İSTANBUL TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM FAKÜLTESİ

BEHAVIOR DRIVEN DEVELOPMENT İLE FARAZİ BİR HESAP UYGULAMASI İÇİN TEST KÜMESİ

Bitirme Projesi

Program: Bilgi Teknolojileri II.Öğretim Tezsiz Yüksek Lisans
Danışman: Prof. Dr. Tolga Ovatman

Ağustos 2024

Doğruluk Beyanı

Yapmış olduğum bitirme projesi çalışmamda, kullanmış olduğum dış kaynaklı dokümanların referanslarının, raporumda açıkça ve eksiksiz bir biçimde verildiğini ve geriye kalan tüm bölümlerde, özellikle çalışmanın temelini oluşturan teorik kısımların ve deneylerin şahsım tarafından yapıldığını bilgilerinize sunarım.

İstanbul, 2024

İçerik

BEHAVIOR DRIVEN DEVELOPMENT İLE FARAZİ BİR HESAP UYGULAMASI İÇİN TEST KÜMESİ

(ÖZET)

Davranış Odaklı Geliştirme (BDD), yazılım geliştirme süreci boyunca teknik ve teknik olmayan ekip üyeleri arasındaki iş birliğini kolaylaştıran bir yöntemdir. BDD, Test Odaklı Geliştirme (TDD) ilkelerine dayanarak geliştirilmiştir ve kullanıcı davranışının analizine yöneliktir. BDD testleri, işletmenin gereksinimleriyle uyumlu kullanıcı hikayeleri veya senaryolar biçiminde oluşturulur ve böylece kavrama ve değerlendirme kolaylaştırılır.

BDD, kullanıcı hikayeleri ve senaryolar kullanarak sistemin davranışını tanımlar ve böylece TDD üzerine inşa edilerek daha yüksek bir seviyede çalışır. Ayrıca, BDD, yazılımın belirli bir iş alanını ele alacak şekilde tasarlanmasını garantilemek için Alan Odaklı Tasarım (DDD) ilkelerinden yararlanır ve böylece iş birimleri ve geliştiriciler arasındaki gelişmiş iletişimi kolaylaştırır.

BDD'nin en önemli avantajlarından biri, test vakalarının Gherkin gibi kolayca anlaşılabilir bir dilde oluşturulmasıdır. Bu, testlerin hem analitik belgeler hem de kullanıcı hikayeleri olarak kullanılmasını sağlar. Ayrıca, sık güncellemelerin standart bir uygulama olduğu yazılım geliştirme bağlamında, BDD test vakalarının güncel kalmasını sağlar ve iş birimlerinin sürece katılımını kolaylaştırır.

Sonuç olarak, BDD teknik personelin test vakaları hazırlaması gerekliliğini ortadan kaldırır. Gherkin sözdizimiyle oluşturulan senaryolar hem kullanıcı hikayeleri hem de test senaryoları olarak kullanılabilir. Bu, yazılım geliştirme sürecinin her aşamasında iş birimlerinin katılımını garanti eder ve güncel belgelerin sürdürülmesini kolaylaştırır. BDD, Çevik yazılım süreçlerinde önemli bir rol oynar ve tüm departmanların çeviklik ve yanıt verebilirlik elde etmesi için vazgeçilmez bir yaklaşımdır.

TEST SET FOR A HYPOTHETICAL CALCULATION APPLICATION WITH BEHAVIOR DRIVEN DEVELOPMENT

(ABSTRACT)

Behaviour-Driven Development (BDD) is a method that facilitates collaboration between technical and non-technical team members throughout the software development process. BDD was developed based on the principles of Test-Driven Development (TDD) and is oriented towards the analysis of user behaviour. BDD tests are constructed in the format of user stories or scenarios that align with the requirements of the business, thereby facilitating comprehension and assessment.

BDD defines the behaviour of the system through the use of user stories and scenarios, thereby operating at a higher level by building on TDD. Furthermore, BDD draws upon the principles of Domain-Driven Design (DDD) to guarantee that the software is designed to address a particular business domain, thereby facilitating enhanced communication between business units and developers.

One of the most significant advantages of BDD is that test cases are constructed in a readily comprehensible language, such as Gherkin. This enables the utilisation of tests as both analytical documents and user stories. Furthermore, in the context of software development, where frequent updates are a standard practice, BDD ensures that test cases remain current and facilitates the involvement of business units in the process.

Consequently, BDD obviates the necessity for technical personnel to prepare test cases. Scenarios created with Gherkin syntax can be utilised as both user stories and test scenarios. This guarantees the involvement of business units at each stage of the software development process and facilitates the maintenance of up-to-date documentation. BDD plays a pivotal role in Agile software processes and is an indispensable approach for all departments to achieve agility and responsiveness.

1. Giriş

Davranış Odaklı Geliştirme (BDD), bir yazılım ürünü veya projesi üzerinde çalışan hem teknik hem de teknik olmayan ekip üyeleri arasındaki iş birliğini içeren bir yazılım geliştirme yöntemidir [1]. Davranış Odaklı Geliştirme (BDD), Test Odaklı Geliştirmeye (TDD) dayalı bir metodolojidir, ancak özel odağı kullanıcı davranışıdır ve bu nedenle kullanıcı davranışı ön plandadır. BDD testleri, işletmenin belirli gereksinimleri ve beklentileriyle uyumlu kullanıcı hikayeleri veya senaryoları biçiminde yazılır. Test senaryolarının bu şekilde yazılması BDD'nin en büyük farklılıklarından biridir.

BDD yaklaşımı, test sürecinin daha da iyileştirilmesini sağlar ve TDD'ye benzer şekilde, kodlama aşamasından önce test vakalarının oluşturulmasına olanak tanır. BDD, Test Odaklı Geliştirmede (TDD) kullanılan yaklaşımlara dayanmaktadır [2]; ancak, TDD'den daha yüksek bir seviyede çalışır ve odağı testten bir sistemin beklenen davranışlarının tanımlanmasına kaydırdığı için TDD'nin bir iyileştirmesi olarak düşünülebilir. Ayrıca, test senaryoları biçiminde ifade edilen müşteri gereksinimlerini ve senaryolarını yönetmek için değerli bir yaklaşım olarak kabul edilir [3].

Test Odaklı Geliştirme (TDD), testlerin oluşturulmasıyla başlayan ve ardından bu testleri geçecek kodun geliştirilmesine doğru ilerleyen bir yazılım geliştirme sürecidir. TDD'nin amacı, yazılımın işlevselliğini daha küçük birimlere bölerek kademeli olarak geliştirmektir. TDD'nin temel öncülü, önce testleri geliştirerek, kodun amaçlanan işlevselliği hakkında daha kapsamlı bir anlayış kazanılabileceğidir. Testler, kodun test edilebilir, sürdürülebilir ve genişletilebilir olmasını sağlayarak kod için bir taslak görevi görür. TDD'de yer alan temel aşamalar şunlardır:

· Başarısız bir test yazılır

· Testi geçmek için kod yazılır

· Kalitesini iyileştirmek için kod yeniden düzenlenir

BDD yaklaşımı TDD ve DDD yaklaşımlarından daha sonra tanıtılmıştır. BDD yaklaşımının tanıtılmasında en büyük amaç TDD yaklaşımındaki sorun olarak görülen konuların ortadan kaldırılmak istenmesidir bu yapılırken Alan Odaklı Tasarım (DDD) yaklaşımından faydalanılan noktalar da olmuştur. Yazılım geliştirmeye yönelik DDD yaklaşımı, söz konusu yazılımın etki alanına odaklanır. Etki alanı, yazılımın çözmesi amaçlanan sorun alanı olarak tanımlanabilir. DDD, yazılımın etki alanı etrafında inşa edilmesini ve sorunu en iyi şekilde ele almasını garanti eder. Eric Evans Alan Odaklı Tasarım (DDD) yaklaşımının nasıl çalıştığını öğrenmek ve anlamak için bazı kavramlardan bahseder [4]:

- Bağlam (Context): Bağlam, bir kelimenin veya ifadenin anlamını belirleyen durum ve çevredir. Örneğin, aynı kelime farklı bağlamlarda farklı anlamlar taşıyabilir.

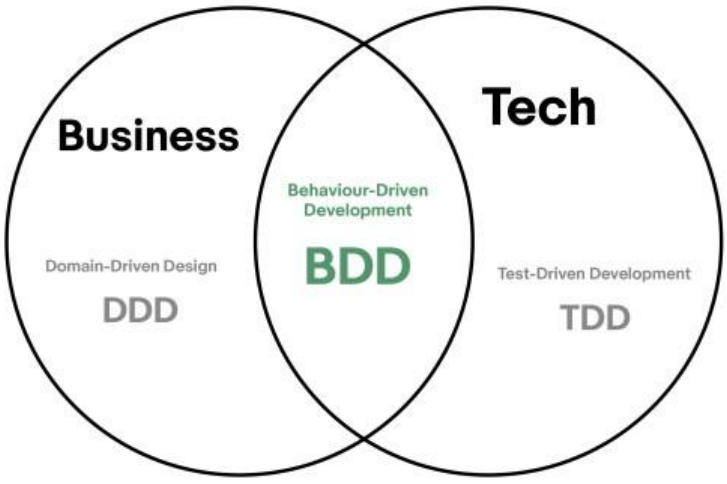
- Alan (Domain): Domain, bilgi, etki veya faaliyet alanını ifade eder. Bir iş alanında veya projede geçerli olan konu ve kapsam alanıdır. Her projenin ve işin bir alanı vardır.

- Model (Model): Model, bir alanın seçilmiş yönlerini soyut bir şekilde tanımlayan ve o alandaki sorunları çözmek için kullanılabilen sistematik bir yapıdır. Bu yapı, iş süreçlerini, nesneleri ve ilişkileri kapsar.

- Özellik	TDD	BDD	DDD
Odak Noktası	Kodun test edilmesi	Kullanıcı davranışları	Alan bilgisi
Katılımcılar	Geliştiriciler	Geliştiriciler ve iş birimi	Geliştiriciler ve alan uzmanları
Dil	Programlama dilleri	İş birimi tarafından anlaşılır dil (Gherkin)	Alan uzmanları tarafından anlaşılır dil
Araçlar	JUnit, NUnit, pytest vb.	Cucumber, SpecFlow, Behave	Alan modelleme araçları

Tablo 1- TDD, BDD ve DDD

Mesela DDD'de kullanılan Her Yerde Geçerli Dil (Ubiquitous Language) yaklaşımı BDD'nin genel mantığı ile çok uyuşmaktadır. Alanın asıl hakimi olan iş birimlerini okunabilirliği yüksek test dokümantasyonu ile sürece dahil ederek ve buna ek olarak testlerin geliştirmelerden önce yazılması ile testlerin işi tarifleyen bir analiz dokümanı gibi kullanılmasını sağlayarak iş ekibi ve yazılım ekibi arasındaki iletişimi üst seviyelere taşımıştır.



Şekil 1- Davranış Odaklı Geliştirme (BDD) – Alan Odaklı Tasarım (DDD) – Test Odaklı Geliştirme (TDD) Kesişimi

2. Davranış Odaklı Geliştirme (BDD)

2.1. Davranış Odaklı Geliştirme (BDD) nedir?

Gereksinimler üzerinden türetilen test senaryoları, çevik ekip ile son kullanıcı arasındaki sözlü etkileşimler için çok önemli bir rol oynar. BDD'de, tanımlanmış bir desende düzenlenmiş düz dil ifadeleri biçiminde yazılmış bir uygulama veya yazılım sistemiyle etkileşim kurmak için bir dizi girdi sağlanır "Verildi-Ne Zaman-O Zaman" [5], ayrıntıları şu şekilde tasvir eder:

a. Verildi (Given) - ilk bağlamı gösterir.

b. Ne Zaman(When) - bir olayın meydana gelmesini sunar.

c. O zaman (Then) - beklendiği gibi bir sonucun vaadini gösterir.

Aşağıda kullanıcının giriş yapması davranışına ilişkin Gherkin sentaksı ile hazırlanmış bir test dokümanına yer verilmiştir. Given – When – Then adımları And ile birden çok koşula sahip olabilir.

```
1  Feature: User login
2    Scenario: Successful login with correct credentials
3      Given I am on the login page
4      When I enter valid credentials
5      Then I should be redirected to the dashboard
```

Şekil 2 - Given - When - Then Kullanıcı Giriş Yap Örneği

BDD betiklemesi, test otomasyonunu kolaylaştıran bir desen kullanır. Gereksinimleri açıklığa kavuşturmanın veya belirsizlikleri azaltmanın geleneksel yöntemleri genellikle proje paydaşları ile teknik ekip arasında yanlış iletişime neden olarak paydaşlarla doğrudan iletişimi zorlaştırır [4]. BDD, tüm ekip üyeleri ile müşteri arasında bir anlatı işbirliğini dahil ederek bu sorunu ele alır. BDD'nin üç hedefi olaylar, sonuçlar ve bağlamdır. Olaylar, beklenen sonuç olan nihai sonuca ulaşmak için gerçekleşen eylemleri ifade eder. Başka bir deyişle, bir olay bir kullanıcının gerçekleştirdiği bir eylemdir ve bir sonuç, eylemden elde edilecek beklenen sonuçtur. BDD'nin birincil hedefi, sistemin kullanıcıdan bekleyebileceği davranış kümesini belirlemektir. Bu sayede tüketicilerin kolayca anlayabileceği bir tanımlama ortaya çıkar [7].

2.2. Davranış Odaklı Geliştirme'nin tarihçesi

Davranış odaklı geliştirme kavramı ilk olarak 2000'lerin başında Daniel Terhorst-North tarafından tanıtıldı. Daniel North bu kavramı ilk defa, 2006'da 'BDD'nin Tanıtılması' başlıklı bir makalede ortaya koydu. Test Odaklı Geliştirmeye (TDD) bir yanıt olarak ortaya çıktı ve yeni çevik takımlardaki programcılara test ve kodlamanın "doğrudan iyi kısmına" geçmeleri için bir araç sunarak yanlış anlaşılma olasılığını azalttı. BDD, kabul seviyesinde hem analizi hem de otomatik testi kapsayan bir metodolojiye dönüştü. BDD alanında bir diğer önemli isim de 2004'te konu hakkında kapsamlı bir şekilde yazmaya ve konuşmaya başlayan Liz Keogh'dur.

2003'te Daniel Terhorst-North, "test" kavramı yerine "davranış" kavramına dayalı bir kelime dağarcığı kullanan JBehave adlı JUnit'in yerini alacak bir şey üzerinde çalışmaya başladı. Ayrıca, Liz Keogh ve Chris Matts erken bir aşamada önemli katkılarda bulundu. "Given/When/Then" şablonu, bir hikayenin kabul kriterlerini yürütülebilir bir biçimde yakalamak için geliştirildi ve Eric Evans'ın Alan Odaklı Tasarımında (DDD) tanıtılan yaygın dil fikrinden etkilendi. İş değerine odaklanması, şablonun belirli bir iş bağlamının özel gereksinimlerini yakalamadaki faydasını yansıtır. BDD kavramı, Rachel Davies tarafından Connextra'da geliştirilen ve tanınan bir standart haline gelen kullanıcı hikayeleri yazmak için "As a..., I ..., So that..." şablonundan etkilendi.

2005 yılında, Ruby dilinde BDD'yi destekleyen RSpec projesi Dave Astels, Steven Baker, Aslak Hellesøy ve David Chelimsky tarafından kuruldu.

BDD'nin kısa tarihçesi:

- **1999** TDD, Kent Beck tarafından "Extreme Programming" (XP) pratiği olarak popüler hale getirildi.
- **2003** Eric Evans, "Domain-Driven Design: Tackling Complexity in the Heart of Software" kitabını yayımladı.
- **2004** Dan North, TDD'nin bazı zorluklarını ve yanlış anlaşılmalarını gözlemledi ve bu durumu iyileştirmek için bir yöntem geliştirdi.
- **2006** Dan North, BDD'yi tanıtan ve açıklayan bir makale yayımladı.

2.3. Gherkin Sentaksı

BDD'nin en büyük farklılıklarından biri Gherkin sentaksıdır. Amacı tüm paydaşların teknik bilgiye ihtiyaç duymadan davranışı anlayabilmesini sağlamaktır.

Projelerimizde her bir davranış için .feature uzantılı bir Gherkin dosyası oluşturulur. Bu feature dosyasına ilgili özelliğin farklı durumlardaki davranışları tanımlanır.

Gherkin sentaksında;

- Given anahtar kelimesi ile ön koşul yani başlangıç durumu tanımlanır,
- When anahtar kelimesi ile olay,
- Then anahtar kelimesi ile de sonuç tanımlanır.

Given-When-Then adımları ile senaryo oluşturulur.

- And / But : Given, When, Then anahtar kelimelerinden birden fazla kullanmak istediğimizde bu anahtar kelimelerden faydalanıyoruz. And olumlu durumlarda, But olumsuz durumlarda kullanılmaktadır.

Gherkin'in en büyük gücü okunabilirliği ve yaşayan bir belge olarak hizmet edebilmesidir.

2.4. Davranış Odaklı Geliştirme Araçları

Davranış odaklı geliştirme (BDD), yazılım geliştirme bağlamında teknik ekipler ve iş paydaşları arasındaki iletişimi kolaylaştırmaya yarayan temel bir metodolojiyi temsil eder. BDD araçları,

çeşitli programlama dilleri genelinde uygulama özelliklerini tanımlama ve test etmede son derece önemlidir.

Bu araçlar etkili iletişimi garanti eder ve yazılım işlevlerinin söz konusu işletmenin özel ihtiyaçlarıyla uyumlu olmasını sağlar. BDD araçları, yazılım testinin manzarasını dönüştürüyor, ekip üyeleri arasında iş birliğini teşvik ediyor ve testlerin son derece verimli bir şekilde otomasyonunu sağlıyor.

Farklı yazılım dilleri ve çerçeveleri için bir çok BDD aracı mevcuttur. Bunlardan çok bilinen birkaç tanesi şunlardır:

2.4.1. Behave

Behave çerçevesi Python programlama diliyle kullanılmak üzere tasarlanmıştır.

- Behave, "features" olarak adlandırılan bir dizinde saklanan üç farklı dosya türüyle çalışmak üzere tasarlanmıştır. Özellik dosyaları, gerçekleştirilecek senaryoları içerir.

- "Steps" olarak etiketlenen dizin, belirtilen senaryolar için Python adım uygulamalarını içerir.

- Ek olarak, adımlardan, senaryolardan, özelliklerden veya tüm test paketinden önce ve sonra yürütülen betikler olan çevresel kontrolleri uygulama seçeneği de mevcuttur.

- Behave özellikleri Gherkin kullanılarak (belirli değişikliklerle) yazılır ve "name.feature" olarak adlandırılır.

- Bir özellik ve senaryoya eklenen etiketlere, kendilerine iletilen "feature" veya "scenario" nesnesi aracılığıyla ortam işlevleri içinde erişilebilir. Bu nesneler, özellikler dosyasında bulundukları sırayla eklenen etiket adlarının bir listesi olan "tags" adlı bir özniteliğe sahiptir.

2.4.2. JBehave

JBehave, BDD test senaryoları yazmak için kullanılan bir diğer açık kaynaklı araçtır. Gherkin dili burada da kullanıcı hikayelerini tanımlamak için kullanılır. JBehave'de kullanıcı hikayeleri esasen test senaryolarıdır. Hikayedeki bu adımların her biri kodlanmıştır - böylece beklenen eylem kümesi hikaye yürütme sırasında gerçekleştirilir. JBheave, tamamı itibarıyla bir Java çerçevesidir.

2.4.3. Pytest

Pytest BDD, Python programlama dilinde testlerin oluşturulması için py.test test çerçevesini kullanan bir davranış odaklı geliştirme (BDD) çerçevesidir. py.test çerçevesinin basitliğini ve gücünü, BDD'de kullanılan Gherkin dilinin etkileyici sözdizimiyle birleştirir.
pytest-bdd, Gherkin sözdizimini kullanarak davranış odaklı geliştirme (BDD) tarzı testlerin oluşturulmasını sağlayan bir py.test eklentisidir.

BDD Aracı	Yazılım Dilleri	Desteklediği Versiyon

Behave	Python	2.7.14
JBehave	Java	1.8
Pytest	Python	3.5

Tablo 2 - BDD Araçları

3. Farazi Bir Hesap Makinası için Davranış Odaklı Geliştirme

Python dilinde yazılmış 4 işlem yeteneğine sahip farazi bir hesap makinası projesi Davranış Odaklı Geliştirme metodu uygulanmıştır. Bu uygulama sırasında öncelikle Gherkin sentaksında test senaryoları oluşturulmuş sonra bu test senaryolarını koşacak test dosyası Python dilinde hazırlanmıştır son olarak ise test senaryolarını başarılı hale getirecek hesap makinası projesi Python dilinde yazılmıştır.

3.1. Gherkin Sentaksında .features dosyasının hazırlanması

Bütün test senaryolarının *Given, When* ve *Then* metoduyla test senaryolarını içerecek calculator.feature dosyası hazırlanmıştır.

Yazılacak Hesap makinasında gerçekleştirilecek işlemler için "Calculator Operations" isimli *Feature* tanımlanmıştır. Bu *Feature* içerisinde;

- Kullanıcıdan *Given* adımında bir sayı girmesi istenmiştir. İkinci sayının girilmesi için *Given* adımı *And* kullanılarak çoklanmıştır.
- *When* adımında kullanıcının seçtiği işlem tipi (toplama, çıkarma, çarpma veya bölme) alınmıştır.
- *Then* adımında ise girilen sayılar ve seçilen işlem tipine göre yapılan hesabın daha önceden belirtilmiş sonuca eşit olduğu kontrolü yapılmıştır.

features > calculator.feature

```
Feature: Calculator Operations
  As a user
  I want to perform basic arithmetic operations
  So that I can do calculations quickly

  Scenario Outline: Perform arithmetic operations
    Given I have entered <first_number> as the first number
    And I have entered <second_number> as the second number
    When I choose to perform "<operation>"
    Then the result should be <result>

    Examples:
      | first_number | second_number | operation | result |
      | 5            | 3             | add       | 8      |
      | 28           | 17            | add       | 45     |
      | 10           | 4             | subtract  | 6      |
      | 56           | 11            | subtract  | 45     |
      | 6            | 7             | multiply  | 42     |
      | 8            | 9             | multiply  | 72     |
      | 15           | 3             | divide    | 5      |
      | 156          | 3             | divide    | 52     |

  Scenario: Division by zero
    Given I have entered 10 as the first number
    And I have entered 0 as the second number
    When I choose to perform "divide"
    Then I should see an error message "Division by zero is not allowed"
```

Şekil 3 - Gherkin Sentaksında hazırlanmış test senaryosu

Examples tablosu ile *Given, When* ve *Then* adımlarında ihtiyaç duyulacak parametreler tanımlanmıştır. Burada yapılan toplam sekiz örnek tanımlamasının her biri farklı bir teste denk gelecektir.

İkinci bir Feature olarak sıfırla bölünme durumunda verilecek hata için tanımlama yapılmıştır.

3.2. Test Adımları dosyasının hazırlanması

Test senaryolarının koşulması için *calculator_steps.py* dosyası hazırlanırken; *calculator.py* dosyasından işlem tipi fonksiyonları, *behave* üzerinden *given,when,then* fonksiyonları ve *pytest* üzerinden *approx* fonksiyonu import edilmiştir.

calculator_steps.py dosyası *calculator.feature* dosyasındaki test tanımlarını çalıştırmak için dizayn edilmiştir.

Dosya terminale *behave* komutu yazılarak çalıştırılır.

features > steps > calculator_steps.py > ...

```python
from behave import given, when, then
from calculator import add, subtract, multiply, divide
from pytest import approx

@given('I have entered {number:d} as the first number')
def step_impl(context, number):
    context.first_number = number

@given('I have entered {number:d} as the second number')
def step_impl(context, number):
    context.second_number = number

@when('I choose to perform "{operation}"')
def step_impl(context, operation):
    context.operation = operation
    if operation == "add":
        context.result = add(context.first_number, context.second_number)
    elif operation == "subtract":
        context.result = subtract(context.first_number, context.second_number)
    elif operation == "multiply":
        context.result = multiply(context.first_number, context.second_number)
    elif operation == "divide":
        try:
            context.result = divide(context.first_number, context.second_number)
        except ValueError as e:
            context.error_message = str(e)

@then('the result should be {result:d}')
def step_impl(context, result):
    assert context.result == approx(result)

@then('I should see an error message "{error_message}"')
def step_impl(context, error_message):
    assert context.error_message == error_message
```

Şekil 4 - Test adımları dosyası

3.3. Proje Dosyasının Hazırlanması (calculator.py)

Proje dosyası tanımlanan tüm test adımlarını başarılı hale getirecek şekilde Python dilinde hazırlanmıştır.

```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        raise ValueError("Division by zero is not allowed")
    return a / b

def calculator():
    while True:
        print("\nCalculator")
        print("1. Add")
        print("2. Subtract")
        print("3. Multiply")
        print("4. Divide")
        print("5. Exit")

        choice = input("Enter your choice (1-5): ")

        if choice == '5':
            print("Thank you for using the calculator. Goodbye!")
            break

        if choice in ('1', '2', '3', '4'):
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))

            if choice == '1':
                print("Result:", add(num1, num2))
            elif choice == '2':
                print("Result:", subtract(num1, num2))
            elif choice == '3':
                print("Result:", multiply(num1, num2))
            elif choice == '4':
                print("Result:", divide(num1, num2))
        else:
            print("Invalid input. Please try again.")

if __name__ == "__main__":
    calculator()
```

Şekil 5 - Proje dosyası (calculator.py)

3.4. Test Sonuçları

Yazılan testlerin tamamı (9 senaryo) başarılı olmuştur. Senaryoların her birindeki 4 adım (given,when,then ve and) toplam 36 adım ayrı ayrı başarılı olmuştur. Testlerin tamamlanması 1ms'nin altında sürmüştür.

```
bekirasil@Bekirs-MBP-2 calculator_v3 % behave
Feature: Calculator Operations # features/calculator.feature:1
  As a user
  I want to perform basic arithmetic operations
  So that I can do calculations quickly
  Scenario Outline: Perform arithmetic operations -- @1.1   # features/calculator.feature:14
    Given I have entered 5 as the first number              # features/steps/calculator_steps.py:5 0.000s
    And I have entered 3 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "add"                          # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 8                             # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.2   # features/calculator.feature:15
    Given I have entered 28 as the first number             # features/steps/calculator_steps.py:5 0.000s
    And I have entered 17 as the second number              # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "add"                          # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 45                            # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.3   # features/calculator.feature:16
    Given I have entered 10 as the first number             # features/steps/calculator_steps.py:5 0.000s
    And I have entered 4 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "subtract"                     # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 6                             # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.4   # features/calculator.feature:17
    Given I have entered 56 as the first number             # features/steps/calculator_steps.py:5 0.000s
    And I have entered 11 as the second number              # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "subtract"                     # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 45                            # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.5   # features/calculator.feature:18
    Given I have entered 6 as the first number              # features/steps/calculator_steps.py:5 0.000s
    And I have entered 7 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "multiply"                     # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 42                            # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.6   # features/calculator.feature:19
    Given I have entered 8 as the first number              # features/steps/calculator_steps.py:5 0.000s
    And I have entered 9 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "multiply"                     # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 72                            # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.7   # features/calculator.feature:20
    Given I have entered 15 as the first number             # features/steps/calculator_steps.py:5 0.000s
    And I have entered 3 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "divide"                       # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 5                             # features/steps/calculator_steps.py:28 0.000s

  Scenario Outline: Perform arithmetic operations -- @1.8   # features/calculator.feature:21
    Given I have entered 156 as the first number            # features/steps/calculator_steps.py:5 0.000s
    And I have entered 3 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "divide"                       # features/steps/calculator_steps.py:13 0.000s
    Then the result should be 52                            # features/steps/calculator_steps.py:28 0.000s

  Scenario: Division by zero                                # features/calculator.feature:23
    Given I have entered 10 as the first number             # features/steps/calculator_steps.py:5 0.000s
    And I have entered 0 as the second number               # features/steps/calculator_steps.py:9 0.000s
    When I choose to perform "divide"                       # features/steps/calculator_steps.py:13 0.000s
    Then I should see an error message "Division by zero is not allowed" # features/steps/calculator_steps.py:32 0.000s

1 feature passed, 0 failed, 0 skipped
9 scenarios passed, 0 failed, 0 skipped
36 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.001s
```

Şekil 6 - Test Sonuçları

4. Sonuç

Sonuç olarak BDD yaklaşımı ile test senaryolarının TDD'deki teknik insanlar tarafından hazırlanması zorunluluğu ortadan kaldırılmıştır. Gherkin sentaksı ile hazırlanan test senaryoları hem kullanıcı hikayesi, analiz dosyası olarak kullanılabilen işi ve yazılımdan beklenen davranışı anlatan dokümanlar olmuş hem de test senaryoları olarak kullanılabilmiştir. Bu durum en

başından itibaren bir yazılımın başarılı olup olmaması sürecinde analiz ve test senaryolarının hazırlanması süreçlerine iş birimlerinin (müşterinin) katılabilmesini sağlamıştır. Öte yandan sürekli geliştirmelerin ve değişikliklerin olduğu yazılım dünyasında dokümanların güncelliğini yitirmemesi işten bile değilken BDD ile her yeni geliştirme veya güncellemede test senaryolarının yeniden yazılması gerekliliği canlılığını yitirmeyen bir dokümantasyon oluşmasını sağlamaktadır.

Kullanıcı Hikayesi (User Story) formatları hem analizleri sadeleştirmekte, hem de kullanıcı ve değer odaklı düşünme sürecini desteklemektedir. Peki tüm bu senaryoları oluşturmadan önce test etmenin mümkün kılındığı bir IT dünyası oluşturulabilir mi? Planlama ve tasarım aşamasının ardından kodlama sürecine geçilmeden tüm senaryoların testini yazabiliyor olmak bize bu noktada ne sağlar? En doğru ve en net yanıt: ‘’Business Agility’’. Bir ürünün/servisin karar verilmiş ilk farklılaştırmaya hızla adapte olabilme yeteneği olarak tanımlayabiliriz bu kavramı.

Nasıl ki organizasyonlarda IT içerisindeki takımların agile olmasının yetmediği, iş kollarının hem vizyon hem de talepleri olgunlaştırma süreçlerinde de agile’ı desteklemeleri gerekliliği gibi; BDD yaklaşımı ile uçtan uca agile döngüsünün tamamlanabileceği gerçeğini göz ardı etmemek gerekir. “Agile metodolojisi ile iş yapıyorsanız ve uygulama testi için BDD kullanmıyorsanız kendinizle çelişiyorsunuzdur.” (Eric N. SHAPIRO, Co-founder of ArcTouch) sözü bu durumda kendini ispatlamış olmakta. Business Agility kavramını tam olarak anlayabilmek/uygulayabilmek için şirketler içlerindeki her halkanın (pazarlama, satış, ürün yönetimi & geliştirme, süreç iyileştirme, test) aynı çeviklikte ve aynı duyarlılıkta olabilmesi için yazılım süreçlerinde BDD yaklaşımının gerekliliği kaçınılmazdır [8].

5. Referanslar

1. L. P. Binamungu, S. M. Embury and N. Konstantinou, "Characterising the quality of behaviour driven development specifications", *Proc. Agile Processes Softw. Eng. Extreme Program. 21st Int. Conf. Agile Softw. Develop. XP*, pp. 87-102, Jun. 2020.

2. M. Diepenbeck, U. Kühne, M. Soeken and R. Drechsler, "Behaviour driven development for tests and verification", *Proc. 8th Int. Conf. Tests Proofs (TAP)*, pp. 61-77, Jul. 2014.

3. M. S. Farooq, U. Omer, A. Ramzan, M. A. Rasheed and Z. Atal, "Behavior Driven Development: A Systematic Literature Review," in *IEEE Access*, vol. 11, pp. 88008-88024, 2023, doi: 10.1109/ACCESS.2023.3302356.

4. Burak Tabakoğlu, "Domain-Driven Design(DDD) Nedir?", Jul 2023 https://medium.com/goturkiye/domain-driven-design-ddd-nedir-750dc6c9641b

5. A. Mishra and A. Mishra, "Introduction to behavior-driven development" in IOS Code Testing: Test-Driven Development and Behavior-Driven Development With Swift, New York, NY, USA:Apress, pp. 317-327, 2017.

6. A. Aitken and V. Ilango, "A comparative analysis of traditional software engineering and agile software development", *Proc. 46th Hawaii Int. Conf. Syst. Sci.*, pp. 4751-4760, Jan. 2013.

7. M. S. Farooq, U. Omer, A. Ramzan, M. A. Rasheed and Z. Atal, "Behavior Driven Development: A Systematic Literature Review," in *IEEE Access*, vol. 11, pp. 88008-88024, 2023, doi: 10.1109/ACCESS.2023.3302356.

8. Bahattin Emre Özdemir, April 2019 https://ba-works.com/blog/yazilim-dunyasinda-yeni-nesil-bir-yaklasim-bdd-behaviour-driven-development/

6. Kaynak Kod

https://github.com/bekoasil/calculator_v3