ücretli, ticarî MBT araçları kullanılmasına rağmen karşılaşılan, kullanılan araçtan çok, sürecin doğası gereği karşılaşılan zorluklardır. Genel olarak MBT süreci kapsamında gerçekleştirilen işlemler aşağıdaki gibi sıralanabilir:

1. Sistemin, kullanıcılar tarafından kullanım şekillerini, sistemden beklenen davranışlarını yansıtan bir model oluşturulmaktadır. Örneğin, biz çalışmamızda MBT aracı olarak GraphWalker aracını kullanıyoruz ve bu

[3] http://www.yworks.com
[4] http://www.seleniumhq.org/

araç için sistem davranışlarının bir sonlu durum makinası (Finite State Machine – FSM) olarak modellenmesi gerekmektedir.
2. Modeldeki bütün durum ve aksiyonlar (durum geçişleri) için ayrı ayrı test kodu veya betikler (script) yazılmaktadır.
3. MBT aracı modeli sistematik bir şekilde dolaşarak test senaryoları oluşturmaktadır. Bu senaryoların her biri, modelde yer alan durumların farklı birleşim ve sıralarda dolaşılması ile oluşan birer dizi şeklindedir.
4. Oluşturulan test senaryoları sistem üzerinde otomatik bir şekilde sırasıyla çalıştırılmaktadır. İkinci aşamada geliştirilen test kodları, senaryolara dâhil olan her durum dizisinin somut test adımlarına dönüşmesini sağlar.

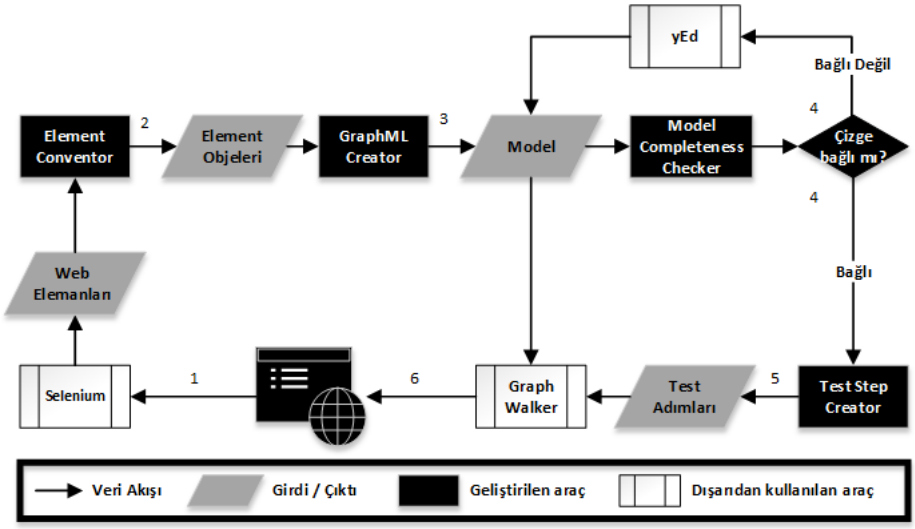
Yukarıdaki adımlardan ilk ikisi manuel olarak gerçekleştirilmektedir. İlk aşamada, sistem modelinin sıfırdan oluşturulması çok fazla zaman ve çaba sarf edilmesine sebep olmaktadır. Ayrıca model geliştirme sırasında hata yapılması olasılığı da mevcuttur. İkinci aşamada ise modelde yer alan her durum ve aksiyona karşılık gelen test kodları veya betikler yazılması gerekmektedir. Bu işlem de manuel bir şekilde gerçekleştirildiğinden emek gerektirir ve hata oluşumuna açıktır. Biz bu çalışmada Web uygulamalarına yönelik MBT kapsamında, manuel iş gerektiren ve hataya açık olan bu işlemlerin, otomasyon desteği ile iyileştirilmeleri için bir çözüm ve araç seti öneriyoruz. Bir sonraki bölümde çözüm yaklaşımımız ve yöntemimiz detaylarıyla açıklanmıştır.

4 Çözüm Yaklaşımı ve Yöntem

Çözüm yaklaşımımız genel hatları ile Şekil 1'de görülmektedir. Bu şekilde hem uygulanan süreç, hem de bu süreci destekleyen araçlar gösterilmektedir. Bu araçların bir kısmı daha önceden Bölüm 2'de tanıttığımız, açık kaynak kodlu, dışarıdan kullandığımız araçlardır (GraphWalker, yEd, Selenium). Diğer araçlar ise, bizim geliştirdiğimiz, tüm araçların bir arada çalışmalarını sağlayan ve sürecin baştan sona otomasyonunu destekleyen araçlardır. Esnekliği ve modülerliği artırmak için tüm araçlar tek başına çalışabilecek şekilde tasarlanmıştır. Bu sayede bu araçlar başka projelerde kullanılabilecek ya da ihtiyaçlara göre kolaylıkla değiştirilebileceklerdir. Bunun yanı sıra, geliştirilen tüm bu araçları ve süreci baştan sona kontrol eden, araçların uyumlu bir şekilde ve sırayla bir arada çalışmalarını sağlayan ayrı bir araç daha geliştirilmiştir[5].

Süreçten kısaca bahsetmek gerekirse, ilk aşamada 'Web Element Extractor' Selenium aracılığıyla açtığı Web sayfasını tarayıp "id" ve "name" nitelikleri olan Web elemanlarını ayıklıyor (1). Yapısı gereği bu araç diğer nitelikleri de ayıklama yetisine sahiptir.

[5] Tüm kaynak kodlara bu adresten ulaşılabilinir: https://github.com/uguurozkan/SeniorProject

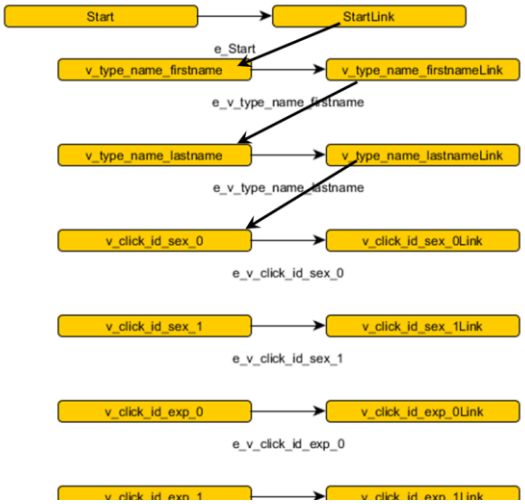


Şekil 1: Genel Süreç ve Araçlar

Ayıklanan Web elemanları, 'Element Convertor' aracılığıyla, 'Element' objelerine çevriliyor (2). Kullanıcı bu elemanlar üzerinden işlemler (seçme, değiştirme, tıklama gibi) yapabileceği için, işlemlerle ilgili olarak isimlendirme yapılıyor ve bu işlemlere ilişkin ayrı ayrı metodlar oluşturuluyor. Elemanların ve metodların isimlendirmeleri Selenium komutları ile tutarlı olacak şekilde yapılıyor ki ilerleyen aşamalarda her eleman için otomatik çalıştırılabilir test adımları oluşturulabilsin.

Oluşturulan 'Element' objeleri, modelin aslî yapısını oluşturmaktan sorumlu 'GraphML Creator' aracına veriliyor ve isteğe bağlı olarak boş, sadece düğümlerin olduğu, ya da düğümlerle beraber ayrıtların da olduğu bir diagramın saklandığı .graphml uzantılı bir dosya oluşturuluyor (3). Oluşan bu dosya sonradan eksikleri tamamlanacak modelin bulunduğu dosyadır.

Bu noktaya kadar süreç araçlar ile otomatik olarak ilerliyor; ama bu aşamada eğer oluşturulan model tamamlanmamışsa (çizge bağlı değilse) manuel müdahalenin yapılması gerekmektedir. Bu aşamada 'Model Completeness Checker', temel olarak modelde izole kalmış düğüm veya düğüm gruplarını tespit ediyor (4). Eğer modelde tüm düğümler birbirlerine bağlı değil ise, bir uyarı mesajı ile birlikte tamamlanmamış model Şekil 2'deki gibi yEd aracı ile kullanıcıya sunuluyor. Testi gerçekleştiren kişi bu araç ile modeli düzeltiyor ve 'Model Completeness Checker' güncellenen modeli tekrar teyit etmeye çalışıyor. Bu döngü teyit işlemi başarılı olana kadar devam ediyor. Genel süreç göz önünde bulundurulduğunda bu aşamanın manuel müdahale gerektiren tek aşama olduğu ve diğer aşamaların otomatik olarak yapıldığı vurgulanmalıdır.



Şekil 2: Başarısız Model Teyit İşlemi

Teyit aşaması başarılı bir şekilde sonlandıktan sonra 'Test Step Creator' ile üzerinde çalışılan proje için Şekil 3'deki arayüz ve bu arayüzden oluşturulan, içine Selenium komutlarının yerleştirildiği test adımları oluşturuluyor (5). Bu yapı .java uzantılı bir dosya olarak kaydediliyor (Bkz. Ek bölümde yer verilen Şekil 4).

```
//Generated by GraphWalker
(http://www.graphwalker.org)
package org.myorg.testautomation;

import ...

@Model(file =
"org/myorg/testautomation/Demo.graphml"
)
public interface Demo {

 @Vertex()
 void v_click_id_exp_6();

 @Vertex()
 void v_type_id_datepickerLink();

// Truncated

@Vertex()
 void v_click_id_profession_0();
}
```

Şekil 3: Test adımlarının tanımlandığı arayüz sınıfı

Bir sonraki aşamada GraphWalker, test adımlarını ve modeli kullanarak, modelin başlangıç noktasından itibaren, önceden tanımlanan algoritmasıyla bitirme koşulu sağlanana kadar yolları takip ediyor. Her düğüm ve ayrıtta, karşılık gelen metot, dolayısıyla metodun içindeki Selenium komutları, sistem üzerinde çalıştırılıyor (6).

Bu süreci örnek bir uygulamada denedik. Bir sonraki bölümde, gerçekleştirdiğimiz kontollü deneye ilişkin değerlendirme ve sonuçları paylaşıyoruz.

5 Değerlendirme

Önerdiğimiz yöntem ve geliştirmiş olduğumuz araçların etkinliğini test etmek ve sürecin manuel bir şekilde hazırlanmasına kıyasla ne kadar kolaylık getirdiğini ölçmek için programlama konusunda deneyimi az olan kişilerin katılımıyla kontrollü bir deney gerçekleştirdik.

Hipotezimiz, bir sistem üzerinde geliştrdiğimiz araçları kullanarak MBT uygulandığında sürenin manuel olarak uygulandığından daha kısa sürdüğü üzerinedir.

Kontrollü deneyimizi, değişkenleri en aza indirmek amacıyla programlama ile alâkalı derslerden en çok iki tane alan ve not ortalaması 2.80 üzerinde olan (14 adet) Endüstri Mühendisliği dördüncü sınıf öğrencileri ile aynı bilgisayar ortamında ve aynı web uygulaması üzerinde gerçekleştirdik.

Deneyde, katılımcılar rastlantısal bir şekilde eşit sayıda iki gruba ayrılmıştır. İlk gruptan öncelikle modelin ve test senaryolarının manuel olarak oluşturulması istenmiştir. Sonrasında ise aynı işlemin, geliştirilen araçlar yardımıyla yapılması istenmiştir. İkinci grupta yer alan öğrencilerden ise tam tersi sırada bu işlemlerin gerçekleştirilmesi istenmiştir. Yani, ikinci grup ilk olarak araç yardımı ile, sonradan ise manuel bir şekilde işlemleri gerçekleştirmiştir.

Deney sırasında işlemlerin gerçekleştirilme süreleri ölçülmüştür[6]. Ölçülen süreler Tablo 1'de gösterilmektedir.

Tablo 1'de görüldüğü üzere, araç yardımı olmadan yapılan işlemlerde ulaşılan en kısa süre 47 dakika iken geliştirilen araçların yardımı ile bu sürenin 3 dakikaya düştüğü görülmüştür[7].

Sonuç olarak, geliştirilen araçlar sayesinde sürecin teyit aşaması hariç tamamıyla otomatikleştirildiği, dolayısıyla süre nezdinde ciddi bir düşüş yaşandığı gözlemlenmektedir.

---

[6] Süre, programların çalıştırılmasından itibaren test sonuçlarının ekrana bastırılmasına kadar olan kısmı kapsamaktadır.

[7] Araçlar yardımıyla tamamlanan test sürecinin kayıtlarına http://youtu.be/8VQVu59J-Jo adresinden ulaşmak mümkündür.

Tablo 1: Deney sonuçları

	Katılımcılar	Geliştirilen araçlar kullanılarak[8]	Geliştirilen araçlar kullanılmadan[8]
Grup 1	Denek 1	3:58	≥ 120:00
	Denek 2	3:09	≥ 120:00
	Denek 3	3:52	≥ 120:00
	Denek 7	3:10	63:39
	Denek 8	3:18	77:43
	Denek 9	3:08	84:32
	Denek 10	3:10	77:39
Grup 2	Denek 4	3:22	≥ 120:00
	Denek 5	3:31	69:47
	Denek 6	3:57	≥ 120:00
	Denek 11	3:48	≥ 120:00
	Denek 12	3:03	47:03
	Denek 13	4:01	62:56
	Denek 14	3:13	49:31

Sürenin yanı sıra, araçların otomatik olarak test senaryolarını üretmesi, testlerin kapsamında da bir artışa neden olmuştur. Manuel yapılan testlerde katılımcıların büyük bir çoğunluğunun bir kaç düğümü modele eklemeyi unutmuş olduğu gözlemlenmiştir. Öte yandan geliştirilen araçlar aracılığıyla yapılan denemelerde, web uygulamasında yer alan elemanların[9] otomatik olarak ayıklanıp diagrama otomatik olarak yerleştirilmesinden dolayı bu tarz sorunlarla karşılaşılmamıştır.

Görüldüğü üzere, geliştirilen araçlar sayesinde test senaryolarının daha çok durumu kapsadığı ve manuel işlemler sırasında yapılabilecek hataların engellendiği gözlemlenmiştir.

Gerçekleştirdiğimiz kontrollü deneyde az sayıda kişiler ile denemeler yapmış olmamıza rağmen test otomasyon yaklaşımımızın potansiyel faydalarını gösteren sonuçlar elde edilmiştir.

Bir sonraki bölümde, literatürde yayınlanmış diğer ilgili çalışmalar özetlenmektedir.

6 İlgili Çalışmalar

Literatürde MBT üzerine yazılmış çok sayıda vaka çalışmaları bulunmaktadır [5].

[6] gibi mobil uygulamalarının GUI'si üzerine ya da [7] gibi yazılımların GUI'si üzerine yazılmış bir çok MBT uygulaması görmek mümkündür. Ama Li et al. [8]'un

[8] Süre dakika:saniye olarak belirtilmiştir.
[9] Bahsedilen düğümler sadece "id" ve "name" niteliklerini içeren elemanlardan oluşmaktadır.

araştırmasından da görülebileceği üzere model bazlı test yaklaşımıyla alâkalı yazıların oranı biraz düşük durumdadır.

Ricca ve Tonella [9], Web uygulamalarının üst düzey soyutlaması için UML modelleri sunmuşlardır. Sunulan bu modeller tamamıyla statik HMTL linklerinden oluşuyor ve herhangi bir Web uygulaması UML modelinin bir oluşumu olarak varsayılıyor. Model, HTML linklerinden oluşan statik bir çizim ve URL'den oluşan test senaryolarını oluşturan iki araçla desteklenmiştir. Araçlardan analiz yapmakla sorumlu olan ReWeb, bu işlemi gerçekleştirmek için bütün ulaşılabildiği Web sayfalarını indirmekte ve ilerleyen aşamalarda bunları testi uygulayan kişi birleştirmektedir. Bu yönüyle Beyaz Kutu (white-box) yaklaşımını sergilemektedir. Öte yandan bizim çalışmamızda Web uygulamasının herkese açık olan kaynak kodundan model üretilmektedir ve herhangi bir indirme işlemi de bulunmamaktadır. Dolayısıyla da testi gerçekleştiren kişi herhangi bir birleştirme işlemi yapmamaktadır. Ayrıca ikinci araç olan ve testlerden sorumlu TestWeb'de test senaryolarında girdilerin manuel bir şekilde girilmesi gerekmektedir. Bu yönüyle bizim çalışmamıza kısmen benzemektedir fakat gene de bizim çalışmamızda öncelikle rastlantısal olarak üretilen değerler kullanılmakta, gerekirse de testi gerçekleştiren kişinin müdahelesi gelmektedir. Ama test senaryolarının başarılı/başarısız olduğu, Ricca ve Tonella'nın çalışmasında testi gerçekleştiren kişi tarafından yapılırken, bizim çalışmamızda bu işlem tamamen otomatik yapılmaktadır.

Yang et al. [10] [11] Web uygulamalarını test etmeye yönelik geliştirilen test araçları için beş alt sistemden (geliştirme, ölçme, uygulama, başarısızlık analizi, yönetim) oluşan bir yapı önermişlerdir. Bu alt sistemlerden ikisi Anderews et al. [12] tarafından karşılanmaktadır. Bizim çalışmadaki sonlu durum makinaları ile yapılan modelleme üstte bahsedilen çalışmalar ile ortak özellik göstermektedir. Ama bizim çalışmamız özellikle varolan bir aracın hızlandırılması üzerine iken bu çalışmalar yeni birer araç ya da yapı oluşturmayı hedeflenmiştir.

Qian et al. [13] ise Web uygulamalarını muhtemel sonlu durum makinaları (Probable Finite State Machine – PFSM) ile test ederek MBT yaklaşımına olasılığı da katmıştır. Ama burada model tamamen manuel olarak oluşturulmak zorundadır.

Bir başka çalışmada [14] model üretmek yerine XML üzerinde yazılmış test senaryolarını çalıştırılmıştır. Bu çalışmada test senaryoları sistem üzerinde otomatik bir şekilde çalıştırılabilmektedir. Fakat bu senaryolar bir model baz alınarak oluşturulmamaktadır; her biri manuel bir şekilde tanımlanmaktadır.

GraphWalker aracı daha önceden MBT sürecini uygulamak üzere kullanılmıştır [15]. Fakat bu çalışmada daha çok çevik yazılım geliştirme süreci kapsamındaki MBT uygulama yöntemlerine odaklanılmıştır. Model oluşturma süreci ve oluşturulan test senaryolarının otomatik çalıştırılması üzerine bir çalışma yapılmamıştır.

7 Sonuç

Bu bildiride, MBT için geliştirilmiş bir araç olan GraphWalker, Web uygulamaları için test sürecini iyileştirmek üzere farklı araçlar ile bir arada kullanılmıştır. MBT sürecinde manuel olarak yapılması gereken, fakat (kısmen) otomasyon desteği sağlanabilecek aktiviteler bulunmaktadır. Öncelikle bu aktiviteler tespit edilmiş ve bu aktivitelerin otomasyonu için mevcut araçlar kullanılmış ve yeni araçlar geliştirilmiştir. Bir araya getirilen farklı araçlar sayesinde, test oluşturma hızı artarken harcanan zaman ve çaba düşmüştür. Buna ek olarak, bu araçlar sayesinde program yazma ihtiyacı ortadan kalkmış, programlamaya dair az veya hiç bilgisi olmayan insanlar için de model ve test oluşturma, testleri otomatik çalıştırma imkânı sunulmuştur.

Gelecekte yapılacak çalışmalar için bu araçların geliştirilmesi ve GraphWalker'ın Selenium ile içiçe nasıl daha etkin bir şekilde çalışacağı araştırılabilir. Bunun yanında, model oluşturma aşamasında makine öğrenme teknikleri ile otomasyonun artırılması da ilerisi için üzerinde durulabilecek araştırma konuları arasındadır.

Teşekkür

Gerçekleştirdiğimiz deneye katılımlarından ötürü Özyeğin Üniversitesi Endüstri Mühendisliği öğrencilerinden B. Uydaşoğlu, C. Özeltürkay, E. Elmas, F. Yorgancı, J. Aras, M. Boncuk, N. Altundağ, R. B. Parlar, S. Aşıtepe, S. Alkaya, T. Balcı, T. Özcan, T. Pınar ve Y. Yurd'a teşekkür ederiz.

Ek Bölüm

```
package org.myorg.testautomation;
import org.graphwalker.core.condition.ReachedVertex;
//Truncated
public class DemoTest extends ExecutionContext implements Demo {
    public final static Path MODEL_PATH =
Paths.get("org\\myorg\\testautomation\\Demo.graphml");
    private WebDriver driver;
    private String baseUrl;

    public void setUp() {
        driver = new FirefoxDriver();
        baseUrl = "http://localhost:801/";
        driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
        driver.get(baseUrl + "/xampp/CS401Website/");
    }
    @Override
    public void e_v_type_name_firstname() {
        driver.findElement(By.name("firstname")).clear();
        driver.findElement(By.name("firstname")).sendKeys("Testi");
    }
    @Override
    public void v_type_name_firstname() { // No need to fill. }
    @Override
    public void v_type_name_firstnameLink() { // No need to fill. }

//Truncated

    @Test
    public void runFunctionalTest() {
       new TestBuilder()
        .setModel(MODEL_PATH)
                .setContext(new DemoTest())
           .setPathGenerator(new RandomPath(new EdgeCoverage(100)))
           .setStart("e_Start")
            .execute();
    }
    @Test
    public void runStabilityTest() {
        new TestBuilder()
             .setModel(MODEL_PATH)
             .setContext(new DemoTest())
             .setPathGenerator(new RandomPath(new TimeDuration(30,
TimeUnit.SECONDS)))
             .setStart("e_Start")
             .execute();
    }
    @Override
    public void e_Start() { setUp(); }
    @Override
    public void StartLink() { // No need to fill. }
}
```

Şekil 4: Test Creator aracının oluşturduğu, test adımlarını içeren sınıf

Kaynaklar

1. Tretmans, J., Brinksma, E.: TorX: Automated Model-Based Testing. First European Conference on Model-Driven Software Engineering, 31-43 (2003)
2. Shafique, M., Labiche, Y.: A systematic review of model based testing tool support. Carleton University, Canada, Tech. Report SCE-10-04 (2010)
3. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco, USA (2006)
4. Binder, R.: Model-based Testing User Survey: Results and Analysis., online, www.robertvbinder.com/docs/arts/MBTUser-Survey.pdf (2012)
5. Neto, A., Subramanyan, R., Vieira, M., Travassos, G.: A survey on model-based testing approaches: a systematic review. In : Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies, Atlanta, pp.31-36 (2007)
6. Yang, W., Prasad, M., Xie, T.: A Grey-box Approach for Automated GUI-Model Generation of Mobile Applications. In: Fundamental Approaches to Software Engineering 7793. Springer Berlin Heidelberg, Rome, Italy (2013) 250-265
7. Nguyen, et al., A.: GUITAR: an innovative tool for automated testing of GUI-driven software. Automated Software Engineering 21(1), 65-105 (2014)
8. Li, Y.-F., Das, P., Dowe, D.: Two decades of Web application testing—A survey of recent advances. Information Systems 43, 20-54 (July 2014)
9. Ricca, F., Tonella, P.: Analysis and testing of Web applications. In : 23rd International Conference on Software Engineering, Toronto, pp.25-34 (2001)
10. Yang, J., Huang, J., Wang, F., Chu, W.: An object-oriented architecture supporting Web application testing. In : First Asian-Pacific Conference on Quality Sofware (APAQS '99), Japan, pp.122-129 (1999)
11. Yang, J.-T., Huang, J.-L., Wang, F.-J., Chu, W.: Constructing an object-oriented architecture for Web application testing. Journal of Information Science and Engineering 18(1), 59-84 (January 2002)
12. Andrews, A., Offutt, J., Alexander, R.: Testing Web applications by modeling with FSMs. Software & Systems Modelling 4(3), 326-345 (2005)
13. Qian, Z., Miao, H.: Towards Testing Web Applications: A PFSM-Based Approach. Advanced Materials Research 204-210, 220-224 (2011)
14. Jia, X., Liu, H.: Rigorous and automatic testing of web applications. In Hamza, M., ed. : Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002), Cambridge, pp.280-285 (2002)
15. Sivanandan, S., Yogeesha, C. B.: Agile Development Cycle: Approach to Design an Effective Model Based Testing with Behaviour Driven Automation Framework. In : ADCOM, Bangalore (2014)