DEU FMD 26(76) (2024) 49-58

Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi
Dokuz Eylul University Faculty of Engineering Journal of Science and Engineering
Elektronik/Online ISSN: 2547-958X

ARAŞTIRMA MAKALESI / RESEARCH ARTICLE

Olay Sıra Çizgeleri için Alana Özgü Bir Dil

A Domain Specific Language for Event Sequence Graphs

Mert Kalecik [1], Tuğkan Tuğlular [1*], Fevzi Belli [1,2]

[1] İzmir Yüksek Teknoloji Enstitüsü, Mühendislik Fakültesi, Bilgisayar Mühendisliği, İzmir, TÜRKİYE
[2] University of Paderborn, Paderborn, GERMANY
*Sorumlu Yazar / Corresponding Author *:* tugkantuglular@iyte.edu.tr

Öz

Yazılımın kalitesi, güvenilir ve hatasız yazılım çözümleri sunabilmek için ulaşılması gereken temel parametrelerden biridir. Sıklıkla model tabanlı testlerle kendini gösteren sistematik test yaklaşımı, yazılım kalitesini artırmak için kullanılan bir yöntemdir. Model tabanlı test yaklaşımına örnek olarak Olay Sıra Çizgeleri (OSÇ) verilebilir. Alana özel dil (AÖD), sınırlı bir alanda geliştirim sürecinde önemli avantajlar sağlayan bildirimsel bir dildir. Bu araştırma karar tablosuyla güçlendirilmiş hiyerarşik OSÇ tasarlama ve görselleştirme için OSÇ-AÖD adı verilen alana özgü bir dil geliştirmeye odaklanmaktadır. Değerlendirme amacıyla bir odak grubu ile çalışılmış ve belli senaryolar için hem Test Suite Designer (TSD) aracı ile hem de OSÇ-AÖD ile OSÇ'ler oluşturulmuştur. Odak grubuna uygulanan anket yoluyla elde edilen karşılaştırma sonuçları, OSÇ-AÖD yaklaşımının TSD yaklaşımına göre kolay kullanım ve tekrar kullanılabilirlik gibi bazı iyileştirmeler içerdiğini göstermektedir.

Anahtar Kelimeler: *Olay Sıra Çizgeleri, Alana Özgü Dil, Model-Tabanlı Sınama*

Abstract

The quality of the software is one of the essential parameters that must be reached in order to provide reliable and error-free software solutions. Systematic testing, which is frequently manifested through model-based testing, is one method for improving the quality of software. Event Sequence Graphs (ESGs) can be given as an example of a model-based testing approach. Domain specific language (DSL) is typically a declarative language that provides significant advantages in the development process in a limited domain. This research is centered on the development of a domain-specific language, which will be referred to as ESG-DSL, for the design and visualization of decision table-augmented hierarchical ESGs. A focus group was worked on for evaluation and ESGs were created with both the Test Suite Designer (TSD) tool and ESG-DSL for certain scenarios. The comparison results obtained through the questionnaire applied to the focus group show that the ESG-DSL approach includes some improvements, such as ease of use and reusability, compared to the TSD approach.

Keywords: *Event Sequence Graphs, Domain Specific Language, Model-Based Testing*

EXTENDED ABSTRACT

Introduction

Model-Based Testing (MBT) is a model-based test design technique that represents the expected behavior of the system under test (SUT). Formal models are derived from model-based testing requirements. An Event Sequence Diagram (ESG) is a model-based approach to representing the expected behavior of SUT. The hierarchical nature of ESGs allows a SUT to be modeled into smaller modules or sub-ASPs. Additionally, ESGs support modeling behavior variations through Decision Tables (DTs).

In this research, a domain-specific language for ESG modeling was designed, developed, and implemented. DT and modularization support for ESG is also provided by the developed DSL called ESG-DSL. The created ESG is visualized using Graphviz. The created ESG is fed to the ESG engine for test generation.

With this research, a DSL and visualization process for an ESG was designed and developed. At the same time, this research tries to answer the following questions:

1. How to design and develop a DSL that supports hierarchy and decision tables for an ESG?

2. Can the developed DSL provide complete visualization for complex ESGs? How can a hierarchical DT-augmented ESG be visualized?

Materials and Methods

In this section, the definitions used within the scope of the study are explained, and the design and use of ESG-DSL are outlined. A DSL is a small, often declarative language that focuses on a specific problem domain and offers specific expressive power to that domain. A DSL provides a more straightforward language than a general-purpose language. It means ease of use. In addition, it offers a robust structure to users by using all the features of a general-purpose language (for example, data structures and file operations). Even though the development process is complicated and requires domain knowledge, DSLs also increase productivity in terms of time and reusability for software components.

DOI:10.21205/deufmd.2024267607

Geliş Tarihi / Received: 20.02.2023
Kabul Tarihi / Accepted: 25.05.2023

Atıf şekli / How to cite:
Kalecik, M., Tuglular, T., Belli, F. (2024). Olay Sıra Çizgeleri için Alana Özgü Bir Dil. DEUFMD, 26 (76), 49-58.

Eclipse XText and JetBrains metaprogramming systems are widely used for APC development. Although they both provide solid grammatical language, Eclipse XText was preferred because it can efficiently process text files.

The primary purpose of the DSL developed in this study is to design ESG using well-defined grammar. In addition, it provides easy modeling and automatic graphical drawing for the mentioned ESG, DT, and sub-ESGs. Additionally, it increases the reusability of ESG, DT, and sub-ESGs.

Candidate entities were identified and validated using domain analysis with ESG-DSL. Following this, conceptual analysis was conducted as the final stage of the domain analysis in order to illustrate entity relationships. Entity identifiers, attribute lists, attribute relationships, and constraints were exposed through this procedure. Furthermore, the inheritance relationships of ESG-DSL grammatical entities are also shown. The outcome of conceptual analysis illustrates the relationship between entities, identifiers, and attributes.

After the DSL domain analysis steps, the DSL application step begins. The framework provides textual parsing, error checking, syntax highlighting, code compiler, general-purpose language support, syntax-to-pattern converter, and generator for target output. In this study, the compiler/application builder model was selected to develop ESG-DSL.

After the DSL grammar was prepared, the AÖD generator class was written to generate the desired output JSON file. The ESG-DSL producer class that extends the basic producer class provided by the Eclipse XText framework is developed. Once the desired JSON file is created, the created file is read by the ESG-Engine project, converted into Java objects, and passed to Graphviz. ESG and its components are drawn in high quality using the DOT language on the Graphviz framework.

Results and Discussion

The case study for ESG-DSL focuses on the login, withdrawal, deposit, invoice printing, and user logout Gherkin scenarios in the bank ATM (Automatic Teller Machine) project. An ATM enables many transactions carried out in bank branches to be carried out reliably, regardless of the branch location. The given Gherkin-based scenarios were implemented with the TSD tool and the ESG-DSL. Among these applications, the comparison is made regarding functional suitability, usability, reliability, maintainability, productivity, compatibility, and meaningfulness.

A total of 10 people participated in this case study, including software managers, analysts, developers, and testers. The participants were divided into two groups. One group used the TSD tool, and the other used the ESG-DSL tool. The ESG was introduced in a 15-minute presentation to both groups. Finally, for the first group, the participants were shown the use of the TSD tool, while for the second group, the use of the ESG-DSL was presented. The evaluation survey consisted of three parts: (i) collecting personal information from the participant, (ii) scoring the ESG-DSL and TSD according to a set of DSL/instrument characteristics, and (iii) open-ended questions.

Regarding functional suitability, ESG and sub-ESG visualization can be performed with ESG-DSL and TSD tools. However, only ESG-DSL can perform DT visualization. Moreover, while the features table of the DT and the DT itself can be automatically drawn with ESG-DSL, the table visualization is not supported by the TSD tool. From a usability perspective, it is not easy to understand how the TSD tool works, and the TSD tool does not have a user-friendly user interface. ESG-DSL, on the other hand, is easy to understand because it works with a natural language-based syntax, and there is no notational difference between it and domain entities.

Regarding reliability, there is an error-checking mechanism, syntax highlighting, and error visualization support for ESG-DSL. The TSD tool, on the other hand, does not have an error prevention method for ESG. Because ESG-DSL communicates with other platforms on a JSON file, it can be integrated with any platform. More development should be performed on TSD with the Java general-purpose language for integration. From an efficiency perspective, both methods increase the productivity of AVF visualization. The figures below give Scenario drawing time in minutes for both methods.

The case study showed that TSD requires less time than ESG-DSL for ESG visualization. However, visualization with the ESG-DSL takes much less time than the TSD tool for the sub-ESGs and DT-augmented ESGs.

Feedback was received from participants for open-ended questions. "Does ESG-DSL make graphic viewing easier?" They mostly answered "yes" to the question. "Is ESG-DSL/TSD useful for graphic visualization?" Those who said "yes" also said that the ESG-DSL made the visualization process more straightforward than the existing TSD tool. "Do you think that ESG-DSL covers all domain models?" All participants answered yes because all scenarios could be implemented with ESG-DSL. The last open-ended question, "Your suggestions for the future of ESG-DSL," was answered mainly by automating the process from design to visualization with ESG-DSL.

Conclusion

ESG-DSL is the name given to a Domain Specific Language (DSL) that has been developed specifically for Event Sequence Graphs (ESGs) as part of this research. In the paper, a concise explanation is provided for ESG visualization, DSL creation, design, implementation, and deployment process. In addition to that, this research offers a method of layered modularization for both the sub-ESGs and the ESGs augmented by decision tables (DTs). Two distinct tables are used to represent the DTs, one for the DT itself and second for the attributes.

The research proposes a DSL that has characteristics that are more similar to those of the natural language. The tool developed in this research produced by using the XText framework has a mechanism for syntax and error highlighting through colors. ESG DSL provides a user-friendly editor as part of this development environment, making it suitable for non-technical people like business colleagues. Because of this, it is possible for a greater number of individuals to participate in event-based modeling even if they lack necessary technical expertise.

ESG-DSL offers a proper environment for graph visualization, which makes it possible to view graphs while also providing support for error checking and syntax highlighting. The time required for ESG DSL development is much longer than that required for TSD graph visualization. When used specifically to a constrained portion of the project, ESG-DSL is able to significantly boost productivity. The case study indicates that following the main ESG and the first sub-ESG visualization, there are considerable gains in sense of time because it permits the reuse of software artifacts. These improvements were made possible by the fact that the software artifacts could be reused. This is one of the reasons why there are significant advantages. The capacity to reuse something improves the result for the individuals using it and indirectly raises the production of the project. In addition, the colorful syntax is simple to memorize, and activities like copying/pasting may boost productivity.

1. Giriş

Model Tabanlı Test (MTT), test edilen sistemin (TES'in) beklenen davranışını temsil eden model tabanlı bir test tasarım tekniğidir. Formel modeller, model tabanlı test gereksinimlerinden türetilmiştir [1]. Bir Olay Sıra Çizgesi (OSÇ), TES'in beklenen davranışını temsil etmeye yönelik model tabanlı bir yaklaşımdır [2]. OSÇ'lerinin hiyerarşik doğası, bir TES'in daha küçük modüller veya alt OSÇ'ler halinde modellenmesini sağlar. Ayrıca, OSÇ'ler, Karar Tabloları (KT'ları) aracılığıyla davranış varyasyonlarının modellenmesini destekler. Bunlara KT ile güçlendirilmiş OSÇ'ler (KT-OSÇ'ler) denir. OSÇ'ler, davranışsal modelleme anlamında sonlu durum makinelerine [3] benzer. Aralarındaki fark, sonlu durum makinelerinin durumlara, OSÇ'lerinin ise olaylara odaklanmasıdır. Sonlu durum makineleri OSÇ'lerine dönüştürülebilir ve bunun tersi de geçerlidir [4]. OSÇ'lerin tercih edilmesinin bir nedeni, modellemeye getirdiği basitlik ve genişletilebilirlik özellikleridir [5]. Başka bir nedeni ise, OSÇ'ler test üretiminde çizge teorisini kullanabilmektedir [5].

Bu araştırmada, OSÇ modellemesi için alana özgü bir dil tasarlanmış, geliştirilmiş ve uygulanmıştır. OSÇ için KT ve modülerleştirme desteği de OSÇ-AÖD adı verilen geliştirilmiş AÖD tarafından sağlanmaktadır. Oluşturulan OSÇ Graphviz [6] kullanılarak görselleştirilir. Oluşturulan OSÇ, test üretimi için OSÇ test motoruna [7] beslenir.

Araştırmanın motivasyonu, eksiksiz OSÇ (alt OSÇ ve KT-OSÇ dahil) modelleme görselleştirmesinin şu anda mevcut olmamasıdır. TSD adı verilen sınırlı bir OSÇ modelleme ve görselleştirme aracı mevcut olmasına rağmen, modelleme maliyeti model büyüdükçe artmaktadır ve tam görselleştirme sağlayamamaktadır. Tam görsel-leştirmeden kastedilen OSÇ'ni oluşturan alt OSÇ ve KT-OSÇ gibi tüm parçaların bir arada görüleceği bir çıktının üretilebilmesidir. Ayrıca, TSD aracının hata denetimi ve hatayı vurgulama desteği yoktur. Buna ek olarak, TSD aracı ile mevcut bir OSÇ'ni güncellemek, yeni bir OSÇ oluşturmaya benzer bir çaba gerektirmektedir. Bu nedenlerle araştırmanın amacı; kullanıcı dostu, yeniden kullanılabilir özelliği olan, anlaşılması ve üzerinde çalışılması kolay, alana özgü bir dil oluşturmaktır.

Yazılım testi, bir uygulamanın, verilen gereksinimleri karşılayıp karşılamadığını ve yazılımın tüm özelliklerinin doğru bir şekilde uygulandığını ortaya koymak için gerçekleş-tirilen doğrulama işlemidir [8].

Uzun zamandır yazılım eserlerinin yeniden kullanılabilirliğini artırılmaya çalışılmaktadır. İlk girişimler mikro diller ve etki alanına özgü diller için küçük diller olarak adlandırıldı [9]. Günümüzde bir AÖD, genel amaçlı bir dilin parçaları üzerine alana özgü yetenekler ekleyerek onu geliştirir [10].

OSÇ-AÖD geliştirmesinde Eclipse Xtext [11] çerçevesi kullanılmıştır. Geliştirilen AÖD ile bir OSÇ tanımlanabilmektedir. Xtext kullanan bir yazılıma örnek olarak Yaktor [12] verilebilir. Yaktor olay güdümlü, eş-zamansız, dağıtılmış, ölçeklenebilir çok taraflı durum makinesi aracıdır ve Yaktor AÖD'ni [13] kullanır. Franca AÖD [14] ve Expression Language AÖD [15] Xtext çerçevesi kullanılarak geliştirilen diğer AÖD örnekleridir.

AÖD ile oluşturulan OSÇ'lerin görselleştirmesi gereklidir. Bu görselleştirme için JGraph çerçevesi [16], Java Universal Network Graph (JUNG) çerçevesi [17] ve PlantUML çerçevesi [18] gibi çerçeveler yerine bunların temelini oluşturan Graphviz [6] çerçevesini tercih edilmiştir. Yazılım mühendisliği grafikleri çiziminde yoğun olarak kullanılan Graphviz [19] grafikleri ve çizimlerini manipüle eden bir araç sunmaktadır [20]. Graphviz yönlendirilmiş grafikleri alan etkin çizmek konusunda bir algoritmaya sahiptir [21].

Bu araştırma ile, bir OSÇ için AÖD ve görselleştirme süreci tasarlanmış ve geliştirilmiştir. Aynı zamanda bu araştırma aşağıdaki soruları cevaplamaya çalışmaktadır:

1. Bir OSÇ için hiyerarşi ve karar tablosu destekleyen bir AÖD nasıl tasarlanır ve geliştirilir?

2. Geliştirilen AÖD, karmaşık OSÇ'ler için tam görselleştirmeyi sağlayabilir mi? Bir hiyerarşik KT-OSÇ her parçası ile nasıl görselleştirilir?

Makale şu sırayla düzenlenmiştir. İkinci bölüm AÖD ve OSÇ gibi temel kavramları tanıtıp OSÇ'ler için bir AÖD ve grafik görselleştirmenin ayrıntılı tasarım ve geliştirme adımlarını içermektedir. Üçüncü bölümde yaklaşımı sınamak için kullanılan vaka çalışması ve yapılan anket temelli incelemeye ait sonuçlar aktarılmıştır. Bölüm 4'te çalışmanın analizine yer verilmiş ve gelecek çalışmalar ifade edilmiştir.

2. Materyal ve Metot

Bu bölümde çalışma kapsamında kullanılan tanımlar açıklanmış, OSÇ-AÖD tasarımı ve kullanımı anlatılmıştır.

2.1. Alana Özgü Diller

Bir AÖD belirli bir sorun alanına odaklanarak bu alana özel ifade gücü sunan küçük, genellikle bildirimsel bir dildir. Bir AÖD genel amaçlı bir dilden daha basit bir dil sunar. Bu da kullanım kolaylığı anlamına gelir. Bununla birlikte genel amaçlı bir dile ait tüm özelliklerden istediklerini (örneğin, veri yapıları ve dosya işlemleri) kullanarak güçlü bir yapıyı kullanıcıların hizmetine sunar. Geliştirme süreci zor olsa ve alan bilgisi gerektirse bile, AÖD'ler zaman anlamında üretkenliği ve yazılım bileşenleri için yeniden kullanılabilirliği de artırır [22].

AÖD geliştirme için Eclipse XText [11] ve JetBrains meta programlama sistemi [23] yaygın olarak kullanılmaktadır. Her ne kadar ikisi de güçlü bir gramer dili sağlasa da metin dosyalarını daha kolay işleyebilmesi sebebiyle Eclipse XText tercih edilmiştir. Ayrıca Eclipse XText, yapı bileşenleri için Eclipse Modeling Framework (EMF) ve kod üretimi sağlar [24]. Kod üretimi, sözdiziminden modelleri ayrıştırmak ve modelleri oluşturulan hedef çıktıya dönüştürmek anlamına gelir.

XText içinde bir dilbilgisi model, alan, yorum ve tip üzerine bir dizi kuraldan oluşur. Bir kural belirteç dizileri kullanılarak tanımlanır. Belirteç ya başka bir kurala referanstır ya da ilkel belirteçlerdir (INT, STRING, ID) [25]. Bir kural, durum veya nitelenmiş ad olduğunda bir açıklama ve ardından durum veya nitelenmiş ad alanları içerir. Nitelikli ad bir kimlik alır ve ayrıca virgülle birleştirilebilir [25].

XText dilbilgisi hazır olduğunda AÖD ile üretim için Generate XText Artifacts iş akışı kullanılır. Artık AÖD bir Eclipse uygulaması olarak kullanıma hazırdır. Eclipse XText editörü bildirimlere ilişkin dilbilgisi sözdizimini denetler ve yazılı dilbilgisinde eksik bir kısım varsa kırmızı renkle uyarır. Tüm bildirimler tamamlandığında, editör ekranının kaydedilmesi dosya oluşturucuyu tetikler ve AÖD dilbilgisi kurallarına dayanarak bir dosya ortaya konur.

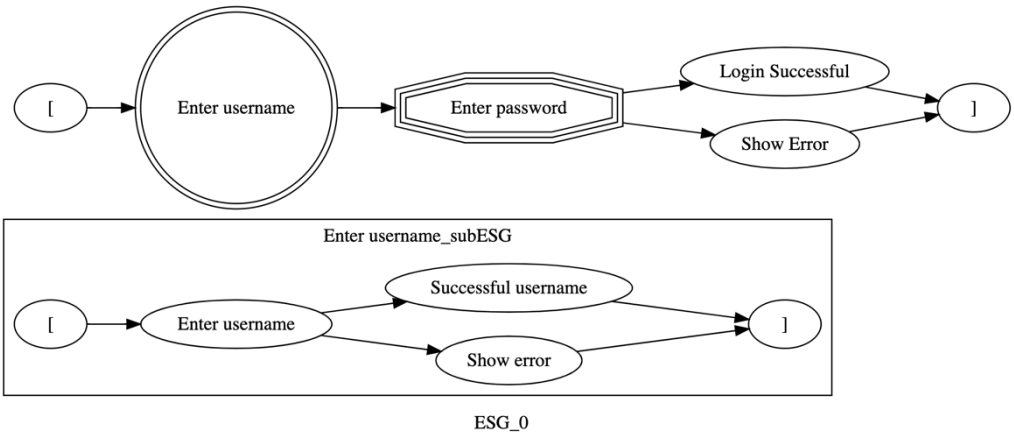
AÖD kullanımlarının yaygın olarak bilinen örnekleri, SQL (ilişkisel veritabanlarıyla kullanılan Yapısal Sorgulama Dili), MATLAB (özellikle bilim adamları ve mühendisler için tasarlanmış programlama dili) ve HTML'dir (web sayfaları

oluşturmak için kullanılan Hiper Metin İşaretleme Dili). Bu örneklerden de görülebileceği gibi, tasarlanan AÖD'in kullanıcı arayüzüne yazılan bildirimler genel amaçlı bir dilde derlenir.

2.2. Olay Sıra Çizgeleri

Olay sıra çizgesi test edilen bir sistemin davranışlarını temsil eden bir çizgedir [26]. OSÇ, etkileşimli bir sistemi kullanıcı eylemleri ve onlara sistemin verdiği tepkileri modeller. Bir olay sıra çizgesi (V, E) bir yönlü çizgedir; öyle ki $V \neq \emptyset$ sonlu bir olaylar (düğümler ya da köşeler) kümesi ve $E \subseteq V \times V$ sonlu bir kenarlar (yaylar) kümesidir. Olay sıra çizgelerinde [ve] sözde olayları sırasıyla başlangıç ve bitiş olaylarını işaretlemek için kullanılır [2].

Şekil 1'de örnek olarak verilen OSÇ modeli, oturum açma sürecini temsil etmektedir. Bu modeldeki "Enter Username" olayı bir başlangıç olayı olup, "Show Login Successfull" ve "Show Error" olayları ise bir bitiş olaylarıdır.



Şekil 1. Oturum açma OSÇ modeli

Figure 1. ESG model of login

OSÇ'lerinde bakım yapılabilirlik, okunabilirlik, yeniden düzenleme ve yeni özellik ekleme süreçleri, büyük ölçekli projelerle uğraşırken çizge çiziminin düzenlenmesi açısından karmaşıklaşmaktadır. OSÇ için alt-OSÇ kullanarak modüler hale getirme bu noktada devreye girmektedir. Bir OSÇ, herhangi bir katmanın altında alt-OSÇ'ler içerebilir. Nihai OSÇ, her bir alt-OSÇ'nin rafine edilmesi ile tek katmanda ifade edilebilir.

Karar tabloları, belirli bir dizi koşul altında eylemleri tanımlamak için grafiksel bir gösterimdir. Bir KT koşul kombinasyonlarını belli eylemler ile ilişkilendirir. Karar tabloları, neden-sonuç ilişkilerini tanımlamak için kullanışlıdır.

KT'nda maksimum koşul kombinasyonu sayısı $2|C|$ ile temsil edilir, burada $|C|$ koşul sayısını temsil etmektedir. Bu koşulların kombinasyonu, OSÇ'nin karmaşıklaşmasına neden olur. Bu karmaşıklığı anlaşılır kılmak için o OSÇ ilgili KT içerecek şekilde yapılandırılır.

Tablo 1 ve Tablo 2'de, Oturum açma OSÇ içinde yer alan "Enter Password" için kullanılan KT verilmiştir. Hangi koşullar altında oturum açmaya izin verileceği, hangi koşullar altında hata mesajı verileceği ve hangi koşullar altında tekrar kullanıcı adı girişine yönlendirileceği bu KT'nda ifade edilmiştir. Burada koşullar C0, C1, C2, C3 ile, aksiyonlar A0, A1, A2 ile ve kurallar R0, R1 ile tanımlanmıştır.

2.3. Olay Sıra Çizgeleri Alana Özgü Dil için Analiz

Bu çalışmada geliştirilen AÖD'nin temel amacı, iyi tanımlanmış gramer kullanarak OSÇ tasarlamaktır. Buna ek olarak, söz konusu AÖD, KT ve alt-OSÇ'ler için kolay modelleme ve otomatik grafik çizim sağlar. Ayrıca, OSÇ, KT ve alt-OSÇ'lerin yeniden kullanılabilirliğini artırır.

Tablo 1. Oturum açma OSÇ modeli karar tablosu

Table 1. Decision table of login ESG model

	R0	R1
C0	T	T
C1	F	T
C2	F	T
C3	T	T
A0	X	-
A1	-	X
A2	X	-

Tablo 2. Oturum açma OSÇ modeli karar tablosu özellikleri

Table 2. Decision table properties of login ESG model

Condition/Action	Table Properties
C0	Password is type of string
C1	Password length is greater than or equal to 10 AND Password length is less than or equal to 100
C2	Password has upper case characters AND Password has lower case characters
C3	Password has special character AND Password contains number
A0	Enter Username
A1	Login Successful
A2	Show Error

Öte yandan, bir AÖD geliştirmenin ve kullanmanım bazı dezavantajları vardır. Başlangıçta, bir AÖD geliştirmek için geliştiriciler gereklidir ve bunun da bir maliyet karşılığı bulunur. Geliştirme süreci bittiğinde, AÖD kullanıcıları için eğitim süresi gereklidir. Özellik seti genişlediğinde ise, AÖD için bakım ve geliştirme maliyetine katlanmak gerekecektir.

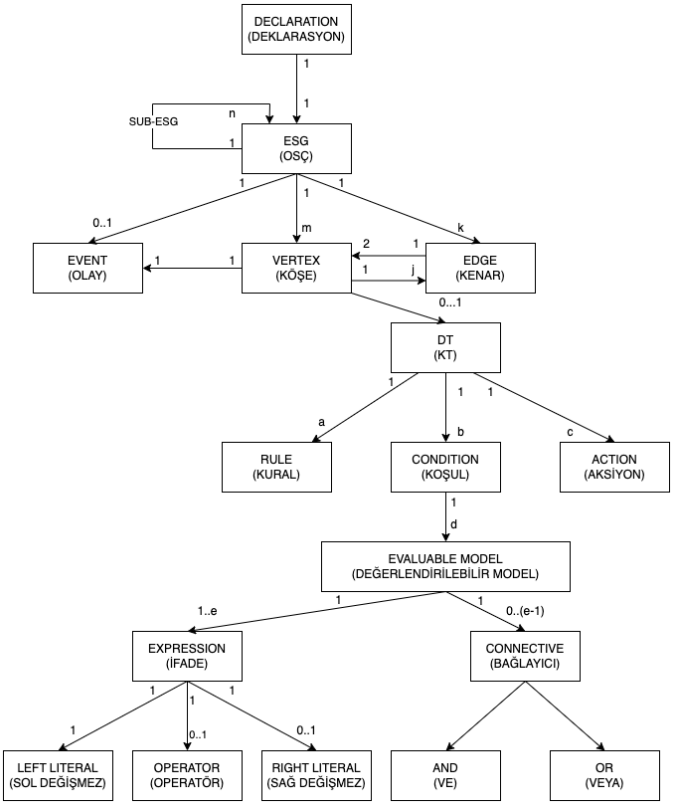
TSD, sürükle ve bırak yaklaşımı ile OSÇ tasarlamak ve ilgili OSÇ'nden test sıraları oluşturmak için kullanılır. Köşeler çift tıklama, kenarlar ise sürükle ve bırak ile oluşturulur. Araçta alt-OSÇ desteği vardır ancak bunları bir arada görselleştirmek mümkün değildir. Kullanıcı deneyimi açısından, TSD aracıyla karmaşık süreç modellemesi çok zordur.

TSD içinde köşeler ve kenarlar için kopyalama ve yapıştırma tutarlı olarak beklendiği gibi çalışmayabilmekte ve kopyalama işlemi sırasında kopyada bazı eksik kısımlar buluna-bilmektedir.

Bir AÖD geliştirme kararı, kullanıcının üretkenliğini ve yazılım eserlerinin yeniden kullanılabilirliğini artırmak için verilir [22]. Ayrıca bir AÖD, sınırlı bir alanda daha az konu ve teknoloji bilgisi gerektirdiğinden bazı avantajlar sağlayan bir soyutlama katmanı oluşturur [27]. OSÇ-AÖD geliştirim kararı bu yaklaşımlarla birlikte kullanıcılar için kolay bir gramer ile daha iyi bir kullanıcı deneyimi amacıyla verilmiştir.

OSÇ-AÖD etki alanı analizi ile aday varlıklar belirlenmiş ve doğrulanmıştır. Daha sonra, alan analizinin son adımı olarak, varlık ilişkilerini göstermek için kavramsal analiz [28] gerçekleştirilmiştir. Bu yaklaşım ile varlık tanımlayıcıları, öznitelik listeleri, öznitelik ilişkileri ve kısıtlamaları ortaya konmuştur. Ayrıca, OSÇ-AÖD gramer varlıklarının kalıtım ilişkilerini de gösterilmiştir. Kavramsal analizin sonucu; varlıklar, tanımlayıcılar, öznitelikler ve bunların ilişkisini göstermektedir.

Kavramsal analizden sonra varlık ilişkileri diyagramı oluşturulmuştur. Şekil 2'de OSÇ-AÖD varlık ilişkisi verilmektedir. Kökte, altında öğeler içeren bir OSÇ vardır. Öğeler ya bir OSÇ ya da bir köşe olmalıdır. Her OSÇ kenarları ve olayları içerir. Bir OSÇ, alt-OSÇ de olabilir. Bu mantıkla tekrarlama ilişkisi devreye girmektedir. Bir köşe üzerinde bir karar tablosu saklama olanağına sahiptir. Bir KT, bazı koşullar altında kullanıcı girdileriyle uğraşırken aksiyonların çıkarılmasına da yardımcı olur. Bir koşul, bağlaçlarla bağlantılı bir ifade içerir.



Şekil 2. OSÇ-AÖD varlık ilişki diyagramı

Figure 2. ESG-DSL entity relationship diagram

2.4. Olay Sıra Çizgeleri için Alana Özgü Dil Gerçekleştirimi

AÖD alan analizi adımları tamamlandıktan sonra AÖD uygulama adımı başlar. Söz konusu çerçeve, metinsel ayrıştırma, hata denetimi, sözdizimi vurgulama, kod derleyici, genel amaçlı dil desteği, sözdiziminden model dönüştürücü ve hedef çıktı için oluşturucu sağlar. Bu çalışmada derleyici/uygulama oluşturucu modeli [22], OSÇ-AÖD geliştirmesi için seçilmiştir.

OSÇ-AÖD'den beklenti, önce alan nesnelerinin Java nesnelerine model çevirilerinin yapılması, ardından hedef dosya olan JSON dosyasının oluşturulmasıdır.

OSÇ-AÖD için uygulama adımı, Eclipse IDE kullanılarak Eclipse XText çerçevesi ile geliştirilmiştir. OSÇ-AÖD'nin dilbilgisi yapısı, Bölüm 2.3'te gösterildiği şekilde oluşturulur. İlk önce, OSÇ'ler ve alt-OSÇ'ler, sonra KT ile artırılmış OSÇ'ler oluşturulur.

İlk olarak, Şekil 3'te görülen olay modeli OSÇ-AÖD dilbilgisi modeli oluşturulmuştur. Olay modeli, ad alanını bir katar olarak alır. Olay; OSÇ, alt-OSÇ ve köşe ile birlikte kullanılır. Renk bilgisi Graphviz ile grafikler çizilirken kullanılır.

Kenar dilbilgisi, kaynak köşeden hedef köşeye yönlü bir kenarı temsil eder. Kaynak köşe tanımlayıcısı ve hedef köşe tanımlayıcısı birer tamsayı olarak ifade edilir. Kenarların da renk bilgisi vardır, bu isteğe bağlı bir bilgidir ve varsayılan olarak siyah gelir. Soru işareti, isteğe bağlı bir bilgi olduğunu belirtir. Köşe

dilbilgisinde bir köşe için tanımlayıcı, olay bilgisi, varsayılan olarak siyah olan isteğe bağlı bir renk bilgisi ve isteğe bağlı bir karar tablosu bilgisi vardır.

```
terminal IDENTIFIER:
  ('0'..'9')*;
ESG: 'ESG' ID = IDENTIFIER
    ('Event' event = EVENT)?
    subESGs += (VERTEX | ESG) (',' subESGs +=
                (VERTEX | ESG))*
    edges += EDGE(',' edges += EDGE)*;
VERTEX: 'Vertex' ID = IDENTIFIER
    'Event' event = EVENT
    ('Color' color= COLOR)?
    (dt = DT)?;
EDGE: 'Edge'
    'Source' source= IDENTIFIER
    'Target' target= IDENTIFIER
    ('Color' color= COLOR)?;
EVENT:
    ID = STRING;
COLOR:
    ID = ("black"|"red"|"green"|"blue"|"orange");
```

Şekil 3. OSÇ-AÖD dilbilgisi modeli

Figure 3. ESG-DSL grammar

OSÇ dilbilgisi kök OSÇ'yi temsil eder. Kök OSÇ olay alanını almadığı için olay bilgisi isteğe bağlıdır. OSÇ dilbilgisi aynı zamanda alt-OSÇ'leri de temsil eder. Ancak her bir alt-OSÇ, kök OSÇ'de rafine edilmek üzere olay bilgisini alır. Her bir alt-OSÇ, bir OSÇ veya bir köşe olabilir. Alt-OSÇ'ler virgülle ayrılır. Yıldız işareti bulunduğu bilgilerin zorunlu olduğunu ifade eder. Her OSÇ, virgülle ayrılmış kenarlar içerir ve kenarlar bilgisi zorunlu bir bilgidir.

İkinci olarak, OSÇ_AÖD dilbilgisi, karar tabloları ile köşe büyütmeyi desteklemek için Şekil 4'te gösterildiği şekilde genişletilmiştir.

Koşul dilbilgisi karar tablosu koşullarını temsil eder. Her koşul, kendisi için bir tanımlayıcı olarak ad bilgisi ve bir dizi değerlendirilebilir model alır. Değerlendirilebilir model, bir ifade veya iki ifade arasında bir bağlantı işlevi gören bir bağlayıcı olmalıdır. İfade dilbilgisi; sol hazır bilgi, bir operatör (<, >, <=, >=) ve sağ hazır bilgi içerir. Bir değişmez aynı zamanda bir dilbilgisi modelidir ve girişi bir dizi veya tamsayı olarak kabul eder.

Kural dilbilgisi modeli, karar tablosunun kurallarını temsil etmek için oluşturulur. Her kuralın katar olarak bir isim bilgisi, tamsayı olarak tanımlayıcı bilgisi ve katar içeren bir değer bilgisi vardır. Değer bilgisinin her karakteri sıralı koşul sonuçlarına karşılık gelir. İk koşul (C0, C1) varsa ve "TF" değeri almış ise C0 = T ve C1 = F anlamına gelir. Kuralların giriş değişkenlerini tutmak için değişken dilbilgisi modeli oluşturulur, kural dilbilgisi modeli virgülle ayrılmış değişkenler kümesine sahiptir ve değişkenler bilgisi zorunludur.

Aksiyon dilbilgisi, kuralın belirli koşullar altında seçilen aksiyonları tutar. Kural dilbilgisinde aksiyon bilgisi zorunludur, bir ya da daha fazla aksiyon olabilir.

```
DT: "DT" ID = IDENTIFIER
    conditions += CONDITION (',' conditions += CONDITION)*
    actions += ACTION (',' actions += ACTION)*
    rules += RULE (',' rules += RULE)*;
CONDITION: "Condition" ID = IDENTIFIER
    evals += EVALUABLE(evals += EVALUABLE)*;
EVALUABLE:
    EXPRESSION | CONNECTIVE;
EXPRESSION:
  "("
    left = LITERAL
    (operand = OPERAND)?
    (right= LITERAL)?
  ")";
CONNECTIVE:
    connective = ("AND" | "OR");
RULE: "Rule" ID = STRING
    "Value" value = STRING
    "Variables" vars += VARIABLE("," vars += VARIABLE)*
    "Actions" actions += [ACTION]("," actions += [ACTION])*;
ACTION: "Action" name = ID
    "ID" ID = IDENTIFIER
    "Event" event = IDENTIFIER;
VARIABLE: 'var' name = STRING
    'value' value = LITERAL;
```

Şekil 4. OSÇ-AÖD KT dilbilgisi modeli

Figure 4. ESG-DSL DT grammar

KT dilbilgisi modeli, artırılmış köşeyi temsil etmek için oluşturulur. KT için katar olarak bir ad bilgisi, tamsayı olarak bir tanımlayıcı bilgisi, virgülle ayrılmış bir dizi koşul, virgülle ayrılmış bir dizi aksiyon ve virgülle ayrılmış bir dizi kural vardır. Koşul, aksiyon ve kural bilgileri zorunludur.

AÖD dilbilgisi hazırlandıktan sonra, istenen çıktı JSON dosyasını oluşturmak için AÖD oluşturucu sınıfı yazılmıştır. Şekil 5'te Eclipse XText çerçevesi tarafından sağlanan temel üretici sınıfı genişleten OSÇ-AÖD üretici sınıfı verilmiştir.

Şekil 6'da çıktı dosyasının yapısı verilmiştir. Tamsayı olarak kimliği, dize olarak adı, kenar dizisini ve hem köşeleri hem de alt OSÇ'leri içeren köşeler dizisini içeren bir JSON dosyası oluşturmak için. OSÇ-AÖD üretici sınıfı, Şekil 6'da gösterilen yapıya uygun bir JSON dosyası üretir.

```
class MkDslGenerator extends AbstractGenerator {
override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
        val simpleClassName = resource.getURI.trimFileExtension.lastSegment
        if(resource.contents?.head == null) { return; }
        val declarations = resource.contents.head.eContents.filter(DECLARATION)
        var count = 0 as int
        fsa.generateFile(simpleClassName + '.json', '''
          ...
}
```

Şekil 5. OSÇ-AÖD için XText üretici

Figure 5. XText generator for ESG-DSL

Üreteç, XText çerçevesi tarafından sağlanan AbstractGenerator'dan "doGenerate" yöntemini devralır. OSÇ-AÖD üretici, model setindeki tüm bildirimleri incelere ve kök OSÇ'yi bulur. Bir kök OSÇ için, üreteç sınıfı, Şekil 6'daki yapıya uygun JSON nesneleri üretir. Bir köşe modelinin KT özniteliği varsa, o zaman köşesi olan KT JSON nesnelerini oluşturur. OSÇ-AÖD üreteci tam sürümü [29] de görülebilir.

```
{
    "ID": 0,
    "name": "refinedESG",
    "vertices": [
                    ...
                    ]
    "edges": [
                    ...
                    ]
}
```

Şekil 6. OSÇ-AÖD çıktı dosyası yapısı

Figure 6. ESG-DSL output file structure

2.5. Olay Sıra Çizgeleri Görselleştirimi

İstenen JSON dosyası oluşturulduktan sonra, oluşturulan dosya ESG-Engine projesi [29] tarafından okunur ve Java nesnelerine dönüştürülür ve Graphviz'e iletilir. Graphviz çerçevesi üzerinde DOT dili kullanılarak alt OSÇ (alt-OSÇ) oluşturulması için alt grafik kümeleri ile bir soyutlama katmanı desteği vardır. Alt-OSÇ de başka bir OSÇ'dir. Alt-OSÇ köşesi Şekil 1'deki çift daire şeklinde görselleştirilir. Şekil 1'te verilen rafine edilmiş OSÇ'nin görselleştirilmesi için, alt-OSÇ'yi temsil etmek üzere alt grafik öğesi kullanılır.

Karar tablosu görselleştirmesinde ilk olarak, ana OSÇ'de karar tablosu köşesi üçlü sekizgen olarak çizilir. Sonra, karar tablosu özellikleri tablosu görselleştirilir. En son olarak, karar tablosu çizilir. Bir örneği Şekil 2'de görülebilir. Şekil 2'de verilen çizimler için, özellikler tablosu ve karar tablosunun kendisi için alt grafik kümeleri oluşturulmuştur.

3. Bulgular ve Tartışma

OSÇ-AÖD için vaka çalışması incelemesi, banka ATM (Otomatik Vezne Makinesi) projesindeki oturum açma, para çekme, para yatırma, fatura yazdırma ve kullanıcı oturumunu kapatma senaryolarına odaklanacaktır. Bir ATM banka şubelerinde gerçekleştirilen birçok işlemin güvenilir bir şekilde şube konumundan bağımsız olarak gerçekleştirilmesini sağlar. ATM işlevlerini ifade eden Gherkin [30] tabanlı bu senaryolar aşağıda verilmiştir:

- Senaryo 1 – Başarılı Oturum Açma
- Senaryo 2 – Başarısız Oturum Açma
- Senaryo 3 – Başarılı Para Çekme
- Senaryo 4 – Başarısız Para Çekme
- Senaryo 5 – Başarılı Para Yatırma
- Senaryo 6 – Başarısız Para Yatırma
- Senaryo 7 – Başarılı Çıktı Yazdırma
- Senaryo 8 – Başarısız Çıktı Yazdırma
- Senaryo 9 – Oturumu Kapatma

Verilen Gherkin tabanlı senaryolar hem TSD aracı ile hem de OSÇ-AÖD ile uygulanmıştır. Bu uygulamalar arasında; işlevsel uygunluk, kullanılabilirlik, güvenilirlik, sürdürülebilirlik, üretkenlik, uyumluluk ve anlamlılık açısından karşılaştırma yapılmıştır. Hem TSD hem de OSÇ-AÖD ile görselleştirilmiş OSÇ'leri [31] de bulunabilir. Karşılaştırma yapmak amacıyla Şekil 7'de OSÇ-AÖD ile görselleştirilmiş oturum açma OSÇ modelinin TSD aracı ile görselleştirilmiş hali Şekil 8'de verilmiştir.

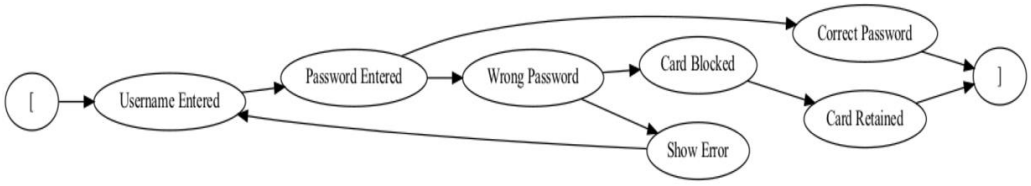
Bu vaka çalışmasında yazılım yöneticileri, analistleri, geliştiriciler ve test uzmanları olmak üzere toplam 10 kişi yer almıştır. Katılımcılar iki gruba ayrılmış ve bir grup TSD diğeri ise OSÇ-AÖD aracını kullanmıştır. Her grup için 15 dakika süren sunum yapılmış ve OSÇ tanıtımı gerçekleştirilmiştir. Son olarak, birinci grup için katılımcılara TSD aracı kullanımı gösterilirken, ikinci grup için OSÇ-AÖD kullanımı sunulmuştur.

Değerlendirme anketi üç bölümden oluşmuştur:

1) katılımcıdan kişisel bilgi toplama

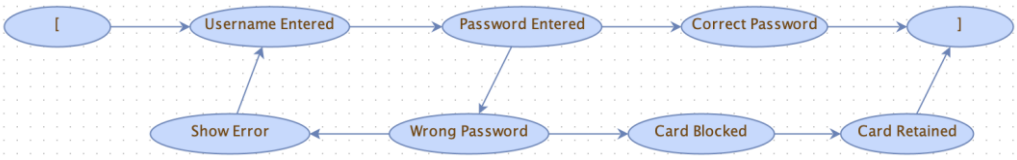
2) OSÇ-AÖD ve TSD'yi bir dizi AÖD/araç özelliğine göre puanlama

3) açık uçlu sorular



Şekil 7. OSÇ-AÖD ile görselleştirilen oturum açma OSÇ modeli

Figure 7. ESG model of login visualized by ESG-DSL



Şekil 8. TSD aracı ile görselleştirilen oturum açma OSÇ modeli

Figure 8. ESG model of login visualized by TSD tool

Son olarak katılımcılardan OSÇ-AÖD'nin işlevselliği ve gelecekteki gelişimi için öneriler hakkında geri bildirim almak için açık uçlu sorular sorulmuştur.

Etki Alanına Özgü Dillerin Niteliksel Değerlendirme Çerçevesi (FQAD) [32], OSÇ-AÖD'nin benimsenmesi için özelleştirilmiştir. Anketin her sorusuna 1 "Çok Kötü" ve 5 "Çok İyi" aralığında yanıtlar istenmiştir.

Anketin ilk bölümü, katılımcı hakkında ad, soyad, mezun olunan bölüm, akademik derece, iş deneyimi, çalıştıkları şirketteki görev gibi bilgileri toplamaya yönelik beş sorudan oluşmaktadır. Katılımcılar, OSÇ_AÖD ve TSD olmak üzere iki platform arasında eşit olarak dağıtılmıştır. Mezun olunan bölümler bazında her iki grupta da aynı dağılım yüzdesine sahiptir. Tüm katılımcılar lisans derecesine sahiptir.

Anketin ikinci bölümü hem TSD aracı hem de OSÇ-AÖD için; işlevsellik, kullanılabilirlik, güvenilirlik, sürdürülebilirlik, üretkenlik, uyumluluk ve ifade gücü olmak üzere yedi alt bölüm ve 18 sorudan oluşmaktadır. Şekil 9'da, her bir kalite özelliği için toplanan puanların ortalaması verilmektedir.

Anketin son bölümünde ise katılımcılara gelecek gelişim planı ve geri bildirimlerini almak için aşağıdaki açık uçlu sorular sorulmuştur:

1) OSÇ-AÖD/TSD grafik görselleştirmeyi kolaylaştırır mı?

2) OSÇ-AÖD/TSD grafik görselleştirme işlemi için faydalı mı?

3) OSÇ-AÖD/TSD'nin tüm etki alanı modellerini kapsadığını düşünüyor musunuz?

4) OSÇ-AÖD/TSD'yi geliştirmek için önerilerinizi ve diğer yorumlarınızı yazınız.

İşlevsel uygunluk açısından hem OSÇ-AÖD hem de TSD aracı ile OSÇ ve alt-OSÇ görselleştirmesi yapılabilmektedir. Ancak sadece OSÇ-AÖD, KT görselleştirmesi yapabilmektedir. Ayrıca, KT'nin özellikler tablosu ve KT'nin kendisi OSÇ-AÖD ile görselleştirilebilirken özellikler tablosu görselleştirmesi TSD aracıyla desteklenmemektedir.

Kullanılabilirlik açısından bakıldığında TSD aracının nasıl çalıştığını anlamak zordur ve TSD aracı kullanıcı dostu bir kullanıcı arayüzüne sahip değildir. OSÇ-AÖD ise doğal dil tabanlı bir sözdizimi ile çalıştığı ve etki alanı varlıklarıyla arasında gösterimsel bir fark olmadığı için anlaşılması kolaydır.

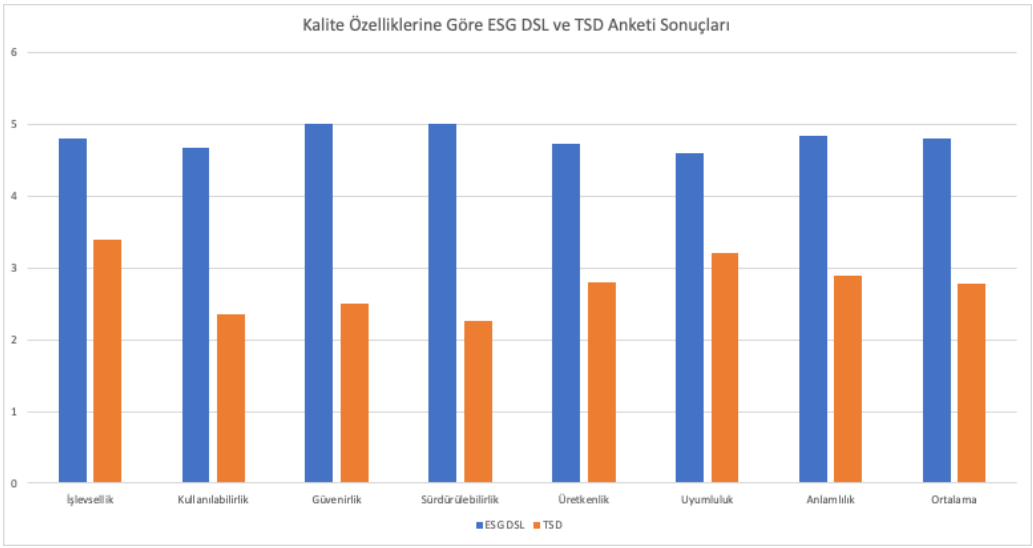
Güvenilirlik açısından, OSÇ-AÖD için hata kontrol mekanizması, sözdizimi vurgulama ve hata görselleştirme desteği vardır. TSD aracı ise OSÇ için bir hata önleme yöntemine sahip değildir.

OSÇ-AÖD diğer platformlarla bir JSON dosyası üzerinde iletişim kurduğu için herhangi bir platformla birleştirilebilir. Bakım açısından TSD'yi sürdürmek daha zordur, Java genel amaçlı dili ile daha fazla geliştirme yapılmalıdır.

Verimlilik açısından bakıldığında her iki yöntem OSÇ görselleştirmenin üretkenliğini artırır. Senaryo çizme süresi her iki yöntem için dakika cinsinden aşağıdaki şekillerde verilmiştir.

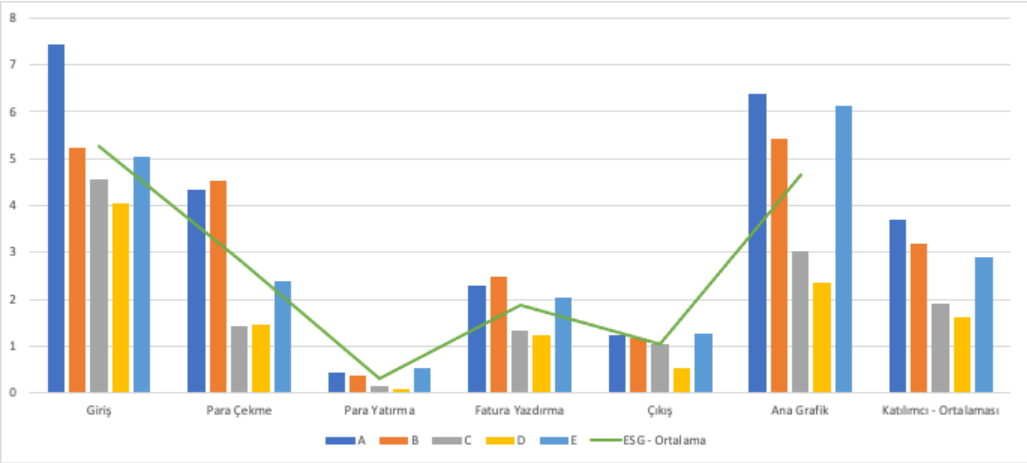
Şekil 10 ve Şekil 11, ana, oturum açma, para çekme, para yatırma, faturayı yazdırma ve oturumu kapatma OSÇ görselleştirme sırasına sahip olarak, TSD görselleştirme işleminin OSÇ-AÖD'den daha az zaman gerektirdiğini göstermektedir. Katılımcıların "A" dan "E" ye kadar olan harflerle gösterilmiştir. Ayrıca, OSÇ-AÖD ile görselleştirme, alt-OSÇ ve KT-OSÇ için TSD aracına göre çok daha az zaman harcar.

Açık uçlu sorular için katılımcılardan geri bildirim alınmıştır. "OSÇ-AÖD grafik görüntülemeyi kolaylaştırır mı?" sorusuna çoğunlukla "evet" yanıtı verdiler. "OSÇ-AÖD/TSD grafik görselleştirme işlemi için faydalı mı?" için "evet" diyenler, OSÇ-AÖD'nin görselleştirme sürecini mevcut TSD aracına göre daha kolay hale getirdiğini söyledi. "OSÇ-AÖD'nin tüm alan modellerini kapsadığını düşünüyor musunuz?" sorusuna katılımcıların tamamı evet yanıtı vermiştir, çünkü tüm senaryolar OSÇ-AÖD ile uygulanabilmiştir. Açık uçlu son soru olan "OSÇ-AÖD'in geleceği için önerileriniz" sorusuna ağırlıklı olarak OSÇ_AÖD ile tasarımdan görselleştirmeye giden sürecin otomatik hale getirilmesi yanıtı gelmiştir.



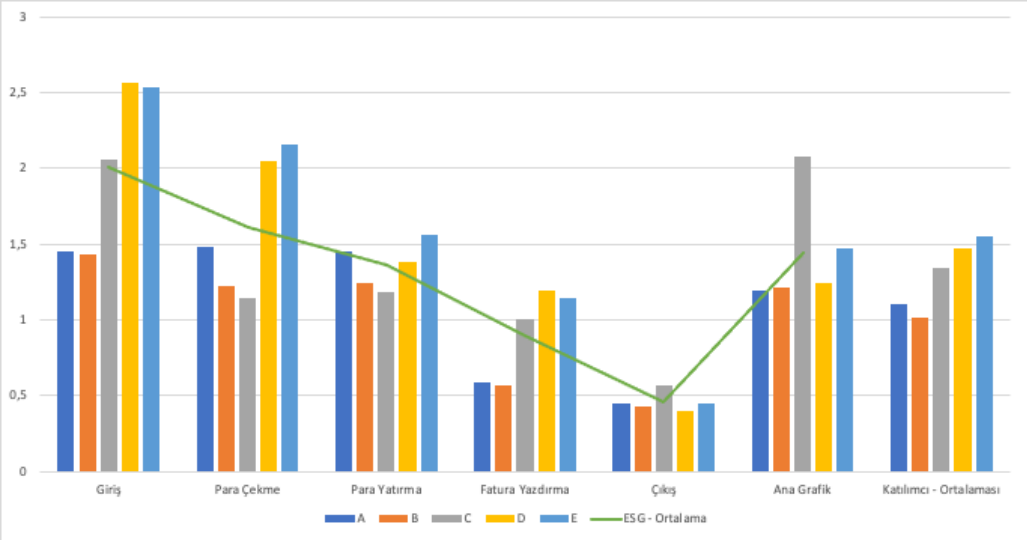
Şekil 9. OSÇ-AÖD ile TSD anket sonuçları

Figure 9. ESG-DSL vs TSD Questionnaire Results



Şekil 10. OSÇ-AÖD ile senaryo çizme süresi

Figure 10. Scenario drawing time with ESG-DSL



Şekil 11. TSD ile senaryo çizme süresi

Figure 11. Scenario drawing time with TSD

4. Sonuçlar

OSÇ-AÖD, bu araştırma kapsamında OSÇ'ler için özel olarak geliştirilmiş bir AÖD'e verilen addır. Makalede, OSÇ görselleştirme, AÖD oluşturma, tasarım, uygulama ve dağıtım süreci açıklanmıştır. Buna ek olarak, bu araştırma hem alt OSÇ'ler hem de KT'ler ile zenginleştirilmiş OSÇ'ler için katmanlı bir modülerleştirme yöntemi sunmaktadır. KT'leri temsil etmek için iki farklı tablo kullanılır: ilk tablo KT'nin kendisini gösterirken, ikinci tablo KT özelliklerini ve bu özelliklerin tanımlarını gösterir.

Bu araştırma, doğal dilin özelliklerine daha çok benzeyen özelliklere sahip bir AÖD önermektedir. XText çerçevesi kullanılarak üretilen bu araştırmada geliştirilen araç, hata denetimi, renkli sözdizimi vurgulama ve bölmede bir hata göstergesi için bir sisteme sahiptir. OSÇ-AÖD, bu geliştirme ortamının bir parçası olarak kullanıcı dostu bir editör sağlar ve iş arkadaşları gibi teknik olmayan kişiler için uygun hale getirir. Bu nedenle, gerekli teknik uzmanlığa sahip olmasalar bile daha fazla sayıda kişinin olay tabanlı modellemeye katılması mümkündür.

OSÇ-AÖD, grafik görselleştirme için uygun bir ortam sunar, bu da grafikleri görüntülemeyi mümkün kılarken aynı zamanda hata kontrolü ve sözdizimi vurgulama için destek sağlar. OSÇ-AÖD geliştirmesi için gereken süre, TSD grafik görselleştirmesi için gerekenden çok daha uzundur. Özellikle projenin kısıtlı bir bölümünde kullanıldığında, OSÇ-AÖD üretkenliği önemli ölçüde artırabilir. Vaka çalışması, ana OSÇ ve ilk alt OSÇ görselleştirmesinden sonra, yazılım eserlerinin yeniden kullanılmasını sağladığı için zaman anlamında önemli gelişmeler olduğunu göstermektedir. Bir şeyi yeniden kullanma kapasitesi, onu kullanan kişiler için sonucu iyileştirir ve dolaylı olarak projenin üretimini artırır. Ek olarak, renkli sözdizimini ezberlemek kolaydır ve kopyalama ve yapıştırma gibi etkinlikler çok az ek iş ile üretkenliği artırabilir.

Gelecek çalışma olarak OSÇ-AÖD ile grafik görselleştirme projesi arasındaki iletişim için bir hat tasarlama ve geliştirme amaçlanmaktadır. Kullanıcıların OSÇ-AÖD'ne erişimini genişletmek için, uygulama bir yazılım programı olarak bulut üzerinden kullanılabilir hale getirilebilir. Ayrıca, karar tablosunun oluşturulması, erişilebilir ve kullanışlı bir sözdizimi oluşturulmasına yönelik geliştirmeler tamamlanacaktır. AÖD

modelleri üzerinden yapılan sadeleştirme işlemi sonucunda OSÇ'lerin görsel olarak analiz edilmesi için harcanan süre kısalacaktır.

Etik kurul onayı ve çıkar çatışması beyanı

Hazırlanan makalede etik kurul izni alınmasına gerek yoktur.

Hazırlanan makalede herhangi bir kişi/kurum ile çıkar çatışması bulunmamaktadır.

Yazar katkılarının beyanı

İlk yazar, ikinci ve üçüncü yazarın ortak yüksek lisans tez öğrencisidir. Bu makale de ilk yazarın yüksek lisans tezi kapsamında yaptığı çalışmalardan oluşturulmuştur.

Kaynaklar

[1] Eeles, P., Sam, H. B., Mistrík, I., Roshandel, R., Stal, M. 2014. Relating System Quality and Software Architecture: Foundations and Approaches. Morgan Kaufmann, Boston, 1-20. DOI: 10.1016/B978-0-12-417009-4.00001-6.

[2] Belli, F. 2001. Finite state testing and analysis of graphical user interfaces. 12th International Symposium on Software Reliability Engineering, 27-30 Kasım, Hong Kong DOI: 10.1109/ISSRE.2001.989456.

[3] Chow, T. S. 1978. Testing Software Design Modeled by Finite-State Machines, IEEE Transactions on Software Engineering, Cilt. 4, 178-187. DOI: 10.1109/TSE.1978.231496.

[4] Belli, F., Linschulte, M., Tuğlular, T. 2016. Karar tablosu destekli olay sıra çizgeleri temelli sınama durum üretim aracı. 10. Ulusal Yazılım Mühendisliği Sempozyumu, 24-26 Ekim, Çanakkale. s. 408-413. DOI: S0218194016500091

[5] Budnik, C. J., Belli, F., Hollmann, A. 2009. Structural feature extraction for GUI test enhancement. International Conference on Software Testing, Verification, and Validation Workshops, 1-4 Nisan, Denver, CO, ABD, s. 255-262, DOI: 10.1109/ICSTW.2009.19

[6] Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., Woodhull, G. 2003. Graphviz and dynagraph – static and dynamic graph drawing tools. Graph Drawing Software, Mathematics and Visualization, Springer, Berlin, Heidelberg, 378 s. DOI: 10.1007/978-3-642-18638-7_6

[7] Öztürk, D. 2020. A Model-Based Test Generation Approach for Agile Software Product Lines. İzmir Yüksek Teknoloji Enstitüsü, Mühendislik ve Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 108s, İzmir. https://gcris.iyte.edu.tr/handle/11147/10970 (Erişim Tarihi: 30.01.2023).

[8] Uddin, A., Anand, A. 2019. Importance of Software Testing in the Process of Software Development, International Journal for Scientfic Research and Development, Cilt.6, 141-145.

[9] Bentley, J. 1986. Programming pearls: Little languages, Communications of the ACM, Cilt. 29, No. 8, s. 711-721. DOI: 10.1145/6424.315691.

[10] Deursen, A., Klint, P., Visser, J. 2000. Domain-Specific Languages: An Annotated Bibliography, ACM Sigplan Notices, Cilt. 35, No. 6, 26–36. DOI: 10.1145/352029.352035

[11] Xtext, 2006. https://www.eclipse.org/Xtext/, (Erişim Tarihi: 30.01.2023).

[12] SciSpike/yaktor, scispike, 2021. https://github.com/SciSpike/yaktor (Erişim Tarihi: 30.01.2023).

[13] SciSpike/yaktor-dsl-xtext, scispike, 2017. https://github.com/SciSpike/yaktor-dsl-xtext (Erişim Tarihi: 30.01.2023).

[14] GitHub, Franca, 2018. https://github.com/franca/franca, (Erişim Tarihi: 30.01.2023).

[15] Expression Language, 2021. https://github.com/intuit/common-xtext-expression-language. (Erişim Tarihi: 30.01.2023).

[16] Bagga, J. S., Heinz, A. 2002. JGraph - A Java Based System for Drawing Graphs and Running Graph Algorithms, Lecture Notes in Computer Science, Cilt. 2265. Springer, Berlin, Heidelberg. DOI: 10.1007/3-540-45848-4_45

[17] JUNG Framework Tech Report, 2016. http://www.datalab.uci.edu/papers/JUNG_tech_report.html#related. (Erişim Tarihi: 30.01.2023).

[18] PlantUML, 2009. https://plantuml.com/ (Erişim Tarihi: 30.01.2023).

[19] Graphviz, 1991. https://graphviz.org/ (Erişim Tarihi: 30.01.2023).

[20] Gansner, E., North, S. 1997. An Open Graph Visualization System and Its Applications to Software Engineering, Journal of Software: Practice and Experiences, Cilt. 30, 1203-1233. DOI: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N.

[21] Gansner, E. R., Koutsofios, E., North, S. C., Vo, K.-P. 1993. A technique for drawing directed graphs, IEEE Transactions on Software Engineering, Cilt. 19, No. 3, 214-230. DOI: 10.1109/32.221135.

[22] Mernik, M., Heering, J., Sloane, A. M. 2005. When and how to develop domain-specific languages, ACM Computing Surveys, Cilt. 37, No. 4, 316–344. DOI: 10.1145/1118890.1118892.

[23] Jetbrains Meta Programming System, https://www.jetbrains.com/mps/ (Erişim Tarihi: 30.01.2023).

[24] Merks, E., Paternostro, M., Budinsky, F., Steinberg, D. 2009. EMF: Eclipse Modeling Framework. 2nd edition, 722s.

[25] Efftinge, S., Völter, M. 2006. oAW xText: a framework for textual DSLs. Workshop on Modeling Symposium at Eclipse Summit, Cilt. 32, 22 Eylül, Flensburg, Germany.

[26] Belli, F., Budnik, C. J. 2004. Minimal Spanning Set for Coverage Testing of Interactive Systems. Theoretical Aspects of Computing, 20-24 Ekim, Guiyang, China, 220–234. DOI: 10.1007/978-3-540-31862-0_17.

[27] Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe , A., Mehandjiev, N. 2004. Meta-Design: A manifesto for End-User Development, Communications of the ACM, Cilt. 47, s. 33–37. DOI: 10.1145/1015864.1015884.

[28] Compatangelo, E., Meisel, H. 2002. Conceptual Analysis of EER Schemas and Ontologies. https://www.semanticscholar.org/paper/Conceptual-Analysis-of-EER-Schemas-and-Ontologies-Compatangelo-Meisel/47d57c45ab8b52643f7a250bd3db8fd3e58a25f6 (Erişim Tarihi: 30.01.2023).

[29] ESG Engine, 2021. https://github.com/esg4aspl/esg-engine (Erişim Tarihi: 30.01.2023).

[30] Gutiérrez, J. J., Ramos, I., Mejías, M., Arévalo, C., Sánchez-Begines, J. M., Lizcano , D. 2017. Modelling gherkin scenarios using uml. 26th Information Systems Development: Advances in Methods, Tools and Management, 6-8 Ekim, Larnaca, Kıbrıs. https://idus.us.es/handle/11441/127202 (Erişim Tarihi: 30.01.2023).

[31] ESG DSL case study outputs, 2022. https://github.com/esg4aspl/esg-dsl/tree/master/case_study_output (Erişim Tarihi: 01.05.2023).

[32] Kahraman, G., Bilgen, S. 2015. A framework for qualitative assessment of domain-specific languages, Software and Systems Modeling (SoSyM), Cilt. 14, No. 4, 1505–1526. DOI: 10.1007/s10270-013-0387-8.