Makale Türü: Araştırma Makalesi
Makale Gönderim Tarihi:08.03.2021
Makale Kabul Tarihi: 10.02.2022

Grafiksel Kullanıcı Arayüzü Testi İçin Bir Uçtan Uca Model Tabanlı Yaklaşım
An End-to-End Model-Based Approach for Graphical User Interface Testing

Alper Silistre[1], *Onur Kılınççeker*[2,3,5], *Fevzi Belli*[2,4], *Moharram Challenger*[5], *Geylani Kardaş*[1]

[1] Uluslararası Bilgisayar Enstitüsü
Ege Üniversitesi, Türkiye
alpersilistre@gmail.com, geylani.kardas@ege.edu.tr

[2] Faculty of Computer Science
Paderborn University, Germany
okilinc@mail.upb.de, belli@upb.de

[3] Bilgisayar Mühendisliği Bölümü
Muğla Sıtkı Koçman Üniversitesi, Türkiye
kilincceker@mu.edu.tr

[4] Bilgisayar Mühendisliği Bölümü
İzmir Yüksek Teknoloji Enstitüsü, Türkiye
belli@upb.de

[5] Department of Computer Science
University of Antwerp and Flanders Make, Belgium
moharram.challenger@uantwerpen.be

Özet

Model tabanlı Grafiksel Kullanıcı Arayüzü (GUI) testi, yazılım GUI testi içerisinde önemli bir yer tutmaktadır. Manuel test, zaman alıcı bir iştir ve büyük ölçüde hataya açıktır. Yazılım test topluluğunun uzun yıllardır üzerinde çalıştığı ve genel kullanımda olan birkaç test modeli vardır. Bu makale, model tabanlı GUI testinde kullanılan farklı modelleri incelemektedir. Test senaryoları oluşturmak ve bunları tek bir modelde birleştirmek amacıyla, kabul gören birkaç modelin Olay Sıra Çizgelerine (ESG) nasıl dönüştürüleceğine ilişkin bir yöntem önerilmiştir ve bunun kullanımını örnekleyen bir vaka çalışması sunulmuştur. Ayrıca bu makalede diğer modellerden dönüşümle elde edilen ESG modelinden test kümelerinin üretilmesi ve çalıştırılmasını içeren bir yaklaşım tanıtılmıştır. Deneysel çalışmalar öne sürülen bu yaklaşımın uygun ve etkili olduğunu göstermiştir. Bu kapsamda ESG'den elde edilen 20 mutant için öne sürülen yaklaşım en yüksek seviyede mutasyon skoru sonucunu vermiştir. Ayrıca gerçek bir sistem için gerçekleştirilen deneysel çalışmalar, ESG modelinden elde edilen test kümelerinin daha kompakt ve hata yakalamada daha başarılı olduğunu göstermektedir.

Anahtar kelimeler: GUI Testi, Model-Tabanlı Test, Sonlu Durum Makinesi, Olay Sıra Çizgesi, Olay Akış Çizgesi, Düzenli İfade

Abstract

Model-based Graphical User Interface (GUI) testing keeps its importance in software GUI testing. Manual testing is time-consuming and highly error prone. There are several test models in general use that the software testing community has been working on for many years. This article examines the different models used in model-based GUI testing. To create test cases and combine them into a single model, a method for how to convert several accepted models into Event Sequence Graphs (ESG) has been proposed, and a case study illustrating its use is presented. In addition, this article introduces an approach involving generating and running test sets over an ESG model obtained by the transformation from other models. Experimental studies have shown that this proposed approach is appropriate and effective. In this context, the application of the proposed approach enabled to receive the highest mutation score for 20 mutants obtained from the ESG. In addition, experimental studies for a real system show that the test sets obtained from the ESG model are more compact and more successful in detecting faults.

Keywords: GUI Testing, Model-Based Testing, Finite State Machine, Event Sequence Graph, Event Flow Graph, Regular Expression

1. Giriş

Grafiksel Kullanıcı Arayüzü (Graphical User Interface; GUI), web, mobil veya masaüstü tüm bilgisayar uygulamalarının önemli bir parçasıdır. Uygulamalar içerisinde dolaşabilmek ve uygulamanın bize sunduğu özelliklerini kullanabilmek için her türlü GUI elemanı ile etkileşime giriyoruz. GUI esas olarak yazılımlar ile iletişim kurabilmemiz için bir arayüzdür. Kullanıcı, uygulamayla etkileşim kurmak için bir düğmeyi tıklayarak veya bir giriş alanına metin yazarak bir eylem gerçekleştirebilir. GUI Testi, GUI test eden kişiler tarafından karar verilen ön koşullara dayalı olarak GUI elemanlarının davranışlarını ve durumlarını kontrol etme ve doğrulama işlemidir [1]. Mevcut yazılım ekosisteminde, GUI elemanlarının arkasındaki iş mantığını doğrulamak büyük öneme sahiptir. Her ne kadar GUI testi bir uygulama için önem arz etse de uygulamada az sayıda GUI öğesi olsa bile test edilmesi gereken çok sayıda olası test senaryosu nedeniyle uygulama geliştiricileri tarafından genellikle ihmal edilir. Aynı eylem, programın durumuna bağlı olarak programı bir hata durumuna sokabilir. Bunu manuel olarak test etmek zordur ve uygulamaların içinde hatalar varken işletilmesine neden olur. Bu nedenle, bir uygulamanın GUI'sini düzgün bir şekilde test etmek ve doğrulamak, hataları ve kusurları ortaya çıkarabilir. Bu, temelde yatan iş mantığını (business logic) test etmek kadar önemlidir. Ayrıca, modern yazılım dünyasında, özellikle akıllı telefonlardaki mobil uygulamalar gibi tüketici hedefli uygulamalar için uygulamanın kullanılabilirliği ayırt edici bir faktördür [2]. GUI testi bu gibi alanlarda önemli yer kaplamaktadır.

Model tabanlı test, yazılımda Kara Kutu Testi (Black-Box Testing) için popüler bir yöntemdir [12]. Sistem modelini daha yüksek bir soyutlama katmanında oluşturmak, bu modele dayalı olarak test senaryolarını oluşturmamıza olanak sağlar. Literatürde, Sonlu Durum Makinesi (Finite-State Machine; FSM) [3], Olay Akış Çizgesi (Event-Flow Graph; EFG) [7], Olay Sıra Çizgesi (Event Sequence Graph; ESG) [6][12] ve Düzenli İfade (Regular Expression; RE) [25][26] gibi farklı modeller mevcuttur. Model tabanlı test, test edilen sistemin (System Under Test; SUT) modeline (soyutlamaya) dayalı test senaryoları oluşturmamıza ve ardından bu testleri, tanımlanmış bir test kâhinine (test oracle) göre çalıştırmamıza olanak tanır. Bu konu etrafında incelenen ve geliştirilen otomatik araçlar ve süreçler mevcuttur. Kod tabanlı yöntemler yerine model tabanlı yöntemleri kullanmak, bu testleri kodla yürütmekten daha verimli bir şekilde test dizileri oluşturmamıza ve çalıştırmamıza olanak sağlar.

Bu çalışmada, bütünsel bir test üretme süreci oluşturmak amacıyla diğer modelleri ESG modeline dönüştürmeyi içeren ve elde edilen birleşik modeli kullanarak model-bazlı test yürütme işlemini sağlayan bir yöntem öneriyoruz. ESG modelinin, yararlanmak istediğimiz diğer modellere göre basitlik, genellik ve ölçeklenebilirlik gibi çeşitli avantajları vardır. Bu yöntem ile test dizileri otomatik olarak oluşturulur ve model tabanlı test oluşturma ve çalıştırma süreçlerini birleştirmek için bunlar ESG modelinde yürütülür. Mevcut modelleri ESG'ye dönüştürmenin ana nedeni, farklı modellerin uçtan uca model tabanlı testlerini uygulamak için farklı süreçlere ve uygulamalara ihtiyaç duymasıdır. Çalışmamızla, bu çabaları, test dizilerini oluşturmak ve çalıştırmak için verimli olan tek bir modelde birleştirmeyi amaçladık.

Deneyimlerimize ve literatür incelemesinden elde ettiğimiz sonuçlara dayanarak, test oluşturma ve çalıştırma adımları için ESG modelini kullanmaya karar verdik. Önerdiğimiz yöntemin kullanımını örneklemek için ayrıca bir durum çalışması da yine bu makalede yer almaktadır. Çalışmamızda ESG modelinin diğer modellere göre avantajları ortaya konulmuş ve elde edilen sonuçlar değerlendirilmiştir.

Bu makale, daha önceki konferans bildirimizin [31] genişletilmiş bir versiyonudur. [31]'de tanıtılan çalışma bu yeni çalışmada önerdiğimiz yöntemin öncül bir tasarımını sunmuştur. Bu makalede ise daha önce verilen bu tasarım genişletilmiş ve örnek bir sistem üzerinde uygulanmıştır. Yeni çalışmanın öncekini nasıl genişlettiği ve iyileştirdiği aşağıdaki maddelerde listelenmektedir:

- Kullanılan modellerin formel tanımlarına bağlı olarak, bu modellerden ESG modeline dönüşüm algoritmaları ve karmaşıklık analizleri verilmiştir.
- Verilen dönüşüm algoritmaları kullanılarak çalışmada adı geçen diğer modellerin ESG'ye dönüşüm işlemleri ve bunların örneklendirilmesi yerine getirilmiştir.
- Önerilen yöntem kapsamında gereken test kümesi üretim ve test koşumu aşamaları detaylandırılmıştır ve uygulamaları örnek çalışma üzerinde gösterilerek sonuçları verilmiştir.

Makalenin geri kalanı şu şekilde düzenlenmiştir: Bölüm 2 ilgili çalışmaları anlatmaktadır. Bölüm 3 önerilen yeni yöntemi tanıtmaktadır Bölüm 4 uygulama sonuçlarını ve mevcut çalışmanın geçerliliğine yönelik olası tehditleri aktarmaktadır. Son olarak, Bölüm 5'te sonuç ve geleceğe yönelik çalışmalar verilmiştir.

2. İlgili Çalışmalar

Bu bölüm, GUI testinde halihazırda var olan modellerle ilgili çalışmaları tanıtmaktadır.

Memon vd. [14] GUI'lerin kapsam kriterlerine odaklanır ve test edilecek önemli test dizilerini belirlemek ve GUI'yi bir hiyerarşide yapılandırmak için GUI bileşeni (component) terimini tanımlar. Bir GUI bileşeninde bulunan GUI elemanları arasındaki etkileşimi tanımlayan EFG'yi kullanarak GUI bileşenini temsil eder ve açıklar.

Memon [15], GUI tabanlı yazılım uygulamalarını test etmek için geleneksel yazılım tekniklerinin ve araçlarının neden GUI testi için uygun olmadığını açıklamaktadır. Ona göre GUI'ler soyutlama seviyeleri açısından uygulama kodlarından farklıdır. GUI testi sürecini ve GUI testini yapanların bu sürece nasıl yaklaşması gerektiğini anlatır. Makalede verilen örnekler yazım zamanı olan 2002'yi yansıtsa da karşılaşılabilecek zorluklar ve süreç günümüzde hala kabul edilir ve uygulanabilir durumdadır.

Belli [6], "bütünsel" (holistic) yaklaşım dediği yeni bir yaklaşımı önermektedir. Bu yaklaşımda, girdiler ve gerçekleştirilen eylemler hatalı olsa bile uygulamanın hata vermeden çalışması gerektiğini göstermek için sistemlerin sadece doğru girdilerle test edilmemesi, bu hatalı girdi ve eylemleri içeren test dizileri ile de test edilmesi gerektiğinden bahsetmiştir. Bu yöntemle, uygulama davranışı açısından

eksiksiz bir test kapsamasına sahip olacağımızdan bahsetmektedir.

Shehady ve Siewiorek [3], sistem tasarımını FSM ile eşdeğer tutarken, bir GUI için FSM'den daha az duruma sahip Değişken Sonlu Durum Makinesi (VFSM) adlı yeni bir model sunar. VFSM, test senaryoları oluşturmak için herhangi bir zamanda buna karşılık gelen bir FSM'e dönüştürülebilir. Toplam durum sayısı daha az olduğu için, bir sistemi VFSM ile modellemek FSM'den daha kolaydır ve bunun daha kısa sürede yapılacağından bahsetmişlerdir. White ve Almezen [5], sistem üzerinde gözlemlenebilir bir etki ile sonuçlanan bir veya daha fazla GUI nesnesini içeren bir etkinliği temsil eden ve sorumluluk (responsibility) adı verilen bir kavramı kullanırlar. Bu tanımlanmış sorumluluk için, bu sorumluluğu çağırabilen tüm eylemlerin ve GUI nesnelerinin bir birleşimi olan Tam Etkileşim Dizilerini (Complete Interaction Sequences; CIS) oluştururlar.

Memon vd. [7], EFG modelinden otomatik test üretimi için yapay zekâ tabanlı bir planlama algoritması olan yeni bir teknik sunar. Tanımlanan operatörlere bağlı olarak, planlama algoritmasını EFG modelinde uygulamak için ilk ve son adımlar oluşturulur. Algoritma, GUI olaylarını ve etkileşimlerini dikkate alarak ilk ve son durumlar arasında test dizileri oluşturur.

Memon [8], olay alanı keşif stratejilerini kullanarak modele dayalı test için yeni bir yöntem sunar. Model tabanlı test için tüm modelleri olay akışı modeli adı verilen ölçeklenebilir tek bir modelde birleştirir. Model oluşturma adımlarının maliyetini ve çabasını azaltmak için prosedürü otomatikleştirir.

Xie ve Memon [9], EIG ve EFG üzerindeki önceki çalışmalarını kullanarak GUI modelinde hatayı göstermek için gereken en kısa olay dizileri olan ve Minimal Etkili Olay Bağlamı (Minimal Effective Event Context; MEEC) adı verilen yeni bir kavram tanımlamışlardır. Bir GUI sisteminde, bir olaya verilen yanıt, bir sistemin mevcut durumuna bağlı olarak ertelenebileceğinden hatayı tespit etmek için test dizisini gereksiz yere uzun hale getirecek bir olay kombinasyonu üzerinden geçilebilir. Bunun yerine MEEC, arızayı tespit etmek için en kısa yolu gösterir.

Huang vd. [10], testin erken sonlandırılması gibi olasılıklar nedeniyle GUI testi için yararsız olan GUI test dizilerini onarmayı amaçlayan bir yöntem geliştirirler. Bu sorunlu test serilerini düzeltmek ve kapsamı artırmak için genetik bir algoritma kullanırlar.

Belli vd. [11], GUI'lerin doğruluğu hakkında deneysel bir anlayış elde etmek için GUI'lerin güvenilirliği ve insan-makine sistemlerinde bir GUI'nin güvenilirlik modelinin seçimi hakkında bir vaka çalışması sunar. GUI testi için uygun bir modelleme tekniğinin seçilmesinin değerlendirme sürecinin ve dolayısıyla yazılımın kalitesini etkilediğini belirtmişlerdir.

Banerjee vd. [1] GUI test çalışmaları hakkında anket yapıp ilgili belgeleri sistematik bir haritalama tekniği ile eşleştirmişlerdir. GUI testi hakkında 1991 ve 2011 arasında yazılan 230 makale havuzundan çalışmalar için seçim kriterlerini belirlemişlerdir. Çalışmaları sınıflandırıp daha fazla çalışma ve araştırma gerektiren mevcut yaklaşımlara ve alanlara genel bir bakış sağlamışlardır. Ayrıca, model tabanlı GUI testi için geleneksel ve modern tekniklerden örnekler sunmuşlardır.

Belli vd. [12] modelleme ve test senaryosu oluşturma tekniklerini dikkate alarak model tabanlı GUI testine ilişkin mevcut çalışmaları ayrıntılı olarak gözden geçirmişlerdir. Bu modellerin ve kullanımlarının gerçek dünyadan örneklerini verirken bu tekniklerin optimizasyonunu incelemişlerdir.

Tablo 1: GUI Testi için Kullanılan Modellerin Avantajları ve Dezavantajları

Model	Avantajları	Dezavantajları
ESG [6]	+Basit bir modelleme mekanizması sunar +Ölçeklenebilirlik sorununa uygun bir çözüm önerir +Test oluşturma için basit ve doğru bir yol sağlar	-ESG modeli, içindeki düğümlerde (node) GUI olaylarını tutması nedeniyle FSM gibi diğer bilinen modellerle karşılaştırıldığında bağlam bilgisi gerektirir.
EFG [14]	+Daha yüksek ifade gücü sayesinde farklı GUI bileşenleri için çeşitli yollarda modellemeleri kolaylaştırır. +Test üretimi için uygulanabilir bir çözüm sunar	-Ölçeklenebilirlik sorunu ile baş edemez -EFG modeli, içindeki düğümlerde GUI olaylarını tutması nedeniyle FSM gibi diğer bilinen modellerle karşılaştırıldığında bağlam bilgisi gerektirir
FSM [3]	+Basit bir modelleme mekanizması sunar	-Ölçeklenebilirlik sorununa çözüm getirmez -Test üretimi için karmaşık bir formalizasyon gerektirir
VFSM [3]	+Ölçeklenebilirlik sorununa uygun bir çözüm önerir	-FSM'den dönüşüm nedeniyle ek maliyete neden olur -Test üretimi için karmaşık bir çözüm gerektirir
RE [22]	+Modelleme için kompakt bir çözüm sunar ve test üretimini basitleştirir	-Ölçeklenebilirliği yönetmez -FSM'den dönüşüm nedeniyle ek maliyete neden olur

Belli vd. [13], test edilen sistemin çok büyük olması durumunda katmanlı merkezli test yöntemini ve ilgili test oluşturma sistemini önererek test senaryolarının sayısını ve maliyetini azaltan bir çalışma gerçekleştirmiştir. Bu metodolojiyi kullanarak, az sayıda test senaryosunda bile birçok hatalı durumun bulunabileceğini göstermiştir.

Kilincceker vd. [22], GUI'yi modellemek ve test etmek için düzenli ifadeyi tanıtıp aynı çalışmada kullanmışlardır. Ayrıca düzenli ifadeden rastgele test dizileri oluşturmuşlardır ve rastgele test üretme algoritmalarını bir vaka çalışması üzerinde değerlendirmişlerdir.

Mercan vd. [23], bir mobil uygulamanın GUI'sini modellemek ve test etmek için sonlu durum makinesini sunmuştur. Ayrıca,

sonlu durum makine modeline göre hataların varlığını ve yokluğunu test etmek için bir metodoloji önermiştir.

Kilincceker ve Belli [24], düzenli ifadeye dayalı bir analiz aracılığıyla GUI testi için dört yeni kapsam kriterini önermektedir. Normal ifadelerin analizinden sonra, kapsam kriterlerini sundukları bağlamsal tablolar elde ederler. Bu kapsam kriterleri, mutasyon testine dayalı kalite değerlendirmesi dahil olmak üzere test oluşturma ve test için [26]'da kullanılmıştır.

Kilincceker ve Belli [27], [43], hem donanım tasarımı hem de yazılım GUI testi için birleşik bir modelleme yöntemi sunmaktadır. Ayrıca, mutasyon testiyle birleştirilmiş bütünsel bir test yöntemi kullanırlar. Modelleme ve test yöntemlerini donanım tasarımı ve yazılım GUI alanından aldıkları iki örnek olay incelemesinde değerlendirirler.

Kilincceker vd. [33], GUI tabanlı sistemler için model-tabanlı mutasyon testini kullanan ve model-tabanlı ideal test olarak adlandırılan bir yöntem öne sürmüşlerdir. Öne sürülen bu yaklaşım mutasyon testi ve Belli [6] tarafından ortaya atılan bütünsel testi (holistic testing) birleştirerek elde edilen yöntemin ideal test olmak için gereken güvenilirlik ve yeterlilik kriterlerini sağladığını iddia etmektedirler. Çalışma kapsamında öne sürülen yöntemin bu kriterleri sağladığını gerek teorik gerekse deneysel çalışmalar ile göstermişlerdir. Ayrıca öne sürülen yöntem, farklı yöntemler ile kıyaslanmıştır. Ancak kullanılan bazı adımlar halen elle gerçekleştirildiği için bu adımların otomatik hale getirilmesi ileriki çalışmalar olarak ifade edilmiştir. Kilincceker vd. [35] ayrıca benzer yaklaşımın gömülü sistemlerin fonksiyonel testleri için de geçerli olduğunu göstermişlerdir.

Mevcut çalışmalardan farklı olarak, Vos vd. [39], GUI testlerinin test betikleri olmadan yine otomatik olarak gerçekleştirilebildiğini göstermiştir. Bunun için TESTAR isimli bir araç geliştirilmiştir ve bu araç GUI tabanlı olayları otomatik olarak tespit edebilmekte, ardından bu olayları yine otomatik olarak mevcut GUI üzerinde koşabilmektedir. Chahim vd. [40], TESTAR aracını endüstriyel seviyede test etmişlerdir.

Valdes vd. [38], GUI tabanlı sistemlerin otomatik testlerine yönelik 30 yıllık çalışmaları özetleyen kapsamlı bir literatür analizini ortaya koymuşlardır. Arzu eden okuyucular, bu konu hakkında detaylı bilgileri, var olan yöntemleri ve mevcut problemleri bu çalışmayı okuyarak elde edebilirler.

Tablo 1 yukarıda değinilen ve şu anda GUI testinde kullanılan modellerin avantaj ve dezavantajlarını özetlemektedir. Önerdiğimiz yeni yöntem, mevcut çalışmalardan farklı olarak ESG modelini avantajlarından dolayı diğer modellerden dönüştürülebilir hale getirerek tekil bir modelleme imkânı sunmaktadır. Böylece diğer modeller ile ifade edilen sistemler öne sürülen yaklaşım ile tekil ESG modeli kullanarak test edilebilir olmaktadır.

3. Önerilen Yöntem

Önerilen yöntem, sistemin modelini temsil etmek için JSON veya XML gibi açık standart dosya formatlarının herhangi birinde bir ESG modeli oluşturmanın bir yolunu sağlar. ESG modeli FSM, Hiyerarşik FSM (Hierarchical Finite State Machine; HFSM), RE ve EFG'den dönüştürülebilmelidir. Bunların tümü, bir GUI sistemini modellememize izin veren literatürdeki modellerdir. Örneğin, bu bir web sitesinde bir kayıt formu veya bir mobil uygulamada kullanıcı etkileşimlerini kabul eden bir ekran olabilir.

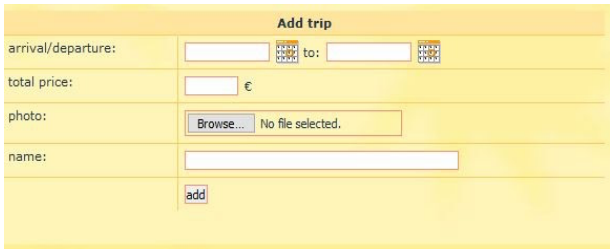
3.1. Kullanılan Modeller ve Dönüşümler

Kullanılan kavramlar bu bölümde resmi (formel) olarak tanımlanmıştır. Bu kavramlar FSM, HFSM, RE ve EFG'dir. Her resmi gösterim için, karşılık gelen modelleri içeren örnek bir GUI sistemi göstereceğiz. Örnek sistem, ISELTA [21] web sitesinin "Special" adı verilen modülünün basitleştirilmiş bir sürümüdür (Şekil 1). ISELTA, otel sağlayıcıları için bir çevrimiçi rezervasyon sistemidir ve buradaki "Special" modülü, yolculuk eklemek için bir formdur. ISELTA uygulaması bölüm 3.4 kapsamında daha detaylı olarak açıklanacaktır. Ayrıca ISELTA uygulamasının gerçek versiyonu yine bölüm 3.4 kapsamında, test koşumu için kullanılacaktır.

3.1.1. FSM

Tanım 1: Aşağıda verilen 5'li küme <Q, Σ, δ, q0, F> bir FSM [20] tanımlar

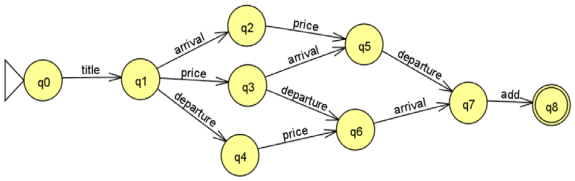
- Q: sonlu bir durum kümesi
- Σ: sonlu bir girdi sembol kümesi (alfabe)
- δ: durum geçiş fonksiyonu
- q0: Q kümesine ait olan başlangıç durumu
- F: Q kümesine ait olan bitiş durumlarının kümesi



Şekil 1: ISELTA Web sitesi "Special" modülü

Örnek 1: Aşağıda verilen 5'li küme ISELTA "Special" modülü için bir FSM tanımlar (bakınız Şekil 2). Burada "t", "d", "p", "a", ve "s" sembolleri sırasıyla (set title, set departure, set price, set arrival, add button) eylemlerini temsil eder. Gerçek sistemde "name" olarak verilen olay, model üzerinde "title" olarak ifade edilmiştir.

- Q: = {q0, q1, q2, q3, q4, q5, q6, q7, q8}
- Σ: = {t, a, p, d, s}
- δ: = {δ(q0, t)=q2, δ(q2, a)=q7, δ(q2, p)=q3, δ(q2, d)=q8, δ(q7, p)=q4, δ(q3, a)=q4, δ(q3, d)=q5, δ(q8, p)=q5, δ(q4, d)=q6, δ(q5, a)=q6, δ(q6, s)=q1}
- q0: = {q0}
- F: ={q1}



Şekil 2: ISELTA "Special" modülü FSM'i

3.1.2. HFSM

Tanım 2: Aşağıda verilen 6'lı küme <Q, Σ, δ, q0, F, L> bir HFSM [7] tanımlar.

- Q: sonlu bir durum kümesi
- Σ: sonlu bir girdi sembol kümesi (alfabe)
- δ: durum geçiş fonksiyonu
- q0: Q kümesine ait olan başlangıç durumu
- F: Q kümesine ait olan bitiş durumlarının kümesi
- L: sonlu bir katman kümesi

Örnek 2: Aşağıda verilen 6'lı küme ISELTA "Special" modül için bir HFSM tanımlar

- Q: = {q0, q1, q2, q3, q4, q5, q6, q7, q8}
- Σ: = {t, a, p, d, s}
- δ: = {δ(q0, t)=q2, δ(q2, a)=q7, δ(q2, p)=q3, δ(q2, d)=q8, δ(q7, p)=q4, δ(q3, a)=q4, δ(q3, d)=q5, δ(q8, p)=q5, δ(q4, d)=q6, δ(q5, a)=q6, δ(q6, s)=q1}
- q0: = {q0}
- F: = {q1}
- L: = {Ø}

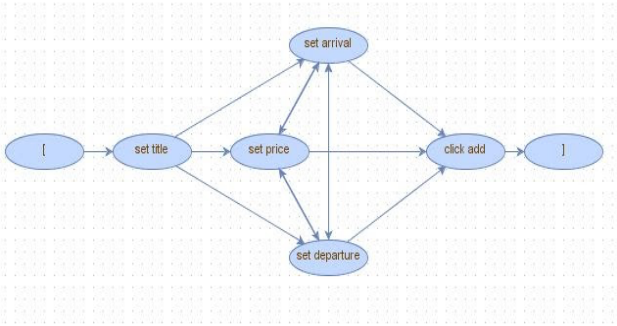
3.1.3. ESG

Tanım 3: Aşağıda verilen 4'lü küme <E, A, S, F> bir ESG [12] tanımlar

- E: eylemleri temsil eden sonlu bir düğüm kümesi
- A: A ⊆ N×N olaylar arasındaki ilişkiyi temsil eden yönlendirilmiş sonlu bir ark kümesi
- S: başlangıç eylemini temsil eden boş olmayan eylemler kümesi
- F: bitiş eylemini temsil eden boş olmayan eylemler kümesi

Örnek 3: Aşağıda verilen 4'lü küme ISELTA "Special" modülü için bir ESG tanımlar (bakınız Şekil 3).

- E: = {[, set title, set arrival, set price, set departure, click add,]}
- A: = {([, set title), (set title, set arrival), (set title, set price), (set title, set departure), (set arrival, set price), (set arrival, set departure), (set arrival, click add), (set price, set arrival), (set price, set departure), (set price, click add), (set departure, set arrival), (set departure, set price), (set departure, click add), (click add,])}
- S: = {[}
- F: = {]}



Şekil 3: ISELTA "Special" modülü ESG'si

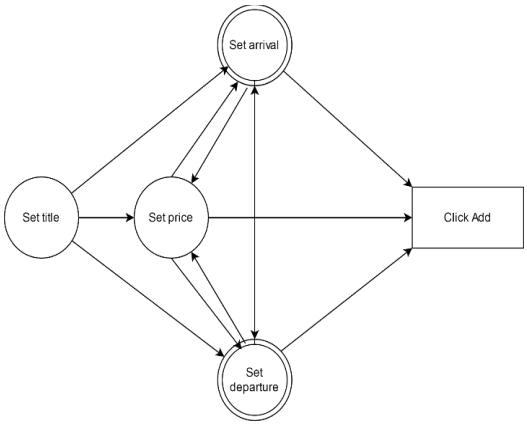
3.1.4. EFG

Tanım 4: Aşağıda verilen 4'lü küme <V, E, B, I> bir EFG [14] tanımlar

- V: tüm eylemleri temsil eden bir tepe kümesi
- E: tepeler arasındaki yönlü ayrıtların bir kümesi
- B: modellenmiş GUI'nin başlangıcında mevcut olan bir tepe kümesi
- I: bir GUI bileşeni için yasak olan (gerçekleşmesi mümkün olmayan) eylemler kümesi

Örnek 4: Aşağıda verilen 4'lü küme ISELTA "Special" modülü için bir EFG tanımlar (bakınız Şekil 4).

- V: = {set title, set arrival, set price, set departure, click add}
- E: = {([set title, set arrival), (set title, set price), (set title, set departure), (set arrival, set price), (set arrival, set departure), (set arrival, click add), (set price, set arrival), (set price, set departure), (set price, click add), (set departure, set arrival), (set departure, set price), (set departure, click add)}
- B: = {set title, set arrival, set price, set departure}
- I: = {Ø}



Şekil 4: ISELTA "Special" modülü EFG'si

3.1.5. RE

Tanım 5: Kurallar aracılığıyla bir RE, x, y, z, ... sembollerinin sırası ile tanımlanır. Semboller, RE'yi tanımlayan aşağıdaki kurallarla ilgili olarak sıfır veya daha fazla kez oluşabilir.

- Birleştirme: "." veya "" (boş) ile gösterilir. Örneğin, "ab", "a" anlamına gelir ve ardından "b" gelir
- Seçim: "+" ile temsil edilir. Örneğin, x + y, "x veya y" anlamına gelir.

Örnek 5: Aşağıda verilen örnek ISELTA "Special" modülü için bir düzenli ifade tanımlar.

R: = (tdpas+tapds+(tpda+tpad)s)

3.2. Model Dönüşümleri

Bölüm 3.1'de verilen formel modeller arası dönüşümlerin nasıl sağlanacağı bu bölümde anlatılmaktadır. Önerdiğimiz yöntemde kullanılan asıl model ESG'dir. Bu yüzden diğer modellerden ESG modeline dönüşümler için gerekli algoritmaların sözde kodları devam eden alt bölümlerde gösterilmektedir. HFSM'den ESG'ye dönüşüm kodu verilmemektedir. Bunun nedeni bu dönüşümün benzerinin FSM'den ESG'ye dönüşüm ile kolaylıkla yapılabilmesidir.

3.2.1. FSM'den ESG'ye Dönüşüm

Bu dönüşüm için gerekli sözde kod Şekil 5'te verilmiştir. Bu sözde kod girdi (FSM) ve çıktı (ESG) satırları ile başlar. Ardından bir döngü ile Lambda fonksiyonu bileşenleri içerisinde gezinilmektedir (bakınız satır 3). Bu döngünün bileşenleri Lambda'yı oluşturan alt fonksiyonlardır (f_i). Döngü içerisinde FSM'de tanımlanan her bir "t_i" bileşeni ESG'nin tepe değerleri kümesi olan "E" içerisine alınır (satır 4-5). Yine bir döngü sayesinde FSM'in Lambda fonksiyonu kullanılarak FSM durumlar arası komşuluk ilişkisi ile ESG'ye ait "A" kümesi (tepeler arası komşuluk) tanımlanır (satır 6-9). Son olarak FSM'e ait başlangıç ve bitiş ayrıtları kullanılarak ESG'ye ait başlangıç ve bitiş tepesi belirlenir (satır 10-15).

```
1  Girdi: FSM <Q, Sigma, Lambda, q0, F>
2  Çıktı: ESG <E, A, S, F>
3  for each function f_i in Lambda(f(sc, t)=sn)
4      sc_yeni = t_i
5      E = E union (sc_yeni)
6      for each function f_j in Lambda(f(sc, t)=sn)
7          if (sn_i neighbor with sn_j)
8              A = A union (sn_yeni, t_j)
9      end for
10     if (sn_i belongs to F)
11         A = A union (sn_i, "]")
12         F = "]"
13     if (sn_i belongs == q0)
14         A = A union (sn_i, "[")
15         S = "["
16 end for
```

Şekil 5: FSM'den ESG'ye Dönüşüm Sözde Kodu

FSM'den ESG'ye dönüşüm için gereken sözde kodun zaman karmaşası Lambda fonksiyonu bileşenlerine (n bileşen için) bağlı olarak ikinci derecedendir (quadratic time) ($O(n^2)$).

3.2.2. EFG'den ESG'ye Dönüşüm

Şekil 6, bu dönüşüm için gereken sözde kodu göstermektedir. Sözde kodda ilk olarak, EFG girdi değeri, ESG çıktı değeri olarak tanımlanır (satır 1-2). Ardından bir döngü ile EFG'ye ait tepeler arası ilişkiyi tanımlayan "E_efg" ikililerinin bileşenleri içerisinde gezinilir (satır 3-14). Bu döngü içerisinde elde edilen bileşenler ESG'ye ait bileşenler kümesini (A) tanımlar. Ardından yine bir döngü ile EFG'ye ait tepe değerleri içinde gezinilir (satır 5-11). Bu döngü sayesinde ESG'ye ait tepe kümesi olan "E_esg" elde edilir (satır 6). Ayrıca yine aynı döngü içerisinde, ESG'ye ait başlangıç ve bitiş tepeleri için ilgili bileşenler belirlenir ve bunlar bileşenler kümesi "A"ya eklenir (satır 7-10). Son olarak ESG'ye ait başlangıç ve bitiş tepeleri oluşturulur (satır 12-13).

```
1  Girdi: EFG <V, E_efg, B, I>
2  Çıktı: ESG <E_esg, A, S, F>
3  for each element i of E_efg(current, next)
4      A  = A union (i)
5      for each element j of V
6          E_esg =     E_esg union (j)
7          if (j is final event)
8              A = A union (j, "]")
9          if (j belongs to B)
10             A = A union ("[", j)
11     end for
12     F = "]"
13     S = "["
14 end for
```

Şekil 6: EFG'den ESG'ye Dönüşüm Sözde Kodu

EFG'den ESG'ye dönüşüm için kullanılan sözde kodun zaman karmaşası EFG tepe ilişkileri bileşenleri ve yine EFG tepe kümesi "V" eleman sayısına (n eleman için) bağlı olarak ikinci derecedendir ($O(n^2)$).

3.2.3. RE'den ESG'ye Dönüşüm

Bu dönüşüm için gereken sözde kod Şekil 7'de verilmiştir. Dönüşüm girdi "RE" ve çıktı "ESG" modeli tanımlamaları ile başlar (satır 1-2). Ardından gereken işlemler 3 adımda ifade edilmektedir. İlk adım RE modelinden kararlı olmayan sonlu durum makinesi (NFA: Non-deterministic Finite Automata) elde edilmesidir. Bu dönüşüm için gerekli adımlar Brüggemann-Klein tarafından [28]'de tanımlanmıştır. Bu dönüşüm yöntemi, Otomata Teorisi'ne (Automata Theory) dayanmaktadır. İlk adım için gereken dönüşüm çıktı olarak bir NFA üretmektedir. Elde edilen NFA'nın kararlı sonlu durum makinesine (DFA: Deterministic Finite Automata) dönüşümü sözde kodda ikinci adımda verilmektedir. Okuyucu, bu dönüşüm için detaylı bilgilere [28]'den ulaşabilir. Sonuç olarak elde edilen DFA modelinin ESG modeline dönüşümü 3. adımda verilmektedir. Bu dönüşüm Şekil 5'te verilen FSM'den ESG'ye dönüşüm sözde kodu ile çok benzerdir.

```
1  Girdi:RE
2  Çıktı:ESG <E, A, S, F>
3  Adım1:RE'den kararlı olmayan sonlu
4        durum makinesine(NFA) dönüşüm
5  Adım2:NFA'dan kararlı sonlu
6        durum makinesine(DFA) dönüşüm
7  Adım3:DFA'dan ESG'ye dönüşüm
```

Şekil 7: RE'den ESG'ye Dönüşüm Sözde Kodu

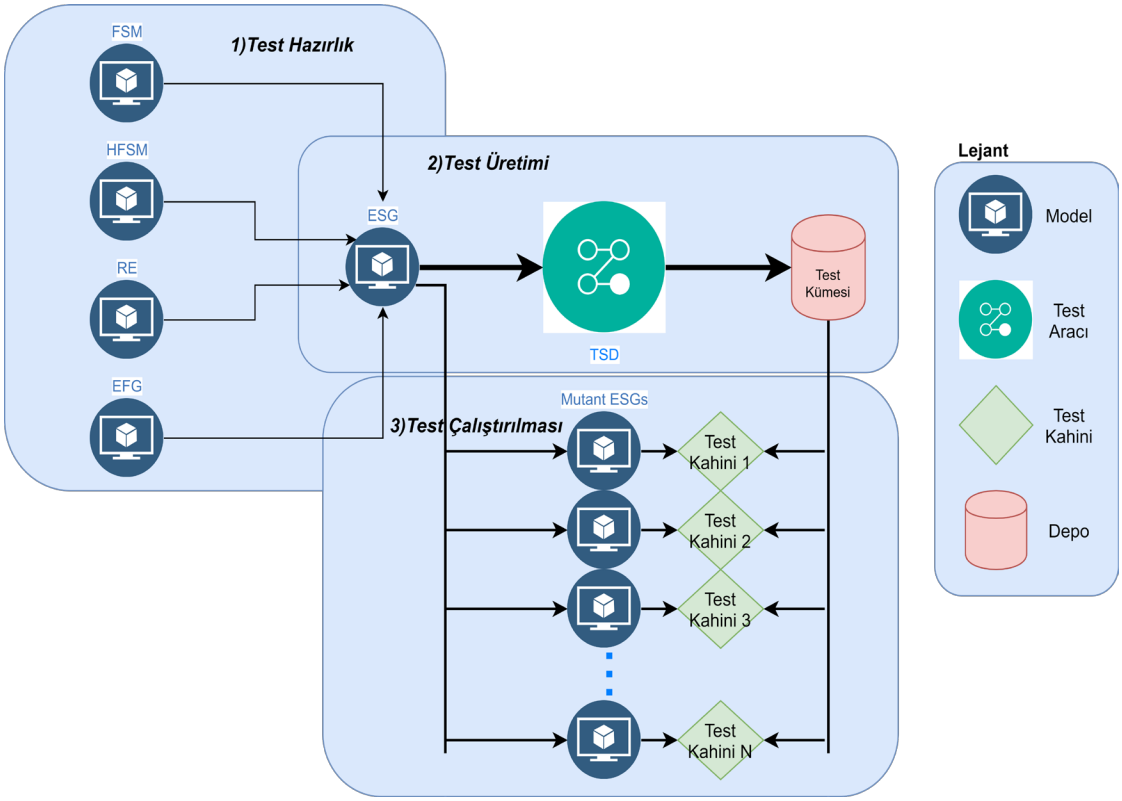
Burada verilen adım 1 için gerekli algoritmanın zaman karmaşası dönüşümlerde kullanılan bileşenlere (n bileşen için) bağlı olarak lineerdir ($O(n)$). Adım 2, genellikle geçerli tipik durumlar için ($O(n^3)$) ve adım 3 ise lineer zamanda gerçekleştirilebilmektedir. Bunun yanında adım 2, en kötü durumda $O(2^n n^2)$ zamanda gerçekleşmektedir. Böylece

RE'den ESG'ye dönüşüm için verilen sözde kodun genel tipik durumlar için zaman karmaşası yine üçüncü derecedendir ($O(n) + O(n^3) + O(n) => O(n^3)$). Zaman karmaşıklığı detayları için arzu eden okuyucular [41] çalışması, sayfa 165'i inceleyebilirler.

3.3. Yöntemin Uygulama Adımları ve Örnek Çalışma

Mevcut çalışma, test ettiğimiz sistemin fonksiyonel hatalarını tespit etmeyi amaçlamaktadır. Bu doğrultuda, önerilen yöntem, test hazırlama, test oluşturma ve test yürütme olmak üzere üç adıma bölünür (bakınız Şekil 8). Önerilen yöntemin bir uygulaması ISELTA web sitesi "Special" modülü üzerinde gerçekleştirilmiştir. Mevcut uygulama adımları çalışma sırasında dışardan müdahaleye gereksinim duymaktadır. Ancak komple bir model tabanlı test otomasyon aracı elde etmek için bu üç adım ileride tamamen otomatikleştirilebilir.

ESG modelinden test dizileri oluşturduktan sonra, sistemi test etmek için model tabanlı test yönteminin modele nasıl uygulanacağını göstereceğiz. Sonuçları göstermek için yaklaşımımızın örnek olay incelemesinde ISELTA web sitesinin formlarını kullanıyoruz (bakınız Şekil 1).



Şekil 8: Önerilen Yöntemin Genel Görünümü

3.3.1. Test Hazırlık

Bölümü 3.2'de ifade edildiği gibi test hazırlık aşamasında FSM, HFSM, RE ve EFG modelleri basitlik, genellik ve ölçeklenebilirlik avantajları nedeniyle ESG modeline dönüştürülür. Dönüşümler için gerekli algoritmalar daha önce yine Bölüm 3.2.'de verilmişti. Görsel olarak ISELTA web sitesinin "Special" formunun diğer modelleri de Bölüm 3.2'deki şekillerde dönüşüm tanımları ve örnekleri aşamasında verilmiştir. Test hazırlık aşamasında diğer modellerden dönüşüm yapılacağı gibi ayrıca kullanıcılar sistemlerini ESG'de modelleyip doğrudan yönteme de aktarabilirler.

ESG modellerinin mutantları, üzerine ekleme (insertion), değiştirme (replacement) ve çıkarma (omission) mutasyon operatörlerinin uygulanacağı orijinal ESG'den elde edilmiştir. Bu kapsamda Şekil 3'te verilen ESG modeline bu mutasyon operatörleri uygulanmış ve toplam 20 mutant elde edilmiştir. Bu mutantlar ESG'de verilen olaylar arası ilişkilerden kaynaklanabilecek fonksiyonel hataları modellemektedir. Bunlardan 2 tanesi ekleme, 9 tanesi değiştirme ve 9 tanesi çıkarma mutantıdır. Değiştirme ve çıkarma mutantları ESG modelinin olası tüm ayrıtlarına uygulanarak elde edilmiştir. Ekleme mutantları elde edilirken oluşan mutantların tamamının hatasız ESG modeline denk modeller ürettiği sonucuna varılmıştır. Bu yüzden deneysel olarak sadece 2 tane ekleme mutantı elde edilmiştir.

3.3.2. Test Kümesi Üretimi

Test oluşturma aşamasında, "graph traversal" algoritması kullanılarak ESG modelinden geçerli bir test dizisi seti içeren bir test grubu oluşturulur. Test Kümesi Üretimi işlemini otomatik olarak gerçekleştirmek için Padeborn Üniversitesi Uygulamalı Veri Teknolojisi bölümü tarafından geliştirilen ve Belli vd. [30]'da tanıtılan bir araç kullanılmıştır. Bu araç ESG modelini kullanmaya imkân vermektedir. Aracın kullanımı ile ilgili detaylı bilgilere [28]'den ulaşılabilir.

Test üretim aracı diğer model-tabanlı test araçlarının aksine ESG modelini kullanma sayesinde bir optimizasyon algoritması işleterek optimum test kümeleri üretimine olanak sağlamaktadır. Optimum test kümeleri üretimi için ESG'ye ait başlangıç ve bitiş tepeleri arasında kalan tepeler arasında tüm tepeleri kapsayan, en kısa yolları bulan bir algoritma kullanılmaktadır. Bu sayede diğer model-tabanlı test araçlarından daha sıkı (compact) ve daha etkili test kümeleri üretimi mümkün olmaktadır. Ayrıca bu araç sayesinde, ölçeklenebilirlik probleminin üstesinden gelmek için katmanlı ESG yapısı kurmak ve bu katmanlı yapıdan otomatik olarak sıkı ve etkili test kümesi elde etmek mümkündür.

Şu aşamada kullanılan programların girdileri, bir önceki aşamada çıkan sonuçların elle programa verilmesi ile sağlanmaktadır. Yöntem kapsamında test üretim aracı ile modeller arası dönüşüm işlemlerini gerçekleştiren araç arasında entegrasyon sağlanacak ve böylece diğer modellerden otomatik olarak dönüşüm yapılarak yine otomatik olarak test kümesi elde edilen bir araç geliştirilecektir.

```
1 [,set title,set price,set arrival,set departure,
2 set arrival,set price,set departure,set price,click add,],
3 [,set title,set arrival, click add,],
4 [, set title, set departure, click add,],
```

Şekil 9: Test Kümesi

Örnek çalışma olarak verilen "Special" modülüne ait ESG'den otomatik olarak elde edilen test kümesi Şekil 9'da verilmektedir.

3.4. Testlerin Çalıştırılması

3.4.1. Model Test Koşumu

Çalışmamız ilk aşamada GUI sistemlerini model-tabanlı olarak test etmek üzerine kurgulanmıştır. Bu sebeple, ilk aşamada test üretim aracının kullanılması sonucu elde edilen test kümeleri gerçek sistem üzerinde çalıştırılmak yerine yine modeller üzerinde çalıştırılır. Bu aşamada kullandığımız yöntem mutant bazlı model tabanlı test çalıştırma yöntemidir.

Hatasız ESG modelinden elde edilen test kümeleri, toplam 20 mutant üzerinde çalıştırılır. Verilen mutant modellerde bu test kümelerini çalıştırma işlemi toplam 20 mutant için yaklaşık olarak 8 dakika gibi bir zaman almıştır. Test çalıştırma işlemi esnasında test kümesinde verilen her bir test durumu için ESG üzerinde ilgili olayın olduğu yere gidilmiştir. Örneğin; test kümesinde {[, set title, set price} test sırası için mutant model üzerinde sırası ile bu test durumları takip edilmiştir. Takip etme işlemi için mutant ESG'nin olaylar arası komşuluk ilişkisi kullanılmıştır. Örneğin; mutant ESG'de "set title" ve "set price" olayları komşu değilse test "başarısız" (fail) veya bu olaylar komşu ise test "başarılı" (success) olmaktadır. Olaylar arası komşuluk ilişkisi test çalıştırma işlemi için test kâhini (test oracle) vazifesi görmektedir.

Test çalıştırma işleminden elde edilen sonuçlar Tablo 2'de verilmektedir. Çalışma sonuçlarına göre "Ekleme" mutasyon operatörü haricinde tüm hatalar tespit edilmiştir. Hata kapsama yüzdeleri "ekleme" operatörü haricinde yüzde 100 ve mutasyon skorları 1'dir. Yani tüm mutantlar başarı ile tespit edilmiştir. "Ekleme" mutasyon operatörü ile elde edilen mutantların yakalanamamasının sebebi bu mutantların hatasız ESG'ye denk olmasıdır. Toplam mutasyon skoru elde edilirken denk mutantların göz ardı edilmesi de gerektiğinden toplam mutasyon skoru 1'dir.

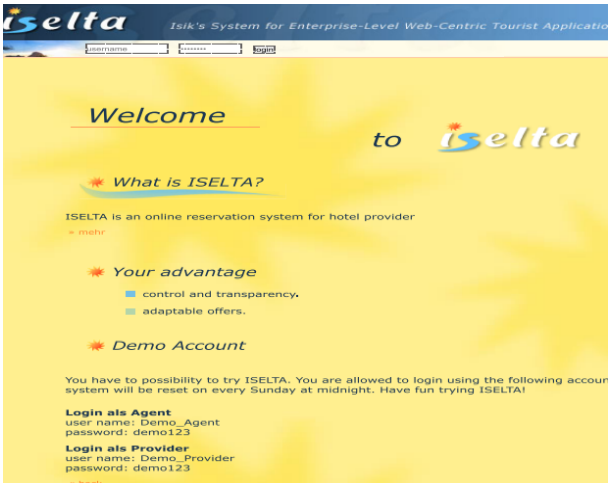
Tablo 2: Model Sistem Test Sonuçları

Mutasyon Operatorü	Mutant Sayısı	Tespit Edilen Hata Sayısı	Hata Kapsama (Yüzde (%))	Mutasyon Skoru
Ekleme	2	0*	0*	0
Çıkarma	8	8	100	1
Değiştirme	8	8	100	1
Toplam	20	16	80	1*

"Ekleme" operatörünün tek başına modele uygulanması sonucu elde edilen tüm mutantlar denk olmaktadır. Bunun sebebi bu operatörün tek başına bir modele uygulanmasının bu modelin hatasız halinde verilen yapısını değiştirmemesi ve sadece ekleme yapmasıdır. Bu yüzden daha sonraki çalışmalarımızda "ekleme" mutasyon operatörünün kullanılmaması düşünülmektedir.

3.4.2. Gerçek Sistem Test Koşumu

Gerçek sistem test koşumları çalışmamızın 2. aşamasıdır. Bu aşama için, "ISELTA" (bakınız Şekil 10) ismi verilen, Paderborn Üniversitesi ve Almanya merkezli ISIK turizm firması tarafından geliştirilen bir turizm web sitesi uygulaması seçilmiştir. Bu web uygulaması sayesinde turizm acentaları ve müşteriler daha kolay bir şekilde iletişim halinde olabilmektedir. Ayrıca acentalar, müşterilerine "Special" adı verilen modül sayesinde özel birtakım kampanyalar sunabilmekte ve müşteriler bu modül üzerinden sunulan kampanyalara rezervasyon yapabilmektedir. ISELTA uygulaması PHP programlama dilinde geliştirilmiştir ve yaklaşık olarak 70000 satır kod içermektedir. Okuyucular dilerlerse, Şekil 10'da verilen demo bilgilerini (kullanıcı adı: Demo_Agent, şifre: demo123 veya kullanıcı adı: Demo_Provider, şifre: demo123) kullanarak sisteme girebilir ve farklı roller için kayıtlar oluşturarak, bunlar için çeşitli durum çalışmaları yapabilirler.



Şekil 10: ISELTA web sitesi

ISELTA web sitesi test koşumları için hazırlanan ilgili materyaller (hatasız model, test kümeleri, test tasarımları ve kullanılan araçlar), [42]'de paylaşılmıştır. Dileyen okuyucular bu sayfayı inceleyerek adımları kendi başlarına gerçekleştirebilirler.

Deneysel çalışmalar kapsamında, ISELTA web sitesine ait toplam 12 adet hatayı gösteren mutant sistemler kullanılmıştır. Bu mutant sistemlerle ilgili bilgiler [33]'te bulunabilir. Bu hatalar literatürde, fonksiyonel hata olarak tanımlanan ve sistemin arzu edilen sonuç olayına ulaşmasına ancak istenilen çıktıları üretememesine neden olan durumlardır.

Elde edilen sonuçlar Tablo 3'te verilmektedir. Deneysel çalışmalar kapsamında ESG, FSM ve RE modelleri ve bunlara ait mutasyon skoru, hata kapsama, test kümesi büyüklüğü, test üretim zamanı ve test koşum zamanları kıyaslanmıştır. FSM tabanlı test üretimi için Graphwalker [34] isimli model-tabanlı test aracı kullanılmıştır. RE modelinden test üretimi için [35] çalışması kapsamında geliştirilen model-tabanlı test üretim aracı kullanılmıştır. Model büyüklükleri olarak ESG, 90 tepe ve 133 ayrıt, FSM, 60 tepe ve 112 ayrıt, RE ise toplam 283 olaydan oluşmaktadır.

Tablo 3: Gerçek Sistem Test Sonuçları

	ESG	FSM	RE
Model Büyüklüğü	90/133	60/112	283
Mutasyon Skoru	1	0.92	0.75
Hata Sayısı/Kapsama	12/12	12/11	12/9
Test Kümesi Büyüklüğü	240	767	228
Test Üretim Zamanı (milisaniye)	320	3870	263
Test Koşum Zamanı (saniye)	116	133	47

Tablo 3'te verilen bilgilere göre ESG modelinden üretilen test kümelerinin mutasyon skoru 1'dir ve tüm hatalar yakalanmıştır. Bunların içinde RE modelinden üretilen test kümeleri en düşük mutasyon skoruna ve hata yakalama kabiliyetine sahiptir. Test kümesi büyüklüğü olarak ESG ile RE birbirine yakın iken, FSM tabanlı Graphwalker ile gerçekleştirilen test kümesi 767 olaydan oluşmaktadır. Ayrıca test üretim zamanı olarak FSM diğerlerinin yaklaşık olarak 10 katıdır. Bunun sebebi Graphwalker aracının ilk çalışma anında geçen zamanın çok fazla olmasıdır. Test koşumları açısından RE tabanlı model en düşük değere sahiptir. Burada ortaya çıkan ilginç sonuç ESG ve FSM'in test koşum zamanlarının çok yakın olmasıdır. Bunun ana sebeplerinden birinin test kümelerinde hatalı olaya gelindiğinde ilgili test dizisinin devam ettirilmek yerine o anda kesilmesi ve devamında gelen test dizisine geçilmesidir. Çünkü hatayı yakalayan test dizisi için devamında gelen olayların bir önemi olmamaktadır.

Elde edilen sonuçlar kapsamında, ESG modeli ve buna bağlı test üretim algoritması [36], [37] diğer modellere göre gerek daha kompakt gerekse hata yakalama kabiliyeti olarak daha başarılı test kümeleri üretimine olanak sağlamaktadır.

4. Tartışma

4.1. Sonuçlar ve Çıkarımlar

Uçtan uca model tabanlı bir test oluşturulmuş, bir çalıştırma yöntemi geliştirilmiş ve bu yöntemi örnekleyen bir çalışma ile sonuçlar elde edilmiştir. Diğer modellerden üretilen veya doğrudan sisteme verilen birleşik bir model (ESG) yaklaşımın ana girdisidir. Çalışmada dönüşümleri mümkün kılacak algoritmalar Bölüm 3.2'de karmaşıklık analizleri ve sözde kodlarıyla birlikte sunulmuştur. Bununla beraber, bir test oluşturma yöntemi, ESG modeli kullanılarak incelenmiştir. Test oluşturma yönteminden üretilen test grubunun kalitesi, mutasyon testinin kullanılmasıyla değerlendirilmiştir. Bu çabaların, yazılım mühendisliğinin GUI testi alanında çalışan daha geniş bir kitle için model tabanlı testi daha erişilebilir hale getireceğini umuyoruz.

Yöntemimizin sonuçlarının kalitesine ve yazılım testi alanında çalışmalar yapan insanların bu yöntemi benimsemesine bağlı olarak, daha önce başka çalışmalarda kullanılmış mevcut sistem modelleri, bu modellerin verdiğimiz yöntemler yardımıyla ESG'ye dönüşümü sayesinde kullanabilir hale gelmiştir. Böylece uçtan uca test oluşturma ve çalıştırma yöntemimizin uygulanabilir olması bizim açımızdan çalışmamızın en iyi sonuçlarından biridir. Ayrıca bu çalışma yazılım test topluluğuna model tabanlı testin kullanımının kolaylığı ve bunun değerlendirmesi hakkında bir görüş sunmaya da imkân vermiştir.

Öne sürülen yaklaşım, GUI tabanlı sistemlerin çok ötesinde, makale kapsamında incelenen modellerin diğer tüm uygulama alanları için geçerlidir. Mevcut çalışmanın, test koşum aşaması farklı sistemlere uyarlanarak uçtan uca gereken tüm adımlar gerçekleştirilebilir. Zaten başka alanlara uygulanması, planlanan çalışmalar arasındadır.

4.2. Geçerliliğe Yönelik Tehditler

4.2.1. Sonuç Geçerliliği

Vaka çalışmamızın örneklem büyüklüğü, metodolojiyi genelleştirmek için potansiyel bir tehdittir. Çalışmada kullanılan örneklemimizin küçük olması nedeniyle, yöntemi doğrulamak ve gözden kaçırabileceğimiz olası sorunları bulmak için, her model (FSM, HFSM, EFG, ESG, RE) için büyük boyutta vaka çalışmalarına ihtiyacımız vardır. Bu olası problem nedeniyle, çalışmalarımızı ileride daha büyük test senaryolarıyla genişletmeyi planlıyoruz. Böylece, örnek model boyutu gerçek hayattaki sistemlere benzer boyutta olacak ve bu durumdaki sorunlar tespit edilebilecektir.

4.2.2. İçsel Geçerlilik

Model tabanlı testin yapısı, doğası gereği içsel geçerliliğe yönelik bir tehdittir. Çünkü yaklaşımın tamamı, gerçek GUI programı yerine modeller üzerinde çalışır. Beyaz kutu test (White-Box Testing) yöntemi ile yazılım kodunda testler çalıştırıyormuş gibi model bazlı testlerle bir sistemi tam olarak test etmek mümkün değildir. Bir model, gerçek yazılım davranışının yalnızca bir temsili ve soyutlamasıdır. Test edilen yazılımın karmaşıklığına bağlı olarak, sistemi doğru şekilde temsil edecek doğru bir model oluşturmak zor olabilir. Bu nedenle, sistemin başlangıç modelinin doğruluğu önemlidir ve yöntemimiz için bir tehdittir. Bir model sistemi yanlış şekilde temsil ediyorsa, tüm dönüştürmeler ve test oluşturma / çalıştır

yaklaşımı sistemi olması gerektiği gibi kapsamayacaktır. Önerilen yaklaşımın sistemi tam olarak temsil eden uygun modellerde uygulanmasını sağlamak için daha gelişmiş örnek bir sistem için modellerimizi oluşturacağız.

4.2.3. Dışsal Geçerlilik

Yaklaşımı çalışmanın içeriği dışında uygulamak, dış geçerliliğe yönelik bir tehdittir. Daha önce de belirtildiği gibi, mevcut çalışma, çoğunlukla oyunların GUI'sinde kullanılan görsel nitelikler ve bunların semantiği gibi diğer hata türleri yerine, işlevsel ve operasyonel hataları tespit etmeyi amaçlamaktadır. Böylece, mevcut çalışma daha çok menü ağırlıklı uygulamalar için uygundur. Bu tip uygulamalar GUI tabanlı uygulamaların büyük bir çoğunluğunu oluşturmaktadır. Bu, model tabanlı testin ne için oluşturulduğu ile ilgilidir. Modeller işlevsel olarak sistemleri temsil ettiğinden, bir sistemin ekrandaki görsel öğelerinin test edilmesi bu yaklaşım için uygun olmayabilir. Normalde, kod tabanlı bir test yaklaşımı bu tür doğrulamalar için daha uygun olabilir. Bununla birlikte, sıralı ve davranışsal modeller Petri-Nets modellemesinden ziyade bir test yönteminde kullanıldığında, önerilen bu yöntem uygulanabilir.

4.2.4. Yapısal Geçerlilik

Modellerin ESG modeline dönüştürülmesi, farklı seviyedeki ifadelerin zorlukları nedeniyle yapısal geçerlilik açısından tehdit oluşturmaktadır. Dönüştürme işleminden sonra, modellerin ifade gücü birleşik modelin ifade gücüne göre artar veya azalır. Bu da çıkarılan modelde bazı sistem özelliklerinin eksik olmasına neden olabilir. Dönüşüm sırasında bu ifade gücünün hangi düzeyde kaybedilebileceğini anlamak için daha büyük örneklem boyutuyla ve farklı durumlarda test edilmesi gereklidir. Daha önce de bahsettiğimiz gibi, ifade gücünün makul bir noktanın ötesine geçmesini önlemek ve bu konunun üstesinden gelmek için çalışmalarımızı daha büyük örneklerle genişletmeyi planlıyoruz.

5. Sonuç

Bu çalışmada, tasarımları yine bu makalede anlatılan ve GUI yazılım testi alanında iyi bilinen GUI test modellerini analiz etmeyi amaçladık. İlk analiz, farklı modellerin kullanımının farklı yetenekler gerektirdiğini ve farklı sözdizim ve semantikle sonuçlandığını göstermektedir. Bu farklılıklar, modellerin temsil etme yeteneklerini ve test üretme ve çalıştırma gibi diğer süreçlerini etkiler. Deneyimlerimize dayanarak önerilen yaklaşımda ESG'yi seçmemizin temel nedeni hem test üretme hem de çalıştırma açısından diğer modellere kıyasla daha uygun olmasıdır.

Önerilen yaklaşımın test hazırlama adımında, modeller ESG'ye dönüştürülüp test kümesi üretimi için hazır hale getirilmiştir. Ardından test oluşturma adımında orijinal modelden mutanlar üretilmiş ve bu orijinal ESG modelinden test dizileri oluşturulmuştur. Son olarak, üretilen test dizileri, test dizilerinin kalitesini değerlendirmek için ESG modelinin mutantları üzerinde çalıştırılmıştır. Deneysel çalışmalar göstermektedir ki öne sürülen yaklaşım model-tabanlı test için uygundur ve etkilidir. Örnek durum üzerinde gerçekleştirilen değerlendirme çalışmasında, denk mutantlar gelen test sürecinden çıkarıldıktan sonra mutasyon skoru en yüksek seviyede elde edilmiştir. Ayrıca gerçek bir sistem üzerinde gerçekleştirilen deneysel çalışma sonuçları göstermektedir ki, ESG tabanlı test kümeleri diğer yöntemlere göre daha kompakt ve hata yakalama kabiliyeti açısından daha başarılı sonuçlar vermektedir.

Bu çalışmanın devamında öncelikle bu makalede tanıttığımız uçtan uca model tabanlı test üretme ve çalıştırma yönteminin adımlarının otomatik olarak uygulanmasına imkân verecek yeni bir aracın geliştirilmesi hedeflenmektedir. Bir diğer çalışma da ise sonuçların geçerliliğini arttırmak amacıyla daha büyük ölçekli modellerin kullanılması planlanmaktadır. Öte yandan katmanlı ESG modellerinin ve topluluk belirleme (community detection) algoritmalarının kullanılması testlerin ölçeklendirilmesine de katkı verebilir. Bununla ilgili elde ettiğimiz ilk sonuçlar [32]'de aktarılmıştır. Bu çalışmada tanıttığımız yönteme [32]'deki yaklaşımımızı da dahil ederek yine mevcut yöntemin daha büyük modeller üzerine uygulanabilmesini kolaylaştırmayı hedefliyoruz.

6. Kaynaklar

[1] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository," Information and Software Technology 55, no. 10, pp. 1679-1694, 2013.

[2] Harrison, R., Flood, D. & Duce, D. Usability of mobile applications: literature review and rationale for a new usability model. J Interact Sci 1, 1 (2013). https://doi.org/10.1186/2194-0827-1-1

[3] R. K. Shehady, D. P. Siewiorek, "A method to automate user interface testing using variable finite state machines," Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing, Seattle, WA, USA, pp. 80-88, 1997.

[4] T. S. Chow, "Testing software design modeled by finite-state machines," in IEEE Transactions on Software Engineering, vol. SE-4, no. 3, pp. 178-187, May 1978.

[5] L. White, H. Almezen, "Generating test cases for GUI responsibilities using complete interaction sequences," Proceedings 11th International Symposium on Software Reliability Engineering. ISSRE 2000, San Jose, CA, USA, pp. 110-121, 2000.

[6] F. Belli, "Finite state testing and analysis of graphical user interfaces," Proceedings 12th International Symposium on Software Reliability Engineering, Hong Kong, China, pp. 34-43, 2001.

[7] A. M. Memon, M. E. Pollack, M. L. Soffa, "Hierarchical GUI test case generation using automated planning," in IEEE Transactions on Software Engineering, vol. 27, no. 2, pp. 144-155, February 2001.

[8] A. Memon. "An event‐flow model of GUI‐based applications for testing," Software testing, verification and reliability 17.3 pp. 137-157, September 2007.

[9] Q. Xie, A. M. Memon, "Using a pilot study to derive a GUI model for automated testing," ACM Trans. Software Eng. Methodol. 18, 2, pp. 1–35, November 2008.

[10] S. Huang, M. B. Cohen and A. M. Memon, "Repairing GUI test suites using a genetic algorithm," 2010 Third International Conference on Software Testing, Verification and Validation, Paris, pp. 245-254, 2010.

[11] F. Belli, M. Beyazit, N. Güler, "Event-Oriented, model-based GUI testing and reliability assessment-approach and case study," Advances in Computers, 85, pp. 277-326, 2012.

[12] F. Belli, M. Beyazıt, C. J. Budnik, T. Tuglular, "Advances in model-based testing of graphical user interfaces," In Advances in Computers, vol. 107, pp. 219-280. Elsevier, 2017.
[13] F. Belli, N. Güler, and M. Linschulte, "Layer-centric testing," FERS-Mitteilungen: Vol. 30, No. 1, pp. 55-62, 2012.
[14] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for GUI testing," Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, pp. 256-267, September 2001.
[15] A. M. Memon, "GUI testing: pitfalls and process," in Computer, vol. 35, no. 8, pp. 87-88, August 2002.
[16] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," in Proceedings of the IEEE, vol. 84, no. 8, pp. 1090-1123, August 1996.
[17] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou and A. Ghedamsi, "Test selection based on finite state models," in IEEE Transactions on Software Engineering, vol. 17, no. 6, pp. 591-603, June 1991.
[18] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model ‐ based testing approaches," Software testing, verification and reliability 22.5, pp. 297-312, 2012.
[19] F. Belli, M. Beyazit and A. Memon, "Testing is an Event-Centric Activity," 2012 IEEE Sixth International Conference on Software Security and Reliability Companion, Gaithersburg, MD, pp. 198-206, 2012.
[20] J. E. Hopcroft, R. Motwani, and J. D. Ullman, Automata theory, languages, and computation." International Edition 24.2.2, 2006.
[21] ISELTA websitesi, Çevrimiçi olarak mevcut: http://iselta.ivknet.de, Son erişim: 21.03.2022.
[22] O. Kilincceker, A. Silistre, M. Challenger and F. Belli, "Random Test Generation from Regular Expressions for Graphical User Interface (GUI) Testing," 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Sofia, Bulgaria, pp. 170-176, 2019.
[23] G. Mercan, E. Akgündüz, O. Kilincceker, M. Challenger, and F. Belli, "Android uygulaması testi için ideal test ön çalışması," CEUR Workshop Proceedings, 2018.
[24] O. Kilincceker, and F. Belli, "Grafiksel kullanıcı arayüzleri için düzenli ifade bazlı test kapsama kriterleri," CEUR Workshop Proceedings, 2017.
[25] O. Kilinccceker, E. Turk, M. Challenger and F. Belli, "Regular expression based test sequence generation for HDL program validation," 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, pp. 585-592, 2018.
[26] O. Kilincceker, E. Turk, M. Challenger and F. Belli, "Applying the ideal testing framework to HDL programs," ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems, Braunschweig, Germany, pp. 1-6, 2018.
[27] O. Kilincceker and F. Belli, "Towards uniform modeling and holistic testing of hardware and software," 2019 1st International Informatics and Software Engineering Conference (UBMYK), Ankara, Turkey, pp. 1-6, 2019.
[28] Brüggemann-Klein, A. (1993). Regular expressions into finite automata. Theoretical Computer Science, 120(2), 197-213.
[29] M. Linschulte, On the Role of Test Sequence Length, Model Refinement, and Test Coverage for Reliability (PhD Thesis, Univ. Paderborn), 2014.
[30] F. Belli, A.T. Endo, M. Linschulte, and A. Simao, A Holistic Approach to Model-Based Testing of Web Service Compositions, Software: Practice and Experience, vol.44, no.2, pp. 201–23, 2014.
[31] A. Silistre, O. Kilincceker, F. Belli, M. Challenger and G. Kardas, "Models in Graphical User Interface Testing: Study Design," 2020 Turkish National Software Engineering Symposium (UYMS), Istanbul, Turkey, 2020, pp. 1-6, doi: 10.1109/UYMS50627.2020.9247072.
[32] A. Silistre, O. Kilincceker, F. Belli, M. Challenger, and G. Kardas, "Community Detection in Model-based Testing to Address Scalability: Study Design", 15th Conference on Computer Science and Information Systems (FedCSIS 2020), Track on Software and Systems Engineering, Advances in Software and Systems Engineering (ASSE 2020), Sofia, Bulgaria, 2020, IEEE, pp. 657-660, DOI: 10.15439/2020F163.
[33] O. Kilincceker, A. Silistre, F. Belli and M. Challenger, "Model-Based Ideal Testing of GUI Programs–Approach and Case Studies," in IEEE Access, vol. 9, pp. 68966-68984, 2021, doi: 10.1109/ACCESS.2021.3077518.
[34] Karl, K., 2013. Graphwalker. *URL: www. graphwalker. org* Son erişim: 21.03.2022.
[35] Kilinccceker, O., Turk, E., Challenger, M., & Belli, F. (2018, July). Regular expression based test sequence generation for HDL program validation. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (pp. 585-592). Ieee.
[36] Budnik, C. J. (2006). Test generation using event sequence graphs (Doctoral dissertation, University of Paderborn, Germany).
[37] Belli, F., Nissanke, N., Budnik, C. J., & Mathur, A. (2005). Test generation using event sequence graphs. University of Paderborn, Institute for Electrical Engineering and Information Technology.
[38] Rodríguez-Valdés, O., Vos, T. E., Aho, P., & Marín, B. (2021, September). 30 years of automated GUI testing: a bibliometric analysis. In International Conference on the Quality of Information and Communications Technology (pp. 473-488). Springer, Cham.
[39] Vos, T. E., Aho, P., Pastor Ricos, F., Rodriguez ‐ Valdes, O., & Mulders, A. (2021). testar ‐ scriptless testing through graphical user interface. Software Testing, Verification and Reliability, 31(3), e1771.
[40] Chahim, H., Duran, M., Vos, T. E., Aho, P., & Fernandez, N. C. (2020, September). Scriptless testing at the GUI level in an industrial setting. In International Conference on Research Challenges in Information Science (pp. 267-284). Springer, Cham.
[41] Aho, Alfred V., Lam, Monica S., Sethi, Ravi., & Ullman, Jeffrey D. (2007). Compilers Principles, Techniques, & Tools. Addison-Wesley.
[42] Emo dergi Github materyaller, Çevrimiçi olarak mevcut: https://github.com/alpersilistre/gui-emo-dergi, Son erişim: 21.03.2022.
[43] Kilincceker, O., Turk, E., Belli, F. et al. Model-based ideal testing of hardware description language (HDL) programs. Softw Syst Model (2021). https://doi.org/10.1007/s10270-021-00934-6

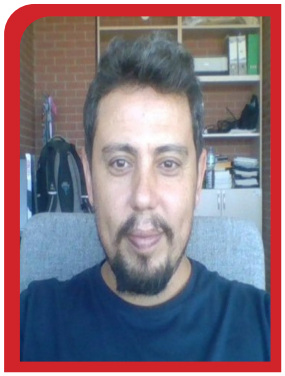
Özgeçmişler

Alper Silistre, 2015 yılında İzmir Ekonomi Üniversitesi'nden yazılım mühendisliği lisans derecesini almıştır. Aynı zamanda Yazılım Mühendisidir. Araştırma ilgi alanları arasında yazılım testi ve yazılım mühendisliği yer almaktadır.

Fevzi Belli (Üye, IEEE) Berlin Teknik Üniversitesi'nden bilgi teknolojisi ve bilgisayar bilimleri alanında B.S., M.S., Ph.D. ve Habilitation (Alman doktora sonrası) derecelerini aldı. Halen Paderborn Üniversitesi ve İzmir Yüksek Teknoloji Enstitüsü'nde Yazılım Mühendisliği Bölümü'nde Fahri Profesör olarak görev yapmaktadır. Araştırma, geliştirme ve yazılım mühendisliği öğretme, doğrulama ve doğrulama, hata toleransı ve kalite güvence konularında 35 yıldan fazla deneyime sahiptir. Uçak endüstrisinde bir programcı olarak başladı ve simülasyon ortamları oluşturmak ve güvenlik açısından kritik özellikleri doğrulamak için programlar yazdı. 1983 yılında Bremerhaven'daki Uygulamalı Bilimler Üniversitesi'nde Profesörlük unvanını aldı; 1989'da Paderborn Üniversitesi'ne geçti. Ayrıca uzun yıllar Maryland Üniversitesi, College Park, MD, ABD ve Avrupa Bölümü'nde Öğretim Üyesi olarak görev yaptı. Aynı zamanda İzmir Ekonomi Üniversitesi Bilgisayar Bilimleri Bölümünün Kurucu Başkanıydı. Yazılım güvenilirliği/hata toleransı, model tabanlı test ve test otomasyonu konularına ilgi ve deneyime sahiptir.

Geylani Kardaş, 2001 yılında Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'nden mezun olduktan sonra bilgi teknolojileri alanındaki yüksek lisans ve doktora derecelerini yine Ege Üniversitesi'nden sırasıyla 2003 ve 2008 yıllarında almıştır. Halen Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü'nde (UBE) doçent doktor olarak görev yapmaktadır ve UBE Yazılım Mühendisliği Araştırma Laboratuvarı'nın (Ege-SERLab) yöneticisidir. Araştırma alanları arasında etmen tabanlı yazılım mühendisliği (AOSE), model güdümlü mühendislik (MDE), alana-özgü diller / alana-özgü modelleme dilleri (DSL'ler / DSML'ler) ve düşük kodlu yazılım geliştirme yer almaktadır. Bu araştırma alanlarında 100'ün üzerinde hakemli makalenin yazarı veya ortak yazarıdır. Dr. Kardaş aynı zamanda Elsevier yayınevinin "Journal of Computer Languages" dergisinin MDE ve DSL bölümünden sorumlu yardımcı editörüdür.

Moharram Challenger (Üye, IEEE) Şubat 2016'da Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü'nden bilgi teknolojisi alanında doktora derecesini aldı. 2005-2009 yılları arasında İAÜ-Shabestar Üniversitesi Bilgisayar Mühendisliği Bölümü'nde Öğretim Üyesi ve Kıdemli Öğretim Görevlisi olarak görev yaptı. 2010-2013 yılları arasında Slovenya ile Türkiye arasında yürütülen ikili bir projede (TÜBİTAK) Araştırmacı ve Takım Lideri olarak görev yaptı. 2012-2016 yılları arasında TÜBİTAK tarafından finanse edilen ulusal bir projeye ve Avrupa'da ITEA ModelWriter ve ITEA Assume adlı iki uluslararası yazılım yoğun projeye liderlik eden UNIT IT Ltd.'de Araştırma ve Geliştirme Direktörü olarak görev yaptı. 2017-2018 yılları arasında Ege Üniversitesi'nde Yardımcı Doçent olarak öğretim üyesi olarak görev yapmıştır. Ocak 2019'dan Temmuz 2020'ye kadar, Flanders Make projelerinde PACo ve DTDesign olarak çalışan Antwerp Üniversitesi'nde Doktora Sonrası Araştırmacı olarak görev yaptı. Halen Antwerp Üniversitesi Bilgisayar Bilimleri Bölümü'nde görev süresi boyunca Yardımcı Doçent olarak görev yapmaktadır. İlgi alanları alana özgü modelleme dilleri, çok etmenli sistemler, siber-fiziksel sistemler ve Nesnelerin İnterneti'dir.



Onur Kılınçeker, M.Sc. (Bilgisayar Müh.), B.Sc. (Matematik) ve Lise. (Elektronik) derecelerine sahiptir. Antwerp Üniversitesi, Bilgisayar Bilimleri Bölümü'nde araştırmacı ve Paderborn Üniversitesi'nde doktora öğrencisidir. İlgi alanları, hem yazılım hem de donanım sistemlerinin model tabanlı doğrulaması ve geçerliliği, model tabanlı testler ve mutasyon testleridir. Şu anda, bütünsel teste (pozitif ve negatif test (diğer adıyla fuzz testi)) ve mutasyon testine dayalı yazılım ve donanım sistemlerinde belirli hataların varlığını ve yokluğunu göstermek için model tabanlı ideal test yöntemleri üzerinde çalışıyor. Aynı zamanda, Simulink modellerinin mutasyon testini endüstriyel ortama entegre etmek için EFFECTS projesinde bilimsel bir araştırmacıdır.