DAVRANIŞ ODAKLI GELİŞTİRME YAKLAŞIMIYLA BİR TEST OTOMASYONUNUN GELİŞTİRİLMESİ VE SÜRECİN DEĞERLENDİRİLMESİ

YÜKSEK LİSANS TEZİ

Fatih KÖKTEN

Danışman
Dr. Öğr. Üyesi Levent ÇELİK

İNTERNET VE BİLİŞİM TEKNOLOJİLERİ YÖNETİMİ
ANABİLİM DALI
Mayıs 2019

AFYON KOCATEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

DAVRANIŞ ODAKLI GELİŞTİRME YAKLAŞIMIYLA BİR TEST OTOMASYONUNUN GELİŞTİRİLMESİ VE SÜRECİN DEĞERLENDİRİLMESİ

Fatih KÖKTEN

Danışman
Dr. Öğr. Üyesi Levent ÇELİK

İNTERNET VE BİLİŞİM TEKNOLOJİLERİ YÖNETİMİ
ANABİLİM DALI

Mayıs, 2019

TEZ ONAY SAYFASI

Fatih Kökten tarafından hazırlanan "Davranış Odaklı Geliştirme Yaklaşımıyla Bir Test Otomasyonunun Geliştirilmesi ve Sürecin Değerlendirilmesi" adlı tez çalışması lisansüstü eğitim ve öğretim yönetmeliğinin ilgili maddeleri uyarınca 24/05/2019 tarihinde aşağıdaki jüri tarafından **oy birliği** ile Afyon Kocatepe Üniversitesi Fen Bilimleri Enstitüsü **İnternet ve Bilişim Teknolojileri Yönetimi Anabilim Dalı'nda YÜKSEK LİSANS TEZİ olarak** kabul edilmiştir.

Danışman : Dr. Öğr. Üyesi Levent ÇELİK

İmza

Başkan : Dr. Öğr. Üyesi Hüseyin ÇAKIR
Gazi Üniversitesi, Gazi Eğitim Fakültesi .....

Üye : Dr. Öğr. Üyesi Fatih ÖZDİNÇ
Afyon Kocatepe Üniversitesi, Eğitim Fakültesi .....

Üye : Dr. Öğr. Üyesi Levent ÇELİK
Afyon Kocatepe Üniversitesi, Eğitim Fakültesi .....

Afyon Kocatepe Üniversitesi
Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
...../...../..... tarih ve
……. sayılı kararıyla onaylanmıştır.

…….
Prof. Dr. İbrahim EROL
Enstitü Müdürü

BİLİMSEL ETİK BİLDİRİM SAYFASI
Afyon Kocatepe Üniversitesi

Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- Görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- Başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- Atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- Kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- Ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

24/05/2019

Fatih KÖKTEN

ÖZET
Yüksek Lisans Tezi

DAVRANIŞ ODAKLI GELİŞTİRME YAKLAŞIMIYLA BİR TEST OTOMASYONUNUN GELİŞTİRİLMESİ VE SÜRECİN DEĞERLENDİRİLMESİ

Fatih KÖKTEN
Afyon Kocatepe Üniversitesi
Fen Bilimleri Enstitüsü
İnternet ve Bilişim Teknolojileri Yönetimi Anabilim Dalı
Danışman: Dr. Öğr. Üyesi Levent ÇELİK

Bu araştırmanın amacı öğrenme yönetim sistemi olan Moodle uygulamasını kullanıcı rolleri açısından fonksiyonel olarak özelliklerinin çalışabilirliğini kontrol edecek test otomasyonunun geliştirilmesidir. Davranış Odaklı Geliştirme yaklaşımıyla manuel test ve otomasyon yazılım testlerini gerçekleştirmektir. Bu süreçte manuel test ve otomasyon test süreçlerini zaman, insan kaynağı, olumlu-olumsuz yönleri, sınırlılıkları ve kullanılabilirlik farklılıklarının karşılaştırılması yapılmıştır.

Bu araştırmada kapsamın, kullanıcı gruplarının belirlenmesinde alan uzmanlarının görüşlerine başvurulmuştur. Bu doğrultuda kullanıcı rollerine ait kullanıcı davranışlarının fonksiyonel analizi yapılmıştır. Sonrasında kullanıcı davranışları Davranış Odaklı Geliştirme yaklaşımına göre kullanıcı hikâyeleri yazılmıştır. Uzman görüşleri alındıktan sonra Moodle uygulamasının manuel yazılım testleri ve yazılım test otomasyonu yapılmıştır. Kullanıcı hikâyeleri Ruby, Cucumber, Gherkin ve Selenium Webdriver yazılım araçları kullanarak otomatize edilmiştir. Optimisazyon aşamasında otomatize edilen senaryoların grup halinde birlikte çalışabilirliği sağlanmıştır. Moodle uygulamasında tanımsız olan class yapılarından dolayı bazı senaryoların test otomasyonu yapılamamıştır. Bu nedenle özellik dosyalarının bir kısmı tekrarlı olarak çalıştırılamamıştır.

Araştırma sonucunda manuel testlerin tamamlanmasının uzun zaman aldığı

görülmüştür. Yazılım test otomasyon sürecinde testlerin otomatize edilmesinin uzun sürdüğü fakat çalışma süresinin kısa sürdüğü görülmüştür. Otomatik testlerin çalıştırılma süreleri ve sonuçlanması manuel testlere göre daha kısa zamanda tamamlanmıştır. Bir birlerine bağımlı test senaryolarının otomatik hale gerilememesinden dolayı otomatize edilen testlerin sürekli çalıştırılamadığı görülmüştür. Davranış Odaklı Geliştirme yaklaşımıyla yazılım testlerinin ilgili yazılım geliştirme ekibinden bağımsız yapılması test aktivitelerinde daha fazla zaman harcanmasına neden olduğu görülmüştür.

2019, xiii + 141 sayfa

Anahtar Kelimeler: Yazılım Testi, Test Otomasyonu, Davranış Odaklı Geliştirme, Çevik Yaklaşım

ABSTRACT
M.Sc. Thesis

IMPROVING A TEST AUTOMATION WİTH BEHAVIOR DRIVEN DEVELOPMENT APPROACH AND EVALUATION OF THE PROCESS

Fatih KÖKTEN

Afyon Kocatepe University

Graduate School of Natural and Applied Sciences

Department of Internet and Information Technologies Management

Supervisor: Asst. Prof. Levent ÇELİK

The purpose of this study is to develop test automation to control the functionality of the Moodle application, a learning management system, in terms of user roles. To conduct manual testing and automation software tests with Behavior Driven Development approach. In this process, time, human resources, positive-negative aspects, manual testing and the limitations of automation test processes and usability differences were compared.

In this study, the opinions of the field experts were used to determine the scope and user groups. In this respect, the functional analysis of the user behaviors of the user roles was made. After that, user stories are written according to the Behavior Driven Development approach. After obtaining expert opinions, manual software tests and software test automation of Moodle application were performed. User stories have been automated using Ruby, Cucumber, Gherkin and Selenium Webdriver software tools. In the optimization phase, grouped automated scenarios are interoperable. Due to the undefined class structures in Moodle application, some scenarios could not be tested. For this reason, some of the feature files cannot be run repeatedly.

As a result of the research, it was observed that the manual tests took a long time to complete. In the software test automation process, it has been observed that the tests

have been automated but the working time is short. The duration and results of the automatic tests were completed in a shorter time than the manual tests. It has been observed that the automated tests have not been able to run continuously due to the automatic regression of the dependent test scenarios. With the Behavior Driven Development approach, software tests were performed independently from the related software development team, resulting in more time spent in test activities.

2019, xiii + 141 pages

Keywords: Software Testing, Test Automation, Behavior Driven Development, Agile Approach

TEŞEKKÜR

Bu araştırmanın konusu, çalışmaların yönlendirilmesi, sonuçların değerlendirilmesi ve yazımı aşamasında yapmış olduğu büyük katkılarından dolayı tez danışmanım Sayın Dr. Öğr. Üyesi Levent ÇELİK' e teşekkürlerimi sunarım.

Her konuda öneri ve eleştirileriyle yardımlarını gördüğüm hocalarıma ve arkadaşlarıma teşekkür ederim.

Bu araştırma boyunca maddi ve manevi desteklerinden dolayı aileme teşekkür ederim.

Fatih KÖKTEN
AFYONKARAHİSAR, 2019

İÇİNDEKİLER DİZİNİ

Sayfa

KISALTMALAR DİZİNİ

Kısaltmalar

API	Application Programming Interface	Uygulama Programlama Arayüzü
AR-GE	Research and development	Araştırma Geliştirme
ASD	Adaptive Software Development	Adaptif Yazılım Geliştirme
BDD	Behavior Driven Development	Davranış Odaklı Geliştirme
BDT	Behavior Driven Traceability	Davranış Odaklı İzlenebilirlik
DSDM	Dynamic Systems Development Methodology	Dinamik Sistemler Geliştirme Yöntemi
EAMS-CBALM	Educational And Academic Management System For Courses Based On Active Learning Methodologies	Aktif Öğrenme Metodolojilerine Dayalı Dersler İçin Eğitim ve Akademik Yönetim Sistemi
FDD	Feature Driven Development	Özellik Odaklı Geliştirme
IDE	Integrated Development Environment	Tümleşik Geliştirme Ortamı
IPTV	Internet Protocol Television	İnternet Protokolü Televizyonu
LSD	Lean Software Development	Yalın Yazılım Geliştirme
NBDT	Normalized Behavior Driven Traceability	Normalleştirilmiş Davranış Odaklı İzlenebilirlik
SAAS	Software as-a Service	Bulut Yazılım Servisleri
TDD	Test Driven Development	Test Odaklı Geliştirme
XP	Extreme Programming	Uç Programlama

ŞEKİLLER DİZİNİ

Sayfa

TABLOLAR DİZİNİ

Sayfa

RESİMLER DİZİNİ

Sayfa

1. GİRİŞ

Yazılım testi, bir yazılım ürünün son kullanıcılara sunulmadan önce beklenen ihtiyaçları ne oranda karşıladığı hakkında bilgiler veren bir eylem olarak nitelendirilmektedir. Analiz, tasarım, geliştirme, test ve ürün olarak belirtilen klasik yazılım yaşam döngüsünde test aktivitesi her aşamada yapılmaktadır. Analiz aşamasında belirtilen yazılım gereksinimlerinin açıklaması, doğruluğu ve istenen özelliğe eş değer olması kontrol edilmektedir. Tasarım aşamasında yapılan tasarımın analiz aşamasındaki belirtilen gereksinimlere göre olması gerekmektedir. Yazılım geliştirme aşamasında yazılan kodlara karşılık gelen ürün çıktısının analiz ve tasarım aşamalarınına göre doğruluklarına bakılmaktadır. Yazılım yaşam döngüsünde her aşamasında yazılım testlerinin yer alması ne kadar gerekli ve önemli olduğunu göstermektedir.

Yazılım geliştirme süreci tamamlandıktan sonra teslim edilen ürünün kaliteli seviyesinin yüksek olması önemli olmaktadır. Bir ürününün kalitesini, doğruluğunun değerlendirilmesinin zor olduğu belirtilmektedir. Bu sebeple yazılım geliştirme sürecinde bazı süreçlere, model ve standartlarına duyulan ihtiyacın gerekliliği ortaya çıkmaktadır. Yazılım geliştirme projelerinde ortaya çıkan bütçe ve zaman kaynaklarının sınırlılığı, başarısızlıklar ve ürünün hatalı ya da istenenden farklı olması organizasyonları etkin bir yazılım kalite yönetim süreci ihtiyacını doğurmaktadır (Serdaroğlu 2015).

Yazılım test süreçlerinde test aktiviteleri genel olarak manuel testlerle yürütülmektedir. Bunun nedeni test araçlarına ihtiyaç duymadan hızlı bir şekilde test edilecek yazılım üzerinde doğrudan test aktivitesine başlanmasdır. Test uzmanı yazılım gereksinimlerine göre geliştirilen yazılımın kontrollerini el ile yapmaktadır. Kısa zamanda sonuç vermesi ve maliyetinin düşük olması nedeniyle ilk aşamada tercih edilmektedir. Projelerin büyüklüğü ve test tekrarlarının fazla olduğu zaman maliyet ve kaynak ihtiyacının arttığı görülmektedir. Bu ve benzer vakallar nedeniyle yazılım test otomasyonu ihtiyacı son yıllarda giderek artmaktadır. Otomasyon testleri bu manuel testlere göre zaman ve maliyetten daha iyi kazanç sağlamaktadır. Otomatik testlerin tercih edilmesinin bir diğer nedeni insan kaynaklı hataları en aza indirmesidir. Bununla birlikte gelişen teknoloji ve

araçları test süreçlerine dâhil ederek teknolojinin merkezde olduğu daha iyi yönetilebilir bir süreç oluşturmak istenmektedir. Otomasyon testlerini hazırlanabilmesi için çeşitli yazılım uygulamalarına ihtiyaç duyulmaktadır. Farklı yazılım dillerinde test otomasyonu geliştirilmektedir.

İhtiyaçların farklılaşması ve teknolojinin ilerlemesiyle yeni süreçler, modeller, yaklaşımlar ve yöntemler ortaya çıkmıştır. Bunların birçoğu karşılaşılan zorlukların aşılması, süreçleri kolaylaştırmak ve karşılanması gereken ihtiyaçlara yönelik olmaktadır. Bu tez çalışmasında örnek bir yazılım test otomasyonu geliştirilmiştir. Geliştirilecek olan test otomasyon uygulamasında basit, anlaşılır, ihtiyaçlara hitap eden, yazılım araçlarını kullanılmıştır. Bununla birlikte yazılım otomasyonu geliştirilecek olan Moodle uygulamasının süreç ve otomasyon çalışmaları benzer web uygulamaları için yazılım test otomasyonu geliştirilmesi çalışmaları için örnek bir çalışma niteliğindedir.

1.1 Amaç

Bu araştırmanın amacı öğrenme yönetim sistemi olan Moodle uygulamasını kullanıcı rolleri açısından fonksiyonel olarak özelliklerinin çalışabilirliğini kontrol edecek test otomasyonunun geliştirilmesidir. Davranış Odaklı Geliştirme yaklaşımıyla manuel test ve otomasyon yazılım testlerini gerçekleştirmektir. Bu süreçte manuel test ve otomasyon test süreçlerini zaman, insan kaynağı, olumlu-olumsuz yönleri, sınırlılıkları ve kullanılabilirlik farklılıklarının karşılaştırılması yapılmıştır. Bu bağlamda araştırmada aşağıdaki sorulara yanıt aranmıştır:

- Ziyaretçi kullanıcı rolü açısından fonksiyonel olarak Moodle uygulamasının çalışabilirliğini kontrol edecek test otomasyonunu geliştirilebilir mi?
- Öğrenci kullanıcı rolü açısından fonksiyonel olarak Moodle uygulamasının çalışabilirliğini kontrol edecek test otomasyonunu geliştirilebilir mi?
- Öğretmen kullanıcı rolü açısından fonksiyonel olarak Moodle uygulamasının çalışabilirliğini kontrol edecek test otomasyonunu geliştirilebilir mi?

2. LİTERATÜR BİLGİLERİ

2.1 Çevik Yazılım Geliştirme

Çevik (Agile) yazılım geliştirme, yazılım dünyasının günümüzün hızlı değişen bilişim teknolojilerine hızlı ayak uydurabilmesini sağlayan yazılım geliştirme yöntemi olarak nitelendirilmektedir. Çevik yazılım geliştirme süreci 2001 yılında 17 yazılım geliştirme uzmanı tarafından bir manifesto halinde ilkesel olarak yayınlanmaktadır. Yayınlanan bu manifestoda:

- Süreçler ve araçlardan ziyade bireyler ve etkileşimlere,
- Kapsamlı dokümantasyondan yerine çalışan yazılıma,
- Sözleşme pazarlıklarından ziyade müşteri ile işbirliğine,
- Bir plana bağlı kalmaktan çok değişime karşılık vermeye,

şeklinde ifade edilmiştir (İnt.Kyn.1). Bu manifestoda özetlenen görüşün yayınlanmasından sonra farklı zamanlarda geliştirilen aşağıdaki çevik yazılım geliştirme yaklaşımlarının kullanımları artmıştır (Campanelli and Parreiras 2015).

- Uç Programlama (Extreme Programming),
- Scrum,
- Kanban,
- Yalın Yazılım Geliştirme (Lean Software Development),
- Dinamik Sistemler Geliştirme Yöntemi (Dynamic Systems Development Methodology),
- Adaptif Yazılım Geliştirme (Adaptive Software Development),
- Özellik Odaklı Geliştirme (Feature Driven Development),
- Test Odaklı Geliştirme (Test Driven Development),
- Davranış Odaklı Geliştirme (Behavior Driven Development)

Çevik yazılım geliştirme sürecindeki modeller artırım ve iterasyon temelli modeller olarak belirtilmektedir. Klasik şelale yazılım geliştirme modelini bölümlere ayırır, her

bir bölüm için aynı süreci gerçekleştirir ve geliştirme döngüsünde tekrarlamaktadır (Cohen *et al.* 2004). Çevik yazılım geliştirme modelleri, geliştirme süresini azaltmak amacıyla projeyi eşzamanlı artırımlara bölmektedir. Şelale yazılım geliştirme modelinde olduğu gibi geliştirme başlamadan tüm ihtiyaçlar analiz edilir ve ihtiyaçlar fonksiyonel artımlar olacak şekilde parçalara ayırmaktadır. Artırımların geliştirilmesi eşzamanlı olduğunda, çok görevlilik mantığı ile yazılım geliştirme süresi azalmaktadır.

İteratif geliştirme yazılım projesini iki ile dört hafta uzunluklarda iterasyonlara bölmektedir. Bu şekilde bölümlenen iterasyonlar kullanılabilir ve dökümante edilmiş ürünler üretilmesi bakımından kolaylık sağlamaktadır. İlk iterasyon en temel ürünü üretmekle başlamaktadır. Sonraki ardışık her iterasyon ürüne yeni özellikler eklemektedir. Her iterasyonda, o iterasyona ait analiz, tasarım, gerçekleştirme ve test etme işlemleri sürdürülmektedir. İteratif geliştirme, sadece bulunulan iterasyondaki ihtiyaçları karşılayacak değişimler ile ilgilenmektedir (Cohen *et al.* 2004). Çevik yazılım geliştirme modelleri olarak nitelendirilen bu yöntemlerden bazıları manifestodan önce de kullanılan modeller olarak belirtilmektedir. Fakat manifestonun yayınlanması ile daha çok kullanılmaya başlanmaktadır (Baran *et al.* 2016).

Çevik yazılım geliştirme modellerin özü uygulama yöntemleridir. Uygulamaların çevik yöntem olabilmesi için manifestodaki her bir temel ilkeye uyulması gerektiğine dikkat çekilmektedir. Manifestodaki belirtilen uygulamalar birbirinden bağımsız olarak kullanılmaktadır. Çevik yöntemlerin odağındaki temel uygulamalar; yinelemeli ve artımlı modelleme, takım çalışması, basitlik ve doğrulamadır. Çevik yazılım geliştirme yöntemi sadece bir uygulama topluluğudur. Uygulamaların bir araya gelme şekilleri ve örgüt kültürü günlük çalışma ortamları, araçlar ve ekip çalışması konularında nasıl desteklenmeleri gerektiği hakkında çok fazla düşünce içermektedir. Birçok çevik yöntem müşteri varlığını, küçük takımları ve yakın iletişimi desteklemektedir (Abrahamsson *et al.* 2017).

Çevik terimi 'hızlı hareket' anlamına gelmektedir. Çevik yöntemde yazılım geliştirme ekibinin değişen yazılım ihtiyaçlarına hızlı cevap verebilecek dinamik bir yapıya sahip olması gerekmektedir. Yazılımların hızlı teslimatı ile müşteri memnuniyeti, gelişmekte

olan değişen gereksinimleri karşılama, geliştirilen yazılımı sık sık verilmesidir. İlkelerin en önemlisi küçük ve kullanışlı yazılımların hızlı ve sürekli teslimatı sağlanarak müşteri memnuniyetidir. Çevik modellerin avantajlarından en önemlisi, projenin değişen gereksinimlerine cevap verebilme yeteneği olarak belirtilmektedir. Müşteriyle yüz yüze iletişim ve sürekli girdiler olduğu için geliştirme ekibi ile müşteri arasında anlaşmazlık olmamaktadır. Çevik modellerin en önemli dezavantajları, projeler küçük ise çevik modelin kullanılması kesinlikle karlı ancak büyük bir proje ise yazılım geliştirme yaşam döngüsünde proje için gereken çabayı ve zamanı değerlendirmek zorlaşmaktadır (Balaji and Murugaiyan 2012).

2.1.1 Uç Programlama

Uç Programlama (Extreme Programming), yazılım geliştirme süreci boyunca son derece kaliteli olmak şartıyla çalıştırılabilir kod bütünü üretmeyi hedeflemiş bir yazılım geliştirme metodolojisidir. Sürecinin en temel, önemli ve sonuç çıktısı olan ürün çalıştırılabilir kod olduğundan, uç programlama metodolojisi sürecin en başından itibaren çalıştırılabilir kodu sürecin merkezinde tutmaktadır. Uç Programlama kısa adımlardan oluşan tekrarlı bir süreç olarak belirtilmektedir. Her bir tekrar süresi birkaç gün ila birkaç haftadan daha uzun olmamasıdır. İlk adımın sonucunda geliştirilmekte olan yazılım sistemin en düşük seviyede çalıştırılabilir bir örneği üretilmiş olmalıdır. Bu ilk adımdan sonraki her bir tekrar adımında bir önceki adıma göre daha fazla niteliğe sahip ve aynı zamanda iyileştirilmiş bir sürüm ortaya çıkarılması amaçlanmaktadır. Bu şekilde sık tekrarlar ile sistemin bütünü tamamlanmaktadır. Uç Programlama, yazılımın her bir adımda daha fazla büyümesi ve süreklilik kazanması özellikleri açısından yüksek kalitede kod üretimine önem vermektedir. Her bir adımın temel hedefi geliştirmek ve kod düzenleme yapmaktır. Geliştirmek, yeni işlevler ve özellikler ekleme olarak belirtilmektedir. Kod düzenleme ise çalışan kodun işleyişine zarar vermeden kodun kalitesinin arttırılması amacıyla yeniden yapılandırılmasıdır. Kod düzenleme çalışması, Uç Programlamanın önemle üzerinde durmuşğu bir işlem olarak gösterilmektedir (İnt.Kyn.2). Uç Programlama'nın özünde aşağıdaki uygulamalar yer almaktadır:

- Planlama: Her tekrar adımına basit bir planla başlanır ve gerekli oldukça bu planda değişikliğe ve iyileştirmeye gidilmesidir.
- Sık ve küçük sürümler: Temek gedef her tekrar adımında çalışan bir sürüm oluşturmaktır. Tekrarlamaların süresi de birkaç haftadan daha uzun olmamalıdır.
- Basit tasarım: Tasarım mümkün olduğunca basit tutulmalı, gerektiğinde düzenlemeler yapılmalıdır.
- Önce test: Kod yazılmadan önce yazılacak kodun testinin yazılmasıdır.
- Refactor etme (kod düzenleme): Kod tekrarlarından uzak durmak, sade ve temiz kod elde etmek için sürekli olarak düzenlemeler yapılmasıdır.
- Çift programcı: Kodun hep çiftler halinde yani eşli olarak yazılmasıdır.
- Sürekli bütünleştirme (Continuos Integration): Her kodlama görevi tamamlanır tamamlanmaz ana sistemle bütünleştirilmelidir.
- Haftada 40 saat çalışma: Geliştirme ekiplerinin verimli çalışabilmesi için aşırı çalışma saatlerinden kaçınılmalıdır.
- Müşteri ile yakın iletişim: Müşterinin tam zamanlı olarak geliştirme ekibiyle yakın temas halinde bulunması ve sürece her an dâhil olabilmesi gerekmektedir.
- Kodlama standartları: İsimlendirme, kaynak kod düzenleme ve dokümantasyon için yaygın standartların tüm ekip tarafından benimsenmesi gerekli görülmektedir.

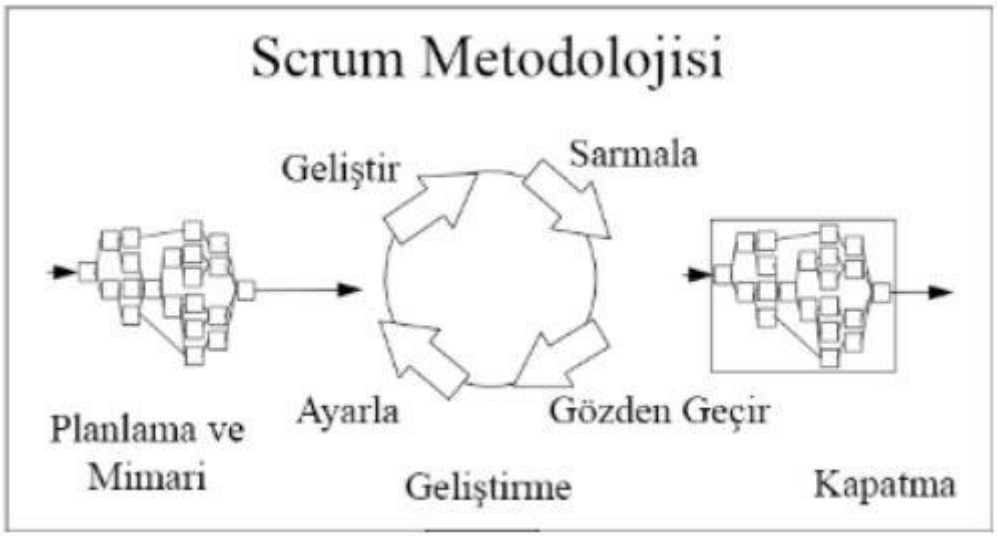
Kent Beck tarafından tanımlanan XP farklı boyutlardaki birçok şirkette ve çeşitli endüstrilerde çok başarılı olduğu kanıtlanmaktadır. XP yaklaşımının odağı müşteri memnuniyeti olmasıdır. Bu yaklaşımla geliştiricilere değişen müşteri gereksinimlerine cevap verebilme, hızlı ve sürekli olarak yüksek kaliteli yazılım sunma yetkisi vermektedir. XP yaklaşımında çalışma yazılımı müşteriye genellikle 1-3 hafta arayla verilmektedir. İletişimi, basitliği, geri bildirimi, saygıyı ve cesareti benimsemektedir. XP yaklaşımı 12 kural içermektedir. Bu kurallar: Planlama Oyunu, Küçük Sürümler, Müşteri Kabul Testleri, Basit Tasarım, Çift Programlama, Teste Dayalı Geliştirme, Yeniden Düzenleme, Sürekli Entegrasyon, Toplu Kod Sahipliği, Kod Standartları, Metafor ve Sürdürülebilir Hız şeklinde belirtilmektedir (Stankovic *et al.* 2013).

XP, geleneksel gelişim modellerinin uzun gelişim döngüsünün neden olduğu

sorunlardan evrilmektedir. İlk olarak geçen yıllarda yazılım geliştirme süreçlerinde etkili bulunan uygulamalarla işlerin yapılması için basit bir fırsat olarak başlamaktadır (Beck 2000). Pratikte yapılan birçok başarılı denemeden sonra XP metodolojisinde kullanılan temel prensipler ve uygulamalar teorik hale getirilmektedir. Denemeler XP'de toplanarak birbirleriyle yeni bir şekilde çalışacak haliyle sıralanmıştır. Böylece yazılım geliştirme için yeni bir metodoloji oluşturulmuştur. "Extreme" terimi bu ortak ilke ve uygulamaları aşırı seviyelere çıkarmaktan gelmektedir (Abrahamsson *et al.* 2017).

2.1.2 Scrum

Scrum, ilk bakışta çok basit kuralları olan bir yönetimsel model olarak belirtilmektedir. Spesifikasyonları net şekilde belli olmayan, değişime açık, karmaşık yazılım projelerinin yönetimi için uygulanmaktadır. Scrum, detaylı bir şekilde projede takip edilmesi gereken adımları belirtmemekte, onun yerine basit ama önemli birkaç olmazsa olmaz kuralıyla esnek bir yönetim sunmaktadır. Scrum'ın sunmakta olduğu en önemli çıktı sürecin şeffaf bir şekle getirilerek süreç içerisinde aksayan noktaların açığa çıkarılmasıdır. Scrum böylelikle proje ekibini ortaya çıkan aksaklıkları çözümleyerek sürekli iyileştirme yapması yönünde motive etmektedir.



Şekil 1.1 Scrum Yapısı

Scrum, tam bir süreç veya metodoloji yerine hafif bir yönetim çerçevesi olarak

görülmektedir. Her tür yinelemeli ve artımlı projeleri yönetmek ve kontrol etmek için geniş bir uygulanabilirliğe sahip oldığu görülmektedir. Scrum basitliği, kanıtlanmış verimliliği ve diğer çevik metodolojiler tarafından desteklenen çeşitli mühendislik uygulamaları için sarıcı olarak davranma kabiliyeti nedeniyle yazılım topluluğunda artan popülariteye kavuşmaktadır (Stankovic *et al*. 2013).

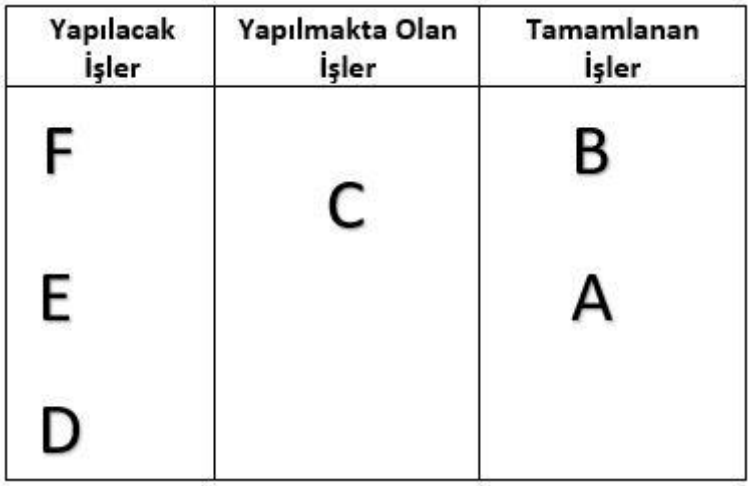
Scrum genel olarak aşağıdaki prensipleri içermektedir (Kniberg and Skarin 2010):

- Organizasyonu küçük, cross-functional, kendini organize eden ekiplere ayırılması (Scrum, her şeyin nasıl yapılacağına dair tam ve ayrıntılı açıklamalar sunmaz ve karar vermek için ekibin kendisine bırakılmaktadır).
- Projenin (yapılması gereken işler) özellikler, hatalar, işlevsel olmayan gereksinimler, vb. gibi küçük, somut teslim edilebilir ürünler içeren ürün biriktirme listesi adı verilen bir listeye bölünmektedir. Liste öncelik sırasına göre sıralanır ve her madde için nispi eforun tahmini yapılmaktadır.
- Zamanı bölmek ve 1 aydan uzun olmayan yinelemeler oluşturmaktır. Değişiklikler iterasyon (sprint) olarak adlandırılır ve müşteriye sunulabilecek ve teslim edilebilecek çalışma kodu ile sonuçlanmaktadır.
- Önceki yinelemeden çıkarılan sonuçları inceleyerek kazanılan iç görülere dayanarak, müşteriyle işbirliği içinde planı optimize etmek ve öncelikleri güncellemektir.
- Her yinelemeden sonra retrospektifler alarak sürecin kendisini optimize etmektir.

2.1.3 Kanban

Kanban genel olarak çalışmaları görselleştirmek, devam etmekte olan işleri sınırlamak ve verimliliği (veya akışı) en üst düzeye çıkarmak için geliştirilen bir yaklaşımdır. Kanban ekipleri bir projenin (veya kullanıcı hikâyesinin) başından sonuna kadar geçen süreyi azaltmaya odaklanmaktadır. Bunu bir kanban tahtası kullanarak yaparlar ve çalışma akışlarını sürekli geliştirmektedirler. Bu yaklaşımda yazılım takımının kanban panosu olduğu belirtilmektedir. Bazı takımlar çevik bir mentor yer almaktadır.

Yaklaşımda yönetim üzerinde ortaklaşa çalışmak ve işleri yerine getirmek tüm ekibin ortak sorumlulu olarak görülmektedir. Kanban yaklaşımında ürünün teslim ve döngü süresi, kanban ekipleri için önemli olmaktadır. Bu durum kanban ekiplerinin başarısını göstermektedir.



Şekil 1.2 Kanban Panosu

Kanban, yalın ve diğer süreçler bir zamanlama sistemi olarak belirtilmektedir. Bir Kanban sürecinde, sürecin başından sonuna kadar ilerleyen Kanban adlı fiziksel (veya sanal) kartlar bulunmaktadır. Amaç, Kanban'ın sürekli bir akışını sağlamanmasıdır. Böylece işlem sonunda envanter gerektiğinden, sadece başlangıçta bu miktarın yaratılması sağlanmaktadır. Yazılım geliştirme için kullanıldığında Kanban, üretim sürecindeki farklı aşamaları temsil etmek için yazılım geliştirme yaşam döngüsü aşamalarını kullanmaktadır. Bu durum özelliklerin akışını (Kanban kartları ile temsil edilir) kontrol etmek ve yönetmektir. Böylece sürece giren özelliklerin sayısı tamamlananlarla eşleşmektedir. Kanban mutlaka yinelemeli olmayan çevik bir metodoloji olarak belirtilmektedir (İnt.Kyn.3).

Scrum gibi işlemler proje yaşam döngüsünü küçük ölçekte taklit eden, her yineleme için ayrı bir başlangıç ve bitişe sahip kısa yinelemelere sahip olmaktadır. Kanban, yazılımın büyük bir geliştirme döngüsünde geliştirilmesine izin vermektedir. Buna rağmen Kanban çevik bir yaklaşımın örneğidir, çünkü çevik manifestonun arkasındaki ilkelerin on ikisini yerine getirmektedir. Kanban'ın ardındaki prensip sınırlı olduğu

belirtilmektedir. Üretilen iş hiçbir yineleme olmadan bir Kanban projesinde bireysel iş kalemleri için belirlenmiş başlangıç veya bitiş noktaları bulunmamaktadır. Her biri birbirinden bağımsız olarak başlayabilir ve bitebilir ve iş öğelerinin bu konuda önceden belirlenmiş bir süresi olmamaktadır. Bunun yerine, yaşam döngüsünün her aşaması herhangi bir zamanda iş için sınırlı bir kapasiteye sahip olarak kabul edilmektedir. Öncelikli ve başlatılmamış gereksinimler listesinden küçük bir iş öğesi yaratılır ve daha sonra genellikle bazı gereksinimlerin detaylandırılması ile geliştirme sürecine başlamaktadır. Bir iş öğesinin, bir kapasite açılıncaya kadar bir sonraki aşamaya geçmesine izin verilmemektedir. Geliştiriciler, herhangi bir anda etkin olan görev sayısını kontrol ederek, çevik ilkelerin uygulanması için fırsat veren genel projeye adım adım yaklaşmaktadır (İnt.Kyn.3).

2.1.4 Yalın Yazılım Geliştirme

Yalın Yazılım Gelişme (Lean Software Development), projede israfını azaltmaya, ürün sahibi için daha az kaynak ile daha çok değer üretmeye ve sürekli gelişmeye odaklanan bir yaklaşım biçimi olarak tanımlanmaktadır. Yalın geliştirme israfı en aza indirmek ve müşteriye değeri en üst seviyeye çıkarmak için üretim hattını optimize etmenin bir yolu olarak ortaya çıkmaktadır. Bu iki hedef aynı zamanda tekrarlanabilir bir süreci izleyen özel kalite standartlarını gerektiren ve yapılması için bir grup uzman çalışanın işbirliğine dayanan yazılım geliştirme aktiviteleri olmaktadır. Fabrikalardaki üretim ve yazılım geliştirme arasında ciddi farklılıklar bulunmaktadır. Yalın prensiplerini uygulamak değer, israf ve diğer önemli yalın kavramlarının nasıl tanımlandığı konuları bir bakış açısı gerektirmektedir. Yazılım geliştirmede yalın prensiplerinin uygulanması, yalın üretim prensiplerine yakın nitelikte yedi ilke ile özetlenebilir (İnt.Kyn.4):

- İsrafın önlenmesi: Müşteriye değer katmayan her şey devre dışı bırakılmaktadır. Gereksiz kod veya işlevsellik müşteriye iletim zamanını geciktirir, geri bildirim döngülerini yavaşlatmaktadır. Tamamlanması zor uzun sürecek işler sisteme gereksiz karmaşıklık katmaktadır.
- Kaliteli üretim: Öncelikli hedef hataları bulmak değil, hataları yaratmaktan kaçınmak gerekmektedir. Bu mümkün görünmüyorsa bir çalışma başlatıp

hataları giderip ilerlemeyi bu şekilde gerçekleştirmek daha verimli olacağı belirtilmektedir. Kod geliştirmeyi basit tutmak ve daha az kodla yazılımı geliştirmeye çalışılmaktadır. Kodun sık sık düzenlenmesi de kaliteli üretim açısından önemli olmaktadır.

- Bilgi oluşturma: Planlama yararlıdır, fakat öğrenmenin önemli olduğu belirtilmektedir. Ekip üyelerinin gerçekte ne istediklerini keşfetmelerine ve hareket etmelerine yardımcı olacak stratejileri sunmak gerekmektedir. Bir takımın ne yapmakta olduğunu düzenli olarak yansıtması ve ardından gelecek stratejilerini geliştirmek için harekete geçmesi de önemli görülmektedir. Bu ilke yalın takımlarını öğrenmeyi doğru bir şekilde belgelemek ve korumak için altyapı sağlamaya teşvik etmektedir.
- Mümkün olduğunca geç karar verin: Uzun zaman öncesinden fazla detay içeren planlar yapmamak, iş gereksinimlerini tam olarak anlamadan fikir ve projelerle ilgili sözler verilmemesi gerekmektedir. Her türlü önemli kararla ilgili bilgileri sürekli olarak toplamak ve analiz etmek ve gerekirse kararın değişmesine zemin hazırlamak anlamına gelmektedir.
- Hızlı Teslimat: İş mümkün olduğunca hızlı sunulması gerekmektedir. Basit bir çözüm oluşturularak müşterilere sunulması önemli görülmektedir. Müşteri geri bildirimlerine göre kademeli olarak geliştirmeler yapılmaktadır. Bu özellikle yazılım açısından önemli olmaktadır. Çünkü hız, inanılmaz bir rekabet avantajı olarak görülmektedir.
- Kişilere Saygı Gösterme: Birbirine saygı duyan kişiler ve bu anlayışın sağlanması takımı güçlendirmektedir. Yalın geliştirme, ekiplerin işleyişlerini, nasıl iletişim kurdukları, aralarındaki problemleri nasıl ele aldıkları, yeni ekip üyelerine yaklaşımı ve daha fazlası ile ilgilenmektedir. Yalın yazılım geliştirme takımı proaktif, etkili iletişim kurulması, sağlıklı bir iletişimin teşvik edilmesi ve işlerini iyi yapmaları için birbirlerine güç vererek insanlara saygıyı teşvik etmesi gerekmektedir.
- Bütünün Optimize Edilmesi: Bir problemin çözümünde etkili olmak istiyorsanız, daha büyük resme bakılması gerekmektedir. Bireysel projelerin desteklediği üst düzey iş süreçlerinin anlaşılması önemli görülmektedir. İlişkili sistemlerin programlarını yönetmek gerekebilir, böylece paydaşlara eksiksiz bir ürün

sunulmasının önemi artmaktadır.

2.1.5 Dinamik Sistemler Geliştirme Yöntemi

Dinamik Sistem Geliştirme Yöntemi (Dynamic Systems Development Methodology) hızlı uygulama geliştirmek amacıyla sonuçlara hızlı ve efektif olarak ulaşmayı sağlamak, zaman, maliyet, risk ve kaliteyi kontrol altında tutmak, kaliteyi maksimum oranda sağlamak amacıyla geliştirilen yazılım geliştirme çerçevesidir. Çevik yaklaşımından çok proje yönetimi yaklaşımı olarak gösterilmektedir. Dinamik Sistem Geliştirme Metodu pek çok yönden XP ve ASD'ye benzemektedir. Yaklaşımın temel prensipleri:

- Aktif kullanıcı katılımı
- Takımlar karar almak için yetkilendirilir
- Ürünlerin sık teslimi
- İş amacı için zindelik
- İteratif ve artımlı teslim
- Geri dönülebilir değişiklikler
- Yüksek seviye bazlı gereksinimler
- Entegre edilmiş olanı test etme
- İşbirlikçi yaklaşım

şeklinde ifade edilmektedir.

DSDM bir çerçeve olduğu kadar bir yöntem olarak gösterilmemektedir. 1990'lı yılların başında ortaya çıkan DSDM aslında hızlı uygulama geliştirme uygulamalarının resmi duruma getirilmesidir. DSDM yaşam döngüsünün altı aşaması bulunmaktadır. Bunlar: Ön proje, Fizibilite Etüdü, İş Etüdü, İşlevsel Model İterasyonu, Tasarım ve Geliştirme İterasyonu, Uygulama ve Proje olarak nitelendirilir. Proje öncesi aşama, projenin başlamaya hazır olduğunu, fonların mevcut olduğunu ve başarılı bir projeye başlamak için her şeyin mevcut olduğunu belirlemektedir. Fizibilite çalışması, fizibilite çalışmasının birkaç haftadan fazla olmamak üzere kısa olması gerektiğini

vurgulamaktadır. Olağan fizibilite faaliyetleriyle birlikte, bu aşama DSDM'nin proje için doğru yaklaşım olup olmadığını belirlemesi gerekliliği belirtilmektedir. İş etüdü aşaması "güçlü bir şekilde işbirliğine dayanır, bilgisini hızlı bir şekilde toplayabilen ve kalkınmanın öncelikleriyle ilgili fikir birliği kazanabilen, bilgili ve güçlendirilmiş personelin katıldığı bir dizi atölye çalışması şeklinde belirtilmektedir". Bu aşamanın sonucu, sistemden etkilenen kullanıcıları, pazarları ve iş süreçlerini tanımlayan iş alanı tanımı olarak tanımlanmaktadır. İşlevsel model yinelemesi, işletme çalışmasında tanımlanan üst düzey gereksinimlere dayanmayı amaçlamaktadır. DSDM çerçevesi, riske dayalı bir dizi prototip oluşturarak çalışır ve bu prototipleri komple sisteme dönüştürmektedir. Bu aşama tasarım ve yapım aşamasıyla ortak bir sürece sahip olmaktadır. Neyin üretileceği belirlenmel, nasıl ve ne zaman yapılacağı kabul edilmesi önemli olmaktadır. Ürünü yaratmak, doğru üretildiğinden emin olunmasıdır. Belgelerin incelenmesi, bir prototip göstererek sistemin bir bölümünü sınamak önemli olmaktadır (Cohen *et al.* 2004).

2.1.6 Adaptif Yazılım Geliştirme

Adaptif Yazılım Geliştirme (Adaptive Software Development), hızlı ve fazla değişen internet ekonomisi için geliştirilmiş bir yazılım geliştirme yöntemi olarak belirtilmektedir. Hızlı ve aşırı değişimli projeler, tahmin edilemez ve geleneksel yöntemlerle işlenemeyecek kadar karmaşık görülmektedir. Geliştirme takımının adaptif olması, yeni değişimlere çabuk cevap vermesi gerekmektedir. Sürekli prototip geliştirerek iteratif ve artırımsal geliştirmeyi teşvik etmektedir. Yöntemde "kaosun sınırında dengeleme" mantığı ile "yeterli yok gösterme ile projelerin kaosa düşmesi engellenebilir" düşüncesi öne çıkmaktadır. Bu yöntemde proje üç bölümde değerlendirilmektedir. Bunlar tahmin etmek, işbirliği yapmak ve öğrenme. Belirsizliğin az olduğu anlarda yapılan plan yerine, bu yöntemde tahmin yapılmaktadır. Birlikte çalışma hızla değişen sistem geliştirmede oldukça önemli görülmektedir. Hatalardan bilgi üretme ve geliştirme aşamasındaki ihtiyaç değişimlerini sağlamak öğrenme aşaması olarak belirtilmektedir. Adaptif Yazılım Geliştirme proje başlama, adaptif çevrim planlama, eşzamanlı özellik geliştirme, kalite gözden geçirme ve final kalite güvence-sürüm yayınlama olmak üzere beş genel adımdan oluşmaktadır. Kalite gözden

geçirme ile çevrim planlama arasında öğrenme geri beslemesi bulunmaktadır (Highsmith 2000).

Adaptif Yazılım Geliştirme James A. Highsmith tarafından geliştirilmektedir. ASD'nin ilkelerinin çoğu Highsmith'in yinelemeli geliştirme yöntemleri konusundaki önceki araştırmalarından kaynaklanmaktadır. ASD temel olarak karmaşık, büyük sistemler geliştirmedeki sorunlara odaklanmaktadır. Metot sürekli prototipleme ile artan yinelemeli gelişimi kuvvetle teşvik etmektedir. Temel olarak ASD "kaosun kenarında dengeleme" ile ilgili olmaktadır. Amacı projelerin kaosa düşmesini engellemek için yeterli rehberliğe sahip bir çerçeve sağlamaktadır (Abrahamsson *et al.* 2017).

2.1.7 Özellik Odaklı Geliştirme

Özellik Odaklı Geliştirme (Feature Driven Development), gelişmekte olan sistemler için çevik ve uyarlanabilir bir yaklaşım olarak belirtilmektedir. FDD yaklaşımı tüm yazılım geliştirme sürecini kapsamamaktadır, bunun yerine tasarım ve yapım aşamalarına odaklanmaktadır. Bir yazılım geliştirme projesinin diğer faaliyetleri ile çalışmak üzere tasarlanmıştır ve kullanılmaları için herhangi bir özel işlem modeli gerektirmemektedir. FDD yaklaşımı endüstride etkili olduğu bilinen en iyi uygulamalarla yinelemeli gelişimi somutlaştırmaktadır. Süreç boyunca kalite unsurlarını vurgular, sık ve somut teslimatları ve projenin ilerlemesinin doğru izlenmesini içermektedir (Palmer and Felsing 2002).

FDD 1997 yılında Jeff De Luca tarafından öne sürülmektedir. Daha sonra Peter Coad ile birlikte Özellik Odaklı Geliştirme modeli olarak geliştirilmektedir. FDD model odaklı, kısa yinelemeli bir yaklaşım olmaktadır. Genel bir model şekli oluşturmak ve özelliklerin tanımlanması ile başlamaktadır. Özellikler daha sonra iş paketlerinde gruplandırılmaktadır. Bir iş paketi tek bir yineleme içinde bitebilir ve aslında müşterinin birlikte oynayabileceği çalışma yazılımını temsil etmektedir. FDD aşağıdaki sekiz uygulamayı kullanarak geliştirme sürecinin geri kalanını özellik teslimatı etrafında tasarlanmaktadır (Stankovic *et al.* 2013):

- Etki alanı nesne modellemesi

- Özelliğe göre geliştirme
- Bileşen ve sınıf sahipliği
- Özellik takımları
- Denetimler
- Yapılandırma yönetimi
- Düzenli oluşturur
- İlerlemenin ve sonuçların görünürlüğü.

2.1.8 Test Odaklı Geliştirme

Test Odaklı Geliştirme (Test Driven Development) yazılım geliştirilirken, geliştirilecek yazılımla ilgili öncelikle test senaryolarının yazıldığı ve genellikle otomatize edildiği yazılım geliştirme yaklaşımıdır. Klasik yazılım geliştirmede önce yazılım tasarlanır sonra kodlanır ve en son test işlemleri yapılmaktadır. Yazılım sona erdikten sonra yapılan test işlemlerinin kapsamı yetersiz olacağından kodun belli kısımları test edilemez ve oluşabilecek hataların tespiti zorlaşmaktır. Bu durumda hatalar sistem kullanıcıları tarafından bulunmaktadır. Bu da istenilen bir durum olmamaktadır. Bunun gibi problemleri çözmek, hataların oluşmasını engellemek için Test Odaklı Geliştirme yaklaşımı geliştirilmektedir. TDD de programı kodlamadan önce yazılımı test edecek şekilde test sınıflarını oluşturularak yazılım işlemine başlanmaktadır. Herhangi bir kod yazmadan testler oluşturulmaktadır. Daha sonrasında çalıştırılır ve yeni eklenen testlerinde çalışması beklenmektedir. Eğer çalışmıyorsa değişiklikler yapılır ve test çalışır duruma getirilirmesi gerekmektedir. Burada gerekli sınıfların en basit şekilde oluşturulmaları önem taşımaktadır. Test edilen sınıf metotları ilk başta boş değeri geri döndürülecek şekilde oluşturulup daha sonra kullanıma göre gereken değişiklikler yapılarak en basit şekilde oluşturulmaktadır. TDD doğru uygulandığında son satırına kadar test edilmiş bir yazılım uygulaması meydana gelmektedir.

Test Odaklı Geliştirme, test edilen kodu yazmadan önce test yazmayı içeren bir geliştirme uygulamasıdır. TDD, çok kısa gelişim döngülerine ve fonksiyonel kod yazmadan, yeniden düzenleme ve sürekli entegrasyondan önce otomatik test yazma konusunda çevik uygulamalara dayanan evrimsel bir yaklaşım olarak görülmektedir.

Her geliştirme döngüsü üç adımdan oluşmaktadır. Bunlar birim testi, uygulama ve yeniden düzenleme olarak belirtilmektedir. Her yinelemenin ilk adımlarında yazılan testlerin tasarım ve uygulamayı yönlendirdiği için TDD olarak adlandırılmaktadır. Bir kod tabanının boyutu arttıkça, yeniden düzenleme adımında daha fazla dikkat ve efor gösterilmektedir. Önceden belirlenmiş olmasa da, tasarım sürekli olarak gelişmekte ve sürekli gözden geçirilmektedir. Bu tanecikli bir seviyede ortaya çıkan tasarımdır ve Test Odaklı Gelişmenin en önemli yan ürünlerinden biri olarak belirtilmektedir (Okolnychyi and Fögen 2016).

Beck (2002) tarafından açıklandığı gibi Test Odaklı Geliştirme, çevik geliştirme tekniklerinden biri olarak belirtilmektedir. Test yazma, başarılı testi mümkün olduğunca çabuk geçen kodları yazma işlemi olmasıdır. Kodu yeniden düzenlemek, bazı problemleri gösteren kod bölümleri gözden geçirmek ve düzenlemektir. Test yazmanın temel nedeni, bir kaynak kodunda değişiklik yapılması talebiyle karşı karşıya kalındığında ilgili durumları kontrol edilme durumu olarak belirtilmektedir. Bu değişiklik geliştirme sırasında kodun yeniden yapılandırılması veya yazılım gereksinimlerinin değiştirilmesi ile ilgili olmaktadır (Nečas 2011).

2.1.9 Davranış Odaklı Geliştirme

Davranış Odaklı Geliştirmede (Behavior Driven Development), yazılan kodları test etmek yerine yazılım uygulamasının kullanıcı davranışlarının test edilmesini benimseyen bir yaklaşımdır. Bu yaklaşımda tüm gereksinimler müşteri davranışı olarak ele alınmaktadır. Dan North ve ekibinin çalışmarı soncunda ortaya çıkardığı yaklaşımda test kelimesi yerine behaviour kelimesi kullanılmaktadır. Maaliyetli olan TDD kavramını daha az maliyet ile nasıl yapılır konusu üzerine ortaya çıkan bir kavram olduğu belirtilmektedir. BDD kaliteli kod üretiminin sağlanması üzerine yoğunlaşmaktadır. Yazılımcıların test üzerine kafa yormak istememeleri iş birimlerinin teknik ortama uzak olmalarından dolayı iletişim eksiklikleri kalitesiz kod yazımına sebep olmaktadır. Bu durumda BDD yaratılan ortak bir dil kullanarak bunu çözmeyi hedeflemektedir. Uygulanan çalışma metodolojisine göre gereksinimlerin belli bir formatta yazılıp uygulama geliştirme ekiplerine iletmektedir. Gherkin dilinde yazılan

gereksinimler yazılımcıların yazmış olduğu BDD birim testlerini otomatik olarak yapmaktadır. BDD yazılımcıların birim test süreçleri kolaylaştırmakta ve test uzmanlarının fonksiyonel test süreleri azaltmaktadır.

BDD, sistemin davranışını tüm ayrıntı düzeylerinde paydaşları açısından tanımlar ve diğer çevik yaklaşımlardan farklılık göstermektedir. BDD, sistemin davranışının bu şekilde açıklanmasına odaklanmanın daha iyi iletişim sağladığını savunmaktadır. Diğer çevik geliştirme yöntemleriyle karşılaştırıldığında paydaşlar için daha büyük bir ürün ürettiğini garanti etmektedir. Davranış spesifikasyonlarını dikkate almak, yazılım mühendislerinin her davranış hakkında daha net düşünmelerini sağlamaktadır. Bunun birlikte bir sınıf veya metot tarafından test edilmelerine daha az güven ve daha iyi çalıştırılabilir dokümantasyona sahip olmalarını sağlamaktadır (Okolnychyi and Fögen 2016).

BDD, domain odaklı tasarım dil kavramını benimsemektedir. Başarılı bir yazılım projesi, iyi bir iletişim gerektirir ve bu da ortak bir dile dayanmaktadır. Domain uzmanları domain dilleri açısından düşünmektedir. Yazılım geliştirme alanındaki kavramları kullanarak geliştiriciler de aynı şeyi yapmaktadır. Analistler ve geliştiriciler bu alanlar arasında çeviri yaparak domain kavramlarını tasarıma eşlemektedir. Bununla birlikte bu çeviride bilgi kaybedilebilir, bu da farklı insanların farklı kavram yorumlara sahip olmalarına neden olmaktadır. Birçok yazılım projesi domain uzmanları ve ekipteki programcılar arasında düşük kaliteli iletişimden şikâyetçi olmaktadır. BDD araçlarıyla yazılmış testler genellikle proje paydaşlarının anlayabileceği bir dil kullanılarak tanımlanmaktadır (Okolnychyi and Fögen 2016).

2.1.9.1 Davranış Odaklı Geliştirme İlkeleri

BDD'yi yazılım geliştirme için kullanmanın asıl amacı müşterilere değer sağlamaktır ve bu anlayışa üç temel ilkeye dayanmaktadır (Nečas 2011):

- Müşteriler için değerli olan bir yazılımı sağlamak için gereken çabayı gösterilmektedir. Bu çaba yazılımı analiz etmek, planlamak ve tasarlamak için

yeterli zaman harcanmasıdır.

- Paydaş kelimesi yalnızca yazılımı alan müşteriye değil, onu kullanması gereken herkese atıfta bulunmaktadır. Ne zaman paydaş değeri getirmeyen veya sağlama yeteneği arttırmayan bir şey yapıldığında, bunu yapmayı bırakmalı ve bu niteliklere sahip bir şeye odaklanmak gerekmektedir.
- Herhangi bir ayrıntı düzeyinde davranışı tanımlamak için aynı dilbilimsel yapılar kullanılmaktadır.

2.1.10 Davranış Odaklı Geliştirme İlgili Araştırmalar

Çıltık vd. (2014) Bankacılık Alanında Doğal Dil İşleme Destekli Davranış Odaklı Geliştirme isimli çalışmalarında, şelale ve çevik yaklaşımları baz alarak yazılım geliştirmede ürününe giden yolu kısaltmak, süreci kaliteli bir şekilde gerçekleştirmek ve ürün kullanıcılarının ihtiyaçlarını odak noktasında tutan yöntemleri benimseyerek bazı çalışmalar yürüttükklerini belirtmektedir. Bu kapsamda bir yaklaşım model geliştirdiklerini söylemektedirler. Bu modelde gereksinimlerin doğrudan yazılım geliştirme sürecini tetiklemesini sorun aşamasından çözüm aşamasına geçişteki karmaşıklığı ve yanlış anlaşılmaları minimuma indirmeyi hedeflemektedirler. Çalışmalarında ilgili alan uzmanlarının bankacılık alanının doğal dilini kullanarak gereksinimleri geliştirme sürecine girdi olarak sağlamayı hedeflemektedirler. Geliştirilen model; proje paydaşları, senaryo oluşturma, bankacılık alanına özgü oluşturulan sözlük ile doğal dil işleme, davranış dönüştürücü ve kod yapısı-test tanımları bölümlerinden oluşmaktadır. Çalışma sonucunda BDD ile ilkelerinin otomatize edilebileceğine değinmektedirler. Gereksinimlerin ya da kabul test senaryolarının tamamına davranış ismini verdikleri Davranış Odaklı Geliştirme yaklaşımına ek olarak önerilen doğal dil işlemi desteğiyle birlikte yazılımın daha kaliteli ve planlanan zaman ve bütçe içinde teslim edileceğini öngörmektedirler.

Akyol ve Gümüşkaya (2016) Çevik Yazılım Geliştirmede BDD/TDD Yöntemlerinin ve Yazılım Kalite Araçlarının Kullanılması: Bir Yazılım Mühendisliği Dersindeki Tecrübe isimli çalışmalarında BDD ve TDD yöntemlerini SaaS uygulamaları geliştirmede birlikte kullanmaktadırlar. Çalışma Gediz Üniversitesi Bilgisayar Mühendisliği Bölümü

Yazılım Mühendisliği dersinde 2013-2015 yılları arasında uyguladığını belirtmektedirler. İlk 2 yıl Ruby/Rails son yılda ise diğer dillerde projeler geliştirdiklerini söylemektedirler. Çalışmada tümleşik bir çevik yazılım geliştirme çerçevesinde, günümüzde kullanılabilecek yazılım test araçları karşılaştırması yapmaktadırlar. Çalışma kapsamında gerçekleştirecekleri dört farklı projede Ruby/Rails, Java EE/Spring, C#/ASP.NET ve PHP ile Zend, Codeigniter, Laravel teknolojileri ve platformları kullandıklarını belirtmektedirler. Microsoft Azure ve Heroku bulut servis sağlayıcılarında çalışan SaaS uygulamaları geliştirmede kullanılan BDD ve TDD test ve kalite araçlarıyla elde edilen tecrübeyi aktarmaktadırlar. Çalışma sonrasında fonksiyonel otomatik testleri geliştirmede kullanılan Cucumber, Ruby programlama dili BDD'yi güçlü olarak sunan ilk ortam olduğunu belirtmektedirler. Çalışmada Ruby dili ve ortamı ile geliştirilmiş projelerin ve bu ortamdaki otomatik test araçları ile SaaS uygulamasının çok hızlı geliştirilip Heroku ortamına taşındığı görülmektedir. Uygulamalarda Ruby dilinin 3-4 haftada öğretilmesinin mümkün olduğu belirtilmektedir. Aynı zamanda Ruby dilini öğrenciler tarafından da kendilerince öğrenilebileceği söylenmektedir.

Oruç ve Ovatman (2016) BDD Kullanarak Web Servislerinin Test Edilmesi isimli çalışmalarında, bir web servisinin doğru çalışabilmesi için test edilmeli ve onaylanması gerektiğini önermektedirler. Web servislerinin test edilmesi yeni zorluklara neden olabilmektedir. Bundan dolayı BDD web servislerinin testinde uygulanabilir olacağını öne sürmektedirler. Web servislerinde test senaryoları tanımlamak üzere Gherkin dilini kullanarak test senaryolarını JMeter test senaryolarına dönüştürmek için bir araç geliştirilmektedir. Geliştirilen araç dinamik olarak test komut dosyalarını oluşturmada Gherkin dili ile BDD yaklaşımını kullanılmaktadır. Çalışmada geliştirilen aracın ilk artısı, Gherkin, alana özel bir dil olduğundan yazılım bilgisi olmayan herhangi bir test uzmanı web hizmeti testlerini oluşturabilir ve çalıştırabilir oldığunu söylemektedirler. İkincisi, JMeter test aracı kullanımından dolayı test geliştiricilerinin birim testlerini manuel olarak yazması gerekli değildir sonuçlarına ulaşmaktadırlar.

Okolnychyi ve Fögen (2016) Davranış Odaklı Geliştirme için Araçlar Çalışması isimli araştırmalarının amacı BDD'de uygulanabilecek ilgili araçları belirlemek, bunları

değerlendirmek ve karşılaştırmak için kriterler geliştirmektir. Bununla birlikte bir araçta başarı elde etmek için hangi araç setinin seçileceğine ilişkin kılavuzlar sağlamaktır. Bu çalışmada kullanılan araştırma yaklaşımı, ilgili literatür taraması ve Java, Groovy, Scala tabanlı diller için mevcut BDD araç setlerinin analiz edilmesi olarak belirtilmektedir. Çalışmada Cucumber, Concordion, Spoct, JBehave ve easyb araçları analiz edilmiş ve karşılaştırılmaktadır. Proje paydaşları tarafından okunabilirliği, otomatik kabul testleri desteği, Web uygulamalarını test etme olanağı, kullanıcı hikâyeleri ve senaryoların düz metin açıklaması olması yönlerinden bu araçların BDD de kullanılabilirlik desteğinin güçlü olduğu belirtilmektedir. Gerçekleştirilen karşılaştırmanın sonuçları, BDD'yi Java, Groovy, Scala tabanlı dillerle mümkün kılan analiz araçlarıyla ana BDD kavramlarının güçlü bir desteğinin olduğunu göstermektedir. Analiz edilen tüm araçlar, farklı diğer dillerle de entegrasyona sahip olduğu belirtilmektedir.

Rocha Silva vd. (2016) Ontolojik Bir Test İle Çok Yapılı Test İçin Bir Yaklaşım: Davranış Odaklı Gelişme Perspektifi isimli çalışmalarında etkileşimli sistemlerin geliştirme süreci boyunca yapıların otomatik olarak değerlendirilmesini desteklemek için BDD temelli bir yaklaşım önermektedir. Çalışmada benzer kavramları paylaşan çoklu yapılar üzerinde çalışabilecek testleri belirlemek için bir ontoloji kullandığını belirtmişler. Görev modelleri, prototipler ve son kullanıcı arayüzlerini test eden bir örnek olay incelemesi, bu yaklaşımın tasarım süreci olduğuna değinmişler. Bu aşamalardan itibaren uygulanabilirliğini göstermek, gereksinimlerin sürekli bir kalite güvencesini sağlamak ve müşterilere ve geliştirme ekiplerine potansiyel sorunları tanımlamaları sağlanmıştır. Farklı yapılar tarafından paylaşılan ortak ontolojik kavramların temeli olarak proje boyunca izlenebilirliği ve test entegrasyonunu desteklemek için bir ontoloji sağlanmış. Her bir kullanıcı arayüzü öğesinin yanıtlayabildiği davranışları temsil ederken, ontoloji ayrıca kullanıcı arayüzü tasarımı için birden fazla çözümü genişletmeye izin vermiş. Birden fazla tasarım seçeneğiyle çalışırken ve daha karmaşık görev modellerini değiştirirken ortaya çıkabilecek olası sorunları ve tutarsızlıkları belirlemek için çalışmaların devam edeceğine değinmişler.

Subramanian vd. (2017) Davranış Odaklı Test Otomasyon Çerçevesi isimli çalışmalarında Behavior Driven Test Automation Framework modelini sunmuşlar. Bu

model kapsamında web uygulaması test otomasyonu geliştirilmesi ve uygulanmasındaki zorluklarla birlikte maliyet kaynaklarına ait problemler için bir çözüm önerisi sunmuşlardır. Önerilen çözüm Behavior Driven Test Automation Framework'ü söz edilen tüm kısıtlamaları ele alabilir, çünkü çerçeve çok fazla maliyet gerektirmeden uygulanabilir ve korunabilir olduğunu öne sürmektedirler. Aynı zamanda üç aşamalı tüm farklı katmanların test otomasyonunu destekleyebilir. Yazılım sisteminin test kapsamını, kalitesini ve güvenilirliğini büyük ölçüde iyileştirmenin ardındaki itici faktör olarak işlev görecek mimari sistem olacağını öne sürmektedirler. Çalışmada yönetim ve proje ekiplerinin test otomasyon sürecine katılmasını sağlar ve test raporlamada görünürlüğü artırır kanısına ulaşmışlar. Bununla birlikte yazılım geliştiriciler, test uzmanları, iş analistleri ve diğer paydaşları içeren proje ekipleri arasındaki iletişimi geliştiğine değinmişler. Çok katmanlı mimari kullanılarak geliştirilen web uygulamaları için güçlü test otomasyon desteğini içermektedir. Bununla birlikte basit ve karmaşık yazılım uygulamaları için web uygulamasının grafiksel kullanıcı arayüzü katmanının, uygulama programlama arayüzü katmanının ve database katmanının otomatik olarak test edilmesini içeren uçtan uca bir test otomasyonu uygulanabileceği sonucuna ulaşmışlar.

Carter (2017) Bhive: Davranış Odaklı Geliştirme buluşmalarında Davranış-Metodu isimli tez çalışmasında çevik yaklaşımlardan yola çıkarak BDD yaklaşımı merkezli BHive yaklaşımı (modeli) geliştirmiştir. Bu modelde "Given", "When" ve "Then" ifadelerini "Precondition", "Command" ve "Postcondition" yapılarına eşleyerek BDD senaryolarının temel dayanaklarına dayandırılmış. Bu eşlemenin Davranış-Metodu gösteriminin oluşturulmasına izin verdiğini ve böyle bir modelin sistem davranışını araştırmak ve gereklilikler ve test planlarındaki boşlukları ortaya çıkarmak için yararlı olduğunu belirtmişler. BDD'yi BHive aracılığıyla Davranış-Metodu ile birleştirilmiş. Çalışmada BHive kodunun Gherkin özellik dosyasından üretildiği belirtilmektedir. Çalışılan örnek Gherkin'e özgü olsa da BHive genel bir yaklaşımdır ve BDD'nin herhangi bir varyasyon için uygulanabilir olduğu belirtilmektedir. BHive'ın davranış odaklı bir yaklaşım kullanılarak geliştirilen bir sisteme etkili bir şekilde uygulanmasına ilişkin bir dizi kural olduğunu belirtilmektedir. Çalışmada IPTV uygulamasının DataLayer'in geliştirilmesine BHive yaklaşımı uygulanmış. Uygulama önemli derecede

karmaşıklık içerdiği görülmektedir. Bununla birlikte BHive'in projeyle ve onun geliştirmeye yönelik yinelemeli yaklaşımıyla uyumlu olduğu gösterilmiştir.

De Souza vd. (2017) Davranış Odaklı Gelişmeyi Yazılım İçin Scrum İle Birleştirmek Eğitim Alanında Geliştirme isimli çalışmalarında, EAMS-CBALM yazılımını Sao Carlos Federal Üniversitesi Tıp Bölümü, Western Oregon Üniversitesi Eğitim Araştırma Enstitüsü, Sırio Libanes Hastanesi ve TokenLab yazılım evi birlikteliğinde gerçekleştirilmiştir. Çalışmalarında bazı EAMS-CBALM bileşenlerini yeniden tasarlamak için BDD ve Scrum yaklaşımlarının birlikte kullanıldığı bir vaka çalışmasını ele almaktadır. Çalışmada ürün sahibi ile geliştirme ekibi arasındaki iletişimin, sistem gereksinimlerini açıkça tanımlamak ve otomatik olarak test süitleri oluşturmak için BDD'yi bir iletişim platformu olarak kullanarak geliştirilebileceğini öne sürmektedirler. Çalışmada EAMS-CBALM için bir örnek olay incelemesi yapılmış. Sonrasında EAMS-CBALM gelişimine katılan aynı ürün sahibi ile düzenli toplantılar yapılarak, BDD sürecinin ardından gereksinim için yeni bir kaynak kodu oluşturulmuştur. Oluşturulan kaynak kodu ürün sahibine sunularak, TokenLab yazılım evi tarafından geliştirilen kodlarla karşılaştırılmıştır. Bu vaka çalışmasında BDD sürecinin EAMS-CBALM gelişiminde kullanılan Scrum yöntemine göre ürün sahibi ile geliştiriciler arasındaki iletişimi geliştirdiğini göstermiştir. Bunun birlikte bir dilin BDD senaryoları ve kabul testleriyle birlikte kullanımı ve ürün sahibinin gelişim süreci boyunca geliştiricileri ile takip etmesini ve uygun şekilde iletişim kurmasını sağlamıştır. Ayrıca bu vaka çalışması BDD'nin yazılım yaşam döngüsünün gereksinim toplama, analiz ve tasarım aşamaları için daha fazla desteğe ihtiyaç duyduğunu göstermiştir.

Lucassen vd. (2017) Otomatik Kabul Testleriyle Davranış Odaklı Gereksinimlerinin İzlenilebilirliği isimli çalışmalarında çevik yazılım geliştirme süreci olan BDD üzerine kurulu olan izlenebilirliği sağlamaya çalışan davranış odaklı izlenebilirlik yöntemini önermektedirler. Otomatik kabul testlerinin çalışma zamanı izlemesini işleyerek kullanıcı hikâyesi gereksinimlerinin kaynak koduna nasıl izleneceğini ve BDT matrisinin normalleştirilmiş sürümlerinin kullanıcıya özel hikâye ve yöntemleri vurguladığını göstermişlerdir. Gelecekteki çalışmalarda prototiplerinin endüstri projeleri üzerinde test ederek ve uygulayıcıları dâhil ederek BDT çıktısının doğruluğunu

değerlendirmeyi amaçlamışlardır. Özellikle NBDT matrisindeki sınıfların ve yöntemlerin hataların giderilmesi gibi yazılım mühendisliği işleri için uygun olup olmadığını anlamak istemişlerdir. Pilot noktada hangi kodun herhangi bir zamanda hangi gereksinimle ilgili olduğunu doğru bir şekilde belirleyebildiğini göstermek için BDT'yi uygulamışlardır. Bir başka ilginç yön ise otomatik değişim etki analizi: Yeni bir kullanıcı hikâyesi eklemenin etkisini tanımlamak için BDT'nin kullanılmasıdır. En son olarak modernizasyon yöntemlerinin bir parçası olarak en gelişmiş özellik konum tekniklerini denemeyi planlamışlardır.

Dimanidis vd. (2018) Otomatik Web API Geliştirmesi için Doğal Dilde Bir Yaklaşım isimli çalışmalarında çevik yaklaşımlardan yola çıkarak BDD merkezli Doğal Dil Odakli Geliştirme modelini (yaklaşımını) geliştirmişlerdir. Web servisleri gereksinimlerinin ortaya çıkarılması ve belirtilmesi aşamalarını kolaylaştırmak için BDD kullanan bu yaklaşımı önermektedirler. Ayrıca bahsedilen bu geliştirme yaklaşımlarınında köprülemeyi amaçlayan bir yazılım aracı olan gherkin2OAS'ı sunmaktadırlar. Önerilen metodoloji ve araç sayesinde, uygun işlevselliği sağlarken hızlı bir şekilde web servisleri hizmetleri tasarlayıp oluşturabilir görüşüne varmışlardır. Doğal dilde ifade edilen gereksinimlerin kaynak koduna dönüştürülmesinin geliştirici verimliliğini artırabileceği gösterilmektedir. Çalışmanın odak noktası iletişim engellerini aşmak için gereklilikleri spesifikasyonlara dönüştürmek için bir yöntem ve araç sağlamak olsa da bu çalışmanın sonuçları web hizmetinin kaynak kodunu OAS şartnamesinden dağıtarak nicel bir şekilde yardımcı olabilir görüşüne varmışlar.

Nečas (2011) Bir Kalite Güvence Aracı Olarak BDD Şartnamesi isimli tez çalışmasının amacı Davranış Odaklı Geliştirme yaklaşımını tanıtmaktır. Bununla birlikte çalışmada kullanıcı hikâyelerini müşteriler ve iş analistleri tarafından doğrudan yazmayı ve iletişimi destekleyen bir araç tasarlanmaya çalışılmıştır. Çalışmada BDD, yazılım geliştirmenin iki yönünü bir araya getirdiğini belirmişlerdir. Bunlardan ilki müşterilerin ihtiyaçlarının analizi ve sistemin otomatik olarak test edilmesidir. Diğeri ise analiz ve gereksinimler için kullanıcı hikâyelerini kullanmasıdır. BDD müşterilerin gereksinimlerine odaklanmıştır ve doğrudan müşteri ile tartışılabilecek kabul testleri sunmuştur. Tasarlanmaya çalışılan araçta Cucumber kullanırken, senaryolarının yazma

şekli açıklayıcı ve anlaşılması açık olduğu belirtilmektedir. Cucumber gibi BDD araçların kullanımları dikkate alındığında bazı dezavantajlara sahiptir olduğu öne sürülmüştür. BDD'de kullanıcı hikâyeleri uygulamanın otomatik olarak test edilmesi için kullanılabilir ve projeyi kalite güvencesi anlamında olumlu yönde etkilediği belirtilmektedir.

2.2 Yazılım Testi

Yazılım testi, yazılımın içinde hata bulma niyetiyle değerlendirme sürecini ifade etmektedir. Bir yazılım ürünün niteliğini, yeteneğini değerlendirmeyi ve kalitesini karşıladığını belirlemeyi amaçlayan bir tekniktir. Yazılım testi, yazılımı güvenilirlik, kullanılabilirlik, bütünlük, güvenlik, yetenek, verimlilik, taşınabilirlik, bakım, uygunluk vb. gibi diğer yazılım kalite faktörleri için de test etmek üzere kullanılmaktadır. Standart mühendislik yöntemleri yerine hazırlanmış bir yöntemdir. Test yapmak pahalı olabilir, ancak yazılımı test etmemek daha maliyetli olmaktadır. Yazılım testi, takip edilmesi gereken belirli hedef ve ilkelere ulaşmayı amaçlamaktadır (Sawant *et al*. 2012).

Yazılım testi, gerçek sonuçların beklenen sonuçlarla eşleşme durumunu kontrol etmek ve yazılım sisteminin hatasız olmasını sağlamak için yapılan bir faaliyet olarak tanımlanmaktadır. İlgilenilen bir veya daha fazla özelliği değerlendirmek için bir yazılım bileşeninin veya sistem bileşeninin yürütülmesini içermektedir. Yazılım testi aynı zamanda gerçek gerekliliklerin aksine hataları, boşlukları veya eksik gereksinimleri tanımlamaya yardımcı olmaktadır. Manuel olarak ve otomatik araçlar kullanılarak yapılmaktadır (İnt.Kyn.5).

Yazılım testi, bir yazılım veya uygulamanın hatasız olduğunu doğrulama ve onaylama işlemi olarak tanımlanmaktadır. Tasarım ve geliştirmesinde rehberlik ettiği şekilde teknik gereklilikleri karşılar ve tüm istisnai ve sınır durumlarını ele alarak kullanıcı gereksinimlerini etkin ve verimli bir şekilde karşılamaktadır. Yazılım testi, yalnızca mevcut yazılımdaki hataları bulmayı değil aynı zamanda yazılımı verimlilik, doğruluk ve kullanılabilirlik açısından iyileştirmeye yönelik önlemleri bulmayı da

amaçlamaktadır. Temel olarak bir yazılım programının veya uygulamasının özelliklerini, işlevselliğini ve performansını ölçmeyi amaçlamaktadır (İnt.Kyn.6).

2.2.1 Yazılım Geliştirmede Test

Ürün geliştirmenin önemli kısmı, ürünün paydaşlarının beklentilerine cevap verdiğinden emin olduğu kalite güvencesi görevidir. Bu durum beklentiler, kullanılabilirlik, güvenilirlik, performans vb. gibi dokundukları özelliklerin niteliğine bağlı olarak gruplara ayrılmaktadır. Yazılım geliştirmede teknik kalite güvence süreci temel olarak testler şeklinde yapılmaktadır. Test, eylemlerin sonucunun analiz edildiği bir ürün üzerinde bir dizi eylemin gerçekleştirildiği bir prosedür olarak tanımlanmaktadır. Faaliyetler manuel veya otomatik sistemler tarafından gerçekleşmektedir. Sonuçlar ürünün, kullanıcılarının, paydaşlarının veya çevrenin tepkileridir (Nilsson 2015).

Test bulguların belirlenmesi, yazılım kalite düzeyinin arttırılması, karar mekanizmaları için veri elde edilmesi ve eksikliklerin önüne geçilmesi bakımından önem taşımaktadır. Genel olarak test ürünün istenen seviyede olduğunu belirlemek, değilse de istenilen seviyeye gelmesini sağlamak için kullanılan bir süreçtir. Test aktiviteleri yazılımın kalite ve güvenilirliğinin artmasını sağlamaktadır. Yazılım testi teknik bir iş olabilir fakat bununla birlikte insan psikolojisini ve bazı önemli ölçütleri de göz önünde bulundurmak gerekmektedir. Normal şartlarda bir yazılım içindeki tüm olası durumlar test edilmek istenir ancak çoğu zaman bu durum gerçekleşmemektedir. Basit görünen bir yazılımın çok sayıda olası girdileri ve çıktıları olabilir. Tüm bu olasılıklar için test aktivitesi oluşturmak güç bir durumdur. Karmaşık bir yazılımın tamamen test edilmesi için uzun zamana ve kaynak ihtiyaç durulmaktadır. Yazılım test eden uzmanlarının test aktiviteleri için uygun bakış açısına sahip olmaları gerekmektedir. Bazı durumlarda test uzmanlarının bakış açısı, işlemesi gereken süreçten daha önemli olmaktadır (Serdaroğlu 2015).

Yazılım geliştirmede test hataları bulma, kalite seviyesi hakkında bilgi kazanma, karar vermek için bilgi sağlama ve hataları önleme gibi hedefleri bulunmaktadır. Kalite kontrol edilmez üretilir düşünce yapısı ve testlerin yazılım geliştirme sürecinin en

başından itibaren işin içine dâhil olması gerekliliği hataların daha yapılmadan önlenmesini sağlamaktadır. Üretilen çıktıların gözden geçirilmesi (analiz dokümanı gibi) sonucunda bulunan hataların tanımlanıp çözülmesi de kodda hataların çıkmasını engellemeye de yardımcı olmaktadır (İnt.Kyn.7).

2.2.2 Yazılım Testinin Hedefleri

Hedefler yazılım sürecinin çıktısıdır. Yazılım testinin hedefleri aşağıda belirtilmektedir (Sawant *et al*. 2012):

- Doğrulama, test ürünün veya yazılımın istenen şekilde çalıştığını doğrulamak ve yazılımın belirtilen şartları yerine getirip getirmediğini doğrulamak için de kullanılmaktadır.
- Öncelik kapsamı, testleri bütçe ve program sınırları dâhilinde verimli ve etkili bir şekilde yapılmalıdır.
- Dengeli test süreci gereksinimleri, teknik kısıtlamaları ve kullanıcı beklentilerini dengelemelidir.
- İlgili dokümanlar, test sürecinin hem başarısı hem de başarısızlığı ile hazırlanmalıdır. Bu yüzden aynı şeyi tekrar test etmeye gerek duyulmamaktadır.
- Deterministtik; ne yaptığımızı, neyi hedeflediğimizi, olası sonucu ne olacağını bilmeliyiz.

Amaç, bir kişinin veya sistemin başarmayı planladığı ve planladığı öngörülen bir durum olmaktadır. Bir hedef, tutarlı ve ölçülebilir olmalı ve bütün hedeflerin birbiriyle bağlantılı olması gerekmektedir. Testte, yazılım test sürecinin amaçlanan çıktıları olarak hedefleri tanımlanmalıdır. Yazılım testinin hedefleri aşağıda belirtilmektedir (Kumar and Syed 2010):

- Doğrulama ve Onaylama: Testlerin yalnızca hataları bulmak için yapıldığını söylemek doğru olmaz. Hatalar yazılımı kullanan herkes tarafından bulunmalıdır. Test, bir ürünün istenen şekilde çalıştığını doğrulamak için kullanılan bir kalite kontrol önlemidir. Yazılım testi, gerçek ürünün ürün

gereksinimlerine kıyasla (yazılı ve kapalı) bir durum raporu sunmaktadır. Test sürecinin, yazılımın yayınlanması kullanımı için belirtilen koşulları sağlayıp sağlamadığını doğrulaması ve doğrulaması gerekmektedir. Test, test edilen yazılımda mümkün olduğu kadar fazla hata ortaya çıkarmalı, gereksinimlerini karşılayıp karşılamadığını kontrol etmeli ve ayrıca kabul edilebilir bir kalite seviyesine getirmelidir.

- Öncelik Kapsamı: Kapsamlı testler mümkün görülmemektedir. Bütçe ve zamanlama kısıtlamaları dâhilinde testleri verimli ve etkili bir şekilde yapılmalıdır. Bu nedenle testin efor sarfedilmesi ve makul şekilde öncelik vermesi gerekmektedir. Genel olarak her özellik en az bir geçerli giriş durumuyla test edilmelidir. Ayrıca yazılımın işletim profiline bağlı olarak giriş izinlerini, geçersiz girişi ve işlevsel olmayan gereksinimleri test edilmektedir. Yüksek oranda mevcut ve sık kullanım senaryoları, nadiren karşılaşılan ve önemsiz senaryolardan daha fazla kapsama sahip olmalıdır.
- Dengeli: Test süreci yazılı gereklilikleri, gerçek dünyadaki teknik kısıtlamaları ve kullanıcı beklentilerini dengelemelidir. Test süreci ve sonuçları tekrarlanabilir ve test cihazından bağımsız, yani tutarlı ve tarafsız olmalıdır. Geliştirme aşamasında kullanılan sürecin yanı sıra, pek çok yazılı veya dolaysız gereklilik olmaktadır. Test ederken, yazılım test ekibi tüm bu gereksinimleri göz önünde bulundurmalıdır. Ayrıca yazılımın kullanıcıları değil geliştirme ekibinin bir parçası olduğumuzu anlamalıdırlar. Test edenlerin kişisel görüşleri pek çok düşünceden yalnızca biri olmaktadır. Test cihazındaki önyargı her zaman kapsama alanında bir önyargıya yol açmaktadır. Son kullanıcının bakış açısı açıkça yazılımın başarısı için hayati önem taşıyor, ancak teknik, bütçe veya zamanlama kısıtlamaları nedeniyle tüm ihtiyaçların karşılanamayacağı kadar önemli olmamaktadır. Her kusur eksiklik zaman ve teknik kısıtlamalarına göre önceliklendirilmelidir.
- İzlenebilir: Hem başarı hem de başarısızlıkları belgelemek, test sürecini kolaylaştırmaya yardımcı olmaktadır. Ne test edildi ve nasıl test edildi devam eden bir test sürecinin bir parçası olarak gereklidir. Bu tür şeyler yinelenen test çabalarını ortadan kaldırmak için bir araç olarak hizmet etmektedir. Test planları okunacak ve anlaşılacak kadar net olmalıdır. Kaostan kaçınmak ve belgelerin

hataların önlenmesinde daha faydalı olmasını sağlamak için bilinen ortak belgeleme yöntemleri üzerinde anlaşılmalıdır.

- Deterministik: Problem tespiti testlerde rastgele olmamalıdır. Ne yaptığımızı, neyi hedeflediğimizi, olası sonucu ne olacağı bilinmelidir. Kapsam kriterleri, kararlaştırılmış nitelikteki ve öncelikteki tüm kusurları ortaya çıkarmalıdır. Ayrıca, daha sonra yüzey kaplama hataları, kapsama alanında hangi bölümün meydana geleceğine göre kategorize edilmeli ve böylece gelecekteki testlerde bu tür kusurları tespit etmede kesin bir maliyet sunulmalıdır. Sürece ilişkin derinlemesine bilgi sahibi olmak, maliyetleri daha iyi tahmin etmemize ve genel gelişimi daha iyi yönlendirmemize olanak sağlamaktadır.

2.2.3 Yazılım Testinin Amacı

Yazılım geliştirmede test, geliştirilen uygulamanın denetlenebilir koşullar altında çalıştırılması ve elde edilen sonuçların değerlendirme sürecidir. Denetlenebilir durumlar hem normal hem de anormal durumları kapsaması gerekmektedir. Bilinçli şekilde hatalı şeyler yaparak olabilecek durumları önceden belirlemeye yönelik olması gerekmektedir. Gerçekleşmesi beklenen durumların olmadığını veya gerçekleşmesi gereken durumların olduğunu ortaya çıkartmak test aktivitesinin amacıdır (Tian 2005). Yazılım testi aşağıda belirtilen amaçlar için yapılır:

- Müşteriye sunulmadan önce ürün kalitesinden emin olmak,
- Yeniden çalışma (düzeltme) ve geliştirme masraflarını azaltmak,
- Geliştirme işleminin erken aşamalarında yanlışları saptayarak ileri aşamalara yayılmasını önlemek, böylece zaman ve maliyetten tasarruf sağlamak,
- Müşteri memnuniyetini arttırmak ve izleyen siparişler için zemin hazırlamak.

Yazılım testi yazılım uygulamalarının ve ürünlerin geliştirilmesinde önemli bir rol oynamaktadır. Aşağıda yazılım testinin amaçlarına değinilmiştir (İnt.Kyn.8).

- Uygun Maliyet: Testin birçok faydası vardır ve en önemlilerinden biri maliyet etkinliğidir. Projemizin zamanında test edilmesi, uzun vadede para tasarrufu

sağlamaktadır. Yazılım geliştirme birçok aşamadan oluşur ve eğer daha erken aşamalarda hatalar yakalanırsa, bu sorunların giderilmesi çok daha düşük maliyetli olmaktadır.

- Güvenlik: Yazılım testinin en hassas ve en önemli kısmı olmaktadır. Kullanıcılar her zaman güvenebilecekleri güvenilir ürünler aramaktadırlar. Sorunları ve riskleri önceden gidermede yardımcı olmaktadır.
- Ürün Kalitesi: Ürününüzün vizyonunun hayata geçmesi için planlandığı gibi çalışması gerekmektedir. Gerekli sonuçları almanıza yardımcı olduğundan ürün gereksinimlerini takip etmek önemlidir.
- Müşteri Memnuniyeti: Bir ürün sahibinin nihai hedefi, en iyi müşteri memnuniyetini sağlamaktır. Mümkün olan en iyi kullanıcı deneyimini sağlamak için yazılım test edilmelidir. Bu pazardaki en iyi ürün olmak, uzun vadede büyük etkileri olacak güvenilir müşteriler kazanılmasına yardımcı olmaktadır.

2.3 Yazılım Test Seviyeleri

Yazılım geliştirme yaşam döngüsünde test yapmak için birçok karakteristik bulunmaktadır. Bunlar yazılım geliştirme aktivitelerinin her bir aşamasına karşılık gelmekte ve belirli hedeflere yönelik olmaktadır. Test seviyeleri projenin akışına ve sistemin mimarisine göre birleştirilir ya da yeniden yapılandırılmaktadır. Farklı test seviyeleri için genel hedefler, yaklaşımlar ve sorumluluklar bulunmaktadır (Serdaroğlu 2015).

Yazılım Test Stratejisi (seviyesi), yazılım test senaryo tasarım yöntemlerini, başarılı bir yazılım inşası ile sonuçlanan iyi planlanmış bir dizi adımla birleştirmektedir. Bu adımlar yazılım test stratejileri test için yol haritası olmaktadır. Bir yazılım test stratejisi, özelleştirilmiş bir test yaklaşımını teşvik edecek kadar esnek olmalı, aynı zamanda da yeterince doğru olmalıdır. Strateji genellikle proje yöneticileri, yazılım mühendisi ve test uzmanı tarafından geliştirilmektedir. Yazılım test stratejileri Birim testi, Entegrasyon testi, Kabul testi ve Sistem testi olarak nitelendirilmektedir (Sawant *et al.* 2012).

2.3.1 Birim Testi

Birim (unit) testi, geliştirilen kodları test etmek için yazılan test kodları ve fonksiyonun belirli bir biriminin davranışını kontrol etme işlemidir. Yazılım, modül ve sınıfların (class) fonksiyonlarını doğrular ve hataları bulmak için aramalar yapmaktadır. Yazılımın geliştirme aşamasının durumuna göre sistemin geri kalan kısmından ayrı uygulanması gerekmektedir. Birim testi temel işleyiş ve yapısal testler gibi hem fonksiyonel hem de fonksiyonel olmayan karakteristikleri kapsamalıdır. Birim testi, test edilmiş olan kodun kullanımıyla ve geliştirme çevresinin desteğiyle birlikte devam etmektedir. Geliştirilen uygulamada ise birim testini kodu geliştiren geliştirici yapmaktadır. Bu aşamada yazılım geliştiriciler buldukları kod geliştirme hatalarını düzeltmektedirler (Serdaroğlu 2015).

Birim testlerinin başarıyla sonuçlanması, yazılım entegrasyonu ve sistem testlerinden önce geliştirilen yazılımın genel olarak beklenen gereksinimleri karşıladığının bir göstergesi olarak değerlendirilmektedir. Birim testinin başarılı olması geliştirilen yazılımın hatasız olduğu anlamına gelmez. Geliştirilen yazılımdan istenen gereksinimlerim tam olarak karşılandığını söylemek için entegrasyon ve sistem testleri ile arayüz testlerinin de başarılı olarak tamamlanması gerekmektedir (Kuday 2014).

2.3.2 Entegrasyon Testi

Entegrasyon testi birimler arasındaki arayüzleri, uygulamanın farklı parçalarının birleştirilmiş hallerinin test edilmesidir. Entegrasyon testinin birden fazla seviyesi bulunmaktadır. Birim entegrasyon testi, birim testleri yapıldıktan sonra ve birimler entegre edildikten sonra birimlerin aralarındaki etkileşimin çalışırlığını görmek için yapılan test seviyesidir. Sistem entegrasyon testi, farklı sistemleri ya da yazılımlar arasındaki etkileşimleri çalışırlığını görmek için yapılan test seviyesidir. Sistem entegrasyon testi sistem testinden sonra yapılması gerekmektedir (Serdaroğlu 2015).

Yazılım geliştirme projelerinde birim testlerin başarılı olarak tamamlandıktan sonra entegrasyon testleri başlamaktadır. Entegrasyon testlerinin amacı birim ya da

modüllerin bir araya gelerek entegre edilmiş yazılımın bileşenleri arasındaki uyumun doğru bir şekilde çalıştığını doğrulamaktır. Bununla birlikte bileşen ya da modüllere ait gereksinimleri de karşıladığının doğrulanmasıdır (Kuday 2014).

2.3.3 Sistem Testi

Sistem testi, yazılım sisteminin belirtilen gerekliliklere uygunluğunu değerlendirmek için eksiksiz ve entegre bir sistem üzerinde yapılan testlerdir. Sistem testi kara kutu (sistemin girdi ve çıktılarına dayalı test) testi kapsamına girmektedir ve bu nedenle yazılan kodların iç tasarımı hakkında hiçbir bilgi gerektirmemelidir (Sawant *et al*. 2012).

Sistem testleri geliştirilen yazılım fonksiyonel olan geresinimleri, fonksiyonel olmayan gereksinimleri ve veri kalite karakteristiklerini kontrol etmektedir. Yazılım test uzmanları eksik ve dokümanda belirtilmeyen gereksinimlerle de ilgilenmek durumunda kalmaktadır. Fonksiyonel gereksinimler için gerçekleştirilecek sistem testleri, sistemin görünen anlaşmada kararlaştırılmış tekniklerinin test edilmesiyle başlamaktadır (Serdaroğlu 2015).

2.3.4 Regresyon Testi

Uygulama ve uygulama ortamlarında gerekli değişiklikler yapıldıktan sonra yeniden yapılan testler regresyon testi olarak tanımlanmaktadır. Böylece önceki yapılan testlerde belirlenen sorunların çözüldüğünden, yeni eklenen özelliklerin problemsiz çalıştığından ve yeni hatalar oluşmadığından emin olunur.

2010 yılında yayınlanan sistemler ve yazılım mühendisliği sözlüğüne göre regresyon testi değişikliklerin istenmeyen etkilere yol açmadığını ve sistemin veya bileşenin belirtilen gereksinimlere uyduğunu doğrulamak için bir sistemin veya bileşenin seçici olarak yeniden test edilmesidir. Yazılım evrimi sırasında değişiklikler yapıldığında regresyon testi yapılmaktadır. Regresyon testinin amacı, yeni yapılan değişikliklerin yazılımın mevcut değişmeyen bir bölümünün davranışlarını engellemediğinden emin

olmak durumudur (Yoo and Harman 2012).

Regresyon testi, değiştirilmiş olan yazılımı tekrar test etme işlemidir. Regresyon testi, ticari yazılım geliştirmede test etme çabasının genişliğini oluşturur ve uygulanabilir herhangi bir yazılım geliştirme sürecinin önemli bir parçasıdır. Büyük bileşenler veya sistemler büyük regresyon test takımlarına sahip olma eğilimindedir. Buna rağmen pek çok geliştirici buna inanmak istemezler ve sistemin bir kısmındaki küçük değişiklikler genellikle sistemin uzak kısımlarında sorunlara neden olmaktadır. Regresyon testi bu tür problemleri bulmak için kullanılmaktadır (Ammann and Offutt 2016).

2.3.5 Kullanıcı Kabul Testi

Kullanıcı kabul testleri yazılım uygulamasının gerçek ortama atılmadan önceki son aşamada yapılan test aktivitesidir. Genellikle yazılım uygulamasını kullanacak olan son kullanıcı tarafından testler yapılmaktadır. Uygulama gereksinimlerin karşılandığını görülünce son kullanıcının onayı alınır ve yazılım teslim edilmektedir. Kullanıcı kabul testi öncesinde geliştirme tamamen bitirilmelidir. Birim, entegrasyon, sistem testleri gibi çeşitli testler tamamlanmalıdır. Yazılımdaki hataların çözümlenmiş olması gerekmektedir. Kullanıcı kabul testinin planlaması yapılmalıdır. Sonrasında test senaryoları olurşturulır ve teste aktivitesine başlanılmaktadır. Test sırasında hata ya da hataların belirlenmesi durumunda hataların çözümlenmesi sağlanır ve testler tamamlanarak kullanıcı kabul testi sonlandırılmalıdır.

Kullanıcı kabul testi, yazılım ürünün standartlara ve belirtilen kriterlere göre geliştirilip geliştirilmediğini ve müşteri tarafından belirtilen tüm gereklilikleri karşıladığını doğrulamak için gerçekleştirilmektedir. Bu tür testler genellikle ürünün harici olarak başka bir tarafça geliştirildiği bir kullanıcı ve müşteri tarafından gerçekleştirilir. Kullanıcı kabul testi kullanıcının sistemin iç işleyişi veya kodlaması ile pek ilgilenmediği ancak sistemin genel işleyişini değerlendirdiği ve kendi belirlediği gerekliliklerle karşılaştırdığı kara kutu test tekniğine girmektedir. Kullanıcı kabul testi ürün teslimi yapılmadan önce, kullanıcı tarafından yapılan en önemli testlerden biri olarak kabul edilmektedir (Sawant *et al.* 2012).

Kullanıcı kabul testi, müşteri tarafından üzerinde anlaşılan şartlara göre sistemi belgelendirmek için gerçekleştirilen bir test türü olarak tanımlanmaktadır. Bu test, yazılım uygulamasını gerçek ortama taşımadan önce testin son aşamasında gerçekleşmektedir. Bu testin temel amacı, iş akışını uçtan uca doğrulamaktır. Kullanıcı arayüzü hatalarına, yazım hatalarına ve sistem testlerine odaklanmamaktadır. Bu test, veri kurulumuna benzer üretim ile ayrı bir test ortamında yapılmaktadır. İki veya daha fazla son kullanıcının dahil olduğu bir tür kara kutu testi olarak belirtilmektedir (İnt.Kyn.5).

2.4 Yazılım Test Teknikleri

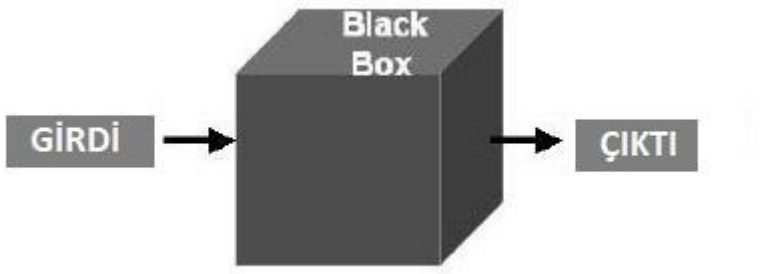
Yazılım test teknikleri, yazılım geliştirmede yazılan kodların kontrolerini yapmak ve fonksiyonların çalışma durumlarını sınamak için kullanılan test teknikleridir. Yazılım testi, yazılım kalite güvencesinin temel bir bileşenidir ve şartname, tasarım ve kodlamanın gözden geçirilmesini temsil etmektedir. Yazılım testinin temel amacı, yazılımı dikkatlice kontrol edilen durumlarda sistematik olarak test ederek yazılım sisteminin kalitesini teyit etmektir, başka bir amaç da yazılımın bütünlüğünü ve doğruluğunu tespit etmektir ve nihayet keşfedilmemiş hataları ortaya çıkarmaktadır. Hata bulmak için kullanılan en önemli üç teknik; kara kutu testi, beyaz kutu testi ve gri kutu testi olarak belirtilmektedir (Khan and Khan 2012).

Test vakaları, daha etkili testler elde etmek için çeşitli test teknikleri kullanılarak geliştirilmektedir. Bu sayede, yazılımın eksiksizliği sağlanmış ve hata bulma olasılığını en yüksek çıkaran test koşulları seçilmiştir. Bu nedenle, test uzmanları hangi test durumlarının seçileceğini tahmin etmemektedir. Gerekli test tekniklerini, test koşullarını sistematik bir şekilde tasarlamalarını sağlamaktadır. Eğer mevcut her türlü test tekniğini bir araya getirilmesi durumunda daha iyi sonuçlar elde edilmektedir. Yazılım iki şekilde test edilebilir, başka bir deyişle iki farklı yöntemi ayırt edebilirsiniz. Bunlar kara kutu testi ve beyaz kutu testidir (Jovanovic 2006).

2.4.1 Kara Kutu Testi

Kara kutu testi (Black box testing), çıktı gereksinimlerine dayalı ve programın içindeki içyapı hakkında herhangi bir bilgi veya kodlama olmadan yazılımı test etme tekniğidir. Temel olarak Kara kutu testi, doğruluk testinin ayrılmaz bir parçasıdır. Ancak fikirleri sadece doğruluk testi ile sınırlı değildir. Amaç bileşenin, bileşen için yayınlanan gereksinime ne kadar uygun olduğunu test etmektir (Jovanovic 2006).

Kara kutu testi, sistemin iç mantıksal yapısını çok az dikkate almakta ya da hiç dikkate almamaktadır. Sadece sistemin temel yönünü incelemektedir. Giriş kriterlerini uygun şekilde kabul edildiğinden ve çıkış kriterlerini doğru üretildiğinden emin olunmasını sağlamaktadır (Khan 2010).



Şekil 1.3 Kara kutu test tekniği

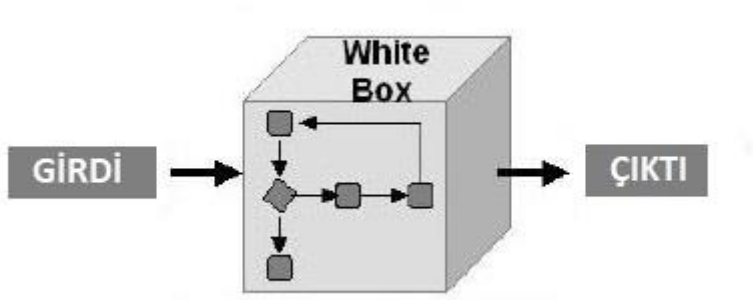
Kara kutu test tekniğiyle şu hata türleri tespit edilmeye çalışılır (Kuday 2014):

- Doğru olmayan veya hiç olmayan işlevlerin tespiti
- Arayüz hataları
- Performans hataları
- Veri tabanlarına ulaşma hataları veya veri yapılarındaki hatalar
- İlklendirme veya sonlandırma hataları
- Sınır değer hataları

2.4.2 Beyaz Kutu Testi

Beyaz kutu testi (White box testing), geliştirilen yazılım uygulamasının kod yapısı bilinerek gerçekleştirilen test tekniğidir. Beyaz kutu testi hataların tespit edilmesinde ve çözümlenmesinde oldukça etkilidir. Çünkü hatalar sorun çıkarmadan önce bulunmalıdır. Beyaz kutu testi, sisteme girdi verme ve sistemin gerekli çıktıyı üretmek için bu girdiyi nasıl işlediğini kontrol etme işlemidir. Beyaz kutu testine beyaz kutu analizi, saydam kutu testi veya saydam kutu analizi de denir. Beyaz kutu testi yazılım test sürecinin entegrasyonu, birim ve sistem seviyelerinde uygulanmaktadır (Jovanovic 2006).

Beyaz kutu testi kod uygulamasının amaçlanan tasarımdan sonra geldiğini ya da gelmediğini doğrulamaktır. Bununla birlikte uygulanan güvenlik işlevselliğini doğrulamak ve yararlanılabilir güvenlik açıklarını ortaya çıkarmak için kullanılabilecek bir güvenlik testi yöntemi olarak kabul edilmektedir (Khan 2010).



Şekil 1.4 Beyaz kutu test tekniği

Beyaz kutu testleri yazılımın içyapısı bilinerek tasarlanmaktadır. Bu nedenle testleri gerçekleştirenler genellikle yazılımın içyapısını bilen yazılım geliştiricilerdir. Beyaz kutu testiyle yazılımın içyapısındaki hataları belirlemeye odaklanmaktadır. Kaynak kod beyaz kutu testlerinin en önemli girdisi olduğundan koda ulaşım olmadan beyaz kutu testleri yapılmamaktadır (Kuday 2014).

2.4.3 Gri Kutu Testi

İç uygulama çalışmaları hakkında çok az bilginin bulunduğu test tekniğidir. Bu teknik

dil ve platformdan bağımsızdır. Gri kutu, beyaz kutu ve kara kutu testinin avantajlarını birleştirmektedir. Test durumlarını tasarlamak için, beyaz kutu testinden daha az ancak kara kutu testinden daha fazla algoritma ve dâhili veri yapısını kullanmaktadır. Gri kutu testi yazılımın bir parçasını özelliklerine göre test edilmesidir. Fakat içsel çalışma konusunda biraz bilgi sahibi olmak için bu teknik kullanılmaldır. Entegrasyon testlerinde yaygın olarak kullanılır bunun yanında çoğu test aşamasına da uygulanmaktadır (Jan *et al.* 2016).

Gri kutu test teknikleri beyaz kutu ve kara kutu test metodolojilerini birleştirmektedir. Gri kutu test tekniği, bir yazılım parçasını kendi özelliklerine göre test etmek için kullanılır, ancak kendi iç çalışma bilgisi de kullanmaktadır. Gri kutu testi ayrıca sınır değerlerini veya hata mesajlarını belirlemek için tersine mühendislik içermektedir. Gri kutu testi, altı çizili kod veya mantık hakkında bilgi sahibi olurken test yazılımını içeren bir işlemdir. Programın içindekilerin gri kutu testinde anlaşılması, kara kutu testinden daha fazla, beyaz kutu testinden daha az olmaktadır (Khan 2010).

2.5 Manuel Yazılım Testi

Manuel test, test uzmanlarının herhangi bir otomasyon aracı kullanmadan test senaryolarını manuel olarak uyguladıkları test aktivitesidir. Tüm test türlerinin en ilkelidir ve yazılım sistemindeki hataların bulunmasına yardımcı olmaktadır. Yeni bir yazılım uygulama testi otomatik duruma getirilmeden önce manuel olarak test edilmektedir. Manuel test daha fazla çaba gerektirmektedir. Yazılım test otomasyonun uygulanabilir olma durumunu kontrol etmek için gerekli olmaktadır. Manuel test, herhangi bir test aracı bilgisi gerektirmemektedir (İnt.Kyn.5).

Manuel test, uzmanların yazılım hatalarını bulmak için program beklentilerini ve gerçek sonuçları karşılaştıran süreç olarak adlandırılmaktadır. Manuel testler yazılımı son kullanıcı olarak kullanan ve ardından yazılımın uygun şekilde hareket edip etmeyeceğini belirleyen test aktivitesidir. Manuel test herhangi bir otomatik araç veya komut dosyası kullanmadan insan tarafından gerçekleştirilmektedir. Manuel test için test uzmanları tarafından test planı ve test senaryosu hazırlanmaktadır. Hazırlanan test

senaryolarına göre yazılım uygulamasının özellikleri sınanmaktadır (Çoşkun 2016).

Manuel Yazılım Testinin Avantajları:

- En önemli özelliği esnekliktir.
- Gerçek kullanıcı sorunlarını bulma olasılığı daha yüksektir.
- Kısa vadeli maliyet düşüktür.
- Test senaryosunun yalnızca bir veya iki kez çalışması gerektiğinde faydalı olmaktadır.
- Test durumlarını her seferinde test etmek için test cihazı aynı miktarda zaman gerektirmektedir.
- Kullanıcı arayüzü testlerinde faydalı olmaktadır.
- Test durumlarını ilk kez manuel test kullanarak yapmak çok faydalı olmaktadır (İnt.Kyn.9, İnt.Kyn.10).

Manuel Yazılım Testinin Dezavantajları:

- Bazı görevlerin manuel olarak yapılması zordur.
- Manuel testler uyarıcı değildir.
- Manuel testler tekrarlı kullanılamaz.
- Manuel testler daha az güvenilirdir.
- Programlanamaz.
- Zaman alıcı ve sıkıcıdır.
- Manuel test kullanarak, farklı işletim sistemi platformlarında farklı makinelerde test yapmak aynı anda mümkün değildir (İnt.Kyn.9, İnt.Kyn.10).

2.6 Yazılım Test Otomasyonu

Yazılım test otomasyonu, çeşitli yazlım araçlarıyla sürekli olarak yapılan manuel testlerin otomatik hale getirilmesidir. Yazılım test otomasyonu tekrar eden testleri ya da tekrar sayısı çok fazla olan testlerin yazılım araçları ile otomatik hale getirilmesidir. Burunla birlikte test otomasyonu otomatize edilen ve manuel olarak koşulması zor olan testleri içermektedir. Yazılım test otomasyonu test yapan kişilerin ilgili özellikler için kodlamaları yaptığı ve yazılımı test etmek için başka yazılımlar kullandığı test

aktivitesidir.

Yazılım projelerinde, sürekli olarak bir kod geliştirme ve bu kod geliştirme doğrultusunda test ihtiyacı artmaktadır. Bununla birlikte aynı senaryoları tekrar tekrar koşturmak zorunluluğu ortaya çıkmaktadır. Bu durumda test otomasyonu ihtiyacı ortaya çıkmaktadır. Testlerin hızlı ve verimli yapılması yazılım geliştirmede fark yaratan bir unsur olmaktadır. Test otomasyonu bu tür ihtiyacın karşılanması için test alanında yapılması zorunlu olan bir yazılım öğesidir. Kritik ve önemli iş süreçlerinde, karmaşık yazılım uygulamalarında ve bunları tehlikeye atan kullanım durumlarında test otomasyonuna odaklanmak önemli görülmektedir. Bu durumlarda testleri otomatize etmenin en büyük yararı zaman ve efor tasarrufu sağlamaktır. Yazılım otomasyon testi test senaryolarının hızlı, art arda ve tekrar tekrar uygulanabilmesini sağlamaktadır. Farklı test türlerinde ve sık sık değişiklik yapılan regresyon testlerinde kolaylık sağlamaktadır. Yazlım uygulamasındaki insan kaynaklı hataları en aza indirmektedir. Yazılımdaki olası hataların daha erken belirlenmesinde etkili olmaktadır. Yazılım uygulamasındaki testlerin kalitesini arttırmaktadır. Testlerin raporlanmasına katkı ve kolaylık sağlamaktadır.

Testleri elle yapmak, testlerin hangi sınıfa ait olduğuna bakılmaksızın zaman alıcı bir işlem olması nedeniyle testleri mümkün olan her yerde otomatize etmek önemli görülmektedir. Kabul ve kullanılabilirlik testleri genellikle etkileşimlidir ve insanların katılımını gerektirmektedir. Bu da onları imkânsız olmasa da otomatize edilmelerini zorlaştırmaktadır. Birim, entegrasyon ve sistem testleri ise etkileşimli olmayan bir niteliktedir. Bu otomatik testleri yürütmek için harcanan zamanın, manuel yapılmasını önemli ölçüde azaltılabileceği anlamına gelmektedir (Nilsson 2015).

Test otomasyonu, otomatik araçlar, komut dosyaları ve yazılımlarla gerçekleşmektedir. Test araçları sistem için uygun programlama dili desteği sunmalıdır. Ayrıca test araçları otomatik testlerde herhangi bir işlem için sistemin geliştirilmesine olanak tanımalıdır (Çoşkun 2016).

Otomatik yazılım testi, geliştirilen yazlımın beklenen sonuçlarını gerçek sonuçlarla

karşılaştırmak için algoritmalara dayalı testleri çalıştırmak için otomatik araçlar kullanmaktadır. Sonuçların doğru olması durumunda, yazılım düzgün çalıştığı sonucuna varılmaktadır. Beklenen sonuç ve gerçekleşen sonuç tutarsız olması durumunda yazılan kodlar kontrol edilmeli ve düzeltilmelidir. Sonuçlar tutarlı ve doğru olana kadar testlere devam edilmektedir (Keus and Dyck 2016).

Otomasyon genellikle test vakalarının her değişiklik yapıldığında yapıldığı birim testi veya regresyon testi gibi tekrarlayan görevleri yerine getirmek için uygulanmaktadır. Test otomasyon sistemlerinin tipik görevleri test komut dosyalarının geliştirilmesi ve yürütülmesi ile test sonuçlarının doğrulanmasıdır. Manuel testin tersine otomatik testler keşif testi veya geç geliştirme doğrulama testleri gibi çok az tekrarlanan işler için uygun olmamaktadır (Kasurinen *et al.* 2010).

2.6.1 Test Otomasyonundan Beklentiler, Avantajları ve Dezavantajları

Yazılım test otomasyonu genel olarak fazla test girdisi ile yapılan tekrar sayısı fazla olan testler için kullanılmaktadır. Yazılım test otomasyonu uzun zaman zarfında iş gücü kazancı sağlaması gerekmektedir. Test otomasyon aktivitelerine başlamadan önce yapılan eksik ölçüm, ön çalışmalarından dolayı zaman, iş gücü, çalışma ekibinde motivasyon ve itibar kaybına neden olduğu görülmektedir. Test otomasyonu zaman, sabır ve uzman bir ekip isteyen zorlu bir süreç olmakla birlikte atılan adımların titizlikle seçilmesi başarıya ulaşmada önemli olmaktadır (Bilgin ve Kasapoğlu 2016). Yazılım test otomasyonunun başarıya ulaşması için beklenen ve beklenmeyen durumlara aşağıda değinilmektedir:

Yazılım test otomasyonundan başlıca beklentiler:

- Regresyon testlerinin tekrarlanması.
- Testlerin kısa zaman aralığında yürütülmesi.
- Manuel olarak yapılması zor testlerin yapılabilmesi.
- Kaynakların daha verimli kullanılması.
- Kararlı ve tekrar edilebilir testlerin gerçekleştirilmesi
- Testlerin tekrar tekrar kullanılabilirliği.

- Yazılım sürümlerin daha erken yayımlanabilmesi.
- Yazılım güvenilirliğinin artırılması.

Yazılım test otomasyonundan beklenilmemesi gerekenler:

- Test aracından hatasız test yapmasını beklemek.
- Test aracının manuel testleri iyileştirebileceğini düşünmek.
- Otomatik testlerin tüm hataları bulacağını düşünmek.
- Test aracının hatasız yazılım çıkarmasını beklemek.
- Bir kez geliştirilen testlerin değişikliğe ihtiyaç duymayacağını düşünmek.
- Test aracının organizasyona hızlı ve kusursuz uyum sağlayacağını düşünmek.

Otomasyon testi, test araçlarının kullanılmasıdır. Bununla birlikte manuel test ve insan katılımlı tekrarlayan gereksiz testlere duyulan ihtiyacı azaltmaktır. Otomatik test, zamandan ve kaynaklardan tasarruf etmek için kullanılmaktadır (Singla and Kaur 2014).

Test Otomasyonunun Avantajları:

- Otomatik test daha güvenilirdir.
- Otomatik testler çok hızlı.
- İnsan kaynaklarına daha az yatırım.
- Programlanabilir.
- Kolay ve etkili bir şekilde yürütebilir.
- Uygun maliyetli olabilir.
- Sonuçlar hızla görebilir.
- Otomasyon testi, bir dizi test senaryosunun sık sık yürütülmesi için faydalı olmaktır.
- Otomasyon test takımları yaptıktan sonra, test durumlarını yürütmek için daha az sayıda test cihazı gerekli.
- Aynı anda farklı işletim sistemi platformu kombinasyonu ile farklı makinede otomasyon testi de yapılabilir.
- Kullanıcı arayüzü testinde yardımcı değildir.
- Otomasyon testi yapı doğrulama testini otomatize etmek için çok faydalıdır.

- Otomasyon testi, kodun sık sık değiştiği testlerde çok yardımcı regresyonlardır (İnt.Kyn.9, İnt.Kyn.10).

Test Otomasyonunun Dezavantajları:

- Takımlara maliyeti pahalı olabilir.
- Araç kullanımı zaman alabilir.
- Araçlar çeşitli sınırlamalar içerebilir (İnt.Kyn.9, İnt.Kyn.10).

2.6.2 Web Otomasyon Testi

Web testi; explorer, chome gibi web tarayıcılarda çalışan yazılım uygulamalarının doğrulama işlemlerine web testi denilmektedir. Maliyetleri düşürmek web uygulamalarını test etmek için gereken çabayı en aza indirmek, yazılım kalitesini artırmak, pazara sunma süresini azaltmak ve yeniden kullanılabilir test durumlarını kullanmaya yardımcı olmaktadır. Hızlı değişen ve rekabet gücü yüksek web tabanlı iş ortamında kuruluşların web uygulamalarını manuel bir test aracı kullanarak test etmeleri çok önemlidir. Bu nedenle testleri otomatize edilen web uygulamalarının ve web servislerinin olağan işlevselliğinin doğru şekilde çalışmasını sağlamalıdır. Testleri birden çok tarayıcıda, platformda, dilde, veritabanında, sunucularda yeniden kullanma ve genişletme tüm kullanıcıların erişime girdiğinden emin olma özelliğini sağlamalıdır (Sharma and Angmo 2014).

Web otomasyon testinin özellikleri:

- Otomatik yazılım testi zaman ve kaynak kazandırır.
- Test etme doğruluğunu artırır.
- Test kapsamını artırır.
- Otomatik yazılım testi, geliştiricilere ve test uzmanlarına yardımcı olmaktadır.

2.6.3 Yazılım Test Otomasyonu İlgili Araştırmalar

Gebizli vd. (2013) tarafından yapılmış Kullanım Modeli Bazlı Otomatik Test Tasarımı isimli çalışmalarında televizyonlar AR-GE aşamasında son kullanıncaya ulaşmadan

önce yazılımların kullanım profillerine göre test senaryolarının ve test datalarının otomatik hazırlaması hedeflenmiştir. Daha sonrasında Vestel test yönetim otomasyon sisteminde senaryoların otomatik çalıştırılması ve sonuçlarının sistemde saklanıp ihtiyaç olduğu durumlarda olası istenen raporların oluşturulabilmesi için bir yöntemden bahsetmektedirler. Çalışmada mevcut çalışan manuel testleri otomatize edip, sonrasında çalışan otomatik çalışan testler ile zamandan ve kaynaktan kazanç sağladıkları sonucuna ulaşmışlardır. Bazı durumlarda ise kritik hataları otomatik testlerle bulanama durumu olabileceği görüşü belirtmektedirler. Çalışmalarında testi kullanıcı profiline uygun ve son kullanıcıların karşılaşa bilecekleri durumları göz önüne alarak otomatik testler tasarlamışlardır. Yeni bir projenin karşılaması gereken ihtiyaçları, test edilecek test durumları ve tamamlanan testlerin sonuçlarına ait verileri tutarak, üretime verilecek yeni yazılımla ilgili bilimsel değerler üretmişlerdir.

Özkan ve Sözer (2015) Web Uygulamaları İçin Model Bazlı Test Süreci Otomasyonu isimli bildirilerinde web uygulamalarını model bazlı test sürecini test otomasyonu yardımıyla iyileştirecekleri görüşünü sunmaktadırlar. Farklı araçların birleştirilmesiyle sistem modelinin yarı otomatik bir yöntemle oluşturulmasını ve oluşturulan test senaryolarını sistem üzerinde otomatik çalıştıracaklarını belirtmektedirler. Çalışmalarında web uygulamalarında test sürecini iyileştirmek için çeşitli araçları birlikte kullanarak GraphWalker isimli model bazlı test aracını geliştirmişlerdir. Çalışmalarında kısmen otomasyon desteği verilebilecek manuel test yapılması gereken aktiviteleri test etmişlerdir. Sonrasında bu aktiviteler için test otomasyonu geliştirmede mevcut araçların kullanımıyla yeni araçlar geliştirmektedirler. Mevcut araçların kullanımıyla oluşturulan yeni araçlar sayesinde test oluşturma hızının arttığını ve harcanan zamanın azaldığını gözlemlemişlerdir. Aynı zamanda harcanan çabanın azaldığını görmüşlerdir. Geliştirilen araç ile programlama bilgisi az olan ya da olmayan insanlara da testleri otomatik üretme ve çalıştırma imkânı sunmuşlardır.

Özalp vd. (2015) Web Projeleri İçin Otomatik Test Senaryosu Üreten Ve Koşan Kara Kutu Test Çatısı isimli çalışmalarında web uygulamalarında ilgili girdi alanları için "Each Choice, Paire-Wise, T-Wise ve All Combination" kapsamları için test aracı geliştirmişlerdir. Geliştirdikleri araçta C# yazılım dilini kullanmışlardır. Bu aracın

geliştirmesinin her dile yapılabileceğini belirtmişler fakat her dil için ayrıca uygulanması gerektiğine değinmişler. Geliştirilecek bu yaklaşım ile yazılım test sürecini zaman ve maliyet konularını azaltacağını öngörmüşlerdir. Belirtilen yapı ile test senaryoları hızlı bir şekilde test edildiğini belirtmektedirler. Bu sebeple yazılım test süreçlerinde daha az test uzmanlarına ihtiyaç duyulacağını söylemektedirler.

Bilgin ve Kasapoğlu (2016) Kullanıcı Arayüzü Yazılımı Test Otomasyonundan Beklentiler ve Riskler isimli çalışmalarında test otomasyonuna duyulan ihtiyacın nasıl belirleneceği ve sonrasında yapılması gerekenleri yaşanılan tecrübeler doğrultusunda anlatmaktadırlar. Çalışmada ilk olarak test otomasyonuna duyulan ihtiyacın belirlenmesi ve test otomasyonundan beklentileri belirtmişlerdir. Sonrasında proje seçimi yapılırken seçilen yazılımın yazılım yasam döngüsünün sonlarında olmamasına dikkat edilmesi konusuna açıklık getirmişlerdir. Çalışmalarında yazılım test otomasyonu maliyet ve fayda analizi, ihtiyacı karşılayacak test aracının araştırılması hususlarına dikkat çekmişlerdir. Altyapı çalışmalarının otomasyona uyarması, testlerin otomasyon aracına aktarılması ve gerekli konfigürasyonların yapılması, sürekli entegrasyon ve test sürecinde ölçüm konularına açıklık getirmişlerdir. Çalışmalarında sonuç olarak yazılım test otomasyonuna geçme kararı alınmadan önce ihtiyaçlar ve beklentiler belirlenmeli, test otomasyon araçlarının araştırılması gerekliliğine değinmişlerdir. Test otomasyonu sürekli gelişmeye açık bir süreç olmasından dolayı yapılacak ölçümler ile eksik veya hatalı bulunan yönleri düzeltilebilir. Sürecin başında test otomasyonuna karar verilmesinde yapılan maliyet ve fayda analizi süreç boyunca tekrarlanabilir olduğunu belirtmişlerdir. Bu tekrarlarda artan tecrübe ve bilgi birikimleri ile gerçeğe daha yakın sonuçlar elde edilebileceğini söylemektedirler.

Büyükyumukoğlu vd. (2017) Test Otomasyonunda Karşılaşılan Zorluklar Ve Öğrenilen Dersler isimli çalışmalarında, teknoloji ve iletişim hizmetleri sağlayıcısı bir firmanın web tabanlı uygulamalarını kullanarak yapılan bir takım işlemlerin manuel test senaryolarının otomatik testlere dönüşümünde yaşanan deneyimleri aktarmaktadırlar. Bu deneyimleri teknoloji ve tasarım, yazılım yaşam döngüsünü oluşturan süreçlere entegrasyon, yazılım ekibi ve kullanıcıların motivasyonu ve değişiklik yönetimi olmak üzere dört boyutta ele almaktadırlar. Çalışmalarında otomasyonu geliştirme sürecinde

basit mimari ve teknoloji kullanmanın test geliştirilecek test otomasyonun başarısını arttıracağı görüşünü belirtmişlerdir. Test otomasyonunda çalışan senaryonun hata alması durumunda ilgili test sorumlusunun işleme el ile müdahale ederek senaryonun kaldığı yerden çalışmasına devam etmesinin sağlanmasının gerekli olacağını belirtmektedirler. Senaryoların otomatize edilmeden önce iş akışlarının detaylı olarak analiz edilmesi gerekliliğini vurgulamışlardır. Test otomasyonunda çalışan senaryoların hata alması durumunda ilgili kişilerin motivasyonuna olumsuz yönde etki yaptığını gözlemlemişlerdir.

Emektar vd. (2018) Sigortacılık Web Servislerinde Test Ve Test Otomasyonu Yaklaşımı isimli çalışmalarında web servislerinin yazılım test süreçlerinde kullanılabilecekleri yazılım test otomasyon çözümleri, yazılım geliştirme maliyetinden elde edilen kazanç, edinilen fayda ve tecrübeler paylaşılmıştır. Çalışmalarında web servisleri üzerine geliştirdikleri test otomasyonu ile web servislerinin tekrarlı test yapılma süreleri daha kısa sürede sonuçlandığını görmüşlerdir. Yazılım test otomasyonu geliştirilirken farklı değer alan parametreler değişken olarak tanımlanarak aynı senaryoları değişen verilere göre testleri koşarak daha geniş örnek alanda test yapılması sağlanmıştır. Geliştirilen test otomasyonunda değişken parametre yapısı sayesinde farklı ortamlarda testleri ek süreye ihtiyaç duymadan gerçekleştirmişlerdir. Test otomasyonu ile senaryoların paralel ve seri koşumlarıyla yük oluşturarak performans değerlendirmeleri yapmışlardır. Geliştirilen otomasyon test sayesinde manuel test eforu ortadan kalktığı ve bu sayede olası insan hatalarının ortadan kalktığı görüşünü belirtmişlerdir.

Haberl vd. (2011) Uygulamada Yazılım Testi isimli çalışmalarını 15'ten fazla ve farklı sektörlerden yazılım geliştirici, testçi ve yönetici olmak üzere üç grupta 1623 katılımcıyla (Almanya'dan %77, İsviçre'den %13, Avusturya'dan %10 katılım oranlarıyla) gerçekleştirmişlerdir. Uygulamalarda yazılım testi çalışmalarındaki ankette katılımcılara; kabul testlerinde, entegrasyon testlerinde, sistem testlerinde ve birim testlerinde test otomasyonu sistematik bir şekilde ve düzenli olarak desteklenir mi? sorusunu yöneltmişlerdir. Bu soruya katılımcılardan gelen cevaplara ait veriler; birim testlere ait senaryoları manuel teste oranla %70-75'i otomatize edilmiş, entegrasyon test

verilerinde test senaryoları manuel teste oranla %30'u otomatize edilmiştir. Kabul testi en düşük otomatik test seviyesi olduğu ve ankete katılanların %40'ından fazlası testleri manuel olarak yaptığı belirtilmektedir. Çalışmada testler için özel ortamların kullanımının çok yaygın olması (>% 80) durumu olumlu sonuç olarak yorumlanmıştır. Bununla birlikte test otomasyonunun kabul, sistem ve entegrasyon testlerinde hala çok az kullanılması şaşırtıcı olduğu söylenmiş. Test otomasyonu birim testlerine odaklanmıştır ancak bu test seviyesi bile tamamen otomatik olmaktan uzaktır yorumu yapılmıştır. Kabul ve sistem testleri yalnızca küçük bir anket katılımcı grubu tarafından büyük ölçüde otomatize edildiği görüşü belirtilmektedir.

Rafi vd. (2012) Otomatik Yazılım Testinin Yararları Ve Sınırlamaları: Sistematik Literatür Taraması ve Uygulayıcı Araştırması isimli çalışmalarında test otomasyonunun faydaları ve sınırlarıyla ilgili görüşleri inceleyerek akademik ve uygulayıcı görüşleri arasında var olan açığı kapatmaya çalışmışlardır. Akademik görüşler sistematik bir literatür taraması ile incelenirken uygulayıcı görüşleri 115 yazılım uzmanından yanıt alınan bir anketle değerlendirmişlerdir. Çalışmaları üç katkı sağlamaktadır. İlk olarak, akademik literatürdeki yazılım test otomasyonu yararları ve sınırlamaları hakkında sistematik bir inceleme yapılarak bu konulardaki verilerin bir bölümü çok fayda sağlamıştır. Verilerin bir bölümü sınırlamanın sadece bir veya iki kaynak tarafından desteklenmesi nedeniyle oldukça sığ olduğunu belirtmişler. İkinci olarak, katılımcıların yazılım test otomasyonu yararları ve sınırlamaları hakkındaki görüşlerine yönelik bir anket yapmışlardır. Bu ankete ait verilerin analizi sonucu test otomasyonunun temel faydalarının test uygulamalarında yeniden kullanılabilirlik, tekrarlana bilirlik ve çaba olduğunu göstermiştir. Bu sonuçlar birkaç regresyon test turuna ihtiyaç duyulduğunda test otomasyonunun üstünlüğünü desteklemektedir. Katılımcılar otomasyonun test kapsamını iyileştirdiğini belirtmektedir. Bu otomasyonun aşırı regresyon testine ihtiyaç duyulmadığında bile faydaları olduğu anlamına gelmektedir. Sınırlamalar ile ilgili olarak otomasyonun test senaryolarının tasarlanmasında, bir test otomasyon aracının satın alınmasında ve personelin eğitilmesinde yüksek bir başlangıç maliyeti getirdiğini tespit etmişlerdir. Şaşırtıcı olmayan bir şekilde, otomatik test vakalarının bakımı da sorunlu olarak algılanmıştır. Ayrıca katılımcıların % 45'i mevcut test otomasyon araçlarının ihtiyaçları için zayıf bir seçim sunduğunu düşünmektedir. Üçüncüsü, ankete

ve literatür taramasına dayanarak, akademik literatür ve katılımcılar arasındaki görüşlerde bazı farklılıkları vurgulamışlardır. Genel olarak katılımcıların ve akademik literatürün görüşleri, test otomasyonunun faydaları ve sınırlamaları ile büyük ölçüde uyumludur. Farklılıktan dolayı birçok akademik kaynak test otomasyonunun hata tespitini artırdığına dair kanıtlar sunarken, katılımcıların % 58'i buna katılmamıştır.

Sharma ve Angmo (2014) Web Tabanlı Otomasyon Testleri ve Araçları isimli çalışmalarında otomasyon testlerini ve otomasyon testleri için mevcut araçları tanımaya yardımcı olacak çeşitli web otomasyon test araçlarını incelemişlerdir. Çalışmalarında ilk olarak statik test (dökümantasyon testi) ve dinamik (yazılım uygulama testi) testlerinden söz etmişlerdir. Sonrasında kara kutu testi (sisteme ait girdiler ve bu girdilere ait çıktıların kontrolleri) ve beyaz kutu testlerine (yazılan kodların testi) açıklık getirmişlerdir. Manuel test ve test otomasyon karşılaştırılması yapıldıktan sonra web otomasyon test araçlarını incelemişlerdir. Bu araçlar Selenium, Watir, Hp-Qtp, Test Complete, FitNesse, Hp Load Runner, Test Ng, Tosca, Silktest ve Win Runner'dır. Çalışma sonucunda web sitelerini test etmenin en kolay yöntemi, web sitelerini kullanırken davranışları kaydedebilen ve ardından web tarayıcısında adımları otomatik olarak izleyebilen Selenium aracını kullanmak olduğunu belirtmişlerdir. Otomatik test araçları kullanmanın en büyük yararı, testleri otomatikleştirerek web sitelerinin her bir özelliğini test etmek için gereken manuel çabayı önlemektir. Web tabanlı otomasyon testi ve bu tür testler için otomatik araçlar üzerine yapılan araştırmalardan, Selenium'un şu ana kadar web uygulamaları için mevcut en iyi otomasyon aracı olduğu sonucuna varmışlardır.

Singla ve Kaur (2014) Selenyum Anahtar Kelimeleri Odaklı Otomasyon Test Çerçevesi isimli çalışmalarının amacı, Selenium Webdriver yazılım test aracını kullanarak web uygulamaları için otomasyon testi yapmaktır. Bu web test aracıyla, Selenium anahtar kelimeleri odaklı bir çerçeve geliştirmektir. Çalışmada Selenium Webdriver'da anahtar kelime odaklı otomasyon çerçevesini Java ile birlikte nasıl tasarlayabileceklerini ve kullanabilecekleri konularına değinmişler. Sonrasında anahtar kelime testinde, her bir anahtar kelime bir fare tıklaması, bir menü öğesinin seçimi, tuş vuruşları, bir pencerenin açılması veya kapatılması veya bir başka eylem gibi tek bir test işlemine karşılık

geldiğini belirtmişlerdir. Anahtar kelime odaklı bir test, anahtar kelime biçimindeki test edilen uygulamadaki kullanıcı eylemlerini simule eden bir işlem dizisi olarak nitelendirmişlerdir. Çalışma sonunda Keyword Driven Framework kullanarak, yönlendirilen web sitesini otomatikleştirmek için birden fazla işlev yazmamışlardır. Bunun yerine anahtar kelimeleri excel dosyalarına belirli bir formatta tanımlamışlardır. Daha sonra bu excel dosyasına senaryo adımlarını yazmışlar ve geliştirdikleri program excelden verileri alıp kullanacak şekilde tasarlamışlardır. Sonrasında test altındaki uygulamaları tüm fonksiyonları, tablodan senaryoları adım adım talimat olarak almıştır. Bu şekilde, test senaryoları anahtar kelime odaklı çerçeve kullanılarak otomatik testlere dönüştürülerek testleri gerçekleştirmişlerdir.

3. MATERYAL ve METOT

3.1 Giriş

Çevik yazılım geliştirme yaklaşımlarında bazı yaklaşımlar bir öncekinin tamamlayıcısı olarak görülmektedir. Fakat her bir yaklaşım değişen ihtiyaçlara ve farklılaşan problemlere göre ortaya atılmıştır. Çevik yazılım yaşam döngüsünde (analiz, tasarım, geliştirme, test, ürün) süreçlerin merkezine kullanıcı davranışlarını odak alan Davranış Odaklı Geliştirme yaklaşımı diğerlerinden bu yönüyle farklılaşmaktadır. Geliştirilecek olarak yazılımın ya da uygulamanın en başında kullanıcı davranışlarına göre planlanması, analiz edilmesi, geliştirilmesi ve test edilmesi Davranış Odaklı Geliştirme yaklaşımının temel niteliğidir. Bu sayede başlangıçta kullanıcı davranışının her hangi bir fonksiyonel özelliği Given-When-Then formatında yazılarak gereksinim maddesi oluşturulmaktadır. Tasarım ve geliştirme bu gereksinime göre yapılmaktadır. Test aşamasında Given-When-Then formatında yazılan gereksinimin test kodlaması yapılmaktadır. Bu formatta yazılan test önemli ölçüde kolaylık sağlamaktadır. Aynı zamanda da zaman bakımından test faaliyetlerinde önemli ölçüde kazanç sağladığı görülmektedir. Bununla birlikte Davranış Odaklı Geliştirme yaklaşımında belirlenen Given-When-Then formatı bir standart oluşturmasından dolayı öğrenim ve kullanım kolaylığı da sağlamaktadır. Projenin her hangi bir aşamasında ekibe dâhil olan yeni bir üye Davranış Odaklı Geliştirme formatında yazılan senaryolara ve testlere göz attığında kolay anlaşılırlığı ve standartlığı sayesinde kolaylıkla sürece dâhil olur ve sistem hakkında bilgi sahibi olmaktadır. Bu çalışmada basit ve kolay anlaşılabilir olması, kullanılabilirlik seviyesinin yüksek olması ve belirli bir standarttı olmasından dolayı test otomasyonu geliştirmede Davranış Odaklı Geliştirme yaklaşımı kullanılmıştır.

Çalışmanın bu bölümünde manuel testlerin yürütülmesi ve Moodle uygulamasının yazılım test otomasyonu geliştirilmesi yapılmıştır. Yapılan bu çalışmalar öncesinde test otomasyonu geliştirmede kullanılacak yazılım dili ve geliştirme teknolojilerine değinilmiştir. Sonrasında otomatize edilecek Moodle uygulamasının kullanıcı arayüzlerinden söz edilmiştir. Bu aşamada var olan farklı rol ve yetkilere sahip kullanıcı rollerine (ziyaretçi, öğrenci yetkili, öğretmen yetkili) ait ekran görüntüleri eklenmiştir.

Farklı türlere ait kullanıcı arayüzlerinin fonksiyonel analizleri yapılarak kullanıcı test senaryoları yazılmıştır. Yazılan test senaryoları belirli ana modül ve alt modüller başlıkları altında gruplandırılmıştır.

Fonksiyonel analiz aşaması tamamlanıp kullanıcı gruplarına ait test senaryoları yazıldıktan sonra manuel yazılım test yapılması aşmasına geçilmiştir. Bu aşamada tüm test senaryolarının manuel testleri gerçekleşecek ve başarılı- tamamlandı, başarısız - tamamlanamadı, kısmen başarılı - tamamlandı olarak nitelendirilmiştir. Burada, başarısız – tamamlanamadı ve kısmen başarılı – tamamlandı şeklinde belirlenen test senaryolarına bulgular bölümünde detaylı olarak değinilmiştir. Manuel yazılım test aşaması tamamlandıktan sonra otomasyon yazılım test aşamasına geçilmiştir. Bu aşamada yazılan test senaryoları test otomasyonu geliştirme araçları ile otomatize edilmiştir. Otomatize edilen ve edilemeyen test senaryoları manuel test aşamasındaki gibi üç statü adı altında nitelendirilecek ve bulgular bölümünde detaylı olarak değinilmiştir.

3.2 Uygulamanın Amacı ve Yöntemi

Bu uygulamanın amacı Moodle uygulamasını kullanıcı ziyaretçi, öğrenci ve öğretmen kullanıcı rolleri açısından fonksiyonel olarak özelliklerinin çalışabilirliğini kontrol edecek test otomasyonunun geliştirilmesidir. Bu çalışma yazılım geliştirme ekibinden bağımsız olarak Davranış Odaklı Geliştirme yaklaşımıyla manuel ve otomasyon yazılım testlerini gerçekleştirilmiştir. Bu süreçte yazılım geliştirme ekibinden bağımsız olarak yapılan yazılım testlerinde yaşanılan zorluklar ve kolaylıkların neler olduğunu belirlemek amaçlanmıştır. Bu çalışmada edinilen deneyimler ile açık kaynak kodlu öğrenme yönetim sistemi uygulamalarının yazılım testlerinin nasıl yapılması gerektiği konusunda örnek çalışma olması amaçlanmaktadır.

Çalışmada Moodle yazılımının manuel ve otomasyon yazılım testleri gerçekleştirilmiştir. Sonrasında manuel test ve otomasyon yazılım test süreçlerinde deneyimlerin karşılaştırmaları yapılmaktadır. Çalışmada yazılım testleri için açık kaynak kodlu olması ve erişim kolaylığı sunulması nedeniyle Moodle yazılım

uygulamasının seçilmesine karar verildi. Çalışma kapsamın belirlenmesi, modüllerin oluşturulması, fonsiyonel analizlerin yapılması ve senaryoların oluşturulması aşamalarında alan uzmanlarının görüşlerine başvurulmuştur. Çalışma bu doğrultuda yürütülmüştür.

3.3 Yazılım Dili ve Geliştirme Araçları

Bu başlık altında tez çalışmasında geliştirilen test otomasyon uygulamasında kullanılan yazılım geliştirme araçlarından söz edilmiş ve tanımlamaları yapılmıştır. Test otomasyonu Ruby yazılım dilinin v2.5.3p105 sürümü ile geliştirilmiştir. Yazılım otomasyonu uygulama geliştirmede Ruby yazılım dili birlikte yazılım test otomasyonu için gerekli olan farklı kütüphane araçları kullanılmıştır. Bu araçlar Ruby kütüphanesinde gem olarak nitelendirilmektedir. Bu araçları Cucumber, Gherkin, Capybara, Selenium WebDriver olmaktadır. Yazılım test otomasyonu geliştirmede kullanılan gem araçlarının her birinin fonksiyonlarına ve kullanım amaçlarına aşağıdaki alt başlıklarda değinilmiştir.

Geliştirme Platformu : Cucumber v3.1.2, Gherkin v5.1.0, Capybara v3.9.0 , Selenium WebDriver v3.10.0
Yazılım Dili **:** Ruby v2.5.3p105
Kullanılacak IDE **:** RubyMine

3.3.1 Ruby

Ruby, 1995 yılında Yukihiro Matsumoto tarafından geliştirilmiştir. Nesne tabanlı olarak yorumlanan ruby okunabilirlik düzeyi oldukça yüksek bir programlama dili olmakla birlikte Python programlama diline benzeyen programlama dilidir. Ruby, nesneye yönelik programlama dili olarak sözdizimi itibarı ile Ada, Perl, Smalltalk gibi programlama dillerinden etkilenerek Python ile bazı ortak sözdizimi özellikleri bulunmaktadır. Genel özellikleri; nesneye yönelim, dört düzeyde değişken tanımlanabilmesi (Global, class, instance ve local), istisna işleme, yüksek taşınabilirlik, geniş standart kütüphane desteği ve Perl benzeri dil seviyesinde doğal düzenli

ifade desteğidir (İnt.Kyn.11).

Ruby'de sınıf ve metot tanımlamaları birtakım anahtar kelimelerle belirlenmektedir. Ruby'de Perl'in göre, değişken isimlerinin başında belirleyici karakterler ($, @, % gibi) kullanmak bir zorunluluk değildir. Python'un aksine girintilerin dilin sözdiziminde bir etkisi bulunmamaktadır. Ruby programlama dili farklı işletim sistemi için yazılmış olup işletim sistemlerin neredeyse hepsinde çalışmaktadır. Bunlardan bazıları; neredeyse tüm Unix türevleri, microsoft platformları, Mac OS X, BeOS, Amiga, OS/2, Symbian işletim sistemleridir (İnt.Kyn.12). Bu çalışmada yazılım geliştirme aracı olarak Ruby programlama dili kullanılacaktır.

3.3.2 Cucumber

Cucumber, programcı olmayanları hedef alan tipik bir etki alanına özgü bir dildir. Cucumber'in temel özelliği, Gherkin dilinin kullanımıyla bir test senaryosu üretmeyi kolaylaştırmaktadır. Cucumber düz metin şeklinde yazılan test senaryolarını Gherkin dilini kullarak otomatize testlere dönüştürülmesini sağlamaktadır. Cucumber testleri Features (Özellik) ve Scenarios (Senaryo) olarak iki kısımdan oluşmaktadır (Evgrafov *et al.* 2015).

Bu çalışmada Davranış Odaklı Geliştirme yaklaşımı yapısına uygun olması ve Ruby programlama dili ile uyumlu çalışmasından dolayı Cucumber dili kullanılmasına karar verilmiştir.

```
Feature: Anamenu Modulu ziyaretci kullanici senaryolari

  Scenario: site haberleri ekrani goruntuleme VTS008
    Given Moodle "http://uzeg.aku.edu.tr/" adresine gitmek istiyorum
    When Site Haberleri alanina tiklarsam
    Then Site Haberleri ekranini gorurum
```

Resim 3.1 Cucumber test senaryosu örneği

Yukarıdaki test senaryosu Cucumber ile çalışır ve bize ilgili çalışan otomasyon test senaryosuna ait test koşum sonuç bilgilerini vermektedir:

Testing started at 20:58 ...

C:\Ruby25\bin\ruby.exe -EUTF-8 C:\Ruby25\bin\cucumber C:/Users/fatih/Desktop/master_Moodle/features/A_visitor/Ziyaretci_Senaryolar.Feature --format Teamcity::Cucumber::Formatter --expand --name "^site haberleri ekrani goruntuleme VTS008$" --color -r features

1 scenario (1 passed)

3 steps (3 passed)

0m18.098s

3.3.3 Gherkin

Cucumber çalıştırılabilir senaryolar yazmak için Gherkin dilini kullanmaktadır. Gherkin dili birtakım anahtar kelimelerle ve formatı olan bir dil olmaktadır. Test otomasyonu yazan geliştiricilere doğal dil kullanarak senaryoları (davranışları) yazma olanağı sunmaktadır. Gherkin diline ait en çok kullanılan anahtar kelimeler; Feature, Background, Scenario, Given, When, Then, And, But olarak belirtilmektedir.

Test otomasyon projesinde her bir özellik (feature) için bir tane ".feature" uzantılı Gherkin dosyası olmaktadır. Özellik dosyasında sistemin ilgili niterliklerin tanımı ve bu niteliğe göre farklı durumlardaki davranışı senaryo adı altında yazılmaktadır. Yazılan senaryolar Given/When/Then formatında, "steps" denilen adımlarla belirlenmektedir. Given ile bir ön koşul, When ile olay, Then ile de sonuç tanımlanmaktadır (İnt.Kyn.13).

```
Given Moodle "http://uzeg.aku.edu.tr/" adresine gitmek istiyorum
And kullanicidi bilgisini "test" giresem
And sifre bilgisini "Test" olarak girersem
And Giris butonuna tiklarsam
And Sag ust kosede kullanici bilgisi ile giris yapildigini gorurum
And Instructional technology dersine tiklarsam
When Konu basliginin yanındaki devami butonuna tiklarsam
Then Konu icerigini gorurum
And Logout icin anasayfaya gidersem
And Logout olmak icin cikis butonuna tiklarim
```

Resim 3.2 Gherkin diline ait en çok kullanılan anahtar kelirmeler

Davranış Odaklı Geliştirme yaklaşımında belirtilen kullanıcı davranışları formatı

Gherkin diline ait anahtar kelimeler kullanılarak yazılmıştır. Bunınla birlikte Gherkin dili kütüphanesi Ruby ve diğer kütüphalerle uyumlu çalışmaktadır. Bu nedenlerden dolayı bu çalışmada Gherkin dili kullanılmasına karar verilmiştir.

3.3.4 Capybara

Capybara, kullanıcıların uygulamalarda nasıl etkileşime gireceğini ve kullanıcı işlemlerini simule eden bir kütüphanedir. Fonksiyonel testler oluşturmak için kullanılan web tabanlı uygulama otomasyon çerçevesi olarak nitelendirilmektedir. Test otomasyonu geliştiricilerin hedeflediği temel özellik testleri kolay yazmak ve bakımı kolay testlere sahip olmaktır (İnt.Kyn.14).

Capybara web tabanlı bir sürücünün üzerine kullanılmak üzere yapılmış bir kütüphanedir. Temel web sürücüsü tarafından yürütülen eylemleri tanımlamak için kullanılan kullanıcı dostu bir etki alanına özel dil sunmaktadır. Sayfa bu özel dili (ve altındaki web sürücüsü) kullanılarak yüklendiğinde, Capybara ilgili öğeyi bulma ve tıklama, bağlantı vb. gibi eylemleri gerçekleştirmeye çalışmaktadır (Evgrafov *et al*. 2015).

```
Given(/^Moodle "([^"]*)" adresine gitmek istiyorum$/) do |anasayfa|
  visit anasayfa
end

When(/^Site Haberleri alanina tiklarsam$/) do
  box=find_by_id("inst1")
  box.find("a", :text =>"Site haberleri").click
end

Then(/^Site Haberleri ekranini gorurum$/) do
  find_by_id("content", :text =>"Genel haberler ve duyurular")
end
```

Resim 3.3 Örnek Capybara özelliği

Davranış Odaklı Geliştirme yapısına, Ruby programla dili ve diğer kütüphalerle uyumlu olmasından dolayı bu çalışmada Cabybara kütüphanesi kullanılmasına karar verilmiştir.

3.3.5 Selenium Webdriver

Selenium WebDriver, daha fazla kontrol ve standart yazılım geliştirme uygulamalarının uygulanmasına izin vermek için çeşitli dillerde API'ler sunmaktadır. Selenyum WebDriver bir web sayfasında tıklama, seçim gibi otomatik işlemleri yapmanızı sağlayan bir tarayıcı otomasyon aracıdır. Özellikle bir web uygulamasının sınama gereksinimlerine yönelik zengin bir test işlevleri kümesi sağlamaktadır. Bu işlemler son derece esnektir ve kullanıcı arayüzü öğelerini bulmak için birçok seçeneğe izin vermektedir. Selenyum WebDriver'ın en önemli özelliklerinden biri, testleri birden çok tarayıcı platformunda yürütme desteği sunmasıdır (Singla and Kaur 2014).

Bu çalışmada test senaryolarının kodlanmasından sonra otomatik senaryolar çalıştırıldığında senaryodaki belirtilen tıklama, yazma, seçme işlemleri Selenium Webdriver aracılığıyla yapılmaktadır. Bu nedenle çalışmada Selenium Webdriver kullanılmasına karar verilmiştir.

3.3.5 RubyMine

Rubymine, Ruby ve Rails için entegre bir geliştirme ortamıdır. JetBrains tarafından geliştirilen Rubymine kodlama yaparken gereksiz kod yazımından kaçınılmasına yardımcı olur ve kod geliştirmede daha hızlı gezinmeyi sağlamaktadır. Örneğin, parantezlerin veya tırnakların kapatılması unutulduğu zaman Rubymine potansiyel hataları vurgular ve geliştirilen kodu biçimlendirmektedir. RubyMine, IntelliJ IDEA platformu üzerine kuruludur. Bir IDE'den beklenilen editör, hata ayıklama araçları, kaynak kontrol entegrasyonu, kod tamamlama vb. tüm temel özellikleri ve Ruby'ye özel birçok özellik sunmaktadır. Kodlama yardımı ve grafiksel kullanıcı arayüz tabanlı bir test çalıştırıcısı ile RSpec, Cucumber, Shoulda, birim test ve testlerini oluşturulması ve çalıştırılmasında kullanıcı dostu olmaktadır (İnt.Kyn.15).

RubyMine geliştirme ortamı Ruby programlama dili, Cucumber, Gherkin, Capybara, Selenium WebDrive vb. kütüpleri bir arada kullanılmasına ve test otomasyonu geliştirmesine kolaylık sağlamaktadır. Bununla birlikte Rubymine üniversite

öğrencilerine ücretsiz erişim ve kullanım hakkı sunmaktadır. Bu nedenle çalışmada RubyMine tümleşik geliştirme ortamının kullanılmasına karar verilmiştir.

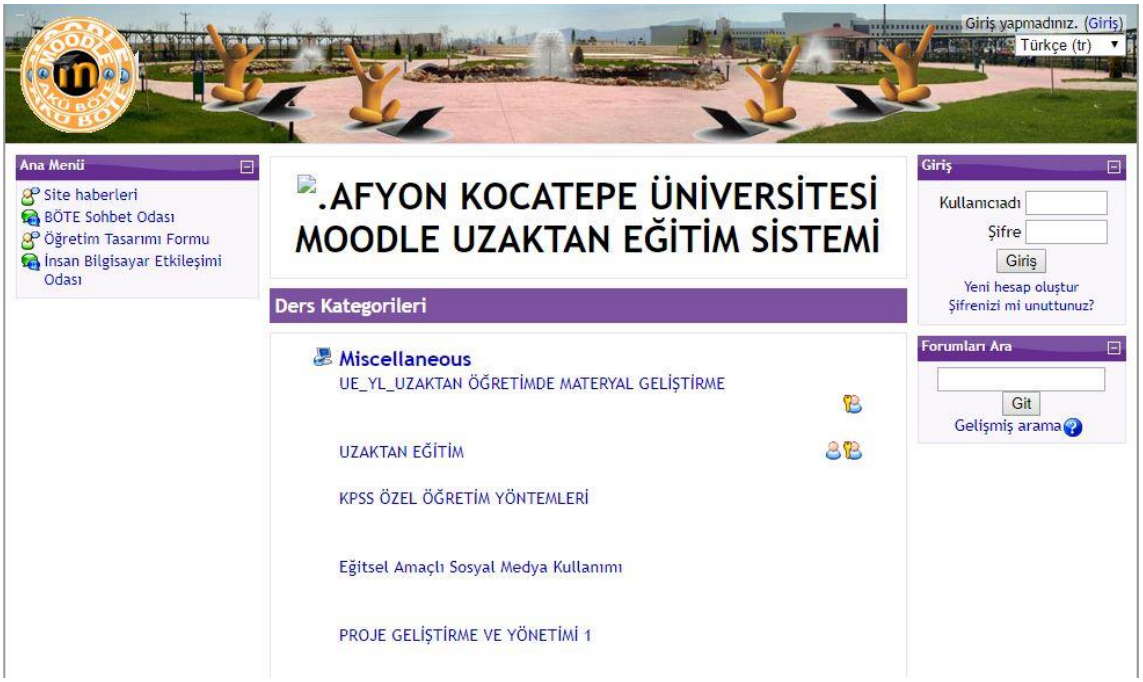
3.4 Moodle Uygulaması Kullanıcı Arayüzü ve Modülleri

Bu başlık altında Moodle uyalamasının yazılacak test senaryolara ilişkin ziyaretçi, öğrenci ve öğretmen kullanıcı rollerine ait uygulama arayüzlerine değinilmiştir. Her bir arayüz kendi içerisinde rol, yetki tanımına göre modüllerden oluşmaktadır. Öğrenci ve öğretmen rollerine ait arayüzler ana modüller alt modüllerden oluşmaktadır. İleriki aşama olan fonksiyonel analiz bölümünde yazılacak test senaryolarının daha anlaşılabilir, sınıflanabilir olması bakımından her bir kullanıcı rolünün arayüzündeki ana modüller ve alt modülleri gruplanarak çalışmaya devam edilmiştir.

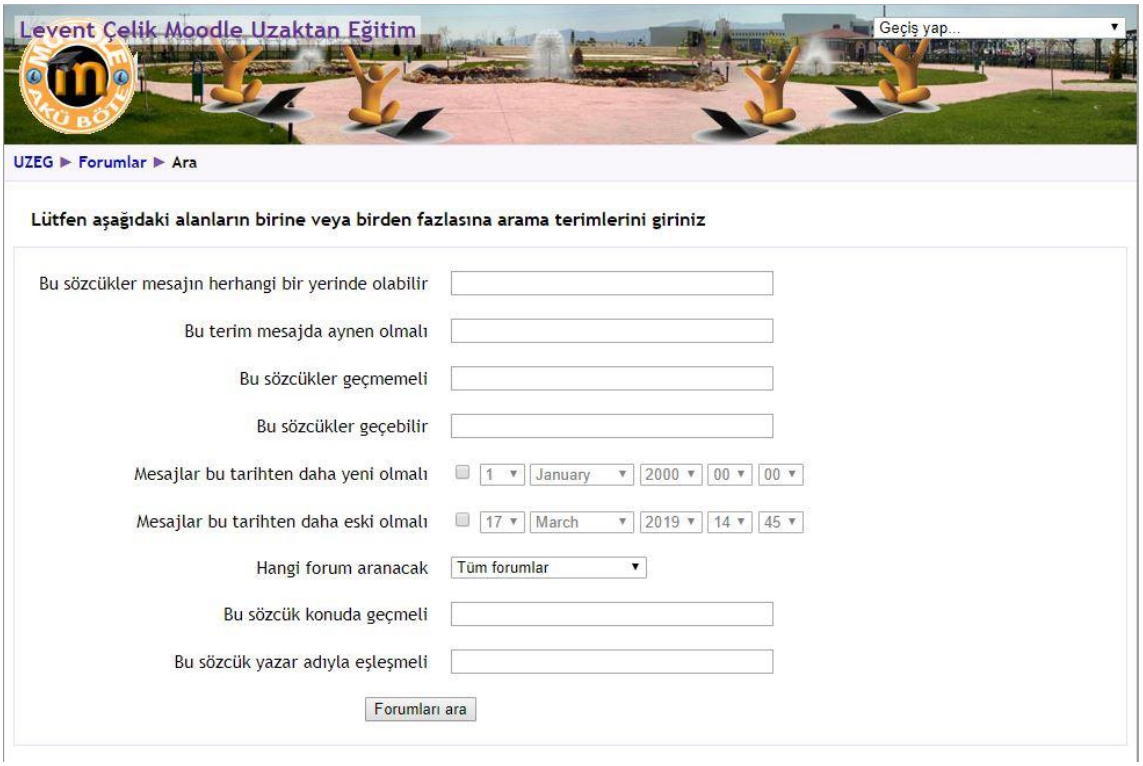
3.4.1 Ziyaretçi Kullanıcı Rolü Arayüzü ve Modülleri

Ziyaretçi kullanıcı arayüzü sisteme üyeliği olmayan kullanıcı rolü olarak nitelendirilmektedir. Moodle uygulamasında üye olmayan kullanıcının görebileceği ekranlar, modüller ve yapabileceği işlemler mevcuttur. Ziyaretçi arayüzünde kullanıcının görebileceği ve kullanabileceği modüller:

- Giriş Modülü
- Forumlarda Ara Modülü
- Ana Menü Modülü
- Ders Kategorileri Modülü



Resim 3.4 Ziyaretçi kullanıcı ekran modülleri



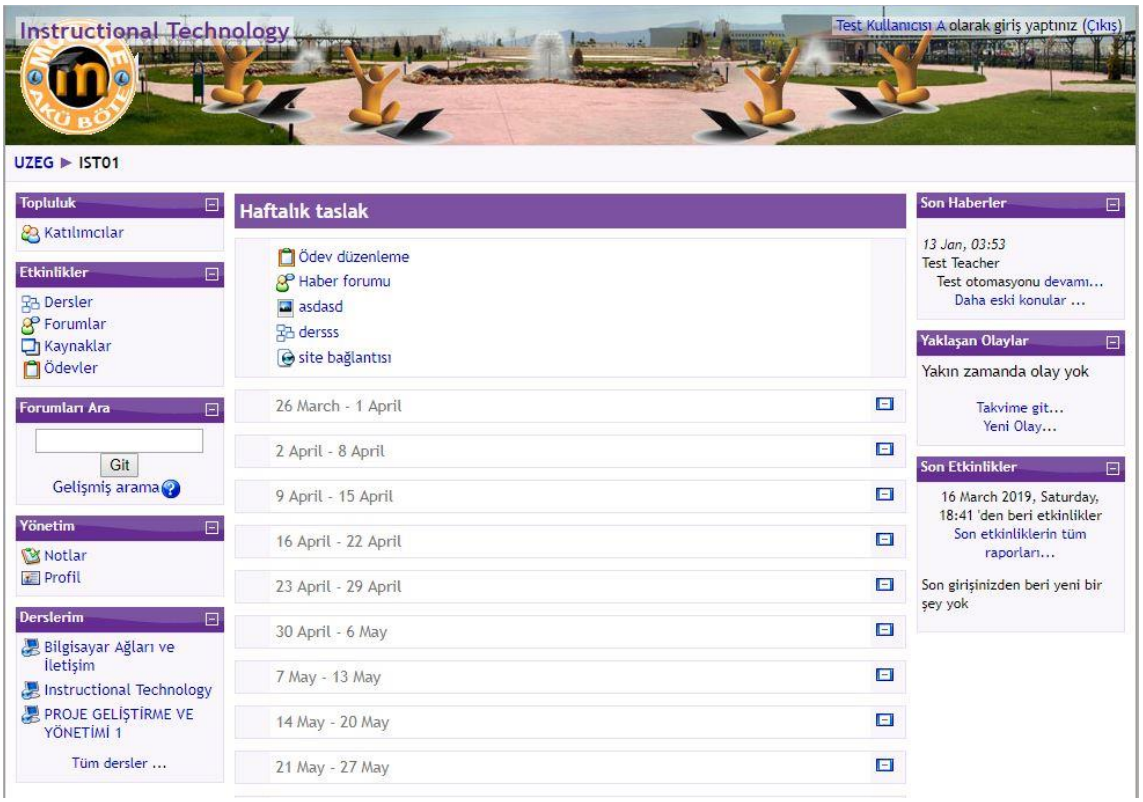
Resim 3.5 Ziyaretçi kullanıcı gelişmiş arama ekranı

3.4.2 Öğrenci Kullanıcı Rolü Arayüzü ve Modülleri

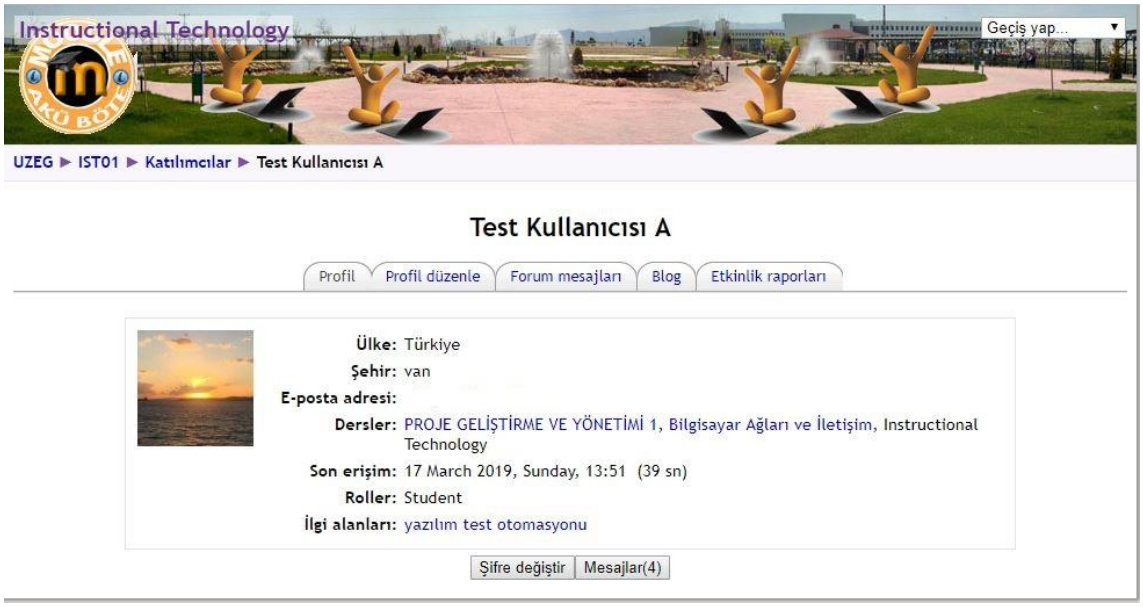
Öğrenci kullanıcı arayüzü sisteme üyeliği olan, kullanıcı rolü ve yetkisi öğrenci rolü olarak tanımlanan üye kullanıcı olmaktadır. Moodle uygulamasında öğrenci rolü ile tanımlı kullanıcının görebileceği ekranlar, modüller ve yapabileceği işlemler ziyaretçi kullanıcıya göre farklılık göstermektedir. Öğrenci arayüzünde kullanıcının görebileceği ve kullanabileceği ana modüllerle birlikte bazı ana modüllere ait alt modüllerde bulunmaktadır. Öğrenci arayüzünde kullanıcının görebileceği ekranlar ve kullanabileceği modül grupları:

- Öğrenci Profil Sayfası (Ekranı)
- Blog Menüsü Modülü
- Forumlarda Ara Modülü
- Ana Menü Modülü
- Derslerim Modülü
    - Haftalık Taslak Alt Modülü
    - Topluluk Alt Modülü
    - Etkinlikler Alt Modülü
    - Forumlarda Ara Alt Modülü
    - Yönetim Alt Modülü
    - Derslerim Alt Modülü
    - Son Haberler Alt Modülü
    - Yaklaşan Olaylar Alt Modülü
    - Son Etkinlikler Alt Modülü

Resim 3.6 Öğrenci yetkili kullanıcı ekran modülleri



Resim 3.7 Öğrenci yetkili kullanıcı derslerim modülü alt modülleri



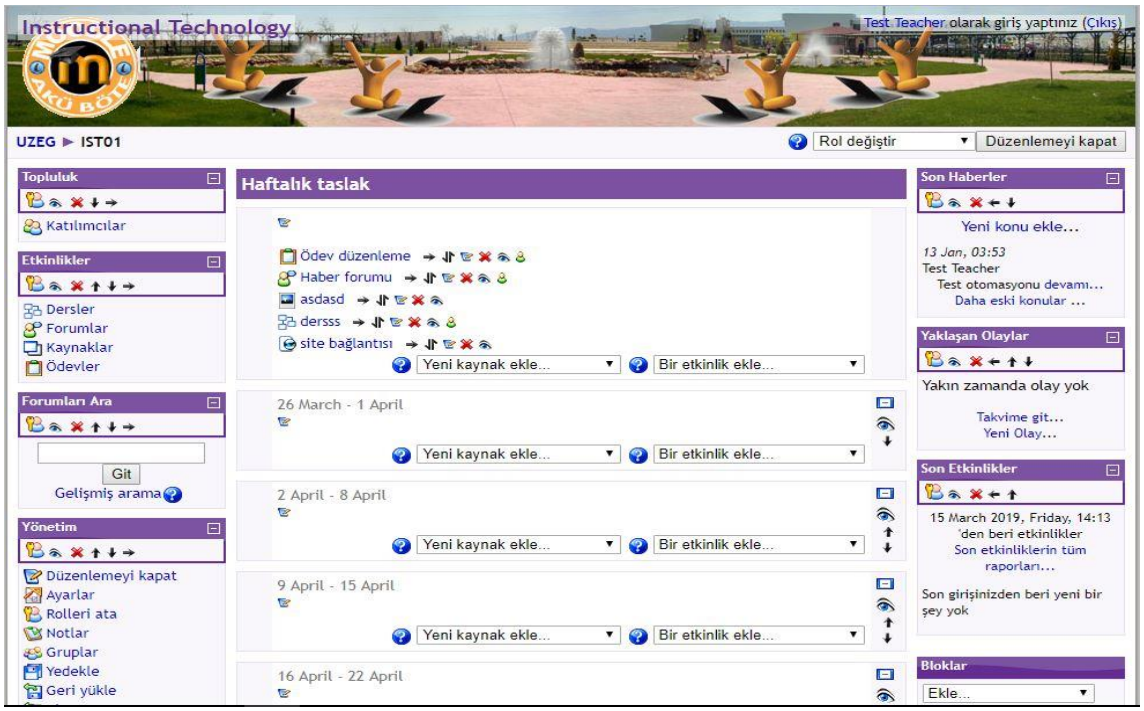
Resim 3.8 Öğrenci yetkili kullanıcı profil ekranı

3.4.3 Öğretmen Kullanıcı Rolü Arayüzü ve Modülleri

Öğretmen kullanıcı arayüzü sisteme üyeliği olan, kullanıcı rolü ve yetkisi öğretmen rolü olarak tanımlanan üye kullanıcı olmaktadır. Moodle uygulamasında öğretmen rolü ile tanımlı kullanıcının görebileceği ekranlar, modüller ve yapabileceği işlemler ziyaretçi ve öğrenci kullanıcılarına göre farklılık gösterir. Öğretmen arayüzünde kullanıcının görebileceği ve kullanabileceği ana modüllerle birlikte bazı ana modüllere ait alt modüllerde bulunmaktadır. Öğretmen arayüzünde kullanıcının görebileceği ekranlar ve kullanabileceği modül grupları:

- Öğretmen profil Sayfası (Ekranı)
- Blog Menüsü Modülü
- Forumlarda Ara
- Ana Menü Modülü
- Derslerim Modülü
    - Haftalık Taslak Modülü
    - Topluluk Modülü
    - Etkinlikler Modülü
    - Forumlarda Ara Modülü

- Yönetim Modülü
- Derslerim Modülü
- Son Haberler Modülü
- Yaklaşan Olaylar Modülü
- Son Etkinlikler Modülü
- Bloklar Modülü



Resim 3.9 Öğretmen yetkili kullanıcı derslerim modülü alt modülleri

Resim 3.10 Öğretmen yetkili kullanıcı profil ekranı

3.5 Fonksiyonel Analiz

Bu bölümde Moodle uygulamasında ziyaretçi, öğrenci ve öğretmen kullanıcı rollerine ait arayüzlerin ve modüllerin fonksiyonel özelliklerine ait analizler yapılmıştır. Yapılan bu analiz çalışması sonrasında belirlenen fonksiyonlara ait testleri için oluşturulan test senaryoları yazılmıştır. Yapılacak olan fonksiyonel analiz çalışmasında alan uzmanlarından görüşler alınarak Moodle yazılımının ziyaretçi, öğrenci ve öğretmen kullanıcı rollerinde sistemin çalışmasını aksatmayacak olan temel fonksiyonlar belirlenerek yürütülmüştür. Tez çalışmasında sınırlı zaman ve kaynak durumları göz önüne alındığında etkili bir kaynak ve zaman yönetimi için çalışma kapsamı bu haliyle sınırlandırılmıştır.

3.5.1 Ziyaretçi Kullanıcı Rolü Test Senaryoları

Ziyaretçi kullanıcı rolü test senaryoları oluşturulmadan önce alan uzmanlarının görüşlerine başvurulmuştur. Bu doğrultuda fonksiyonel özellikler belirlenip senaryoları yazılmıştır. Moodle uygulamasında bir ziyaretçi kullanıcının kullanabileceği alanlar beş farklı modül başlığı altında gruplanmıştır. Ziyaretçi kullanıcıya ait test senaryoları VTS ön kodu ile kodlanmıştır. Ön kod sonrasında gelen rakamlar senaryo numarasını ifade etmektedir. Örneğin VTS010 olarak belirtilen test senaryosu ziyaretçi kullanıcı on numaralı test senaryosunu belirtmektedir. Belirlenen fonksiyonel özelliklere ait test senaryoların örnekleri ilgili başlıklar altında belirtilmiştir. Ziyaretçi kullanıcı rolü test senaryoların tamamı ekler bölümünde yer almaktadır.

VTS001: Bir Moodle ziyaretçi kullanıcı olarak Moodle web adresine gitmek istiyorum böylece Moodle ana sayfasını görüntülerim.
VTS002: Bir Moodle ziyaretçi kullanıcı olarak “Yeni hesap oluştur” alanına tıklamak istiyorum böylece yeni hesap oluştur ekranını görüntülerim.
VTS008: Bir Moodle ziyaretçi kullanıcı olarak “Site haberleri” alanına tıklamak istiyorum böylece site haberleri sayfasını görürüm.
VTS009: Bir Moodle ziyaretçi kullanıcı olarak “Böte Sohbet Odası” alanına tıklamak istiyorum böylece böte sohbet odası sayfasını görürüm.

VTS022: Bir Moodle ziyaretçi kullanıcı olarak “Miscellaneous” ders kategorisi alanına tıklamak istiyorum böylece miscellaneous ders kategorisi sayfasını görüntülerim.

3.5.2 Öğrenci Kullanıcı Rolü Test Senaryoları

Öğrenci kullanıcı rolü test senaryoları oluşturulmadan önce alan uzmanlarının görüşlerine başvurulmuştur. Bu doğrultuda fonksiyonel özellikler belirlenmiştir. Belirlenen kapsama göre senaryolar yazılmıştır. Moodle uygulamasında bir öğrenci yetkisine sahip kullanıcının kullanabileceği alanlar yedi farklı ana modül başlığı altında gruplanmıştır. Öğrenci yetkili kullanıcıya ait test senaryoları STS ön kodu ile kodlanmıştır. Ön kod sonrasında gelen rakamlar senaryo numarasını ifade etmektedir. Örneğin STS025 olarak belirtilen test senaryosu öğrenci yetkili kullanıcı yirmi beş numaralı test senaryosunu belirtmektedir. Belirlenen fonksiyonel özelliklere ait test senaryoların örnekleri ilgili başlıklar altında belirtilmiştir. Öğrenci kullanıcı rolü test senaryoların tamamı ekler bölümünde yer almaktadır.

Profil Sayfası Senaryoları

STS001: Bir Moodle öğrenci yetkili kullanıcı olarak ekranın sağ üst köşesindeki üye kullanıcı “Ad Soyad” yazan alana tıklamak istiyorum böylece kullanıcı profili ekranını görürüm.
STS034: Bir Moodle öğrenci yetkili kullanıcı olarak “Site haberleri” alanına tıklamak istiyorum böylece site haberleri sayfasını görürüm.
STS035: Bir Moodle öğrenci yetkili kullanıcı olarak “Böte Sohbet Odası” alanına tıklamak istiyorum böylece böte sohbet odası sayfasını görürüm.
STS039: Bir Moodle öğrenci yetkili kullanıcı olarak “Haber forumu” alanına tıklamak istiyorum böylece haber forumunu görürüm.
STS040: Bir Moodle öğrenci yetkili kullanıcı olarak “Proje nedir” alanına tıklamak istiyorum böylece proje nedir ders materyali içeriğini görürüm.

3.5.3 Öğretmen Kullanıcı Rolü Test Senaryoları

Öğretmen kullanıcı rolü test senaryoları oluşturulmadan önce alan uzmanlarının görüşlerine başvurulmuştur. Bu doğrultuda fonksiyonel özellikler belirlenmiştir. Belirlenen kapsama göre senaryolar yazılmıştır. Moodle uygulamasında bir öğretmen yetkisine sahip kullanıcının kullanabileceği alanlar yedi farklı ana modül başlığı altında gruplanmıştır. Öğretmen yetkili kullanıcıya ait test senaryoları TTS ön kodu ile kodlanmıştır. Ön kod sonrasında gelen rakamlar senaryo numarasını ifade etmektedir. Örneğin TTS100 olarak belirtilen test senaryosu öğretmen yetkili kullanıcı yüznumaralı test senaryosunu belirtmektedir. Belirlenen fonksiyonel özelliklere ait test senaryoların örnekleri ilgili başlıklar altında belirtilmiştir. Öğretmen kullanıcı rolü test senaryoların tamamı ekler bölümünde yer almaktadır.

TTS001: Bir Moodle öğretmen yetkili kullanıcı olarak ekranın sağ üst köşesindeki üye kullanıcı adı soyadı yazan alana tıklamak istiyorum böylece kullanıcı profil ekranını görürüm.
TTS020: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili "Ders etkinliğine" tıklamak istiyorum böylece ders kaynağının açıldığını görürüm.
TTS050: Bir Moodle öğretmen yetkili kullanıcı olarak katılımcılar tabında da geçerli rolü "Öğretmen yetkili" olarak seçmek istiyorum böylece öğretmen yetkili rolündeki katılımcıları görürüm.
TTS082: Bir Moodle öğretmen yetkili kullanıcı olarak "Düzenlemeyi aç" alanına tıklamak istiyorum böylece düzenleme araçlarını görürüm.
TTS083: Bir Moodle öğretmen yetkili kullanıcı olarak "Düzenlemeyi kapat" alanına tıklamak istiyorum böylece düzenleme araçlarının kapandığını görürüm.

3.6 Manuel Yazılım Test Süreci

Bu bölünde fonksiyonel analiz aşamasında yapılan çalışma sonucu oluşturulan test senaryolarının manuel test süreçleri gerçekleştirilmiştir. Manuel test sürecinde senaryolarda belirtilen fonksiyonel özelliklerin testleri yapılmıştır. Manuel test süreci öncelikli olarak sisteme üye olmayan ziyaretçi kullanıcının test senaryolarının manuel

testleri yapılmıştır. Bu bölümdeki testler tamamlandıktan sonra öğrenci yetkili kullanıcının testleri yapılmıştır. Öğrenci yetkili kullanıcı test senaryolarına ait testler tamamlandıktan sonra da öğretmen yetkili kullanıcısına ait test senaryolarının manuel test süreçleri yapılmıştır. Manuel yazılım testleri bu şekilde planlanmıştır. Manuel testlerin yapılabilmesi üzere Moodle yazılımında bir ders oluşturulmuştur. Fonksiyonel analiz aşamasında yazılan test senaryolarının tamamına yakını bu ders içerisinde tamamlanmak üzere testler tasarlanmıştır.

3.6.1 Ziyaretçi Kullanıcı Rolü Manuel Testleri

Ziyaretçi kullanıcının manuel testleri fonksiyonel analiz bölümünde belirtilen test senaryolarına göre yapılmıştır. Manuel testlere ilk olarak giriş modülündeki test senaryoları ile başlanmıştır. Giriş modülüne ait yedi adet test senaryonun testleri tamamlanmıştır. Sonrasında ana menü modülü testleri ile manuel testlere devam edilmiştir. Ana menü modülündeki dört adet test senaryonun testleri tamamlanmıştır. Forumlarda ara modülündeki on adet test senaryosu ve ders kategorileri modülüne ait beş adet test senaryolarının testleri de tamamlanmıştır.

Tablo 3.1 Ziyaretçi kullanıcı rolü modül bazlı manuel test durumları

Modül Adı	**Senaryo Sayısı**	**Test Sonucu**
Giriş	7	Başarılı-Tamamlandı
Ana Menü	4	Başarılı-Tamamlandı
Forumlarda Ara	10	Başarılı-Tamamlandı
Ders Kategorileri	5	Başarılı-Tamamlandı
Toplam Senaryo	**26**	

Ziyaretçi kullanıcıya ait yirmi altı test senaryosunun testleri tamamlandı ve bu kullanıcı rolüne ait manuel test aktivitesi sonlandırılmıştır. Bu bölümdeki test senaryolarına ait bilgiler Tablo 3.1 de yer almaktadır.

3.6.2 Öğrenci Kullanıcı Rolü Manuel Testleri

Öğrenci yetkili kullanıcının manuel testleri fonksiyonel analiz bölümünde belirtilen test senaryolarına göre yapılmıştır. Manuel testlere ilk olarak profil sayfasına ait test

senaryoları ile başlanmıştır. Profil sayfasına ait yirmi adet test senaryonun testleri tamamlanmıştır. Sonrasında blog modülü testleri ile manuel testlere devam edilmiştir. Blog modülündeki on adet test senaryonun testleri tamamlanmıştır. Forumlarda ara modülündeki üç adet test senaryosu ve ana menü modülüne ait dört adet test senaryolarının testleri de tamamlanmıştır. Son olarak derslerim ana modülü ve bu modüle ait alt modüllerle birlikte kırk yedi test senaryosunun testleri tamamlanmıştır.

Tablo 3.2 Öğrenci yetkili kullanıcı rolü modül bazlı manuel test durumları

Modül Adı	**Senaryo Sayısı**	**Test Sonucu**
Profil Sayfası	20	Başarılı-Tamamlandı
Blog Menüsü	10	Başarılı-Tamamlandı
Forumlarda Ara	3	Başarılı-Tamamlandı
Ana Menü	4	Başarılı-Tamamlandı
Derslerim	47	Başarılı-Tamamlandı
Toplam Senaryo	**84**	

Öğrenci yetkili kullanıcıya ait seksen dört test senaryosunun testleri tamamlandı ve bu kullanıcı rolüne ait manuel test aktivitesi sonlandırılmıştır. Bu bölümdeki test senaryolarına ait bilgiler Tablo 3.2 de yer almaktadır.

3.6.3 Öğretmen Kullanıcı Rolü Manuel Testleri

Öğretmen yetkili kullanıcının manuel testleri fonksiyonel analiz bölümünde belirtilen test senaryolarına göre yapılmıştır. Manuel testlere ilk olarak profil sayfasına ait test senaryoları ile başlanmıştır. Profil sayfasına ait beş adet test senaryonun testleri tamamlanmıştır. Sonrasında blog modülü testleri ile manuel testlere devam edilmiştir. Blog modülündeki beş adet test senaryonun testleri tamamlanmıştır. Forumlarda ara modülündeki üç adet test senaryosu ve ana menü modülüne ait dört adet test senaryolarının testleri de tamamlanmıştır. Son olarak derslerim ana modülü ve bu modüle ait alt modüllerle birlikte yüz on altı test senaryosunun testleri tamamlanmıştır.

Tablo 3.3 Öğretmen yetkili kullanıcı rolü modül bazlı manuel test durumları

Modül Adı	Senaryo Sayısı	Test Sonucu
Profil Sayfası	5	Başarılı-Tamamlandı
Blog Menüsü	5	Başarılı-Tamamlandı
Forumlarda Ara	3	Başarılı-Tamamlandı
Ana Menü	4	Başarılı-Tamamlandı
Derslerim	116	Başrılı-Tamamlandı
Toplam Senryo	**133**	

Öğretmen yetkili kullanıcıya ait yüz otuz üç test senaryosunun testleri tamamlandı ve bu kullanıcı rolüne ait manuel test aktivitesi sonlandırılmıştır. Bu bölümdeki test senaryolarına ait bilgiler Tablo 3.3 de yer almaktadır.

Ziyaretçi rolü, öğrenci rolü ve öğretmen rolü kullanıcılara ait test senaryolarına göre manuel testler koşumları yapılarak manuel test süreci tamamlanmıştır. Bu süreçte tamamlanan testler kullanıcı grupları, senaryo grup kodu, senaryo sayısı ve test sonuçlarına ait veriler tablo 3.4 de belirtilmiştir.

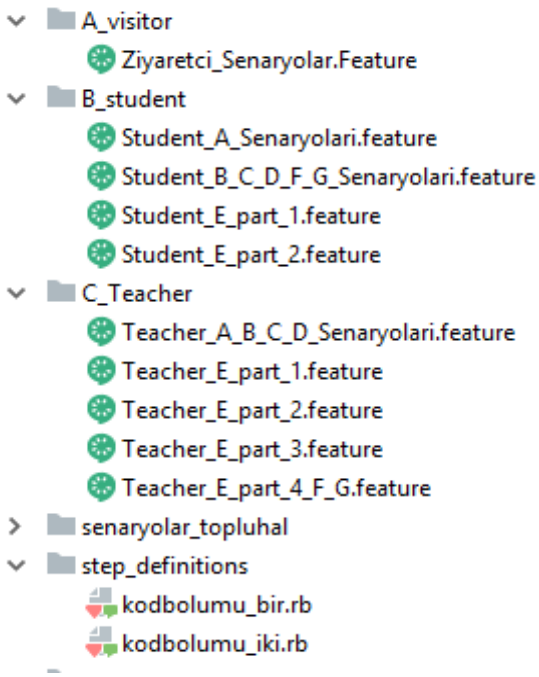
Tablo 3.4 Grup bazlı manuel test durumları

Kullanıcı Grubu Adı	Test Senaryo Grup Kodu	Senaryo Sayısı	Test Sonucu
Ziyaretçi Kullanıcı Rolü	VTS	26	Başarılı-Tamamlandı
Öğrenci Kullanıcı Rolü	STS	84	Başarılı-Tamamlandı
Öğretmen Kullanıcı Rolü	TTS	133	Başarılı-Tamamlandı
Toplam Senaryo		**243**	

3.7 Yazılım Test Otomasyonu Süreci

Bu bölümde fonksiyonel analiz bölümünde oluşturulan test senaryoları Davranış Güdümlü Geliştirme yaklaşımıyla Gherkin, Cucumber, Capybara yapılarını kullanarak Ruby programlama dilinde kodlanarak otomatize edilmiştir. Otomatize edilen test senaryoları kullanıcı grup bazlı (ziyaretçi, öğrenci ve öğretmen) olarak gruplanmıştır. Ziyaretçi kullanıcı rolü için bir, öğrenci kullanıcı rolü için dört ve öğretmen kullanıcı rolü için beş özellik (feature) dosya türü (Resim 3.11) oluşturulmuştur. Given, When,

Then formatında düzenlenen test senaryolar bu özellik dosyalarına yazılmıştır. Test senaryolarının kodlarını -.rb uzantılı dosyalara yazılmıştır.



Resim 3.11 Feature ve rb dosyaları

3.7.1 Ziyaretçi Kullanıcı Rolü Otomasyon Testleri

Bu bölümde ziyaretçi kullanıcının test senaryoları otomatize edilmiştir. Ziyaretçi kullanıcı rolünde bir adet özellik dosyası oluşturulmuştur. Bu kullanıcı rolüne ait yirmi altı adet test senaryosu Given, When, Then formatında düzenlenerek bu özellik dosyalarına yazılmıştır. Özellik dosyasındaki senaryo formatına karşılık gelen kodlar ilgili -.rb uzantılı dosyaya yazılmıştır. Yazılan senaryoların bir örneği aşağıda verilmiştir.

Tablo 3.5 de görüldüğü üzere ilk kısımda VT008 numaralı site haberlerini ekranını görüntüleme senaryosu metin olarak yer almaktadır. Tablonun ikinci bölümde metin olan yazılan senaryo Seneario, Given, When, Then formatında dönüştürülerek yazılmıştır. Daha sondasında bu bölümdeki satırların her birinin fonksiyonlarını gerçekleştirecek kodlar yazılmıştır. Yazılan kodlar tablonun üçüncü bölümünde yer görülmektedir. Senaryo çalıştırıldığında ilgili web adresi açma, site haberleri alanına

tıklama, site haberleri ekran görüntüsünün açılması ve kontrolü işlemleri otomatik yapılmaktadır.

Tablo 3.5 VTS008 numaralı senaryonun otomasyon örneği

Bir Moodle ziyaretçi kullanıcı olarak "Site haberleri" alanına tıklamak istiyorum böylece site haberleri sayfasını görürüm.
Scenario: site haberleri ekrani goruntuleme VTS008 **Given** Moodle "**http://uzeg.aku.edu.tr/**" adresine gitmek istiyorum **When** Site Haberleri alanina tiklarsam **Then** Site Haberleri ekranini gorurum
Given(/^Moodle "([^"]*)" adresine gitmek istiyorum$/) **do** \|*anasayfa*\| visit *anasayfa* **end** *When*(/^Site Haberleri alanina tiklarsam$/) **do** *box*=find_by_id(**"inst1"**) *box*.*find*(**"a"**, **:text** =>**"Site haberleri"**).click **end** *Then*(/^Site Haberleri ekranini gorurum$/) **do** find_by_id(**"content"**, **:text** =>**"Genel haberler ve duyurular"**) **end**

3.7.2 Öğrenci Kullanıcı Rolü Otomasyon Testleri

Bu bölümde öğrenci yetkili kullanıcının test senaryoları otomatize edilmiştir. Öğrenci yetkili kullanıcı rolünde dört adet özellik (feature) dosyası oluşturulmuştur. İlk özellik dosyasında profil sayfası senaryoları yazılmıştır. İkinci özellik dosyasına blog, forumlarda ara ve ana menü senaryoları yazılmıştır. Üç ve dördüncü özellik dosyalarına derslerim modülü senaryoları yazılmıştır. Öğrenci yetkili kullanıcı rolüne ait seksen dört adet test senaryosu Given, When, Then formatında düzenlenerek bu özellik dosyalarına yazılmıştır. Özellik dosyasındaki senaryo formatına karşılık gelen kodlar ilgili -.rb uzantılı dosyaya yazılmıştır. Yazılan senaryoların bir örneği aşağıda verilmiştir.

Tablo 3.6 da görüldüğü üzere ilk kısımda öğrenci yetkili kullanıcının STS001 numaralı senaryosunun metin olarak yer almaktadır. Tablonun ikinci bölümde metin olan yazılan senaryo Seneario, Given, When, Then, And formatında dönüştürülerek yazılmıştır. Daha sondasında bu bölümdeki satırların her birinin fonksiyonlarını gerçekleştirecek kodlar yazılmıştır. Yazılan kodlar tablonun üçüncü bölümünde yer görülmektedir. Senaryo çalıştırıldığında web tarayıcısı otomatik açılır ve sonrasında ilgili web adresinin açılması, kullanıcı adı ve şifrenin girilmesi, giriş butonuna tıklama işlemi ve sayfa kontrolü yapılması işlemleri otomatik yapılmaktadır.

Tablo 3.6 STS001 numaralı senaryonun otomasyon örneği

Bir Moodle öğrenci yetkili kullanıcı olarak ekranın sağ üst köşesindeki üye kullanıcı "Ad Soyad" yazan alana tıklamak istiyorum böylece kullanıcı profili ekranını görürüm.
Scenario: Student kullanici profili goruntuleme STS001 **Given** Moodle "**http://uzeg.aku.edu.tr/**" adresine gitmek istiyorum **And** kullanicidi bilgisini "**kullanıcıadı**" giresem **And** sifre bilgisini "**şifre**" olarak girersem **When** Giris butonuna tiklarsam **Then** Sag ust kosede kullanici bilgisi ile giris yapildigini gorurum

```
Given(/^Moodle "([^"]*)" adresine gitmek istiyorum$/) do |anasayfa|
  visit anasayfa
end
And(/^kullanicidi bilgisini "([^"]*)" giresem$/) do |username|
  find_by_id("login_username").set username
end
And(/^sifre bilgisini "([^"]*)" olarak girersem$/) do |password|
  find_by_id("login_password").set password
end
When(/^Giris butonuna tiklarsam$/) do
  box=find("div.c1.btn")
  box.find("input").click
end
Then(/^Sag ust kosede kullanici bilgisi ile giris yapildigini gorurum$/) do
  find("div.logininfo", :text =>"olarak giriş yaptınız")
end
```

3.7.3 Öğretmen Kullanıcı Rolü Otomasyon Testleri

Bu bölümde öğretmen yetkili kullanıcının test senaryoları otomatize edilmiştir. Öğretmen yetkili kullanıcı rolünde beş adet özellik (feature) dosyası oluşturulmuştur. İlk özellik dosyasında profil sayfası, blog, forumlarda ara ve ana menü senaryoları yazılmıştır. Diğer özellik dosyalarına derslerim modülü senaryoları yazılmıştır. Öğretmen yetkili kullanıcı rolüne ait yüz otuz üç adet test senaryosu Given, When, Then formatında düzenlenerek bu özellik dosyalarına yazılmıştır. Özellik dosyasındaki senaryo formatına karşılık gelen kodlar ilgili -.rb uzantılı dosyaya yazılmıştır. Yazılan senaryolarından bir örneği aşağıda verilmiştir.

Tablo 3.7 de görüldüğü üzere ilk kısımda öğretmen yetkili kullanıcının TTS053 numaralı senaryosunun metin olarak yer almaktadır. Tablonun ikinci bölümde metin olan yazılan senaryo Seneario, Given, When, Then, And formatında dönüştürülerek yazılmıştır. Daha sondasında bu bölümdeki satırların her birinin fonksiyonlarını gerçekleştirecek kodlar yazılmıştır. Yazılan kodlar tablonun üçüncü bölümünde yer görülmektedir. Senaryo çalıştırıldığında web tarayıcısı otomatik açılır ve sonrasında ilgili web adresinin açılması, kullanıcı adı ve şifrenin girilmesi, giriş butonuna tıklama, sayfa kontrolü yapılması, Instructional technology dersine tıklama ve ders kontrolü işlemleri otomatik yapılmaktadır.

Tablo 3.7 TTS053 numaralı senaryonun otomasyon örneği

Bir Moodle öğretmen yetkili kullanıcı olarak ilgili katılımcıyı seçip, seçili kullanıcılarla alanından "Yeni not ekle" seçip, not bilgisi girip, durum bilgisini kişisel seçip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece ilgili katılımcıya kişisel not bilgisi eklerim.
Scenario: Teacher kullanici katilimcilara yeni not ekleme durum bilgisi kisisel TTS053
Given Moodle "**http://uzeg.aku.edu.tr/**" adresine gitmek istiyorum
And kullanicidi bilgisini " **kullanıcıadı** " giresem
And sifre bilgisini "**şifre**" olarak girersem.
And Giris butonuna tiklarsam
And Sag ust kosede kullanici bilgisi ile giris yapildigini gorurum
When Instructional technology dersine tiklarsam

Tablo 3.7 (Devam) TTS053 numaralı senaryonun otomasyon örneği

```
    Then Duzenlemeyi ac butonuna tiklarsam.
```

```
Given(/^Moodle "([^"]*)" adresine gitmek istiyorum$/) do |anasayfa|
  visit anasayfa
end
When(/^kullanicidi bilgisini "([^"]*)" giresem$/) do |username|
  find_by_id("login_username").set username
End
And(/^sifre bilgisini "([^"]*)" olarak girersem$/) do |password|
find_by_id("login_password").set password
end
And(/^Giris butonuna tiklarsam$/) do
  box=find("div.c1.btn")
box.find("input").click
end
Then(/^Sag ust kosede kullanici bilgisi ile giris yapildigini gorurum$/) do
  find("div.logininfo", :text =>"olarak giriş yaptınız")
end
When(/^Instructional technology dersine tiklarsam$/) do
  find("a", :text =>"Instructional Technology").click
end
```

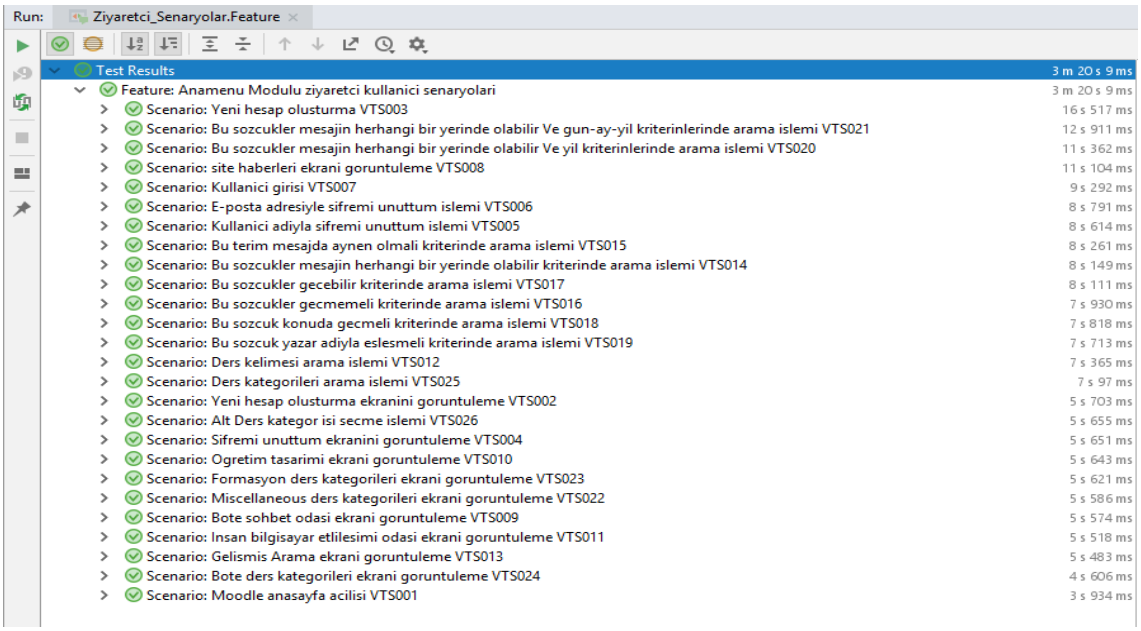
3.8 Otomatik Testlerin Optimizasyonu

Test senaryoları otomatize edildikten sonra özellik dosyalarında arka arkaya sıralı çalıştırılabilmesi için yazılan senaryoların optimizasyonu yapılmıştır. Optimizasyon tamamlandıktan sonra bir özellik dosyası çalıştırıldığında sıralamaya göre o dosya içerisindeki testler otomatik olarak çalışmaktadır.

3.8.1 Ziyaretçi Kullanıcı Rolü Testlerinin Optimizasyonu

Çalışmanın bu bölümünde ziyaretçi kullanıcı giriş, forumlarda ara, ana menü ve ders kategorilerine ait yirmi altı adet yazılan otomatik test senaryosunun optimizasyon

çalışması yapılmıştır. Bu çalışma ile otomatik senaryoların yazıldığı özellik dosyası çalıştırıldığında, otomatik test senaryolarının arka arkaya problemsiz olarak çalışması sağlanmıştır. Özellik dosyasındaki senaryolar VTS008, VTS009, VTS010, VTS011, VTS022, VTS023, VTS024, VTS025, VTS026, VTS012, VTS013, VTS014, VTS015, VTS016, VTS017, VTS018, VTS019, VTS020, VTS021, VTS001, VTS002, VTS003, VTS004, VTS005, VTS006 ve VTS007 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.

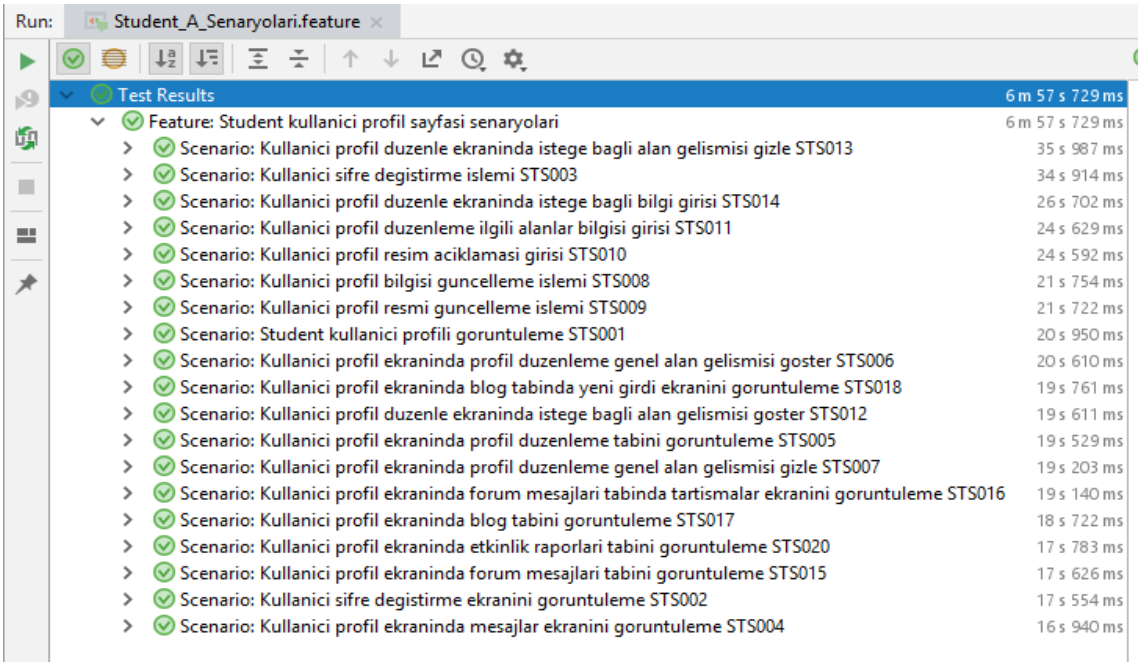


Resim 3.12 Ziyaretçi kullanıcı feature dosyası

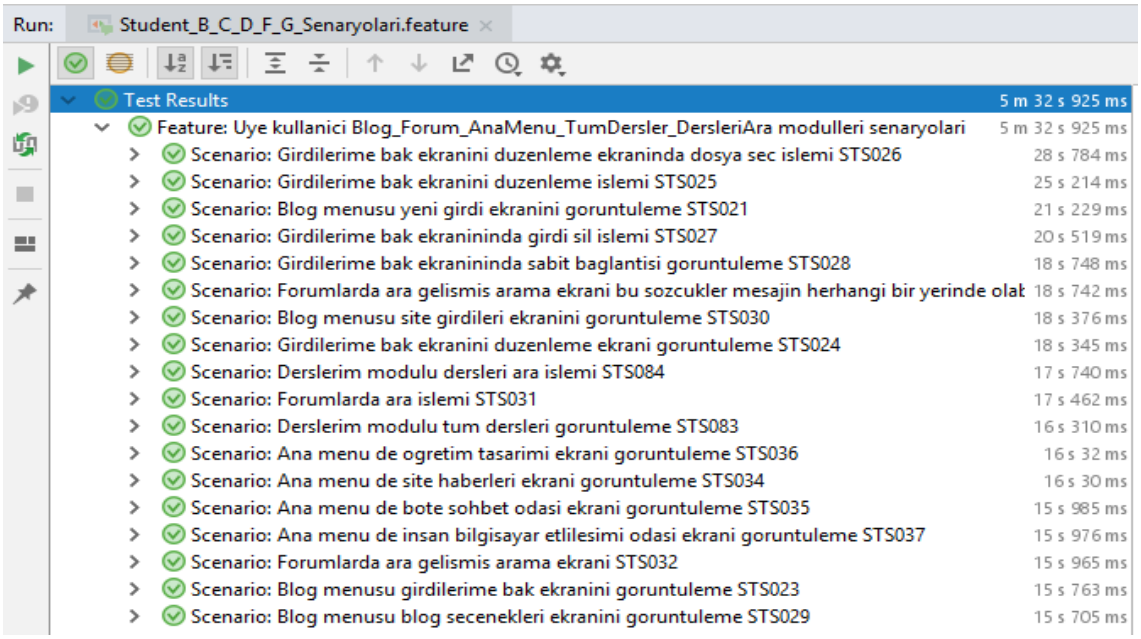
3.8.2 Öğrenci Kullanıcı Rolü Testlerinin Optimizasyonu

Çalışmanın bu bölümünde öğrenci yetkili kullanıcı rolüne ait seksen iki adet otomatize edilen otomatik test senaryosunun optimizasyon çalışması yapılmıştır. Öğrenci kullanıcı test senaryoları gruplandırılarak dört farklı özellik dosyasında yazıldı ve her bir feture dosyası ayrı ayrı optimize edilmiştir. Birinci grupta profil sayfası senaryoları optimize edilmiştir. İkinci grupta Blog menüsü, forumlarda ara ve ana menü modüllerine ait otomatik senaryolar optimize edilmiştir. Üçüncü ve dördüncü grupta derslerim modülü otomatik senaryoları optimize edilmiştir.

İlk özellik dosyasına ait optimize edilip çalıştırılan öğrenci kullanıcı profil sayfasına ait otomatik testler resim 3.13 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar STS001, STS002, STS003, … , STS020 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.

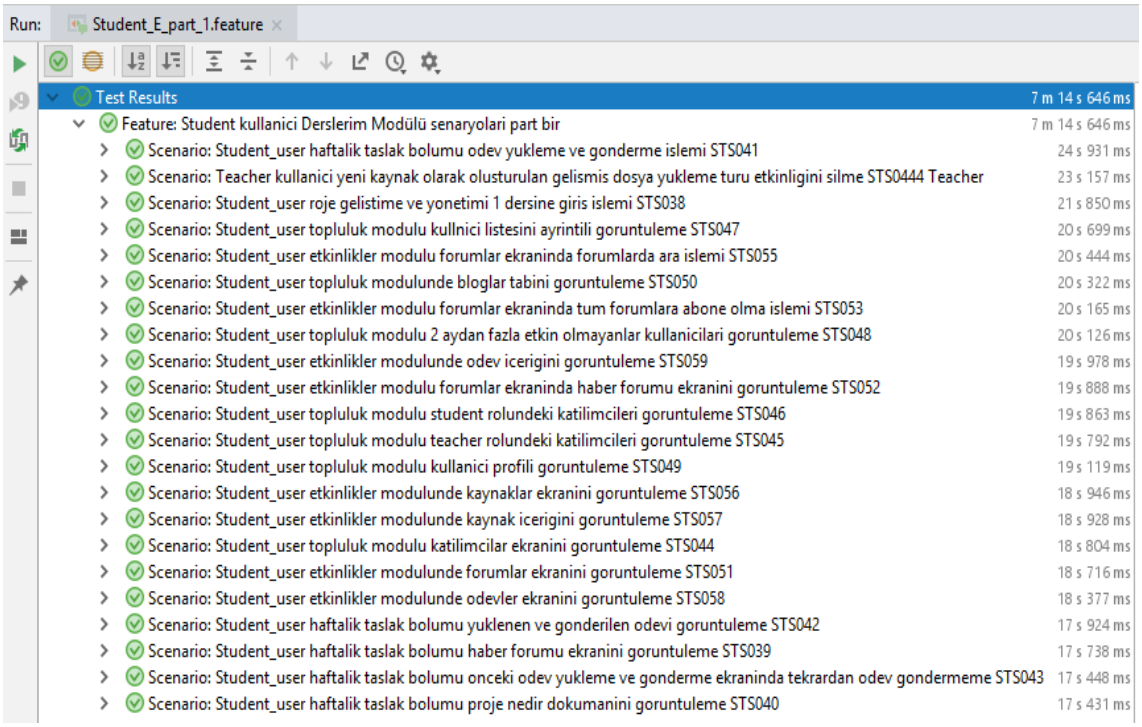


Resim 3.13 Öğrenci yetkili kullanıcı feature-1 dosyası



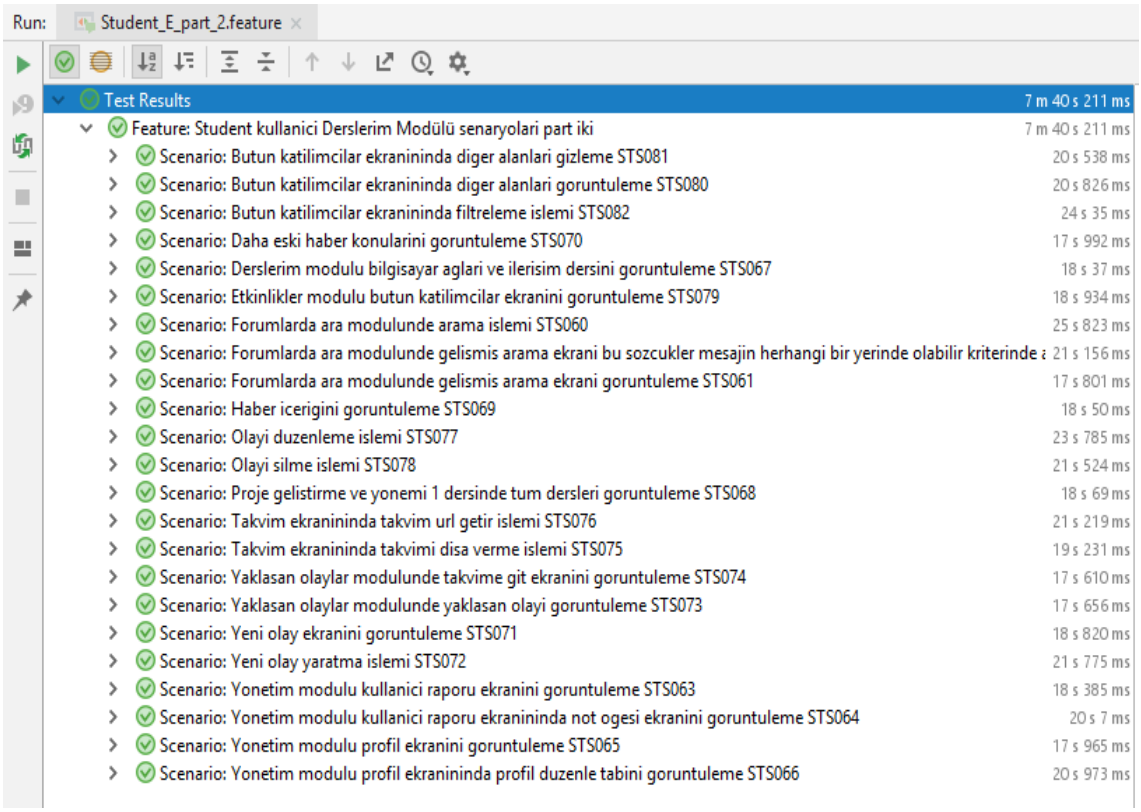
Resim 3.14 Öğrenci yetkili kullanıcı feature-2 dosyası

İkinci özellik dosyasına ait optimize edilip çalıştırılan otomatik testler resim 3.14 da görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar STS021, STS022, … , STS026, STS028, STS029, STS030, STS027, STS031, STS032, … , STS037, STS083, STS084 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.



Resim 3.15 Öğrenci yetkili kullanıcı feature-3 dosyası

Üçüncü özellik dosyasına ait optimize edilip çalıştırılan otomatik testler resim 3.15 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar STS038, STS039, STS040, TTS012, STS041, STS042, STS044, TTS033, … , STS045, STS046, … , STS058, STS059 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.

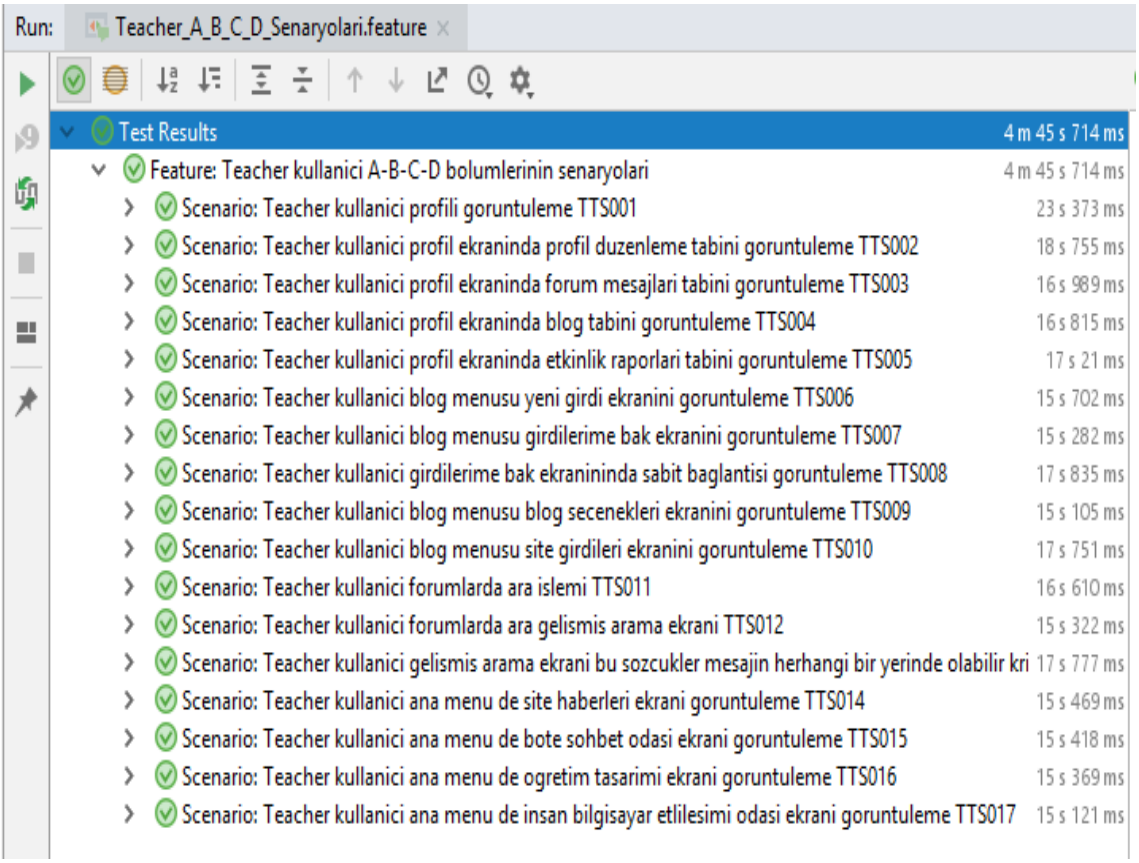


Resim 3.16 Öğrenci yetkili kullanıcı feature-4 dosyası

Dördüncü ve son özellik dosyasına ait optimize edilip çalıştırılan otomatik testler resim 3.16 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar STS060, STS061, … , STS081, STS082 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.

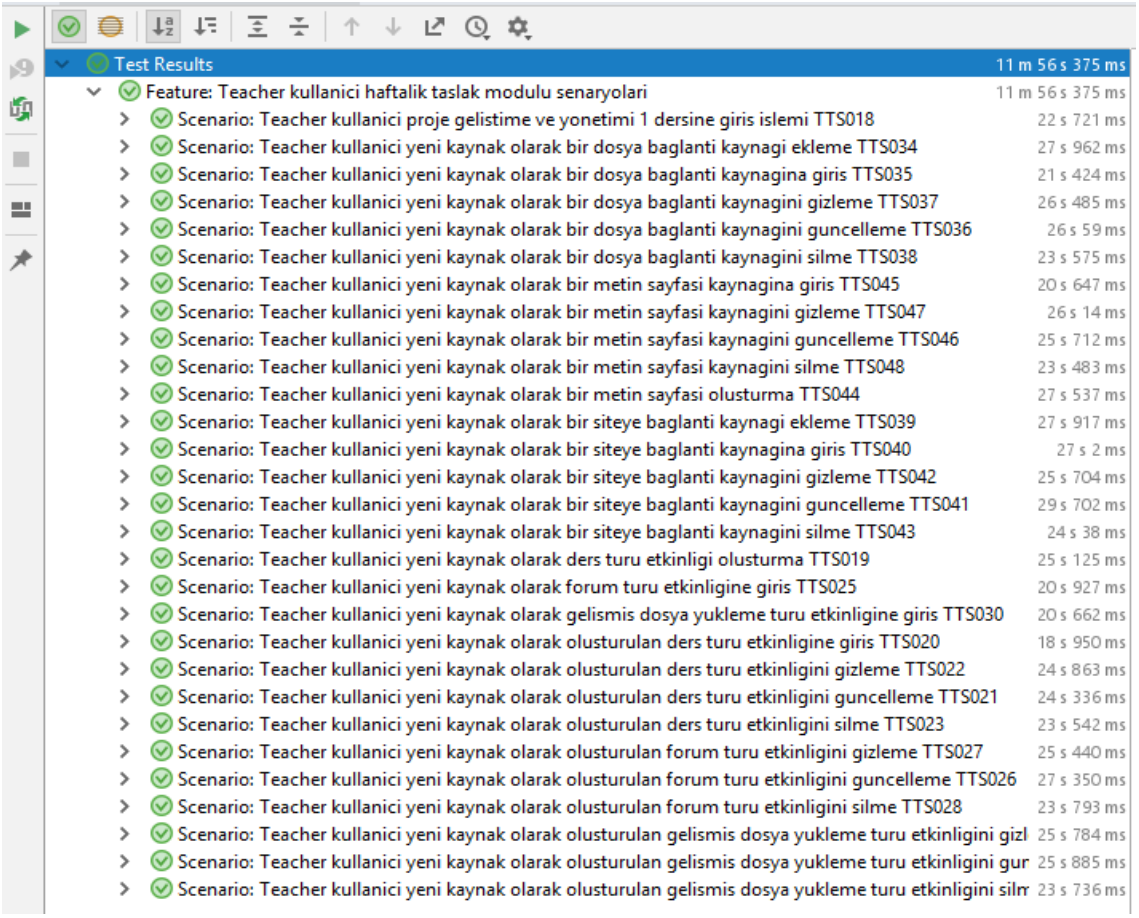
3.8.3 Öğretmen Kullanıcı Rolü Testlerinin Optimizasyonu

Çalışmanın bu bölümünde öğretmen yetkili kullanıcı rolüne yüz yirmi sekiz adet otomatize edilen otomatik test senaryosunun optimizasyon çalışması yapılmıştır. Öğretmen kullanıcı test senaryoları gruplandırılarak beş farklı özellik dosyasında yazıldı ve her bir feture dosyası ayrı ayrı optimize edilmiştir. Birinci grupta profil sayfası, blog menüsü, forumlarda ara ve ana menü modüllerine ait otomatik senaryolar optimize edilmiştir. İkinci, üçüncü, dördüncü ve beşinci grupta derslerim modülü otomatik senaryoları optimize edilmiştir.



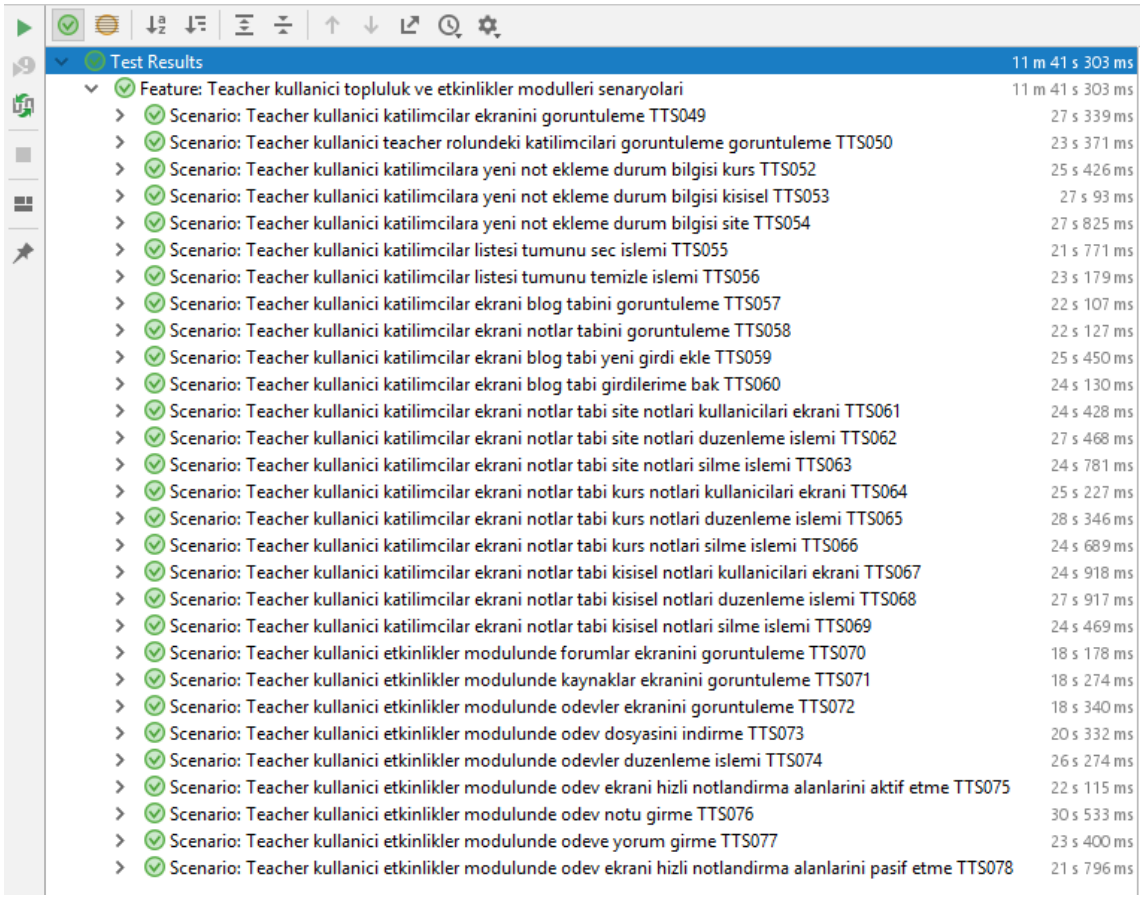
Resim 3.17 Öğretmen yetkili kullanıcı feature-1 dosyası

İlk özellik dosyasına ait optimize edilip çalıştırılan öğretmen kullanıcı profil sayfasına ait otomatik testler resim 3.17 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar TTS001, TTS002, TTS003, … , TTS017 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.



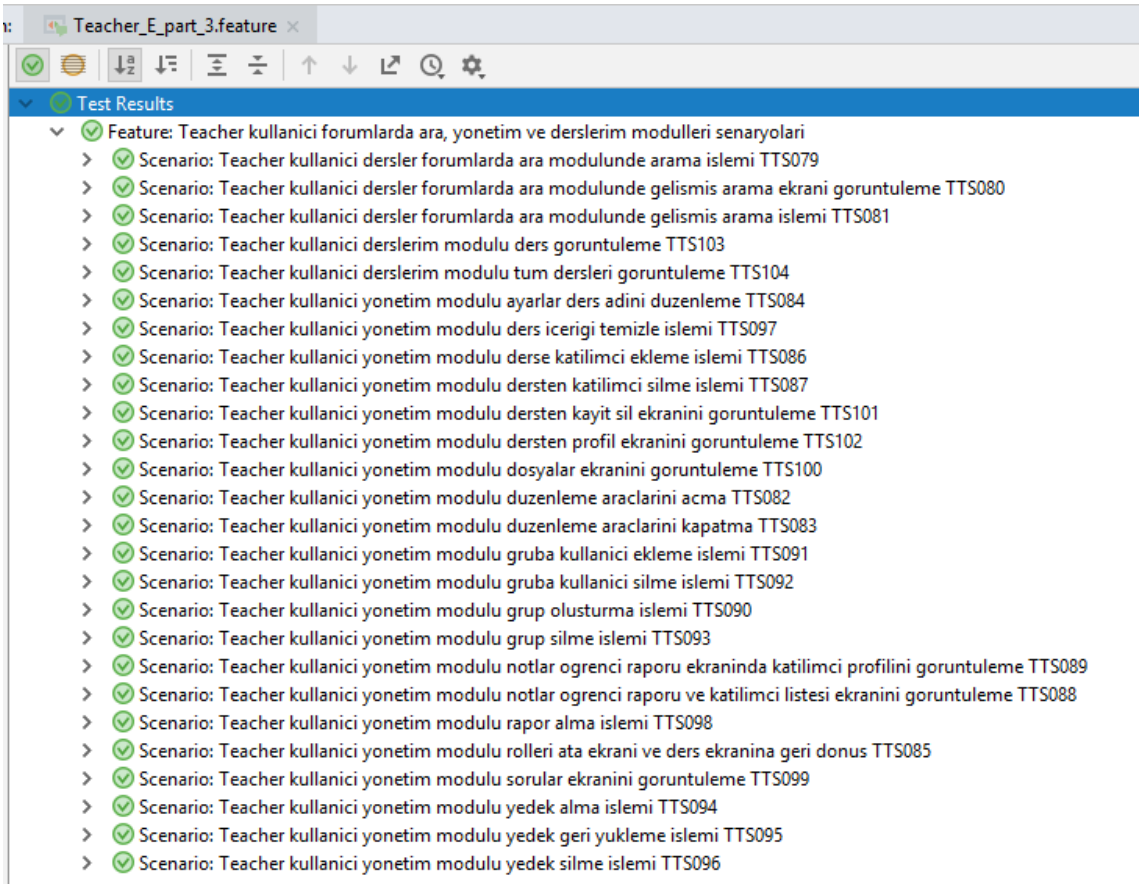
Resim 3.18 Öğretmen yetkili kullanıcı feature-2 dosyası

İkinci özellik dosyasına ait optimize edilip çalıştırılan öğretmen kullanıcıya ait otomatik testler resim 3.18 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar TTS018, TTS019, TTS020, … , TTS048 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.



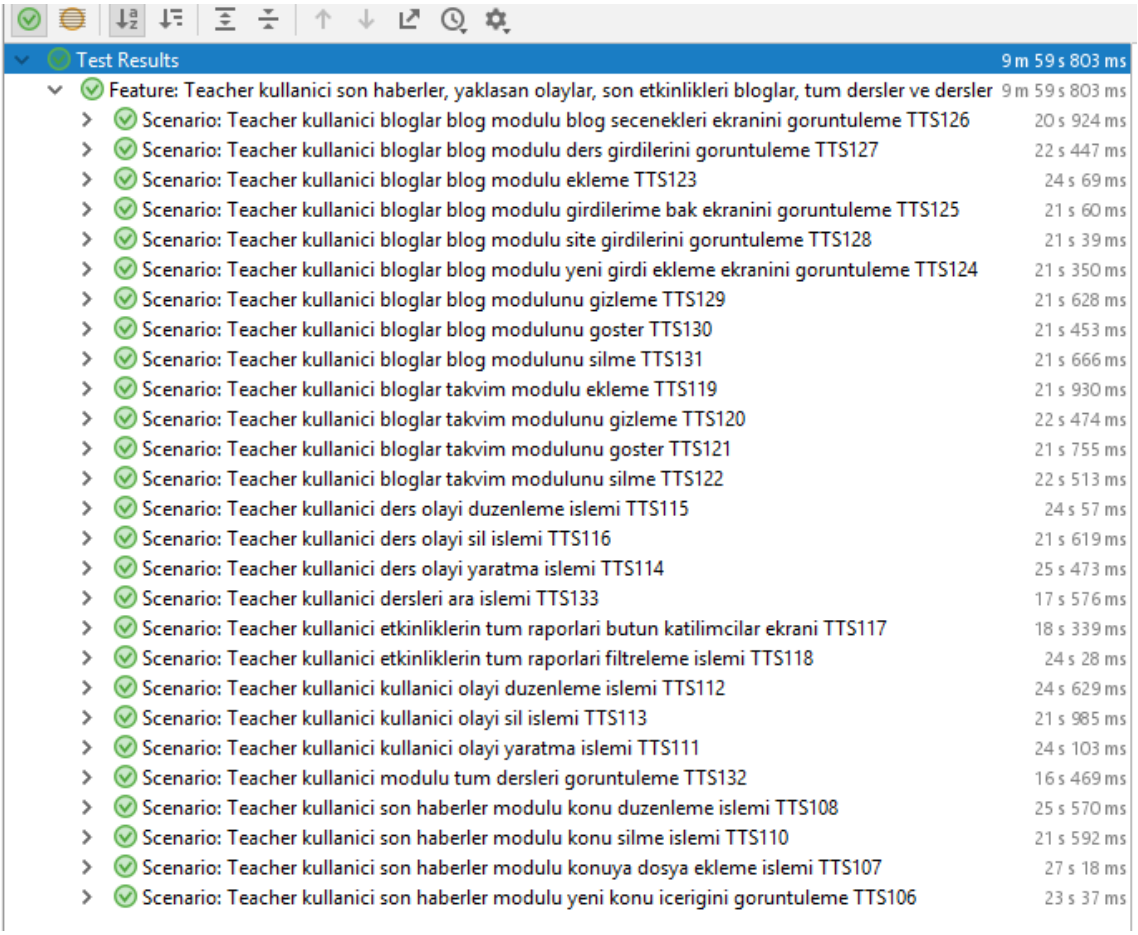
Resim 3.19 Öğretmen yetkili kullanıcı feature-3 dosyası

Üçüncü özellik dosyasına ait optimize edilip çalıştırılan öğretmen kullanıcıya ait otomatik testler resim 3.19 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar TTS049, TTS050, TTS051, … , TTS078 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.



Resim 3.20 Öğretmen yetkili kullanıcı feature-4 dosyası

Dördüncü özellik dosyasına ait optimize edilip çalıştırılan öğretmen kullanıcıya ait otomatik testler resim 3.20 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar TTS079, TTS080, TTS081, … , TTS104 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.



Resim 3.21 Öğretmen yetkili kullanıcı feature-5 dosyası

Beşinci özellik dosyasına ait optimize edilip çalıştırılan öğretmen kullanıcıya ait otomatik testler resim 3.21 de görülmektedir. Bu özellik dosyası çalıştırıldığında özellik dosyasındaki senaryolar TTS105, TTS106, TTS107, … , TTS133 sıralaması şeklinde optimize edilmiştir. Optimizasyon işlemi tamamlandıktan sonra çalıştırılan otomatik testler başarılı olarak tamamlanmıştır.

4. BULGULAR

4.1 Fonksiyonel Analiz Bulguları ve Değerlendirme

Bu araştırmada, Moodle yazılımının farklı kullanıcı rollerine göre fonksiyonel özelliklerinin analizleri yapılarak Davranış Odaklı Geliştirme yaklaşımı formatında metinsel senaryoları yazılmıştır. Fonksiyonel analiz sırasında sistemin ana kullanım özelliklerinin senaryoları yazılmaya çalışılmıştır. Bu nedenle uç detay fonksiyonel özelliklerin senaryoları yazılmamıştır. Bunun nedeni çalışmayı belirli bir kapsamda sınırlandırmak ve mevcut kaynakları etkin bir biçimde kullanarak tez çalışmasının yürütülmesidir. Fonksiyonel analiz aşamasında Moodle yazılımını tasarlayan ve yazılım geliştirmesini yürüten uzmanların olmaması çalışmanın verimliliğini azalttığı görülmüştür. Çünkü test uzmanı bu çalışmayı yaparken tasarlama uzmanından sistemin iş kuralları ve iş akışlarıyla ilgili konularda bilgi alabilmesi fonksiyonel analiz aşamasını daha verimli olmasını sağlayacağı belirtilmektedir. Bununla birlikte oluşturulan senaryolar daha nitelikli ve amacına uygun kapsayıcı olarak yazılacağı düşünülmektedir.

4.2 Manuel Test Bulguları ve Değerlendirme

Fonksiyonel analiz aşamasında hazırlanan test senaryolarına göre manuel testler yapılmıştır. İlk olarak ziyaretçi kullanıcı rolü, sonrasında öğrenci kullanıcı rolü ve son olarakta öğrenmen kullanıcı rolü testleri yapılmıştır. Ziyaretçi kullanıcı manuel testleri sırasında her hangi bir bulguya (bug) rastlanılmamıştır. İlgili senaryolar başarılı olarak tamamlanmıştır. Ziyaretçi kullanıcı ekranına sınırlı erişim olmasından dolayı test senaryolarını tamamlamak zaman alıcı olmamıştır.

Öğrenci yetkili kullanıcı rolü ekranlarındaki öğrenci profil sayfası, blog menüsü, forumlarda ara, ana menü modüllerine ait test senaryolarının manuel testleri problemsiz olarak tamamlanmıştır. Derslerim modülünde haftalık taslak, topluluk, etkinlikler, forumlarda ara, yönetim, derslerim, yaklaşan olaylar, son haberler ve son etkinlikler alt modüllerine ait senaryoların manuel testleri başarılı olarak tamamlanmıştır. Derslerim

modülü ve alt modüllerine ait test senaryoları yürütülürken bazı senaryoların öğretmen kullanıcı senaryolarına bağlılığı olduğu görülmektedir. Bu bağlılıklar, STS039 haber forumu görüntüleme test senaryonun koşulması için öncesinde öğretmen yetkili kullanıcı TTS024 forum oluşturma test senaryosu koşularak ilgili forum kaynağının oluşturulması gerekmektedir. STS040 ilgili dosya kaynağını görüntüleme test senaryonun koşulması için öncesinde öğretmen yetkili kullanıcı TTS034 dosya türünde kaynak oluşturma test senaryosu koşularak ilgili dosya kaynağının oluşturulması gerekmektedir. STS041 ilgili ödev dosyasını yükleme test senaryonun koşulması için öncesinde öğretmen yetkili kullanıcı TTS029 gelişmiş dosya yükleme türünde etkinlik oluşturma test senaryosu koşularak ilgili ödev yükleme etkinliğinin oluşturulması gerekmektedir. STS052 ve STS056 test senaryolarının koşumu için yukarıda belirtilen TTS024 ve TTS034 test senaryoları tamamlanmalıdır. STS069 son haberleri görüntüleme test senaryonun koşulması için öncesinde öğretmen yetkili kullanıcı TTS105 yeni konu ekle foruma gönder test senaryosu koşularak ilgili dosya kaynağının oluşturulması gerekmektedir.

Öğrenci yetkili kullanıcı ekranlarına ait senaryolar ziyaretçi senaryolarına göre sayısal olarak daha fazla ve işlemsel olarak da daha uzun sürdüğü görülmüştür. Buna bağlı olarak testlerin tamamlanma süresi daha fazla sürmüştür. Bazı test senaryoların öğretmen yetkili kullanıcıya bağımlı olması testlerin tamamlama sürelerini artırdığı görülmüştür.

Öğretmen yetkili kullanıcı ekranlarındaki öğretmen profil sayfası, blog menüsü, forumlarda ara, ana menü modüllerine ait test senaryolarının manuel testleri problemsiz olarak tamamlanmıştır. Derslerim modülünde haftalık taslak, topluluk, etkinlikler, forumlarda ara, yönetim, derslerim, yaklaşan olaylar, son haberler, son etkinlikler ve bloglar alt modüllerine ait senaryoların manuel testleri başarılı olarak tamamlanmıştır. Bu kullanıcı rolüne ait senaryolar öğrenci kullanıcı rolüne göre daha fazla aşamalı olduğu görülmektedir. Öğretmen kullanıcı rolüne ait senaryoların öğrenci kullanıcıya göre de daha fazla olmasında dolayı testlerin tamamlanma süresi diğer kullanıcı rollerine göre daha fazla zaman almıştır.

4.3 Test Otomasyonu Bulguları ve Değerlendirme

Çalışmanın bu bölümünde fonksiyonel analiz aşamasında yazılan test senaryolarının otomatize edilmesi, otomatize işlemi tamamlandıktan sona otomatik testlerin optimizasyonu, manuel ve test otomasyonunun karşılaştırması konularına değinilmektedir. Tablo 4.1 de görüldüğü üzere ziyaretçi, öğrenci ve öğretmen kullanıcı rollerine göre toplamda 243 test senaryosu yazılmıştır.

Tablo 4.1 Kullanıcı gruplarına göre yazılan ve otomatize edilen test senaryoları

Kullanıcı Rolü	**Yazılan Senaryo Sayısı**	**Otomatize edilen Senaryo Sayısı**	**Otomatize edilemeyen Senaryo Sayısı**
Ziyaretçi	26	26- (100%)	0
Öğrenci	84	82 - (97,6%)	2 - (2,4%)
Öğretmen	133	128 - (96,2%)	5 - (3,8%)
Toplam	**243**	**236 - (97,1%)**	**7 - (2,9%)**

Ziyaretçi kullanıcı için yazılan 25 test senaryosunun tamamı otomatize edilmiştir. Öğrenci kullanıcı için yazılan 84 adet test senaryosundan (yüzde 97,6) 82 si otomatize edildi, 2 adet senaryo ise otomatize edilememiştir. Öğretmen kullanıcı için yazılan 133 test senaryosundan (yüzde 96,2) 128 i otomatize edildi, kalan 5 senaryo ise otomatize edilememiştir. Otomatize edilemeyen bu senaryoların detaylarına ileriki bölümlerde değinilmiştir. Genel olarak bakıldığında yazılan 243 senaryodan 236 i otomatize edilip kalan 7 senaryo ise otomatize edilememiştir.

Tablo 4.2 de otomatik test senaryolarının yazıldığı özellik dosyalarına ait bilgiler yer almaktadır. Ziyaretçi kullanıcı rolüne ait özellik dosyasında yer alan 26 test senaryosu 113 adımdan oluşmaktadır. Bu özellik dosyasındaki otomatik senaryoları toplam 3 dakika 28 saniye gibi bir sürede çalışmaktadır. Öğrenci kullanıcı rolüne ait özellik dosyalarında yer alan 82 test senaryosu 861 adımdan oluşmaktadır. Bu özellik dosyaları çalıştırıldığında 28 dakika 7 saniye gibi bir sürede otomatik test senaryolarının testleri tamamlanmaktadır.

Tablo 4.2 Kullanıcılara ait feature dosyalarındaki senaryo, adım sayıları ve test koşum süresi

Feature Adı	Senaryo Sayısı	Adım Sayısı	Süre(dk:sn)
Ziyaretçi Kullanıcı	26	113	03:28
VTS - Toplam	**26**	**113**	**03:28**
Öğrenci Kullanıcı 1	19	221	07:08
Öğrenci Kullanıcı 2	18	184	05:42
Öğrenci Kullanıcı 3	22	234	07:26
Öğrenci Kullanıcı 4	23	252	07:51
STS - Toplam	**82**	**861**	**28:07**
Öğretmen Kullanıcı 1	17	162	04:54
Öğretmen Kullanıcı 2	29	355	12:11
Öğretmen Kullanıcı 3	29	372	11:56
Öğretmen Kullanıcı 4	26	321	10:25
Öğretmen Kullanıcı 5	27	310	10:15
TTS - Toplam	**128**	**1520**	**49:41**
Genel Toplam	**236**	**2524**	**81:16**

Öğretmen kullanıcı rolüne ait özellik dosyalarında yer alan 128 test senaryosu 1520 adımdan oluşmaktadır. Bu özellik dosyaları çalıştırıldığında 49 dakika 41 saniye gibi bir sürede otomatik test senaryolarının testleri tamamlanmaktadır. Genel olarak bakıldığında 2524 adımdan oluşan 236 otomatik test senaryosunun toplam koşum süresi 81 dakika 16 saniyedir.

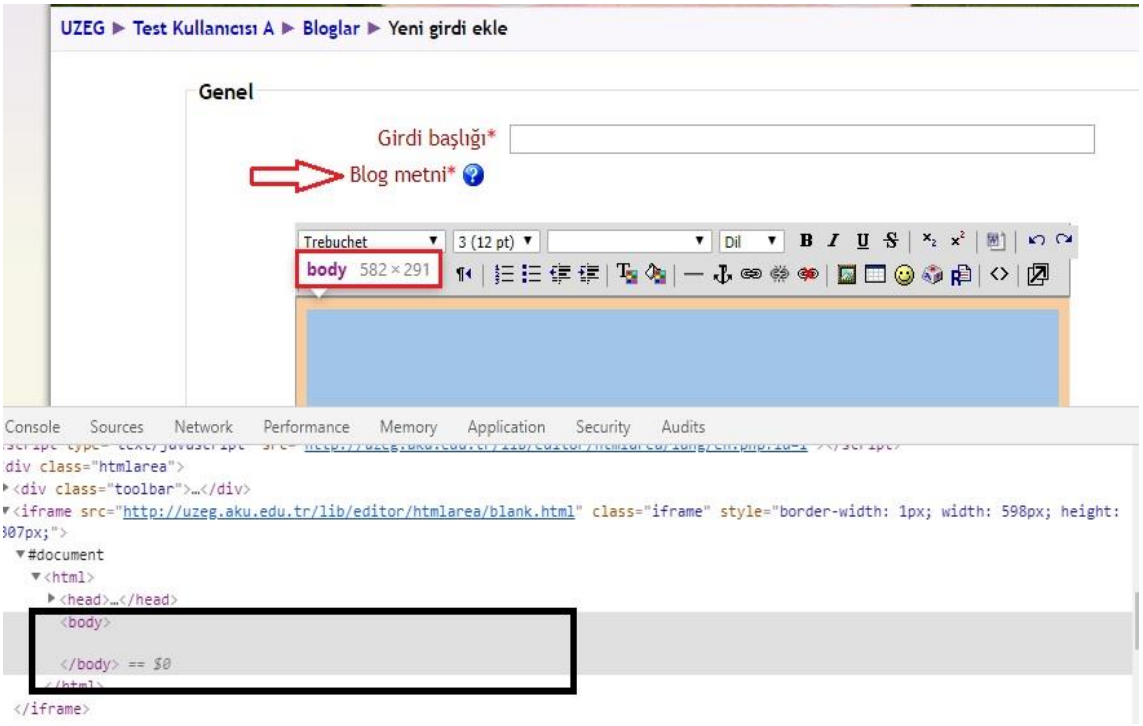
4.3.1 Test Otomasyon Süreci Bulguları ve Değerlendirme

Çalışmanın bu aşamasında fonksiyonel analiz aşamasında yazılan test senaryolarının otomatize edilme sürecinde karşılaşılan bulgulara değinilmiştir. İlk olarak ziyaretçi kullanıcı sonrasında öğrenci yetkili kullanıcı rolü ve son olarak da öğretmen yetkili kullanıcı rolüne ait durumlardan söz edilmiştir. Ziyaretçi kullanıcı rolüne ait 26 test senaryosunun test otomasyonu başarılı olarak otomatize edilerek test senaryolarının tamamı otomatik hale getirilmiştir. Bu aşamada otomatize etme sürecinde her hangi bir bulgu ile karşılaşılmamıştır.

Öğrenci yetkili kullanıcı rolüne ait seksen dört adet test senaryosundan seksen iki adeti başarılı olarak otomatize edilmiştir. İki adet test senaryosu ise otomatize edilememiştir. Otomatize edilemeyen test senaryolarının numaraları STS019 ve STS022 dir. Her iki senaryoda aynı durumdan dolayı otomatize edilememiştir. Senaryo STS019 Bir Moodle öğrenci yetkili kullanıcı olarak yeni girdi ekleme ekranında "Girdi Başlığı" ve "Blog Metni" ekleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni girdi eklendiğini görürüm. Bu senaryodaki yeni girdi ekleme ekranında girdi başlığı ve blog metni zorunlu veri girişi yapılması gereken alanlar olarak tasarlanmış. Resim 4.1 de girdi başlığı ve blog metni olanlarının zorunlu olduğu (*) işareti ile gösterilmiş.

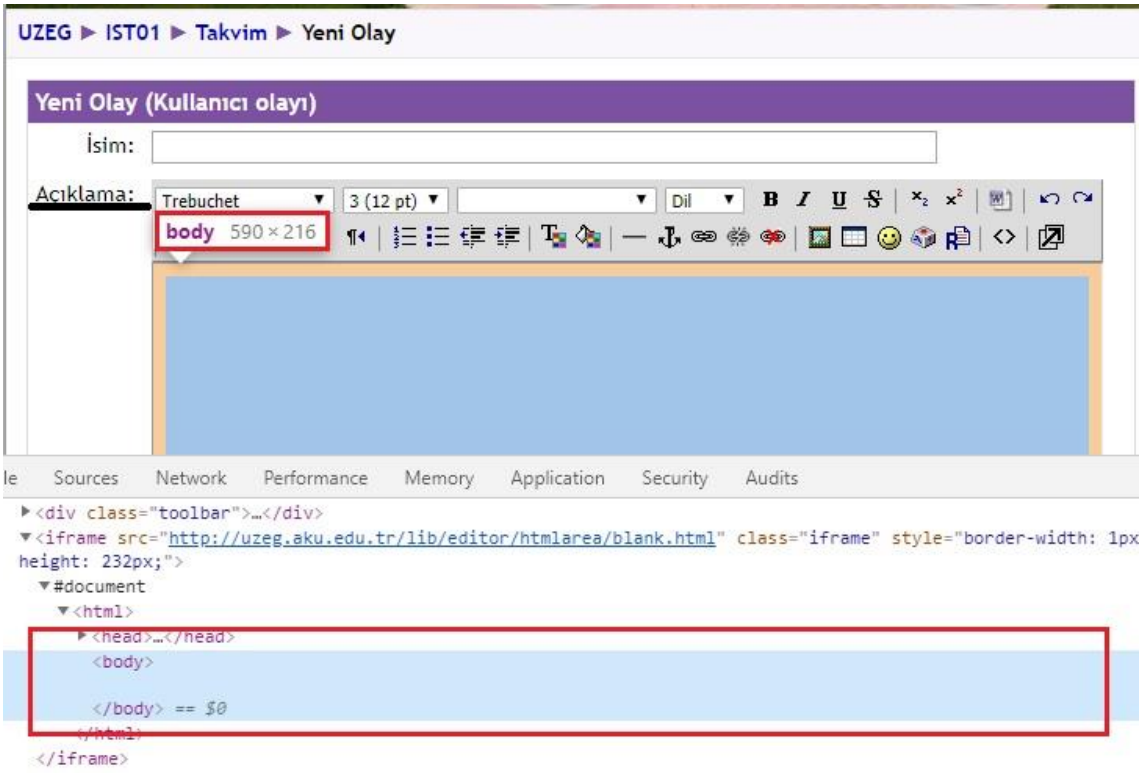
Resim 4.1 Yeni girdi ekle ekranı

Yazılım geliştirme sürecinde girdi başlığında id css yapısı kullanılarak(id=id_subject) bu alana özgü bir id tanımlanmış. Fakat blog metni alanı için bu dudum söz konusu değildir. Blog metni alanında css kullanılmamış herhangi bir class ya da id tanımlanmamış. Test senaryolarının otomatize edilmesi sırasında ilgili alanları seçme, tıklama, veri girişi gibi durumlardan class ve id tanımları kullanılarak ilgili olaylar gerçekleştirilmektedir. Yeni girdi ekle ekranında ise blog metni alanı için class ya da id tanımlaması yapılmamıştır. Bu nedenle de test otomasyonu senaryonun bu adımında durmuş ve senaryo otomatize edilememiştir. Resim 4.2 de blog metni alanına metin girişi yapılan body alanında css tanımlama bilgisinin olmadığı görülmektedir.



Resim 4.2 Yeni girdi ekle blog metni alanı

Öğrenci yetkili kullanıcı STS072 senaryosunda yeni olay ekleme ekranında açıklamada alanında da css tanımlaması bulunmamaktadır. Fakat bu ekranda açıklama alanı zorunlu veri girişi olarak tasarlanmamış. Bu nedenle ilgili alana veri girişi zorunlu olmadığından bu senaryoda açıklama alanına veri girişi yapılmadan test otomasyonuna devam edildi ve senaryonun otomasyonu yapılmıştır. Senaryo STS072**:** Bir Moodle öğrenci yetkili kullanıcı olarak yeni olay ekranında ilgili alanları doldurup "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni olay yaratma işlemini tamamlarım. Resim 4.3 de görüldüğü üzere açıklama alanının zorunlu olduğuna dair (*) işareti yoktur ve kırımızı olarak belirtilmemiştir.

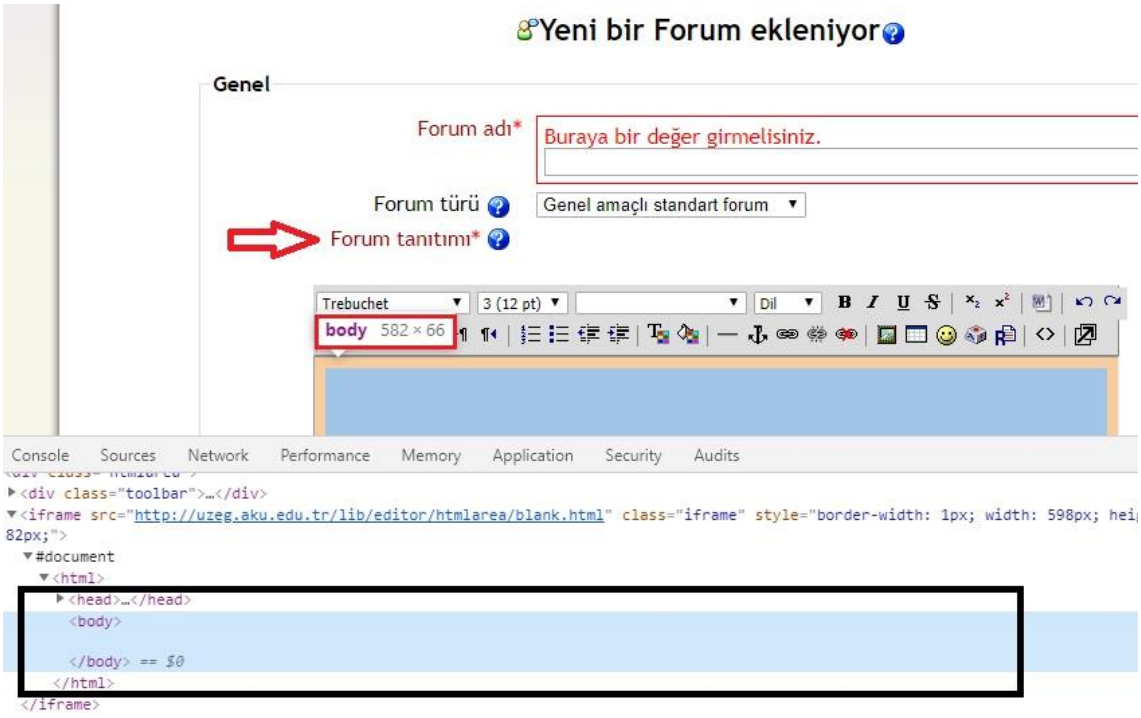


Resim 4.3 Yeni olay ekleme ekranı

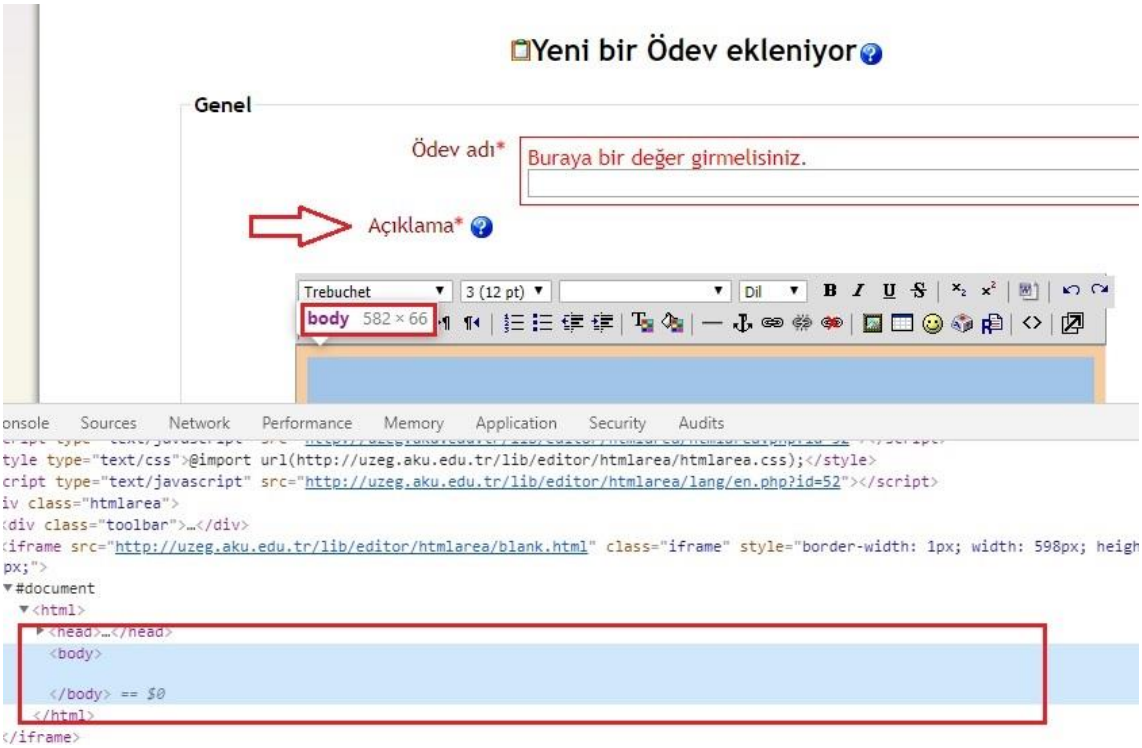
Öğretmen yetkili kullanıcı rolüne ait yüz otuz üç adet test senaryosundan yüz yirmi sekiz âdeti başarılı olarak otomatize edilmiştir. Beş adet test senaryosu ise otomatize edilememiştir. Otomatize edilemeyen test senaryolarının numaraları TTS024, TTS029, TTS051, TTS105 ve TTS109 dur. Bu senaryolar farklı olsalar da otomatize edilememe nedenleri aynı durumdan dolayıdır. Bu test senaryolarının Given, When, Then ve And formatında adım adım otomatize süreçlerinde bulguların karşılaşıldığı adımlarda otomatik test süreci durmuş ve testler otomatize edilememiştir. Buradaki karşılalan durum engelleyici bulgu türü olmaktadır.

Senaryo TTS024: Bir Moodle öğretmen yetkili kullanıcı olarak "Yeni kaynak olarak forum etkinliğini" seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece forum türünde yeni etkinliğin eklendiğini görürüm. Yeni bir form ekleme ekranında forum tanımı yazılım tasarlama aşamasında zorunlu bir alan olan belirtilmektedir. Yazılım geliştirme sürecinde bu alanda ilgili class ve id css tanımlamaları yapılmadığından dolayı test otomasyon geliştirme işlemi bu adımda durmuş ve otomatize edilememiştir. Resim 4.4 de zorunlu alanlar ve forum tanımı body

alanında css tanımının olmadığı görülmektedir.

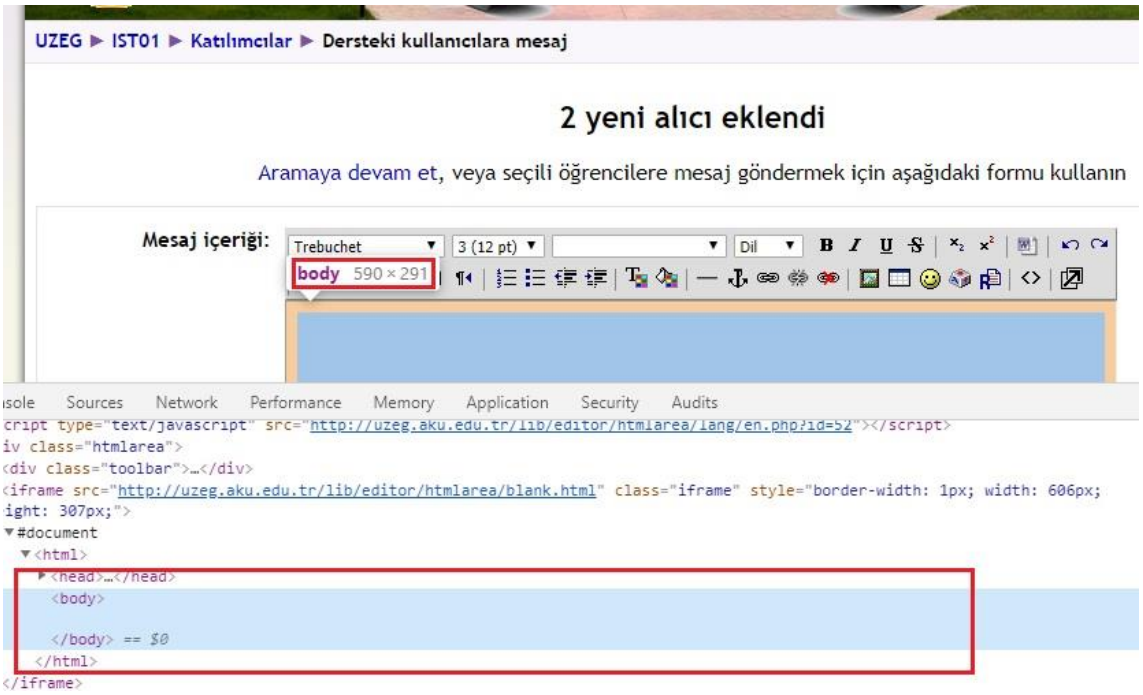


Resim 4.4 Yeni bir forum ekleme ekranı



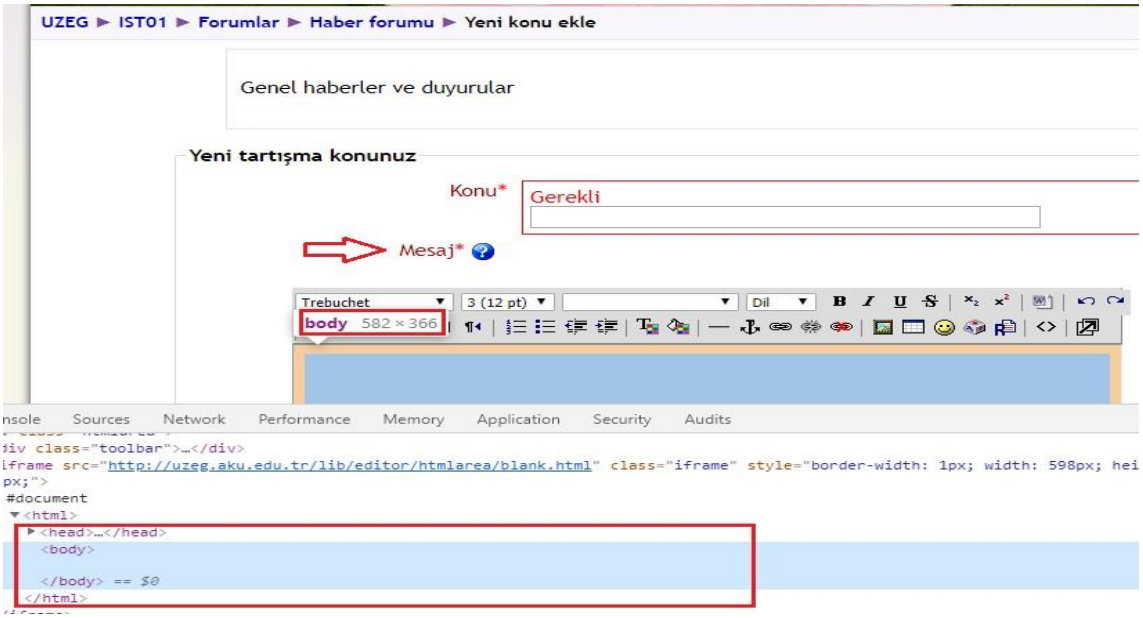
Resim 4.5 Ödevler gelişmiş dosya yükleme etkinliği ödev ekleme ekranı

Senaryo TTS029: Bir Moodle öğretmen yetkili kullanıcı olarak yeni kaynak olarak "Ödevler gelişmiş dosya yükleme etkinliğini" seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece ödevler gelişmiş dosya yükleme türünde yeni etkinliğin eklendiğini görürüm. Yeni bir ödev ekleme ekranında açıklama alanı yazılım tasarlama aşamasında zorunlu bir alan olan belirtilmektedir. Yazılım geliştirme sürecinde bu alanda ilgili class ve id css tanımlamaları yapılmadığından dolayı test otomasyon geliştirme işlemi bu adımda durmuş ve otomatize edilememiştir. Resim 4.5 de zorunlu alanlar ve ödev açıklaması body alanında css tanımının olmadığı görülmektedir.



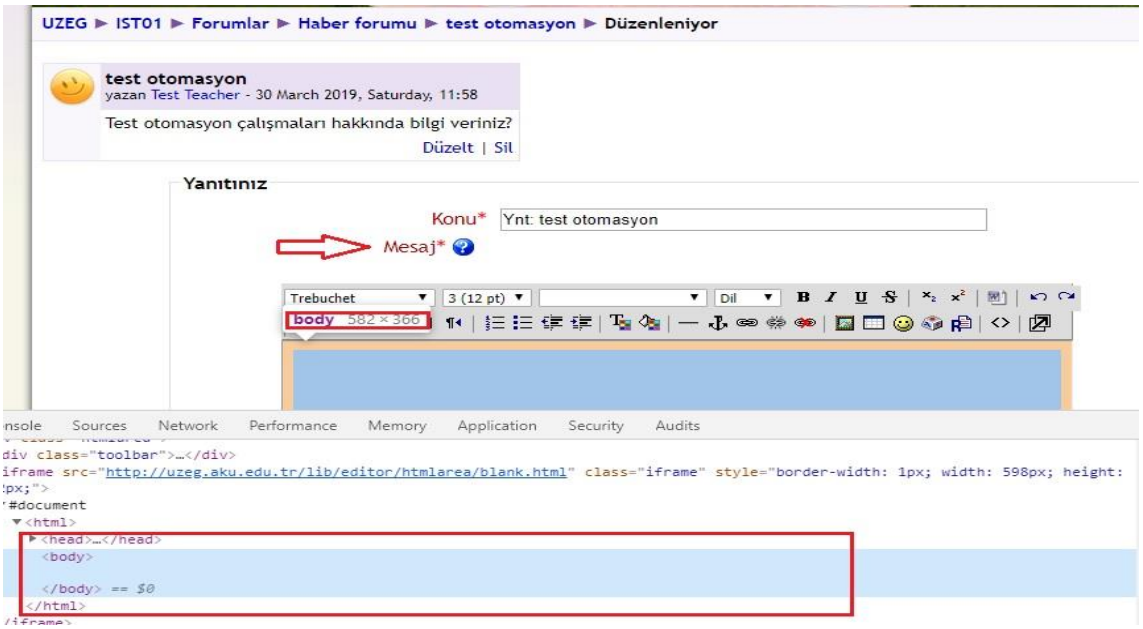
Resim 4.6 Ders katılımcılarına mesaj gönderme ekranı

Senaryo TTS051: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili katılımcı ya da katılımcıları seçip, seçili kullanıcılarla alanından ekle mesaj gönderi seçip, mesaj bilgisi girip, ön izleme butonuna tıklayıp, mesajı gönder butonuna tıklamak istiyorum böylece ilgili katılımcılara mesaj gönderirim. Ders katılımcılarına mesaj gönderme ekranında mesaj içeriği alanında ilgili class ve id css tanımlamaları yapılmadığından dolayı test otomasyon geliştirme işlemi bu adımda durmuş ve otomatize edilememiştir. Resim 4.6 da mesaj içeriği body alanında css tanımının olmadığı görülmektedir.



Resim 4.7 Yeni konu ekleme ekranı

Senaryo TTS105: Bir Moodle öğretmen yetkili kullanıcı olarak "Yeni konu ekle" alanına tıklayıp, ilgili alanları doldurup "Foruma gönder" butonuna tıklamak istiyorum böylece yeni konu eklemiş olurum. Yeni konu ekle ekranında mesaj içeriği alanında ilgili class ve id css tanımlamaları yapılmadığından dolayı test otomasyon geliştirme işlemi bu adımda durmuş ve otomatize edilememiştir. Resim 4.7 de mesaj içeriği body alanında css tanımının olmadığı görülmektedir.

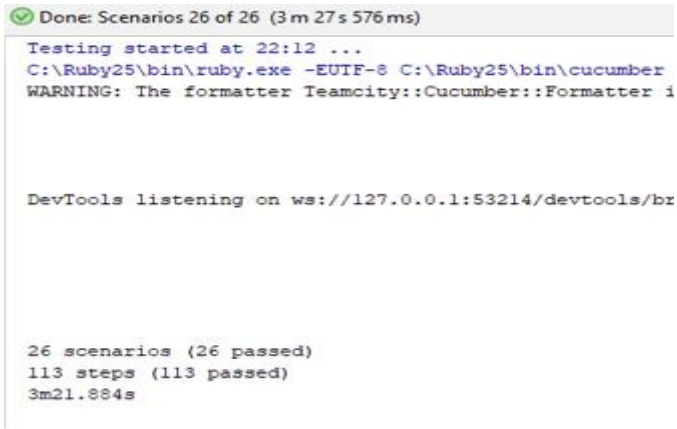


Resim 4.8 Forumlarda konu yanıtlama ekranı

Senaryo TTS109: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili konuyu seçip, “Yanıtla” alanına tıklayıp, ilgili mesajı yazıp “Foruma gönder” butonuna tıklamak istiyorum böylece yanıtlama işlemini tamamlarım. Forumda ilgili konuyu yanıtlama ekranında mesaj içeriği alanında ilgili class ve id css tanımlamaları yapılmadığından dolayı test otomasyon geliştirme işlemi bu adımda durmuş ve otomatize edilememiştir. Resim 4.8 de mesaj içeriği body alanında css tanımının olmadığı görülmektedir.

4.3.2 Test Optimizasyonu Bulguları ve Değerlendirme

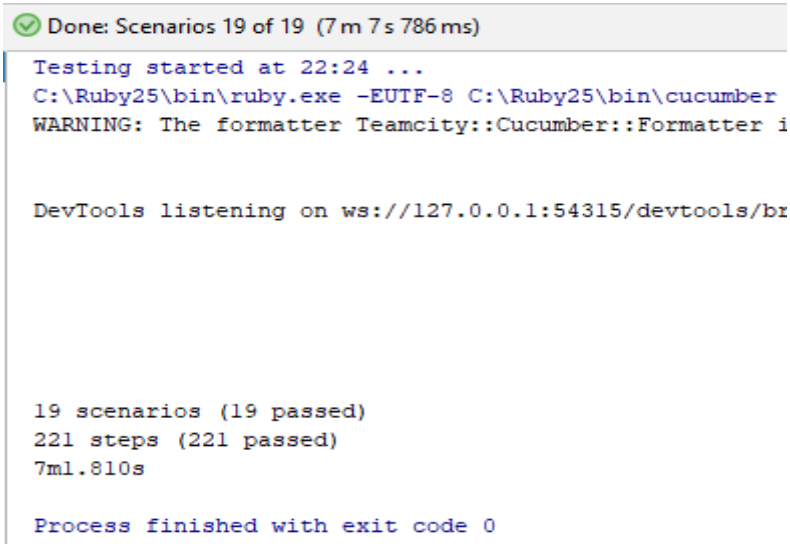
Test senaryolarının neredeyse tamamına yakını Given, When, Then ve And formatında birer birer otomatize edildi ve çalıştırıldı. Otomatize edilen senaryolar ilgili özellik dosyalarında gruplandırıldı. Sonrasında özellik dosyası içerisindeki otomatik senaryoların grup halinde çalıştırılması için optimizasyon çalışması yapılmıştır. Bu çalışma sırasında karşılan durumlara bu bölümde değinilmiştir. Otomatize edilemeyen testlere ilişkin durumlar bir önceki bölümde belirtilmişti. Bu durumlar optimizasyon sırasında bir takım kısıtlamalara neden olduğu görülmektedir.



Resim 4.9 Ziyaretçi kullanıcı senaryoları optimizasyonu

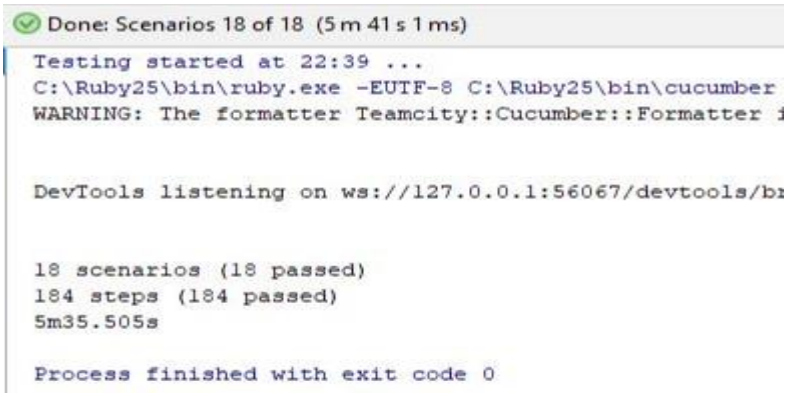
Resim 4.9 da görüldüğü üzere ziyaretçi kullanıcı rolüne ait otomatize edilen yirmi altı senaryonun optimizasyonu tamamlandı ve sonrasında çalıştırılan özellik dosyasında otomatik testleri başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 3 dakika 27 saniye 576 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında

ki belirtilen 3 dakika 21 saniye ve 884 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 26 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 113 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir.



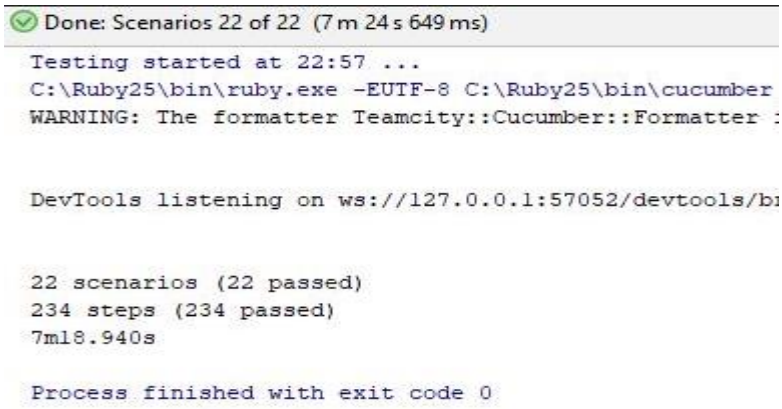
Resim 4.10 Öğrenci yetkili kullanıcı feature-1 optimizasyonu

Resim 4.10 da görüldüğü üzere öğrenci kullanıcı rolüne ait ilk özellik dosyasındaki otomatize edilen on dokuz senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 7 dakika 7 saniye 786 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 7 dakika 1 saniye ve 810 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 19 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 221 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir. Bu özellik dosyasında çalıştırılmadan önce otomatize edilemeyen STS019 senaryosun testi manuel olarak yapılmalıdır. Bu durum otomatik testlerin çalıştırılmasında manuel test yapılmasını gerekli kıldığından ciddi bir kısıt oluşturmaktadır.



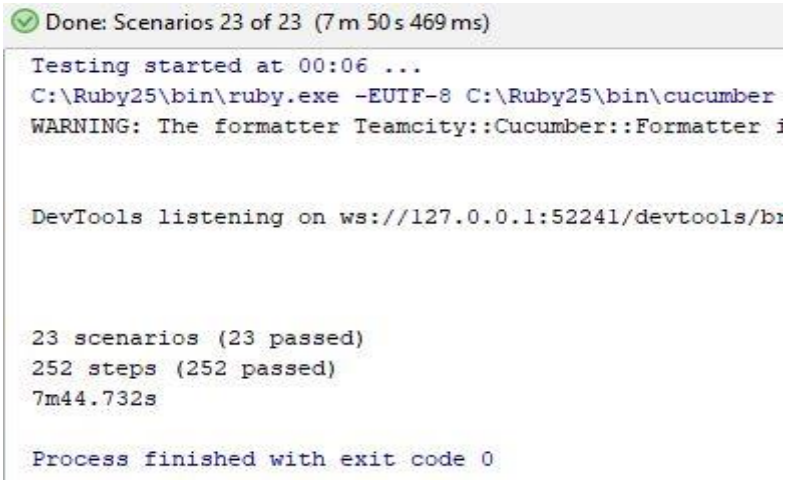
Resim 4.11 Öğrenci yetkili kullanıcı feature-2 optimizasyonu

Resim 4.11 de görüldüğü üzere öğrenci kullanıcı rolüne ait ikinci özellik dosyasındaki otomatize edilen on sekiz senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 5 dakika 41 saniye 1 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 5 dakika 35 saniye ve 505 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 18 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 184 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir. Bu özellik dosyasında çalıştırılmadan önce otomatize edilemeyen STS022 senaryosun testi manuel olarak yapılmalıdır. Bu durum otomatik testlerin çalıştırılmasında manuel test yapılmasını gerekli kıldığından ciddi bir kısıt oluşturmaktadır.



Resim 4.12 Öğrenci yetkili kullanıcı feature-3 optimizasyonu

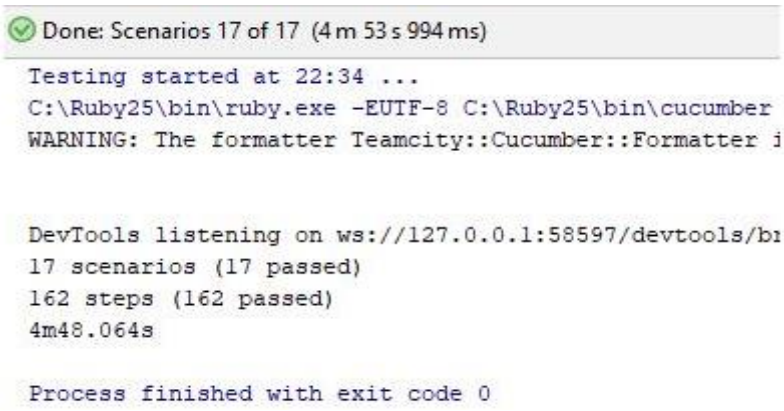
Resim 4.12 de görüldüğü üzere öğrenci kullanıcı rolüne ait üçüncü özellik dosyasındaki otomatize edilen yirmi iki senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 7 dakika 24 saniye 649 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 7 dakika 18 saniye ve 940 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 22 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 234 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir. Bu özellik dosyasında çalıştırılmadan önce otomatize edilemeyen TTS029 senaryosun testi manuel olarak yapılmalıdır. TTS029 senaryosu ödev dosyası yüklenmesi için öncesinde ilgili etkinliğin öğretmen yetkili kullanıcı tarafından manuel koşulması gerekmektedir. Bir öğrenci kullanıcısı bir ödev etkinliğine dosyayı yükleyip gönderdiği zaman bir daha ödev gönderemez. Bu nedenle STS041 öğrenci ödev yükleme test senaryosu çalıştırılmadan önce her zaman TTS029 senaryosu manuel koşmalıdır. Bu durum otomatik testlerin çalıştırılmasında manuel test yapılmasını gerekli kıldığından ciddi bir kısıt oluşturmaktadır.



Resim 4.13 Öğrenci yetkili kullanıcı feature-4 optimizasyonu

Resim 4.13 de görüldüğü üzere öğrenci kullanıcı rolüne ait dördüncü özellik dosyasındaki otomatize edilen yirmi üç senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı

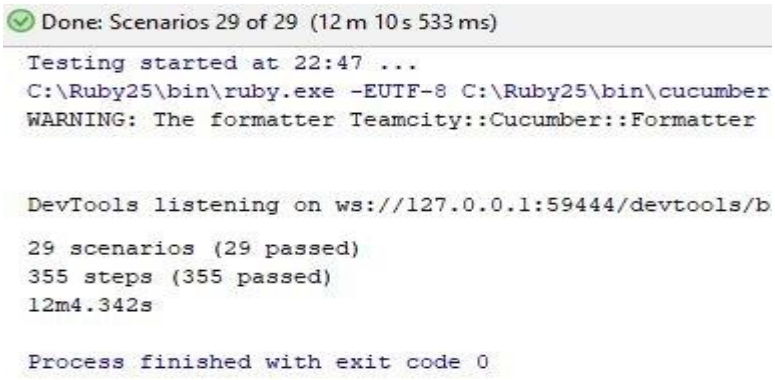
görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 7 dakika 50 saniye 469 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 7 dakika 44 saniye ve 732 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 23 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 252 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir. Bu özellik dosyasında çalıştırılmadan önce otomatize edilemeyen TTS105 senaryosun testi manuel olarak yapılmalıdır. TTS105 senaryosu öğrenci kullanıcı son haberler modülüne ilgili haberi açabilmek için öncesinde ilgili haber konusunun öğretmen yetkili kullanıcı tarafından manuel koşulması gerekmektedir. STS069 ilgili haberi görüntüleme test senaryosu çalıştırılmadan önce her zaman TTS105 senaryosu manuel koşmalıdır. Bu durum otomatik testlerin çalıştırılmasında manuel test yapılmasını gerekli kıldığından ciddi bir kısıt oluşturmaktadır.



Resim 4.14 Öğretmen yetkili kullanıcı feature-1 optimizasyonu

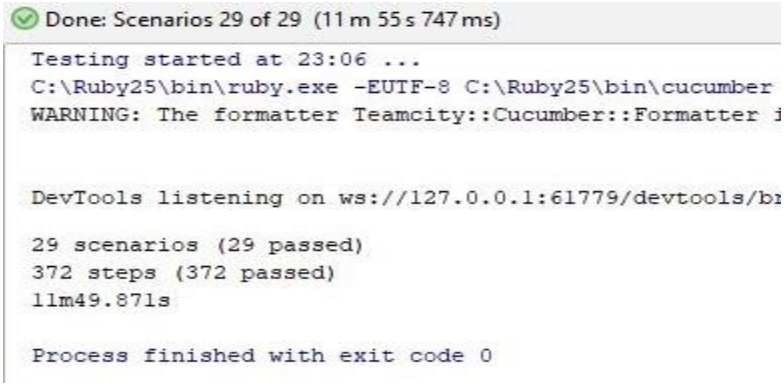
Resim 4.14 de görüldüğü üzere öğretmen kullanıcı rolüne ait ilk özellik dosyasındaki otomatize edilen on yedi senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 4 dakika 53 saniye 994 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 4 dakika 48 saniye ve 064

milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 17 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 162 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir.



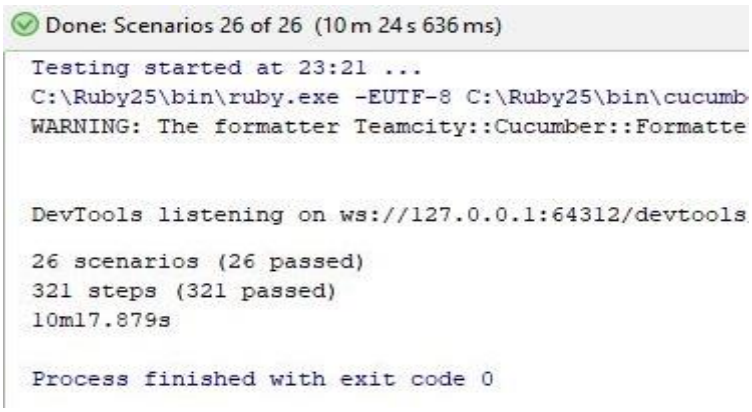
Resim 4.15 Öğretmen yetkili kullanıcı feature-2 optimizasyonu

Resim 4.15 de görüldüğü üzere öğretmen kullanıcı rolüne ait ikinci özellik dosyasındaki otomatize edilen 29 senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 12 dakika 10 saniye 553 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 12 dakika 4 saniye ve 355 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 29 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 355 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir. Bu özellik dosyasında çalıştırılmadan önce otomatize edilemeyen TTS024, TTS029 senaryolarının testleri manuel olarak yapılmalıdır. Bu durum otomatik testlerin çalıştırılmasında manuel test yapılmasını gerekli kıldığından ciddi bir kısıt oluşturmaktadır.



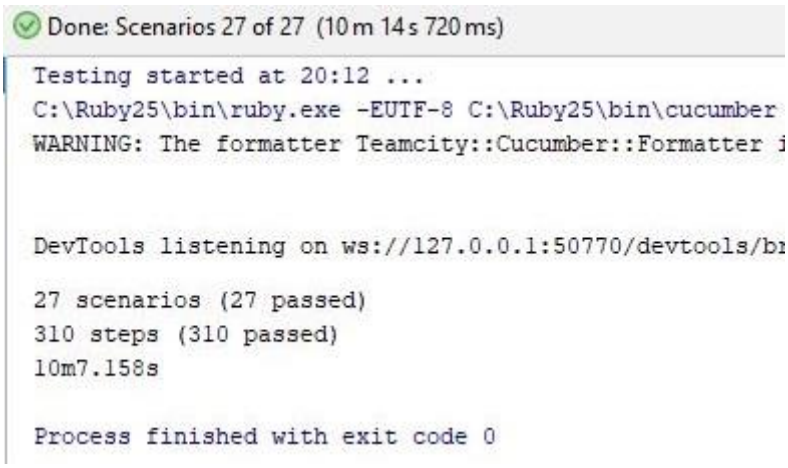
Resim 4.16 Öğretmen yetkili kullanıcı feature-3 optimizasyonu

Resim 4.16 da görüldüğü üzere öğretmen kullanıcı rolüne ait üçüncü özellik dosyasındaki otomatize edilen 29 senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 11 dakika 55 saniye 747 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 11 dakika 49 saniye ve 671 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 29 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 372 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir. Bu özellik dosyasında çalıştırılmadan önce otomatize edilemeyen TTS051 senaryosun testi manuel olarak yapılmalıdır. Bu durum otomatik testlerin çalıştırılmasında manuel test yapılmasını gerekli kıldığından ciddi bir kısıt oluşturmaktadır.



Resim 4.17 Öğretmen yetkili kullanıcı feature-4 optimizasyonu

Resim 4.17 de görüldüğü üzere öğretmen kullanıcı rolüne ait dördüncü özellik dosyasındaki otomatize edilen 26 senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 10 dakika 24 saniye 536 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 10 dakika 17 saniye ve 879 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 26 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 321 adımdan oluştuğu ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir.



Resim 4.18 Öğretmen yetkili kullanıcı feature-5 optimizasyonu

Resim 4.18 de görüldüğü üzere öğretmen kullanıcı rolüne ait son özellik dosyasındaki otomatize edilen 27 senaryonun optimizasyonu tamamlanmıştır. Sonrasında çalıştırılan özellik dosyasındaki otomatik testlerin başarılı olarak çalıştığı görülmektedir. Resmin üst kısmında bu otomatik testlerin tamamının toplam çalışma süresi 10 dakika 14 saniye 720 milisaniye olduğu görülmektedir. Bu süre çalıştırma, web tarayıcının açılması, testlerin sırayla çalıştırılması ve sonlanması gibi uçtan uca geçen zaman bilgisidir. Resmin alt kısmında ki belirtilen 10 dakika 7 saniye ve 158 milisaniyelik zamanda otomatik test senaryolarının çalıştığı toplam zamanı göstermektedir. Bu alandaki diğer bilgi 27 test senaryosunun tamamı passed (başarılı) olduğu görülmektedir. Senaryoların tamamı 310 adım olup ve tüm adımlarında başarılı olarak tamamlandığı görülmektedir.

4.4 Test Otomasyonu ve Manuel Test Karşılaştırması

Manuel testlerin uygulama hızı, test insan kaynağı tarafından yapıldığından yavaştır, bu nedenle zaman alıcı ve sıkıcı olmaktadır. Otomasyon testi, test durumlarını insan kaynaklarından çok daha hızlı yürütmektedir. Test otomasyonunda test senaryoları otomasyon aracı kullanılarak gerçekleştirilir, böylece otomasyon testinde daha az test cihazı gerekmektedir. Bu nedenle insan kaynaklarına daha az yatırım ihtiyacı görülmektedir. Manuel testte, zor olan karmaşık testler yazmak için programlama yapılamamaktadır. Test otomasyonunda ise test cihazlarında programlanabilen otomasyon testleri, zor testler ortaya çıkarmak için karmaşık testler programlanabilir. Manuel testler daha az güvenilirdir, çünkü testler insan hataları nedeniyle her seferinde hassas şekilde yapılamamaktadır. Otomasyon testi daha güvenilir ve daha az hata içermektedir.

Çalışmanın bu bölümünde Moodle uygulamasının manuel test süreçleri ve test otomasyon süreçleri karşılaştırılarak değerlendirilmektedir. Bu değerlendirme her iki test sürecinde harcanan zaman, insan kaynağı, olumlu ve olumsuz yönleri, sınırlılıkları ve kullanılabilirliği durumlarına göre karşılaştırılmıştır.

4.4.1 Zaman

Manuel test süreçlerinde tüm test senaryoları el ile manuel olarak yapılmasından testlerin tamamlanmasının zaman aldığı görülmüştür. Tüm test senaryolarının manuel tamamlanma süresi ortalama değeri 2 adam gün (1 test uzmanı kaynağı için) sürdüğü görülmüştür. Bu sürenin saat karşılığı (2*8) 16 saate yakın bir zaman dilimidir. Bu süreçte yapılan işleri test senaryoların gözden geçirilmesi, test verilerinin oluşturulması, testlerin yürütülmesi ve raporlanması olarak belirtilmektedir. Otomasyon test sürecinde ise test senaryolarının otomatize edilme süresi ortalama 20 adam gün (1 test uzmanı kaynağı için) sürdüğü görülmüştür. Bu süre (20*8) 160 saate karşılık gelmektedir. Fakat bu zaman ilk başta testlerin oluşturulması için harcanan zamandır. Sonrasında testler her defasında otomatik çalıştırıldığında ortalama 80 dakika zaman almaktadır. Bu süreçte yapılanlar otomasyon test yazılımların kurulması, test projesinin oluşturulması,

test senaryoların otomatize edilmesi ve optimizasyon süreci olarak belirtilmektedir. Bu durumda çevik yazılım geliştirmede yazılım paketlerinin iki haftada (iterasyon) bir teslim edildiğini göz önüne alındağında manuel testlere her iterasyonda 2 gün (16 saat) zaman ayrılması gerekmektedir. Fakat testler otomatize edildiği durumda her iterasyondaki ayrılması gereken zaman ortalama 80 dakika diğer operasyonel süreçlerle (otomatik testlerin bakım ve revizyonları) birlikte bu süreyi maksimum 3 saat belirtilmektedir. Bu durum yazılım test otomasyonun manuel teste göre tercih sebebi olabilir.

Tablo 4.3 de görüldüğü üzere otomatize edilen 236 test senaryosu otomatik test aracı yardımıyla çalıştırıldığında geçen süre 81 dakika 16 saniye olarak görülmektedir. Bu değerin saat karşılığı 1 saat 21 dakika olmaktadır. Bu süre otomatik test senaryoların seti 1 kez çalıştırıldığında geçen süredir. Manuel testler ile bu senaryolar yaklaşık olarak 16 saat gibi bir sürede tamamlanmaktadır.

Tablo 4.3 Kullanıcı tiplerine ait test koşum süreleri

Kullanıcı Rolü Kodu	**Senaryo Sayısı**	**Süre(dk:sn)**
VTS	26	03:28
STS	82	28:07
TTS	128	49:41
Genel Toplam	**236**	**81:16**

4.4.2 İnsan Kaynağı

Bu çalışmada Moodle uygulamasının manuel test sürecinde test uzmanı testleri gerçekleştirirken her hangi bir test aracı kullanmamıştır. Genel olarak manuel testlerde test sırasında test araçlarına pek ihtiyaç duyulmamaktadır. Test araçları genellikler test bulgularını ve test raporlarının oluşturtulması için test süreci sonrasında kullanılmaktadır. Bu nedenle manuel test sürecinde test uzmanının öğrenmesi ya da kullanması gereken ekstra bir yazılım olmaması test uzmanı açısından kolaylık oluşturmuştur. Fakat yazılım test otomasyonu geliştirme sürecinde durum böyle değildir. Test otomasyonu geliştirecek yazılım test uzmanı en bir yazılım dili ve o

yazılım diline uyumlu test otomasyon yazılım araçlarını bilmesi ya da öğrenmesi gereklidir. Bununla birlikte otomatik testleri geliştirdikten sonra testlerin otomatik yürütüleceği sistem ve yazılım teknolojilerini bilmesi ya da öğrenmesi gerekmektedir. Bu çalışmada test uzmanı Ruby yazılım dilini ve diğer test araçları yazılımlarını kullanmıştır. Çalışma öncesinde ilgili yazılım araçlarını biliyor olması çalışma sırasında büyük kolaylık sağladığı görülmüştür. Bu durumlar göz önüne alındığında yazılım test otomasyonu geliştirmesi yapan test uzmanları kendilerini ilgili sistem ve yazılım araçları teknolojileri öğrenme konusunda geliştirmelidir. Bununla birlikte güncel yenilikleri ve teknolojileri takip etmeleri önem taşımaktadır.

4.4.3 Olumlu ve Olumsuz Yönler

Tablo 4.4 de belirtilen, çalışmada manuel test sürecinde testlerin uzun sürdüğü ve zaman alıcı olduğu görülmektedir. Manuel testlerde test uzmanının her hangi bir test aracı ihtiyaç olmadan testleri tamamlaması test uzmanına kolaylık sağladığı görülmüştür. Bu test uzmanı tarafından olumlu bir durumdur. Herhangi bir eğitime ihtiyaç duyulmadan mevcut deneyimleriyle testlere doğrudan başlanması istenilen bir durumdur. Manuel testlerde her hangi bir yazılım kullanılmadığından lisans ücretlerin olmaması maliyet yönünden olumlu bir durumdur.

Tablo 4.4 Manuel test - test otomasyonu olumlu ve olumsuz yönleri

Manuel Yazılım Testi	Testler zaman alıcı.	Olumsuz
	Test aracı olmaması.	Olumlu
	Eğitim süreci olmaması	Olumlu
	Test aktivitesine doğrudan başlanması.	Olumlu
	Test aracı olmaması.	Olumlu
	Lisans maliyeti olmaması.	Olumlu
Otomasyon Testi	Testlerin hazırlık süresi zaman alıcı.	Olumsuz
	Testlerin çalışma süresi kısa.	Olumlu
	Eğitim ve öğrenme süreci zaman alıcı	Olumsuz
	Programlama bilgisi olmaması.	Olumsuz
	Test araçlarının lisans maliyetleri.	Olumsuz

Çalışmada otomasyon test sürecinde otomatik testleri geliştirme aşaması zaman alıcı fakat sonraki evrelerde testlerin çalıştırılmasında ciddi anlamda zaman kazancı sağladığı görülmektedir. Otomasyon testi süreçlerinde farklı yazılım geliştirme ve test yazılım araçlarının kullanılması otomasyon geliştirme tecrübesi olmayan ve yazlım geliştirme bilgisi olmayan test uzmanları için olumsuz bir durumdur. Çalışmada kullanılan IDE aracı RubyMine lisansı edu uzantılı mail adresi olan öğrenci ve üniversite grubuna bir yıl ücretsiz kullanım hakkı sunması ve diğer yazılım araçlarına ücretsiz erişim sağlanabilmesi yazılım lisans maliyetinin olmaması bakımından olumlu bir durumdur.

4.4.4 Sınırlılıklar

Çalışmada manuel testlerin belirli zaman aralıklarında tekrarlanması durumu göz önüne alındığında sabit insan kaynağı gerekliliği ve zaman alması yönleriyle sınırlılıklar oluşturduğu görülmüştür. Çalışmada otomasyon testi süreçlerinde otomatize edilemeyen test senaryoları otomatik testlerin tekrarlanarak çalıştırılmasında sınırlılık oluşturduğu görülmektedir. Otomatize edilemeyen test senaryoları otomatik testler çalıştırılmadan önce manuel olarak çalıştırılması gerekmektedir. Eğer çalıştırılmadığı durumlardan bu test senaryolarına bağlı diğer otomatik testler çalıştığında hata almaktadır. Otomatize edilemeyen test senaryolarındaki bulgularla ilgili yazılım geliştirmelerin yapılması sonrasında testlerin otomatize edilmesiyle bahsedilen sınırlılıkların kalkacağı ön görülmektedir. Diğer bir durum otomatik testleri geliştirme sırasında Moodle sistem tasarım ve geliştirme uzmanlarıyla birlikte çalışılması gerekli görülmektedir.

4.4.5 Kullanılabilirlik

Manuel test süreçleri dikkate alındığında testlerin manuel ilerletilmesi ve herhangi bir ek yazılım ya da araçlara ihtiyaç duyulmamasından dolayı kullanım kolaylığı olduğu görülmektedir. Bu nedenle de kullanılabilirlik seviyesinin yüksek olduğu söylenmektedir. Otomasyon testi süreçlerinde testlerin otomatize edilmesinde yazılım dili ve diğer test araçlarının gerekliliği bulunmaktadır. Bu araçların öğrenilmesi ve sonrasında kullanılması bağlılık oluşturduğu görülmektedir. Bu nendenle kullanılabilirlik seviyesinin düşük olduğu görülmüştür.

5. TARTIŞMA ve SONUÇ

Bu tez çalışmasında Moodle uygulamasının yazılım testleri gerçekleştirilmiştir. Davranış Odaklı Geliştirme yaklaşımıyla Moodle uygulamasının kullanıcı rolleri açısından fonksiyonel olarak özelliklerinin çalışabilirliğini kontrol edecek test otomasyonu geliştirilmiştir.

Moodle uygulaması ziyaretçi, öğrenci ve öğretmen kullanıcı rol gruplarına ayrılarak çalışma planlaması yapılmıştır. Kullanıcı rollerine ait fonksiyonel analiz sonrasında her bir durum için senaryolar oluşturulmuştur. Oluşturulan kullanıcı senaryoları Davranış Güdümlü Geliştirme yaklaşımında belirtilen formata göre yazılmıştır. Bu yaklaşımına göre belirlenen formatta yazılan senaryolar için alanında uzman kişilerin görüşleri sade, basit, anlaşılabilir olduğu ve karmaşıklığa yol açabilecek bir yazım dilinin olmadığı yönde alınan geribildirimler, Okolnychyi ve Fögen (2016) yaptıkları çalışmalarında ulaştıkları sonuçlarla benzerlik göstermektedir.

Yazılan test senaryolarının bazıları Moodle sistemi hakkında bilgisi olmayan ve Davranış Odaklı Geliştirme yaklaşımına göre hazırlanan senaryolarla test yapmamış test uzmanları tarafınca testlerin yapılması sağlanmıştır. Test uzmanlarının ortak görüşleri Davranış Odaklı Geliştirme yaklaşımına uygun formatta yazılan test senaryolarının sistem hakkında bilgisi olmamalarına rağmen basit ve anlaşılır olmasından dolayı testleri yapabildikleri geribildirimi alınmıştır. Bu geribildirimlerin Nečas (2011) çalışmasında ulaşmış olduğu sonuçlarla benzerlik göstermektedir. Buna bağlı olarak Moodle sistemi hakkından bilgisi olmayan alanında uzman kişilerin BDD yaklaşımın sunduğu basit ve kolay anlaşılabilir olması sayesinde Moodle sistemine ait testleri yürütebilecekleri söylenebilir.

Tablo 4.5 de görüldüğü üzere 243 adet yazılan test senaryosundan 236 âdeti otomatik hale getirilerek test otomasyonu yazılmıştır. Kalan 7 adet test senaryosunun otomatik testleri yazılamamıştır. Genel olarak bakıldığında testleri otomatik hale getirme işleminde başarı oranı yüzde 97,1 olarak görülmektedir. Bu sonuç çalışmanın verimliliği açısında istenen düzeyde ve olumlu bir sonuç olarak görülmektedir.

Genelden özele doğru gittiğimizde ise ziyaretçi rolündeki kullanıcıya ait senaryoların tamamının otomatik hale getirilmiştir. Buradaki başarı sonucu yüzde 100 olarak görülmektedir. Öğrenci rolündeki kullanıcıya ait senaryoların 82 âdeti otomatik hale getirilmiştir. Buradaki başarı sonucu yüzde 97,6 olarak görülmektedir. Öğretmen rolündeki kullanıcıya ait senaryoların 128 adeti otomatik hale getirilmiştir. Buradaki başarı sonucu yüzde 96,2 olarak görülmektedir.

Tablo 5.1 Yazılan test senaryoları ve otomatize edilen test senaryoları

Kullanıcı Rolü	**Yazılan Senaryo Sayısı**	**Otomatize edilen Senaryo Sayısı**	**Otomatize edilemeyen Senaryo Sayısı**
Ziyaretçi	26	26- (100%)	0
Öğrenci	84	82 - (97,6%)	2 - (2,4%)
Öğretmen	133	128 - (96,2%)	5 - (3,8%)
Toplam	**243**	**236 - (97,1%)**	**7 - (2,9%)**

Otomasyon testlerini geliştirme aşamasında Ruby dili kullanılarak Davranış Odaklı Geliştirme yaklaşımına formatında yazılan bir senaryo adımına karşılık, rb dosyasına kod satırı yazılmıştır. Özellik dosyasındaki adım ne kadar tekrarda kullanılırsa kullanılsın rb kod dosyasına ekstradan kod yazılmasına gerek olmadığı görülmektedir. Bu durumda test otomasyonunda gereksiz kod yazımlarının ve kod kalabalığının önüne geçilmiştir. Akyol ve Gümüşkaya (2016) çalışmalarında Ruby yazılım dili diğer dillere göre öğrenilmesi daha kolay ve kısa zaman alan bir yazılım dili olması sonucu bu çalışmadaki edinilen deneyimlerle benzer ölçüdedir. Bu na bağlı olarakda Ruby, Cucumber, Gherkin ve Selenium Webdriver yazılım araçları kullanarak test otomasyon çalışmasının yapılabileceği görüşmüştür.

Çalışmada manuel ve otomatik test süreçleri değerlendirildiğinde test otomasyonu ile otomatize edilen testlerin ileriki süreçlerde zamandan ciddi oranda kazanç sağladığı sonucuna varılmıştır. Test süreçlerinde ve aktivitelerinde ileriki zamanlarda daha az insan kaynağına ihtiyaç duyulduğu görülmektedir. Gebizli vd. (2013) çalışmalarındaki görüşleride bener olarak belirtmişlerdir.

Çalışmada test otomasyon sürecinin diğer paydaşlardan (ürün sahibi, analiz ve yazlım

geliştirici) bağımsız yürütülmesi ve Moodle uygulamasının fonksiyonel analiz, manuel ve otomasyon testi süreçlerinin uzamasına neden olduğu sonucuna ulaşılmıştır. Çalışma süresince fonksiyonel özelliklerin analizi, iş akışlarının olmaması ve geliştirilen kodlar hakkında bilgi alınamaması gibi zorluklarla karşılaşıldığı görülmektedir. Davranış Odaklı Geliştirme yaklaşımında paydaşların bir arada olmasıyla etkili iletişim ve etkileşimi sağlayarak test otomasyon sürecinin verimliği artacağı görüşüne varılmıştır.

Otomatik test süreçlerinde kullanıcılara ait fonksiyonel özelliklerin otomatize edilmesi sırasında karşılaşılan hataların neredeyse tamamına yakını yazılım geliştirme aşamasında ilgili yazılım geliştiricinin eksik ya da hiç yapmadığı css tanımlamalarında olduğu görülmektedir. Bu nedenle yazılım geliştirme sırasında geliştirilen web sayfasındaki elementlerin css tanımlamalarının yapılması gerekmektedir. Css tanımlamalarında kullanılan class ve id yapıları bu konuda önem taşımaktadır. Yazılım geliştirmede benzer yapıda class ve id yapıların kullanılması test otomasyon sürecinde bir takım problemlere yol açtığı görülmektedir. Bu problemler aynı class ya da id ye ait elementleri seçme veya veri girişlerinde ilgili elementi bulma ve seçme işleminde probleme yol açmaktadır. Geliştirmede aynı class, id isimlerinin kullanılması test otomasyonu sırasında fazladan kodlar yazılmasında neden olmakta ve bu durumda gereksiz kod kalabalığına neden olmaktadır. Herhangi bir elementte css tanımında benzeri olmayan tek bir id ya da class adı verilmesi durumunda test otomasyonu sırasında bir satır kod ile işlem yapılırken aynı id ya da class tanımları kullanıldığında üç ya da dört satır kod yazarak otomasyon işlemi yapıldığı görülmektedir.

Çalışmada otomatik test süreçleri aşamasında bazı fonksiyonel kullanıcı hikâyelerinin otomatize edilemediği görülmektedir. Testlerin otomatize edilmesinin amaçlarından biri de otomatik testlerin istenilen zamanda belirlenen aralıklarla çalışmasıdır. Bu durum göz önüne alındığında otomatikleştirilmeyen testler bu durumda engelleyici problemler oluşturduğu görülmektedir. Otomatize edilen testler istenilen zamanlarda otomatik çalıştırılamayacaktır. Bu nedenden dolayı ilgili senaryoların otomatize edilmesi önem taşımaktadır. Sistemin belirlenen fonksiyonel kullanıcı davranışlarının çalışırlığının doğruluğu açısından bu önemlidir.

Çalışmada otomatik testlerin optimizasyonu sırasında otomatize edilemyen senaryolar belirlenen yer ve zamanda manuel olarak yapılmıştır. Optimizasyon sürecince test setinin sürekli ve tekrarlı çalışılabilirliği durumu göz önüne alındığında manuel çalıştırılan testler istenmeyen bir durumdur. Bu durum aynı zamanda testlerin optimizasyon süreçlerini zamansal olarak ciddi oran da uzattığı görülmektedir.

Bu tez çalışmasında Moodle uygulamasının fonksiyonel analizi sürecinde sistem analisti ve yazılım geliştiricilerin bir arada ve iletişim halinde olmamasından dolayı fonksiyonel analiz sürecinin uzamasına neden olmuştur. Bu nedenle yapılacak benzer çalışmalarda sistemlerin fonksiyonel analizi yapılırken ve kullanıcı hikâyeleri oluşturulurken test uzmanı ilgili sistem analistleri ve yazılım geliştirme uzmanlarıyla birlikte hareket etmesi önerilir. Çalışmada testlerin otomatim hale getirilmesi sırasında sistemin mevcut durumundan dolayı gereksiz kodların yazıldığı bölümler oldu. Yapılacak benzer çalışmalarda gereksiz kod yazımının olmaması için test otomasyonu sürecinde yazılım geliştiriciyle birlikte çalışılması gerekliliği önerilir. Yazılım geliştirici test otomasyon yapısına uygun kod yazması önerilir. Bu sayede testlerin otomatize edilmesinde gereksiz kod yazımından uzak durulabilir ve test otomasyon süresi kısalabilir.

6. KAYNAKLAR

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Ammann, P., and Offutt, J. (2016). Introduction to software testing. Cambridge University Press.

Akyol, G., ve Gümüşkaya, H. (2016). Çevik yazılım geliştirmede BDD/TDD yöntemlerinin ve yazılım kalite araçlarının kullanılması: Bir yazılım mühendisliği dersindeki tecrübe. CEUR Workshop Proceedings.

Balaji, S., and Murugaiyan, M. S. (2012). Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management, **2*:** 26-30.

Baran, A., Akar, F., ve Aslay, F(2016). Çevik Yazılım Geliştirme Yöntemlerinde Etkili Test Araçları. Uluslararası Bilim ve Teknoloji Konferansı, Vienna, Avusturya, pp.1-1.

Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. arXiv preprint arXiv:1205.6904.

Beck, K. (2003). Test Driven Development: By Example Addison-Wesley. *Reading, MA*.

Bilgin, Ö., ve Kasapoğlu, M. (2016) Kullanıcı Arayüzü Yazılımı Test Otomasyonundan Beklentiler ve Riskler. 10. Ulusal Yazılım Mühendisliği Sempozyumu, Çanakkale.

Büyükyumukoğlu, G., Ersoy, E., Özdemir, M. Ş., Bağrıyanık, S., ve Karahoca, A. (2017). Test Otomasyonunda Karşılaşılan Zorluklar ve Öğrenilen Dersler: Telekomünikasyon Sektöründen Deneyimler. 11. Ulusal Yazılım Mühendisliği Sempozyumu, Alanya.

Campanelli, A. S. and Parreiras, F. S. (2015). Agile methods tailoring – A systematic literature review. *Journal of Systems and Software*, ***110*****:** 85–100.

Carter, J. D.,(2017). BHive: Behaviour-Driven Development Meets B-Method. The University of Guelph, Computer Science, Canada.

Chelimsky, D., Astels, D., Helmkamp, B., North, D., Dennis, Z., and Hellesoy, A. (2010). The RSpec Book: Behaviour Driven Development with Rspec. Cucumber, and Friends. Pragmatic, United States.

Cohen, D., Lindvall, M. and Costa, P. (2004). An Introduction to Agile Methods. *Advances in Comp.*, ***62*****:** 1–66.

Çıltık, A., Bağcı, V. H., Turgut, U. O., ve Güngör, T. (2014). Bankacılık Alanında Doğal Dil İşleme Destekli Davranış Odaklı Geliştirme. 8. Ulusal Yazılım Mühendisliği Sempozyumu, ODTÜ, KKTC.

Çoşkun, R. (2016). Practıcal Model Based Software Testıng. Yüksek Lisans Tezi, Atılım Üniversitesi, Fen Bilimleri Enstitüsü, Ankara.

Dees, I., Wynne, M., and Hellesoy, A. (2013). Cucumber Recipes: Automate Anything with BDD Tools and Techniques. Pragmatic, United States.

De Souza, P. L., do Prado, A. F., de Souza, W. L., Pereira, S. M. D. S. F., and Pires, L. F. (2017). Combining Behaviour-Driven Development with Scrum for Software Development in the Education Domain. In *ICEIS (2)* (pp. 449-458).

Dimanidis, A., Chatzidimitriou, K. C., and Symeonidis, A. L. (2018). A Natural Language Driven Approach for Automated Web API Development: Gherkin2OAS. In *Companion of the The Web Conference 2018 on The Web Conference 2018* (pp. 1869-1874).

Emektar, M., Altınsoy, Y., ve Erdem, U. (2018). Sigortacılık Web Servislerinde Test ve Test Otomasyonu Yaklaşımı. 12. Ulusal Yazılım Mühendisliği Sempozyumu, İstanbul.

Evgrafov, I., Hocke, R., and Smirnova, E. V. (2015). Analysis of Domain Specific Languages for GUI testing: RSpec and Cucumber for Sikuli. *Journal of Multidisciplinary Engineering Science and Technology*, **2.**

Gebizli, C. Ş., Metin, D., ve Vestel, A. R. G. E. (2013). Kullanım Modeli Bazlı Otomatik Test Tasarımı. 7. Ulusal Yazılım Mühendisliği Sempozyumu, İzmir.

Haberl, P., Spillner, A., Vosseberg, K., and Winter, M. (2011). Survey 2011 Software Test in Practice . *Translation of Umfrage*.

Highsmith, J. (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Journal of Evolutionary Biology (C. 12)

Jan, S. R., Shah, S. T. U., Johar, Z. U., Shah, Y., and Khan, F. (2016). An innovative approach to investigate various software testing techniques and strategies. *International Journal of Scientific Research in Science, Engineering and Technology*.

Jovanović, I. (2006). Software testing methods and techniques. The IPSI BgD Transactions on Internet Research, **5**.

Kasurinen, J., Taipale, O., and Smolander, K. (2010). Software test automation in practice: empirical observations. *Hindawi Publishing Corporation Advances in Software Engineering*, Article ID 620836.

Keus, A., Dyck, A. (2016). How Much Does Testing Cost?. *Full-scale Software Engineering/Current Trends in Release Engineering*, **1.**

Khan, M. E. (2010). Different forms of software testing techniques for finding errors. *International Journal of Computer Science Issues*, **7(3)**.

Khan, M. E., and Khan, F. (2012). A comparative study of white box, black box and grey box testing techniques. Int. J. Adv. Comput. Sci. Appl, **3(6)**.

Kniberg, H., and Skarin, M. (2010). Kanban and Scrum-making the most of both. C4Media, United States.

Koskela, L. (2008). Test driven. Manning Publications, United States.

Kuday, G. (2015). Yazılım Mühendisliği Yöntemleriyle Yazılım Test Süreci. Yüksek Lisans Tezi, Haliç Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul.

Kumar, P., and Syed, K. (2010). Software testing–goals, principles, and limitations. *International Journal of Engineering Science & Advanced Technology*, **1**.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E., Brinkkemper, S., and Zowghi, D. (2017). Behavior-Driven Requirements Traceability via Automated Acceptance Tests. In *2017 IEEE 25th International Requirements Engineering Conference Workshops* (pp. 431-434).

Nečas, I. (2011). BDD as a specification and QA instrument, Master Thesis, Masarykova Univerzita, Fakulta Informatiky, Czech Republic.

Nilsson, N. (2015). Test-Driven Development in Clojure: An Analysis Of How Differences Between Clojure And Java Affects Unit Testing And Design Patterns. Master Thesis, Kth Royal Instıtute Of Technology, School Of Computer Scıence And Communıcatıon, Sweden.

Okolnychyi, A., and Fögen, K. (2016). A study of tools for behavior-driven development. *Full-scale Software Engineering/Current Trends in Release Engineering*.

Oruç, A. F., and Ovatman, T. (2016). Testing of Web Services using Behavior-Driven Development. *Proceedings of the 6th International Conference on Cloud Computing and Services Science, Italy*.

Özalp, N., Bütün, G., ve Akagündüz, S. (2015) Web Projeleri İçin Otomatik Test Senaryosu Üreten ve Koşan Kara Kutu Test Çatısı. 9. Ulusal Yazılım Mühendisliği Sempozyumu, İzmir.

Özkan, U., ve Sözer, H. (2015). Web Uygulamaları İçin Model Bazlı Test Süreci Otomasyonu. 9. Ulusal Yazılım Mühendisliği Sempozyumu, İzmir.

Palmer, S. R., and Felsing, M. (2001). A practical guide to feature-driven development. Pearson Education, United States.

Rafi, D. M., Moses, K. R. K., Petersen, K., and Mäntylä, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proceedings of the 7th International Workshop on Automation of Software Test* (pp. 36-42).

Robbins, M. (2013). Application testing with Capybara. Packt Publishing, United Kingdom.

Rocha Silva, T., Hak, J. L., and Winckler, M. A. (2016). An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development. *Complex Systems Informatics and Modeling Quarterly*, *7*, 81-107.

Royce, W. (1970). Managing the Development of Large Software Systems, *Proceedings of IEEE WESCON*, **26:** 1-9.

Sharma, M., and Angmo, R. (2014). Web based automation testing and tools. *International Journal of Computer Science and Information Technologies*, ***5*:** 908-912.

Sawant, A. A., Bari, P. H., and Chawan, P. M. (2012). Software testing techniques and strategies. *International Journal of Engineering Research and Applications*, ***2*(3):**, 980-986.

Serdaroğlu, D. (2015). Yazılım Test Süreci Ve TMMI Modelinde Prisma Yaklaşımı Uygulaması. Yüksek Lisans Tezi, Başkent Üniversitesi, Fen Bilimleri Enstitüsü, Ankara.

Singla, S., and Kaur, H. (2014). Selenium Keyword Driven Automation Testing Framework. *International Journal of Advanced Research in Computer Science and Software Engineering Issn*, *2277*, **6:** 125-130.

Subramanian, R., Chen, N., and Zhu, T. (2017). Behavior Driven Test Automation Framework. In *Proceedings of the International Conference on Software*

Engineering Research and Practice (pp. 17-23). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing.

Stankovic, D., Nikolic, V., Djordjevic, M., and Cao, D. B. (2013). A survey study of critical success factors in agile software projects in former Yugoslavia IT companies. *Journal of Systems and Software*, 86(6), 1663-1678.

Tian, J. (2005). Software quality engineering: testing, quality assurance, and quantifiable improvement. IEEE Computer Society, Canada.

Yoo, S., and Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, **22(2)**, 67-120.

Wynne, M., Hellesoy, A., and Tooke, S. (2017). The cucumber book: behaviour-driven development for testers and developers. Pragmatic, United States.

İnternet Kaynaklari

1) https://agilemanifesto.org/, 19.05.2019
2) http://www.yazilimprojesi.com/, 19.05.2019
3) https://www.inflectra.com, 19.05.2019
4) https://medium.com/, 19.05.2019
5) https://www.guru99.com/, 19.05.2019
6) https://www.geeksforgeeks.org/, 19.05.2019
7) https://www.keytorc.com, 19.05.2019
8) https://www.edureka.co/, 19.05.2019
9) http://www.softwaretestingclass.com, 19.05.2019
10) http://www.softwaretestingmentor.com, 19.05.2019
11) https://www.ruby-lang.org/tr, 19.05.2019
12) https://www.pythontr.com, 19.05.2019
13) http://mucahit.blogspot.com, 19.05.2019
14) https://www.sitepoint.com, 19.05.2019
15) http://www.rubyinside.com, 19.05.2019

ÖZGEÇMİŞ

Adı Soyadı	: Fatih Kökten
Doğum Yeri ve Tarihi	: Muğla - 1988
Yabancı Dili	: İngilizce
İletişim	: fatihkoktenn@gmail.com

Eğitim Durumu (Kurum ve Yıl)

Lise	: Ticaret Meslek Lisesi, (2002-2005)
Lisans	: Afyon Kocatepe Üniversitesi, Bilgisayar ve Öğretim Teknolojileri Öğretmenliği, (2010-2014)

Çalıştığı Kurum/Kurumlar ve Yıl :

Afyon Kocatepe Üniversitesi	: 2011 - 2014
Ericsson	: 2014 - 2015
Huawei	: 2015 - 2016
CodeSpace	: 2016 - 2018
Netaş	: 2018 -

EKLER

EK 1. Ziyaretçi Kullanıcı Rolü Test Senaryoları

Giriş Modülü Senaryoları

VTS001: Bir Moodle ziyaretçi kullanıcı olarak Moodle web adresine gitmek istiyorum böylece Moodle ana sayfasını görüntülerim.

VTS002: Bir Moodle ziyaretçi kullanıcı olarak "Yeni hesap oluştur" alanına tıklamak istiyorum böylece yeni hesap oluştur ekranını görüntülerim.

VTS003: Bir Moodle ziyaretçi kullanıcı olarak "Yeni hesap bilgileri alanlarına veri girişleri yapıp" "Yeni Hesabımı oluştur" butonuna tıklamak istiyorum böylece hesap oluşturulduğunu gösteren "Bu e-posta adresinize (**eposta @deneme.com.tr**) bir mesaj gönderildi." mesajını görürüm.

VTS004: Bir Moodle kullanıcı olarak "Şifrenizi mi unuttunuz?" alnına tıklamak istiyorum böylece unutulan şifre ekranını görüntülerim.

VTS005: Bir Moodle ziyaretçi kullanıcı olarak unutulan şifre ekranında "Kullanıcı adı "alanını doldurup tamam butonuna tıklamak istiyorum böylece "Geçerli bir kullanıcı adı veya e-posta adresi belirttiyseniz size şu anda bir e-posta gönderildi." mesajını görürüm.

VTS006: Bir Moodle ziyaretçi kullanıcı olarak unutulan şifre ekranında "E-posta adresi "alanını doldurup tamam butonuna tıklamak istiyorum böylece "Geçerli bir kullanıcı adı veya e-posta adresi belirttiyseniz size şu anda bir e-posta gönderildi." mesajını görürüm.

VTS007: Bir Moodle ziyaretçi kullanıcı olarak "Kullanıcı adı" ve "Şifre" alanlarını doldurup "Giriş" butonuna tıklamak istiyorum böylece Moodle sistemine giriş yapabilirim.

Ana Menü Modülü Senaryoları

VTS008: Bir Moodle ziyaretçi kullanıcı olarak "Site haberleri" alanına tıklamak istiyorum böylece site haberleri sayfasını görürüm.

VTS009: Bir Moodle ziyaretçi kullanıcı olarak "Böte Sohbet Odası" alanına tıklamak istiyorum böylece böte sohbet odası sayfasını görürüm.

EK 1 (Devam). Ziyaretçi Kullanıcı Rolü Test Senaryoları

VTS010: Bir Moodle ziyaretçi kullanıcı olarak Öğretim Tasarımı Formu alanına tıklamak istiyorum böylece öğretim tasarımı formu sayfasını görürüm.
VTS011: Bir Moodle ziyaretçi kullanıcı olarak "İnsan Bilgisayar Etkileşimi Odası" alanına tıklamak istiyorum böylece insan bilgisayar etkileşimi odası sayfasını görürüm.

Forumlarda Ara Modülü Senaryoları

VTS012: Bir Moodle ziyaretçi kullanıcı olarak arama alanına "Ders" yazıp "Git" butonuna tıklamak istiyorum böylece arama sonuçları ekranını görüntülerim.
VTS013: Bir Moodle ziyaretçi kullanıcı olarak "Gelişmiş Arama" alanına tıklamak istiyorum böylece gelişmiş arama ekranını görüntülerim.
VTS014: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "Öğretim tasarımı" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.
VTS015: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu terim mesajda aynen olmalı" alanına "öğretim tasarımı" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.
VTS016: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler geçmemeli" alanına "Ders" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.
VTS017: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler geçebilir" alanına "öğretim tasarımı" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.
VTS018: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcük konuda geçmeli" alanına "tasarım" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.
VTS019: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcük yazar adıyla eşleşmeli" alanına "serdar" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

EK 1 (Devam). Ziyaretçi Kullanıcı Rolü Test Senaryoları

VTS020: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "öğretim tasarımı" yazıp, "Mesajlar bu tarihten daha yeni olmalı" kriterini 2013 seçip "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

VTS021: Bir Moodle ziyaretçi kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "öğretim tasarımı" yazıp, "Mesajlar bu tarihten daha eski olmalı" kriterini 01 April 2013 olarak seçip "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

Ders Kategorileri Modülü Senaryoları

VTS022: Bir Moodle ziyaretçi kullanıcı olarak "Miscellaneous" ders kategorisi alanına tıklamak istiyorum böylece miscellaneous ders kategorisi sayfasını görüntülerim.

VTS023: Bir Moodle ziyaretçi kullanıcı olarak "Formasyon" ders kategorisi alanına tıklamak istiyorum böylece formasyon ders kategorisi sayfasını görüntülerim.

VTS024: Bir Moodle ziyaretçi kullanıcı olarak "Böte" ders kategorisi alanına tıklamak istiyorum böylece böte ders kategorisi sayfasını görüntülerim.

VTS025: Bir Moodle ziyaretçi kullanıcı olarak ders arama alanına "Bilim Tarihi" yazıp ve "Git" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

VTS026: Bir Moodle ziyaretçi kullanıcı olarak alt ders kategorilerinden bir derse tıklamak istiyorum böylece alt ders kategorilerinden bir dersin sayfasını görüntülemek için kullanıcı giriş sayfasının açıldığını görürüm.

EK 2. Öğrenci Kullanıcı Rolü Test Senaryoları

Profil Sayfası Senaryoları

STS001: Bir Moodle öğrenci yetkili kullanıcı olarak ekranın sağ üst köşesindeki üye kullanıcı "Ad Soyad" yazan alana tıklamak istiyorum böylece kullanıcı profili ekranını görürüm.

STS002: Bir Moodle öğrenci yetkili kullanıcı olarak profil ekranında "Şifre Değiştir" butonuna tıklamak istiyorum böylece şifre değiştirme ekranını görürüm.

STS003: Bir Moodle öğrenci yetkili kullanıcı olarak şifre değiştir ekranında "Şimdiki Şifre", "Yeni Şifre", "Yeni şifre (Tekrarla)" bilgilerini girip değişiklikleri "Kaydet" butonuna tıklamak istiyorum böylece şifre değişikliğinin yapıldığını görürüm.

STS004: Bir Moodle öğrenci yetkili kullanıcı olarak profil ekranında "Mesajlar" butonuna tıklamak istiyorum böylece mesajlar ekranını görürüm.

STS005: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profili ekranında "Profil düzenle" tabına tıklamak istiyorum böylece profil düzenleme tabı ekranını görürüm.

STS006: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında "Gelişmişi Göster" butonuna tıklamak istiyorum böylece zorunlu olmayan diğer bilgi alanlarının geldiğini görürüm.

STS007: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında "Gelişmişi Gizle" butonuna tıklamak istiyorum böylece zorunlu olmayan diğer bilgi alanlarının gizlendiğini görürüm.

STS008: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında ilgili bilgileri güncelleyip "Profili güncelle" butonuna tıklamak istiyorum böylece profil bilgilerinin güncellendiğini görürüm.

STS009: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında "Resmi" bölümünde "Dosya Seç" butonundan resim seçip "Profili güncelle" butonuna tıklamak istiyorum böylece profil resmi eklendiğini görürüm.

STS010: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında "Resmi" bölümünde resim açıklaması yazıp "Profili güncelle" butonuna tıklamak istiyorum böylece resim açıklaması eklendiğini görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

STS011: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında "Resmi" bölümünde ilgi alanları açıklaması yazıp "Profili güncelle" butonuna tıklamak istiyorum böylece ilgi alanları açıklaması eklendiğini görürüm.

STS012: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında isteğe bağlı bölümünde "Gelişmişi Göster" butonuna tıklamak istiyorum böylece zorunlu olmayan diğer bilgi alanlarının geldiğini görürüm.

STS013: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında isteğe bağlı bölümünde "Gelişmişi Gizle" butonuna tıklamak istiyorum böylece zorunlu olmayan diğer bilgi alanlarının gizlendiğini görürüm.

STS014: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı profil düzenle ekranında isteğe bağlı bölümünde ilgili bilgileri girip "Profili güncelle" butonuna tıklamak istiyorum böylece isteğe bağlı bilgilerinin güncellendiğini görürüm.

STS015: Bir Moodle öğrenci yetkili kullanıcı olarak profil ekranında "Forum mesajları" tabına tıklamak istiyorum böylece Forum mesajları tabı ekranını görürüm.

STS016: Bir Moodle öğrenci yetkili kullanıcı olarak forum mesajları ekranında "Tartışmalar" butonuna tıklamak istiyorum böylece tartışmalar ekranını görürüm.

STS017: Bir Moodle öğrenci yetkili kullanıcı olarak profil ekranında "Blog" tabına tıklamak istiyorum böylece blog tabı ekranını görürüm

STS018: Bir Moodle öğrenci yetkili kullanıcı olarak "Blog" ekranında "Yeni girdi ekle" butonuna tıklamak istiyorum böylece yeni girdi ekle ekranını görürüm.

STS019: Bir Moodle öğrenci yetkili kullanıcı olarak yeni girdi ekleme ekranında "Girdi Başlığı" ve "Blog Metni" ekleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni girdi eklendiğini görürüm.

STS020: Bir Moodle öğrenci yetkili kullanıcı olarak profil ekranında "Etkinlik raporları" tabına tıklamak istiyorum böylece etkinlik raporları tabı ekranını görürüm.

Blog Menüsü Modülü Senaryoları

STS021: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Yeni girdi ekle" alanına tıklamak istiyorum böylece yeni girdi ekle ekranını görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

STS022: Bir Moodle öğrenci yetkili kullanıcı olarak yeni girdi ekleme ekranında "Girdi Başlığı" ve "Blog Metnini" girip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni girdi eklendiğini görürüm.
STS023: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Girdilerime bak" alanına tıklamak istiyorum böylece girdilerime bak ekranını görürüm.
STS024: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Düzenle" alanına tıklamak istiyorum böylece düzenleme ekranını görürüm.
STS025: Bir Moodle öğrenci yetkili kullanıcı olarak girdi düzenleme ekranında ilgili bilgileri düzenleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece blog girdisinin düzenlendiğini görürüm.
STS026: Bir Moodle öğrenci yetkili kullanıcı olarak girdi düzenleme ekranında "Dosya seç" ile ilgili dosyayı ekleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece blog girdisine dosya eklendiğini görürüm.
STS027: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Sil" alanına tıklayıp sonrasında "Evet" butonuna tıklamak istiyorum böylece ilgili girdinin silindiğini görürüm.
STS028: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Site bağlantısına bak" alanına tıklamak istiyorum böylece site bağlantısının açılmadığını görürüm.
STS029: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Blog seçenekleri" alanına tıklamak istiyorum böylece blog seçenekleri ekranını görürüm.
STS030: Bir Moodle öğrenci yetkili kullanıcı olarak blog menüsünde "Site girdilerine bak" alanına tıklamak istiyorum böylece site girdilerine ekranını görürüm.

Forumlarda Ara Modülü

STS031: Bir Moodle öğrenci yetkili kullanıcı olarak arama alanına "Ders" yazıp "Git" butonuna tıklamak istiyorum böylece arama sonuçları ekranını görürüm.
STS032: Bir Moodle öğrenci yetkili kullanıcı olarak "Gelişmiş arama" alanına tıklamak istiyorum böylece gelişmiş arama ekranını görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

STS033: Bir Moodle öğrenci yetkili kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "öğretim tasarımı" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

Ana Menü Modülü Senaryoları

STS034: Bir Moodle öğrenci yetkili kullanıcı olarak "Site haberleri" alanına tıklamak istiyorum böylece site haberleri sayfasını görürüm.
STS035: Bir Moodle öğrenci yetkili kullanıcı olarak "Böte Sohbet Odası" alanına tıklamak istiyorum böylece böte sohbet odası sayfasını görürüm.
STS036: Bir Moodle öğrenci yetkili kullanıcı olarak "Öğretim Tasarımı Formu" alanına tıklamak istiyorum böylece öğretim tasarımı form sayfasını görürüm.
STS037: Bir Moodle öğrenci yetkili kullanıcı olarak "İnsan Bilgisayar Etkileşimi Odası" alanına tıklamak istiyorum böylece insan bilgisayar etkileşimi odası sayfasını görürüm.

Derslerim Modülü Senaryoları

STS038: Bir Moodle öğrenci yetkili kullanıcı olarak "Proje Geliştirme ve Yönetimi 1" dersine tıklamak istiyorum böylece proje geliştirme ve yönetimi 1 dersine giriş yapmış olurum.

Haftalık Taslak Alt Modülü

STS039: Bir Moodle öğrenci yetkili kullanıcı olarak "Haber forumu" alanına tıklamak istiyorum böylece haber forumunu görürüm.
STS040: Bir Moodle öğrenci yetkili kullanıcı olarak "Proje nedir" alanına tıklamak istiyorum böylece proje nedir ders materyali içeriğini görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

STS041: Bir Moodle öğrenci yetkili kullanıcı olarak "Ödev-Rapor" teslimi alanına tıklayıp dosya seçip "Bu dosyayı yükle" butonuna tıklamak istiyorum böylece ödev gönderme işlemini tamamlamış olurum.
STS042: Bir Moodle öğrenci yetkili kullanıcı olarak "Ödev-Rapor" teslimi alanına tıklayıp yüklenen dosya adına tıklamak istiyorum böylece gönderilen ödevi görüntülerim.
STS043: Bir Moodle öğrenci yetkili kullanıcı olarak "Ödev-Rapor" teslimi alanına tıklayıp dosya seçip "Bu dosyayı yükle" butonuna tıklamak istiyorum böylece ödev gönderilemediğini görürüm.

Topluluk Alt Modülü

STS044: Bir Moodle öğrenci yetkili kullanıcı olarak topluluk modülünde ki "Katılımcılar" alanına tıklamak istiyorum böylece katılımcılar ekranını görürüm.
STS045: Bir Moodle öğrenci yetkili kullanıcı olarak katılımcılar ekranının da geçerli rolü "Öğretmen yetkili" seçmek istiyorum böylece öğretmen yetkili rolündeki katılımcıları görürüm.
STS046: Bir Moodle öğrenci yetkili kullanıcı olarak katılımcılar ekranının da geçerli rolü "Student" seçmek istiyorum böylece student rolündeki katılımcıları görürüm.
STS047: Bir Moodle öğrenci yetkili kullanıcı olarak katılımcılar ekranının da kullanıcı listesini "Daha fazla ayrıntılı" olarak seçmek istiyorum böylece katılımcı listesini daha ayrıntılı görürüm.
STS048: Bir Moodle öğrenci yetkili kullanıcı olarak katılımcılar ekranının da "Bu günden fazla etkin olmayanları" 2 ay olarak seçmek istiyorum böylece 2 ay'dan fazla etkin olmayanlar kullanıcıları görürüm.
STS049: Bir Moodle öğrenci yetkili kullanıcı olarak katılımcılar ekranının da "Kullanıcı resmi" alanına tıklamak istiyorum böylece katılımcının profilini görürüm.
STS050: Bir Moodle öğrenci yetkili kullanıcı olarak katılımcılar ekranının da "Bloglar" tabına tıklamak istiyorum böylece bloglar ekranını görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

Etkinlikler Alt Modülü

STS051: Bir Moodle öğrenci yetkili kullanıcı olarak "Forumlar" alanına tıklamak istiyorum böylece forumlar ekranını görürüm.
STS052: Bir Moodle öğrenci yetkili kullanıcı olarak forumlar ekranında "Haber forumuna" tıklamak istiyorum böylece forum içeriğini görürüm.
STS053: Bir Moodle öğrenci yetkili kullanıcı olarak forumlar ekranında "Tüm forumlara abone ol" alanına tıklamak istiyorum böylece ilgili derste ki tüm forumlara abone olurum.
STS054: Bir Moodle öğrenci yetkili kullanıcı olarak forumlar ekranında "Tüm forumlardan aboneliği kaldır" alanına tıklamak istiyorum böylece ilgili derste ki tüm forumlardan aboneliği kaldırırım.
STS055: Bir Moodle öğrenci yetkili kullanıcı olarak forumlar ekranında arama alanına "Bilgisayar" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece forumlarda arama işlemi yaparım.
STS056: Bir Moodle öğrenci yetkili kullanıcı olarak "Kaynaklar" alanına tıklamak istiyorum böylece kaynaklar ekranını görürüm.
STS057: Bir Moodle öğrenci yetkili kullanıcı olarak kaynaklar ekranında kaynak adına tıklamak istiyorum böylece kaynak içeriğini görürüm.
STS058: Bir Moodle öğrenci yetkili kullanıcı olarak "Ödevler" alanına tıklamak istiyorum böylece ödevler ekranını görürüm.
STS059: Bir Moodle öğrenci yetkili kullanıcı olarak ödevler ekranında ödev adına tıklamak istiyorum böylece ödevin içeriğini görürüm.

Forumlarda Ara Alt Modülü

STS060: Bir Moodle öğrenci yetkili kullanıcı olarak arama alanına "Ders" yazıp "Git" butonuna tıklamak istiyorum böylece arama sonuçları ekranını görürüm.
STS061: Bir Moodle öğrenci yetkili kullanıcı olarak "Gelişmiş arama" alanına tıklamak istiyorum böylece gelişmiş arama ekranını görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

STS062: Bir Moodle öğrenci yetkili kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "öğretim tasarımı" yazıp "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

Yönetim Alt Modülü

STS063: Bir Moodle öğrenci yetkili kullanıcı olarak "Notlar" alanına tıklamak istiyorum böylece kullanıcı raporu ekranını görürüm.
STS064: Bir Moodle öğrenci yetkili kullanıcı olarak kullanıcı raporu ekranında "Not ögesine" tıklamak istiyorum böylece not ögesi ekranını görürüm.
STS065: Bir Moodle öğrenci yetkili kullanıcı olarak "Profil" alanına tıklamak istiyorum böylece profil ekranını görürüm.
STS066: Bir Moodle öğrenci yetkili kullanıcı olarak "Profil düzenle" tabına tıklamak istiyorum böylece profil düzenleme tabı ekranını görürüm.

Derslerim Alt Modülü

STS067: Bir Moodle öğrenci yetkili kullanıcı olarak "Bilgisayar Ağları ve İletişim" alanına tıklamak istiyorum böylece bilgisayar ağları ve iletişim dersi ekranını görürüm.
STS068: Bir Moodle öğrenci yetkili kullanıcı olarak "Tüm dersler" alanına tıklamak istiyorum böylece tüm dersler ekranını görürüm.

Son Haberler Alt Modülü

STS069: Bir Moodle öğrenci yetkili kullanıcı olarak ilgili haber başlığının yanındaki "devamı…" alanına tıklamak istiyorum böylece haber içeriğini görürüm.
STS070: Bir Moodle öğrenci yetkili kullanıcı olarak daha "Eski konular" alanına tıklamak istiyorum böylece ilgili foruma ait tüm haberleri görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

Yaklaşan Olaylar Alt Modülü

STS071: Bir Moodle öğrenci yetkili kullanıcı olarak "Yeni Olay" alanına tıklamak istiyorum böylece yeni olay ekranını görürüm.
STS072: Bir Moodle öğrenci yetkili kullanıcı olarak yeni olay ekranında ilgili alanları doldurup "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni olay yaratma işlemini tamamlarım.
STS073: Bir Moodle öğrenci yetkili kullanıcı olarak yaklaşan olay menüsü altında ilgili alana tıklamak istiyorum böylece ilgili yaklaşan olayın ekranını görürüm.
STS074: Bir Moodle öğrenci yetkili kullanıcı olarak "Takvime git…" alanına tıklamak istiyorum böylece takvim ekranını görürüm.
STS075: Bir Moodle öğrenci yetkili kullanıcı olarak "Takvimi dışa ver" butonuna tıklayıp açılan ekranda dışa ver butonuna istiyorum böylece ilgili takvim dosyasının indirildiğini görürüm.
STS076: Bir Moodle öğrenci yetkili kullanıcı olarak "Takvimi dışa ver" butonuna tıklayıp açılan ekranda "Takvim URL getir" butonuna istiyorum böylece ilgili takvim url bilgisini görürüm.
STS077: Bir Moodle öğrenci yetkili kullanıcı olarak takvim ekranında olayı "Düzelt ikonuna" tıklayıp ilgili düzenlemeden sonra "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece olay düzenleme işlemini tamamlarım.
STS078: Bir Moodle öğrenci yetkili kullanıcı olarak takvim ekranında olayı sil "İkonuna" tıklayıp sonra sil butonuna tıklamak istiyorum böylece olay silme işlemini tamamlarım.

Son Etkinlikler Alt Modülü

STS079: Bir Moodle öğrenci yetkili kullanıcı olarak "Son etkinliklerin tüm raporları" alanına tıklamak istiyorum böylece bütün katılımcılar ekranını görürüm.

EK 2 (Devam). Öğrenci Kullanıcı Rolü Test Senaryoları

STS080: Bir Moodle öğrenci yetkili kullanıcı olarak bütün katılımcılar ekranında "Gelişmişi Göster" butonuna tıklamak istiyorum böylece zorunlu olmayan diğer bilgi alanlarını görürüm.
STS081: Bir Moodle öğrenci yetkili kullanıcı olarak bütün katılımcılar ekranında "Gelişmişi Gizle" butonuna tıklamak istiyorum böylece zorunlu olmayan diğer bilgi alanlarının gizlendiğini görürüm.
STS082: Bir Moodle öğrenci yetkili kullanıcı olarak bütün katılımcılar ekranında sırala kriterini "Tarih en yeniler-önce" seçip tarihi "Pasifleştir" alanını işaretleyip "Son etkinlikleri göster" butonuna tıklamak istiyorum böylece ilgili sonuç listesini görürüm.

Tüm dersler

STS083: Bir Moodle öğrenci yetkili kullanıcı olarak "Tüm dersler" butonuna tıklamak istiyorum böylece tüm dersler ekranını görürüm.

Dersleri ara

STS084: Bir Moodle öğrenci yetkili kullanıcı olarak "Dersleri ara" alanına "İnsan Bilgisayar etkileşimi" yazıp "Git" butonuna tıklamak istiyorum böylece arama sonuçları ekranını görürüm.

EK 3. Öğretmen Kullanıcı Rolü Test Senaryoları

Profil Sayfası Senaryoları

TTS001: Bir Moodle öğretmen yetkili kullanıcı olarak ekranın sağ üst köşesindeki üye kullanıcı adı soyadı yazan alana tıklamak istiyorum böylece kullanıcı profil ekranını görürüm.

TTS002: Bir Moodle öğretmen yetkili kullanıcı olarak kullanıcı profili ekranında "Profil düzenle" tabına tıklamak istiyorum böylece profil düzenleme tabı ekranını görürüm.

TTS003: Bir Moodle öğretmen yetkili kullanıcı olarak profil ekranında "Forum mesajları" tabına tıklamak istiyorum böylece forum mesajları tabı ekranını görürüm.

TTS004: Bir Moodle öğretmen yetkili kullanıcı olarak profil ekranında "Blog" tabına tıklamak istiyorum böylece blog tabı ekranını görürüm

TTS005: Bir Moodle öğretmen yetkili kullanıcı olarak profil ekranında "Etkinlik Raporları" tabına tıklamak istiyorum böylece etkinlik raporları tabı ekranını görürüm.

Blog Menüsü Modülü Senaryoları

TTS006: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Yeni girdi ekle" alanına tıklamak istiyorum böylece yeni girdi ekle ekranını görürüm.

TTS007: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Girdilerime bak" alanına tıklamak istiyorum böylece girdilerime bak ekranını görürüm.

TTS008: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Site bağlantısına bak" alanına tıklamak istiyorum böylece site bağlantısının açılmadığını görürüm.

TTS009: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Blog seçenekleri" alanına tıklamak istiyorum böylece blog seçenekleri ekranını görürüm.

TTS010: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Site girdilerine bak" alanına tıklamak istiyorum böylece site girdilerine ekranını görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

Forumlarda Ara Modülü

TTS011: Bir Moodle öğretmen yetkili kullanıcı olarak arama alanına "Ders" yazıp Git butonuna tıklamak istiyorum böylece arama sonuçları ekranını görürüm.
TTS012: Bir Moodle öğretmen yetkili kullanıcı olarak "Gelişmiş arama" alanına tıklamak istiyorum böylece gelişmiş arama ekranını görürüm.
TTS013: Bir Moodle öğretmen yetkili kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "öğretim tasarımı" yazıp ve "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

Ana Menü Modülü Senaryoları

TTS014: Bir Moodle öğretmen yetkili kullanıcı olarak "Site haberleri" alanına tıklamak istiyorum böylece site haberleri sayfasını görürüm.
TTS015: Bir Moodle öğretmen yetkili kullanıcı olarak "Böte Sohbet Odası" alanına tıklamak istiyorum böylece böte sohbet odası sayfasını görürüm.
TTS016: Bir Moodle öğretmen yetkili kullanıcı olarak "Öğretim Tasarımı Formu" alanına tıklamak istiyorum böylece öğretim tasarımı form sayfasını görürüm.
TTS017: Bir Moodle öğretmen yetkili kullanıcı olarak "İnsan Bilgisayar Etkileşimi Odası" alanına tıklamak istiyorum böylece insan bilgisayar etkileşimi odası sayfasını görürüm.

Derslerim Modülü Senaryoları

TTS018: Bir Moodle öğretmen yetkili kullanıcı olarak "Instructional technology" dersine tıklamak istiyorum böylece instructional technology dersine giriş yapmış olurum.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

Haftalık Taslak Alt Modülü

TTS019: Bir Moodle öğretmen yetkili kullanıcı olarak yeni kaynak olarak "Ders etkinliğini" seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece ders türünde yeni etkinliğin eklendiğini görürüm.

TTS020: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili "Ders etkinliğine" tıklamak istiyorum böylece ders kaynağının açıldığını görürüm.

TTS021: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ders etkinliğinin "Güncelleme ikonuna" tıklayıp ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece ders kaynağın güncellendiğini görürüm.

TTS022: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ders etkinliğinin "Gizle ikonuna" tıklamak istiyorum böylece ders kaynağın gizlendiğini görürüm.

TTS023: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ders etkinliğinin sil "İkonuna" tıklayıp açılan ekranda evet butonuna tıklamak istiyorum böylece ders kaynağın silindiğini görürüm.

TTS024: Bir Moodle öğretmen yetkili kullanıcı olarak "Yeni kaynak olarak forum etkinliğini" seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece forum türünde yeni etkinliğin eklendiğini görürüm.

TTS025: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili forum etkinliğine tıklamak istiyorum böylece forum kaynağının açıldığını görürüm.

TTS026: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili forum etkinliğinin "Güncelleme ikonuna" tıklayıp ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece forum etkinliğinin güncellendiğini görürüm.

TTS027: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili forum etkinliğinin "Gizle ikonuna" tıklamak istiyorum böylece forum etkinliğinin gizlendiğini görürüm.

TTS028: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili forum etkinliğinin Sil ikonuna" tıklayıp açılan ekranda "Evet" butonuna tıklamak istiyorum böylece forum etkinliğinin silindiğini görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS029: Bir Moodle öğretmen yetkili kullanıcı olarak yeni kaynak olarak "Ödevler gelişmiş dosya yükleme etkinliğini" seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece ödevler gelişmiş dosya yükleme türünde yeni etkinliğin eklendiğini görürüm.

TTS030: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ödevler etkinliğine tıklamak istiyorum böylece ödevler kaynağının açıldığını görürüm.

TTS031: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ödevler gelişmiş dosya yükleme etkinliğinin "Güncelleme" ikonuna tıklayıp ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece ödevler gelişmiş dosya yükleme etkinliğinin güncellendiğini görürüm.

TTS032: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ödevler gelişmiş dosya yükleme etkinliğinin "Gizle İkonuna" tıklamak istiyorum böylece ödevler gelişmiş dosya yükleme etkinliğinin gizlendiğini görürüm.

TTS033: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili ödevler gelişmiş dosya yükleme etkinliğinin "sil ikonuna" tıklayıp açılan ekranda evet butonuna tıklamak istiyorum böylece ödevler gelişmiş dosya yükleme etkinliğinin silindiğini görürüm.

TTS034: Bir Moodle öğretmen yetkili kullanıcı olarak yeni kaynak türünü "Bir dosya siteye bağlantı" olarak seçip ilgili alanları doldurup dosyayı ekleyip "Kaydet ve göster" butonuna tıklamak istiyorum böylece dosya bağlantı türünde yeni kaynağın eklendiğini görürüm.

TTS035: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili dosya bağlantı kaynağına tıklamak istiyorum böylece dosya bağlantı kaynağının açıldığını görürüm.

TTS036: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili dosya bağlantı kaynağının "Güncelleme ikonuna" tıklayıp ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece dosya bağlantı kaynağının güncellendiğini görürüm.

TTS037: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili dosya bağlantı kaynağının "Gizle ikonuna" tıklamak istiyorum böylece dosya bağlantı kaynağının gizlendiğini görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS038: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili dosya bağlantı kaynağının "Sil ikonuna" tıklayıp açılan ekranda "Evet" butonuna tıklamak istiyorum böylece dosya bağlantı kaynağının silindiğini görürüm.

TTS039: Bir Moodle öğretmen yetkili kullanıcı olarak yeni kaynak türünü "Bir dosya siteye bağlantı olarak" seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece siteye bağlantı türünde yeni kaynağın eklendiğini görürüm.

TTS040: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili siteye bağlantı kaynağına tıklamak istiyorum böylece siteye bağlantı kaynağının açıldığını görürüm.

TTS041: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili siteye bağlantı kaynağının "Güncelleme ikonuna" tıklayıp ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece siteye bağlantı kaynağının güncellendiğini görürüm.

TTS042: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili siteye bağlantı kaynağının "Gizle ikonuna" tıklamak istiyorum böylece siteye bağlantı kaynağının gizlendiğini görürüm.

TTS043: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili siteye bağlantı kaynağının "Sil ikonuna" tıklayıp açılan ekranda evet butonuna tıklamak istiyorum böylece siteye bağlantı kaynağının silindiğini görürüm.

TTS044: Bir Moodle öğretmen yetkili kullanıcı olarak yeni kaynak türünü "Metin sayfası oluşturun" olarak seçip ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece siteye bağlantı türünde yeni kaynağın eklendiğini görürüm.

TTS045: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili metin sayfası kaynağına tıklamak istiyorum böylece metin sayfası kaynağının açıldığını görürüm.

TTS046: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili metin sayfası kaynağının "Güncelleme ikonuna" tıklayıp ilgili alanları doldurup "Kaydet ve göster" butonuna tıklamak istiyorum böylece metin sayfası kaynağının güncellendiğini görürüm.

TTS047: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili metin sayfası kaynağının "Gizle ikonuna" tıklamak istiyorum böylece kaynağının gizlendiğini görürüm.

TTS048: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili metin sayfası kaynağının "Sil ikonuna" tıklayıp açılan ekranda evet butonuna tıklamak istiyorum böylece metin sayfası kaynağının silindiğini görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

Topluluk Alt Modülü

TTS049: Bir Moodle öğretmen yetkili kullanıcı olarak "Katılımcılar" alanına tıklamak istiyorum böylece katılımcılar ekranını görürüm.

TTS050: Bir Moodle öğretmen yetkili kullanıcı olarak katılımcılar tabında da geçerli rolü "Öğretmen yetkili" olarak seçmek istiyorum böylece öğretmen yetkili rolündeki katılımcıları görürüm.

TTS051: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili katılımcı ya da katılımcıları seçip, seçili kullanıcılarla alanından ekle mesaj gönderi seçip, mesaj bilgisi girip, ön izleme butonuna tıklayıp, mesajı gönder butonuna tıklamak istiyorum böylece ilgili katılımcılara mesaj gönderirim.

TTS052: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili katılımcı ya da katılımcıları seçip, seçili kullanıcılarla alanından "Yeni not ekle" seçip, not bilgisi girip, "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece ilgili katılımcılara kurs not bilgisi eklerim.

TTS053: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili katılımcıyı seçip, seçili kullanıcılarla alanından "Yeni not ekle" seçip, not bilgisi girip, durum bilgisini kişisel seçip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece ilgili katılımcıya kişisel not bilgisi eklerim.

TTS054: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili katılımcı ya da katılımcıları seçip, seçili kullanıcılarla alanından "Yeni not ekle" seçip, not bilgisi girip, durum bilgisini site seçip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece ilgili katılımcılara site not bilgisi eklerim.

TTS055: Bir Moodle öğretmen yetkili kullanıcı olarak "Tümünü seç" butonuna tıklamak istiyorum böylece tüm katılımcıların seçildiğini görürüm.

TTS056: Bir Moodle öğretmen yetkili kullanıcı olarak "Tümünü temizle" butonuna tıklamak istiyorum böylece tüm katılımcılardaki işaretlerin kalktığını görürüm.

TTS057: Bir Moodle öğretmen yetkili kullanıcı olarak katılımcılar ekranının da "Bloglar" tabına tıklamak istiyorum böylece bloglar tabı ekranını görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS058: Bir Moodle öğretmen yetkili kullanıcı olarak katılımcılar ekranının da "Notlar" tabına tıklamak istiyorum böylece notlar tabı ekranını görürüm.

TTS059: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Yeni girdi ekle" alanına tıklamak istiyorum böylece yeni girdi ekle ekranını görürüm.

TTS060: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Girdilerime bak" alanına tıklamak istiyorum böylece girdilerime bak ekranını görürüm.

TTS061: Bir Moodle öğretmen yetkili kullanıcı olarak site notları başlığı altında "Hakkında not yazılacak kullanıcıları seçiniz" alanına tıklamak istiyorum böylece katılımcılar tabının açıldığını görürüm.

TTS062: Bir Moodle öğretmen yetkili kullanıcı olarak site notlarında ilgili notun "Düzenle" alanına tıklayıp, ilgili alanları düzenleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece düzenleme işleminin tamamlandığını görürüm.

TTS063: Bir Moodle öğretmen yetkili kullanıcı olarak site notlarında ilgili notun "Sil" alanına tıklayıp, gelen ekranda "Evet" butonuna tıklamak istiyorum böylece silme işleminin tamamlandığını görürüm.

TTS064: Bir Moodle öğretmen yetkili kullanıcı olarak kurs notları başlığı altında "Hakkında not yazılacak kullanıcıları seçiniz" alanına tıklamak istiyorum böylece katılımcılar tabının açıldığını görürüm.

TTS065: Bir Moodle öğretmen yetkili kullanıcı olarak kurs notlarında ilgili notun "Düzenle" alanına tıklayıp, ilgili alanları düzenleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece düzenleme işleminin tamamlandığını görürüm.

TTS066: Bir Moodle öğretmen yetkili kullanıcı olarak kurs notlarında ilgili notun "Sil" alanına tıklayıp, gelen ekranda "Evet" butonuna tıklamak istiyorum böylece silme işleminin tamamlandığını görürüm.

TTS067: Bir Moodle öğretmen yetkili kullanıcı olarak kişisel notlar başlığı altında "Hakkında not yazılacak kullanıcıları seçiniz" alanına tıklamak istiyorum böylece katılımcılar tabının açıldığını görürüm.

TTS068: Bir Moodle öğretmen yetkili kullanıcı olarak kişisel notlarda ilgili notun "Düzenle" alanına tıklayıp, ilgili alanları düzenleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece düzenleme işleminin tamamlandığını görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS069: Bir Moodle öğretmen yetkili kullanıcı olarak kişisel notlarda ilgili notun "Sil" alanına tıklayıp, gelen ekranda "Evet" butonuna tıklamak istiyorum böylece silme işleminin tamamlandığını görürüm.

Etkinlikler Alt Modülü

TTS070: Bir Moodle öğretmen yetkili kullanıcı olarak "Forumlar" alanına tıklamak istiyorum böylece forumlar ekranını görürüm.
TTS071: Bir Moodle öğretmen yetkili kullanıcı olarak "Kaynaklar" alanına tıklamak istiyorum böylece kaynaklar ekranını görürüm.
TTS072: Bir Moodle öğretmen yetkili kullanıcı olarak "Ödevler" alanına tıklamak istiyorum böylece ödevler ekranını görürüm.
TTS073: Bir Moodle öğretmen yetkili kullanıcı olarak ödevler ekranında "Ödev gönderisine bak" alanına tıklayıp, ilgili ödev dosyasına tıklamak istiyorum böylece ödev dosyasının indirildiğini görürüm.
TTS074: Bir Moodle öğretmen yetkili kullanıcı olarak gönderilen ödevler ekranında "Ödev güncelle" butonuna tıklayıp, ilgili güncellemeleri yapıp "Kaydet ve göster" butonuna tıklamak istiyorum böylece ilgili ödeve ait güncelleme işleminin yapıldığını görürüm.
TTS075: Bir Moodle öğretmen yetkili kullanıcı olarak gönderilen ödevler ekranında "Hızlı notlandırma izni" seçeneğini seçip seçenekleri "Kaydet" butonuna tıklamak istiyorum böylece hızlı notlandırma alanlarının aktif olduğunu görürüm.
TTS076: Bir Moodle öğretmen yetkili kullanıcı olarak gönderilen ödevler ekranında "Hızlı notlandırı" seçip, ilgili kullanıcıların not girişlerini yapıp "Tüm geri bildirimleri kaydet" butonuna tıklamak istiyorum böylece kullanıcıların not girişlerinin yapıldığını görürüm.
TTS077: Bir Moodle öğretmen yetkili kullanıcı olarak gönderilen ödevler ekranında ilgili ödev için yorum girişini yapıp "Tüm geri bildirimleri kaydet" butonuna tıklamak istiyorum böylece kullanıcı ödevlerine yorum eklendiğini görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS078: Bir Moodle öğretmen yetkili kullanıcı olarak gönderilen ödevler ekranında hızlı notlandırma izni seçeneğindeki işareti kaldırıp "Seçenekleri kaydet" butonuna tıklamak istiyorum böylece hızlı notlandırma alanlarının pasif olduğunu görürüm.

Forumlarda Ara Alt Modülü

TTS079: Bir Moodle öğretmen yetkili kullanıcı olarak arama alanına "Ders" yazıp Git butonuna tıklamak istiyorum böylece arama sonuçları ekranını görürüm.
TTS080: Bir Moodle öğretmen yetkili kullanıcı olarak "Gelişmiş arama" alanına tıklamak istiyorum böylece gelişmiş arama ekranını görürüm.
TTS081: Bir Moodle öğretmen yetkili kullanıcı olarak gelişmiş arama ekranında "Bu sözcükler mesajın herhangi bir yerinde olabilir" alanına "öğretim tasarımı" yazıp ve "Forumlarda ara" butonuna tıklamak istiyorum böylece arama sonucunu görürüm.

Yönetim Alt Modülü

TTS082: Bir Moodle öğretmen yetkili kullanıcı olarak "Düzenlemeyi aç" alanına tıklamak istiyorum böylece düzenleme araçlarını görürüm.
TTS083: Bir Moodle öğretmen yetkili kullanıcı olarak "Düzenlemeyi kapat" alanına tıklamak istiyorum böylece düzenleme araçlarının kapandığını görürüm.
TTS084: Bir Moodle öğretmen yetkili kullanıcı olarak "Ayarlar" alanına tıklayıp, ilgili alanları güncelleyip "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece ders düzenleme işleminin yapıldığını görürüm.
TTS085: Bir Moodle öğretmen yetkili kullanıcı olarak Rolleri ata" alanına tıklayıp gelen roller ekranında "Ders girmek için buraya tıklayın" alanına tıklamak istiyorum böylece ders ekranın açıldığını görürüm.
TTS086: Bir Moodle öğretmen yetkili kullanıcı olarak roller ekranında "Student" alanına tıklayıp, arama alanına test yazıp "Ara" butonuna tıklayıp, ilgili kullanıcıyı seçip "Ekle" butonuna tıklamak istiyorum böylece derse istenen kullanıcının eklendiğini görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS087: Bir Moodle öğretmen yetkili kullanıcı olarak roller ekranında "Student" alanına tıklayıp, ilgili kullanıcıyı seçip "Sil" butonuna tıklamak istiyorum böylece derse istenen kullanıcının silindiğini görürüm.

TTS088: Bir Moodle öğretmen yetkili kullanıcı olarak "Notlar" alanına tıklamak istiyorum böylece öğrenci raporu ekranı ve katılımcı listesini görürüm.

TTS089: Bir Moodle öğretmen yetkili kullanıcı olarak öğrenci raporu ekranında ilgili katılımcıya tıklamak istiyorum böylece o katılımcıya ait profil ekranını görürüm.

TTS090: Bir Moodle öğretmen yetkili kullanıcı olarak "Gruplar" alanına tıklayıp, açılan ekranda "Grup oluştur" butonuna tıklayıp, gelen ekranda ilgili alanları doldurup "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece grup oluşturma işleminin tamamlandığını görürüm.

TTS091: Bir Moodle öğretmen yetkili kullanıcı olarak gruplar ekranında grubu seçip kullanıcı ekle "Sil" butonuna tıklayıp, ilgili kullanıcıyı seçip "Ekle" butonuna tıklamak istiyorum böylece gruba üye ekleme işleminin tamamlandığını görürüm.

TTS092: Bir Moodle öğretmen yetkili kullanıcı olarak gruplar ekranında grubu seçip kullanıcı "Ekle sil" butonuna tıklayıp, ilgili kullanıcıyı seçip "Sil" butonuna tıklamak istiyorum böylece gruptan üye silme işleminin tamamlandığını görürüm.

TTS093: Bir Moodle öğretmen yetkili kullanıcı olarak gruplar ekranında grubu seçip seçili grubu "Sil" butonuna tıklayıp, açılan ekranda "Evet" butonuna tıklamak istiyorum böylece grup silme işleminin tamamlandığını görürüm.

TTS094: Bir Moodle öğretmen yetkili kullanıcı olarak "Yedekle" alına tıklayıp, açılan ekranlarda "Devam" butonlarına tıklamak istiyorum böylece yedek oluşturma işleminin tamamlandığını görürüm.

TTS095: Bir Moodle öğretmen yetkili kullanıcı olarak "Geri yükle" alına tıklayıp, ilgili dosya satırında ki "Geri yükle" alanına tıklayıp, açılan sayfalarda "Devam" butonlarına tıklamak istiyorum böylece yedek geri yükleme işleminin tamamlandığını görürüm.

TTS096: Bir Moodle öğretmen yetkili kullanıcı olarak "Geri yükle" alına tıklayıp, ilgili dosyayı seçip "Hepsini" butonuna tıklayıp, açılan ekranda "Evet" butonuna tıklamak istiyorum böylece yedek dosyasını silme işleminin tamamlandığını görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS097: Bir Moodle öğretmen yetkili kullanıcı olarak "Temizle" alanına tıklayıp, açılan ekranda ilgili alanları seçip dersi "Temizle" butonuna tıklayıp, gelen ekranda devam butonuna tıklamak istiyorum böylece ilgili temizleme işleminin tamamlandığını görürüm.

TTS098: Bir Moodle öğretmen yetkili kullanıcı olarak "Raporlar" alanına tıklayıp, gelen ekranda katılım raporunu seçip, gelen ekranda ilgili alanları doldurup "Git" butonuna tıklamak istiyorum böylece ilgili rapor sonucunu görürüm.

TTS099: Bir Moodle öğretmen yetkili kullanıcı olarak "Sorular" alanına tıklamak istiyorum böylece sorular ekranını görürüm.

TTS100: Bir Moodle öğretmen yetkili kullanıcı olarak "Dosyalar" alanına tıklamak istiyorum böylece dosyalar ekranını görürüm.

TTS101: Bir Moodle öğretmen yetkili kullanıcı olarak "Dersinden kaydımı sil" alanına tıklayıp gelen ekranda "Evet" butonuna tıklamak istiyorum böylece dosyalar ekranını görürüm.

TTS102: Bir Moodle öğretmen yetkili kullanıcı olarak "Profil" alanına tıklamak istiyorum böylece profil ekranını görürüm.

Derslerim Alt Modülü

TTS103: Bir Moodle öğretmen yetkili kullanıcı olarak "Bilgisayar Ağları ve İletişim" alanına tıklamak istiyorum böylece bilgisayar ağları ve iletişim dersi ekranını görürüm.

TTS104: Bir Moodle öğretmen yetkili kullanıcı olarak Tüm dersler" alanına tıklamak istiyorum böylece tüm dersler ekranını görürüm.

Son Haberler Alt Modülü

TTS105: Bir Moodle öğretmen yetkili kullanıcı olarak "Yeni konu ekle" alanına tıklayıp, ilgili alanları doldurup "Foruma gönder" butonuna tıklamak istiyorum böylece yeni konu eklemiş olurum.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS106: Bir Moodle öğretmen yetkili kullanıcı olarak konu başlığının yanındaki "Devamı…" alanına tıklamak istiyorum böylece konu içeriğini görürüm.

TTS107: Bir Moodle öğretmen yetkili kullanıcı olarak "Konu düzelt" alnına tıklayıp, Dosya seç" butonundan resim ilgili dosyayı seçmek istiyorum böylece doyanın eklendiğini görürüm.

TTS108: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili konuyu seçip, "Düzelt" alanına tıklayıp, ilgili düzenlemeleri yapıp "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece düzenleme işlemini tamamlarım.

TTS109: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili konuyu seçip, "Yanıtla" alanına tıklayıp, ilgili mesajı yazıp "Foruma gönder" butonuna tıklamak istiyorum böylece yanıtlama işlemini tamamlarım.

TTS110: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili konuyu seçip, "Sil" alanına tıklayıp, gelen ekranda "Evet" butonuna tıklamak istiyorum böylece silme işlemini tamamlarım.

Yaklaşan Olaylar Alt Modülü

TTS111: Bir Moodle öğretmen yetkili kullanıcı olarak "Yeni olay" alanına tıklayıp, kullanıcı olayını seçili ise "Tamam" butonunu tıklayıp, ilgili alanları doldurup "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni kullanıcı olayı yaratırım.

TTS112: Bir Moodle öğretmen yetkili kullanıcı olarak kullanıcı olayı seçip, olayı "Düzelt" ikonuna tıklayıp, ilgili düzenlemeden sonra "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece kullanıcı olayı düzenleme işlemini tamamlarım.

TTS113: Bir Moodle öğretmen yetkili kullanıcı olarak kullanıcı olayı seçip, olayı "Sil ikonuna" tıklayıp, "Sil" butonuna tıklamak istiyorum böylece kullanıcı olayı silme işlemini tamamlarım.

TTS114: Bir Moodle öğretmen yetkili kullanıcı olarak "Yeni olay" alanına tıklayıp, ders olayı seçip "Tamam" butonunu tıklayıp, ilgili alanları doldurup "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece yeni ders olayı yaratırım

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS115: Bir Moodle öğretmen yetkili kullanıcı olarak ders olayı seçip, olayı "Düzelt ikonuna" tıklayıp, ilgili düzenlemeden sonra "Değişiklikleri kaydet" butonuna tıklamak istiyorum böylece ders olayı düzenleme işlemini tamamlarım.
TTS116: Bir Moodle öğretmen yetkili kullanıcı olarak ilgili olayı seçip, olayı "Sil ikonuna" tıklayıp, "Sil" butonuna tıklamak istiyorum böylece ders olayı silme işlemini tamamlarım.

Son Etkinlikler Alt Modülü

TTS117: Bir Moodle öğretmen yetkili kullanıcı olarak son etkinliklerin "Tüm raporları" alanına tıklamak istiyorum böylece bütün katılımcılar ekranını görürüm.
TTS118: Bir Moodle öğretmen yetkili kullanıcı olarak bütün katılımcılar ekranında sırala kriterini "Tarih en yeniler-önce" seçip tarihi "Pasifleştir" alanını işaretleyip "Son etkinlikleri göster" butonuna tıklamak istiyorum böylece ilgili sonuç listesini görürüm.

Bloklar Alt Modülü

TTS119: Bir Moodle öğretmen yetkili kullanıcı olarak ekle alanından "Takvimi" seçmek istiyorum böylece takvim modülünü eklerim.
TTS120: Bir Moodle öğretmen yetkili kullanıcı olarak takvim modülünde "Gizle ikonuna" tıklamak istiyorum böylece takvim modülünü gizlerim.
TTS121: Bir Moodle öğretmen yetkili kullanıcı olarak takvim modülünde "Göster ikonuna" tıklamak istiyorum böylece takvim modülünün gizliliğini kaldırırım.
TTS122: Bir Moodle öğretmen yetkili kullanıcı olarak takvim modülünde "Sil ikonuna" tıklamak istiyorum böylece takvim modülünü silerim.
TTS123: Bir Moodle öğretmen yetkili kullanıcı olarak ekle alanından "Blog" seçmek istiyorum böylece blog modülünü eklerim.
TTS124: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Yeni girdi ekle" alanına tıklamak istiyorum böylece yeni girdi ekle ekranını görürüm.

EK 3 (Devam). Öğretmen Kullanıcı Rolü Test Senaryoları

TTS125: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Birdilerime bak" alanına tıklamak istiyorum böylece girdilerime bak ekranını görürüm.
TTS126: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Blog seçenekleri" alanına tıklamak istiyorum böylece blog seçenekleri ekranını görürüm.
TTS127: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Ders girdilerine" bak alanına tıklamak istiyorum böylece ders girdilerine ekranını görürüm.
TTS128: Bir Moodle öğretmen yetkili kullanıcı olarak blog menüsünde "Site girdilerine bak" alanına tıklamak istiyorum böylece site girdilerine ekranını görürüm.
TTS129: Bir Moodle öğretmen yetkili kullanıcı olarak blog modülünde "Gizle ikonuna" tıklamak istiyorum böylece blog modülünü gizlerim.
TTS130: Bir Moodle öğretmen yetkili kullanıcı olarak blog modülünde "Göster ikonuna" tıklamak istiyorum böylece blog modülünün gizliliğini kaldırırım.
TTS131: Bir Moodle öğretmen yetkili kullanıcı olarak blog modülünde "Sil ikonuna" tıklamak istiyorum böylece blog modülünü silerim.

Tüm dersler

TTS132: Bir Moodle öğretmen yetkili kullanıcı olarak "Tüm dersler" butonuna tıklamak istiyorum böylece tüm dersler ekranını görürüm.

Dersleri ara

TTS133: Bir Moodle öğretmen yetkili kullanıcı olarak "Dersleri ara" alanına "İnsan Bilgisayar etkileşimi" yazıp "Git" butonuna tıklamak istiyorum böylece arama sonuçları ekranını görürüm.