Bankacılık Alanında Doğal Dil İşleme Destekli Davranış Güdümlü Geliştirme

Ali Çıltık[1], Volkan Halil Bağcı[1], Umut Orçun Turgut[1] , ve Tunga Güngör[2]

[1] Cybersoft, İstanbul, Türkiye,
{ali.ciltik,volkan.bagci,umut.turgut}@cs.com.tr
[2] Boğaziçi Üniversitesi, İstanbul, Türkiye, gungort@boun.edu.tr

Özet. Şelale yöntemi, çevik yaklaşımlar gibi farklı yazılım geliştirme yaşam döngülerinin ilk amacı nihai ürüne zamanında, eksiksiz ve kaliteli bir şekilde ulaşmaktır. Nihai yazılım ürününe giden yolu kısaltmak ve bunu kaliteli bir şekilde gerçekleştirmek için son kullanıcının ihtiyacını odak noktasında tutan yöntemleri benimseyerek bazı çalışmalar yürütmekteyiz. Bankacılık alan uzmanları ile yazılım geliştirme uzmanlarının birlikte çalıştığı ortamda alan uzmanlarının belirlediği gereksinimlere yazılım uzmanları problemi kendi evrenlerine taşıyarak çözüm bulmaya çalışırlar. Burada sunmaya çalıştığımız yaklaşımda gereksinim kümesinin doğrudan yazılım geliştirme sürecini tetiklemesini, problem evreninden çözüm evrenine geçişteki karmaşıklığı ve yanlış anlaşılmaları en aza indirmeyi hedefliyoruz. Çalışmamızda bankacılık alan uzmanlarının "bankacılık alanının doğal dilini" kullanarak gereksinimleri geliştirme sürecine girdi olarak sağlamalarından sonra davranış güdümlü geliştirme ilkelerinin otomatize edilebileceğini gösteriyoruz.

Anahtar Kelimeler: Temel Bankacılık, Davranış Güdümlü Geliştirme (BDD), Doğal Dil İşleme (NLP), Bankacılık Alanının Doğal Dili

1 Giriş

Yazılım projelerinin zamanında bitirilememesi, müşterinin taleplerine doğru ve zamanında cevap verilememesi günümüz bilişim dünyasının en önemli sorunlarından biridir. Gereksinim analizinin doğru yapılması, paydaşların doğru belirlenmesi ve paydaşlar arasındaki iletişim sadece bütçe ve takvimi değil, teslim edilen ürünün kalitesini de doğrudan etkilemektedir. Gerçek ihtiyaç sahibinin gereksinimlerini kendi ifade ettiği şekilde yazılım geliştirme sürecinde kodlamaya doğrudan sokması ve kabul kriterlerinin aracısız olarak kaynak koda dönüşümünün sağlanması yukarıda vurgulanan sorunlara çözüm olacaktır.

Davranış güdümlü geliştirme (DGG) yönteminde odak noktası, müşterinin (ihtiyaç sahibinin) kabul test senaryoları olarak belirlediği davranışlardır. [1] Bu yöntemde kabul kriterleri ön planda tutulmaktadır ve kabul kriterlerinden kaynak koda giden süreç için bazı araçlar geliştirilmiştir. [2] Bu araçlar kabul test senaryolarının alana özgü bir dille ifade edilmesinden sonra yazılım geliştiricinin üzerinde çalışabileceği

koçanlar (stub) üretmektedir. Bankacılık alan uzmanlarının belirlediği kabul kriterlerinin doğrudan kaynak koda dönüşmesini ve kaynak kodun koçandan ziyade nihai ürüne yakın olması modelini öneriyoruz. Doğal dil ile ifade edilmiş ve kodlama sürecine girdi olarak sağlanmış kabul kriterlerinin kaynak koda çevrilmesinde kontrollü dil (controlled language) kullanılması ve bu dilin pasif cümle içermemesi gibi kısıtlamalar önerdiğimiz modelin başarısını artıracaktır. Bu kontrollü dile "bankacılık alanının doğal dili" olarak adlandırıyoruz. [3]

Önerdiğimiz modelde bankacılık alan uzmanının belirlediği senaryolar, yazılım ürünün teslimininin sonrasında bile işlevsel testlerde kullanılabilir. Otomatik işlevsel testler uygulanması Sürekli Tümleştirme (Continuous Integration)[4] kavramını farklı bir boyuta taşıyacaktır.

2 Motivasyon

2.1 Daha Sağlıklı İletişim

Davranış güdümlü geliştirme sürecinde müşteri ile proje katılımcıları arasındaki iletişim senaryolar üzerinden sağlanırken kabul test kriterleri de bu süreçte oluşmaya başlar. Senaryoların oluşumunda kullanılacak dilin sadeliği iletişimin daha sağlıklı olmasını sağlayacak ve oluşabilecek yanlış anlaşılmaların geliştirme sürecinin ön fazlarında tespit edilip düzeltilmesini kolaylaştıracaktır.

2.2 Üretkenlik Artışı

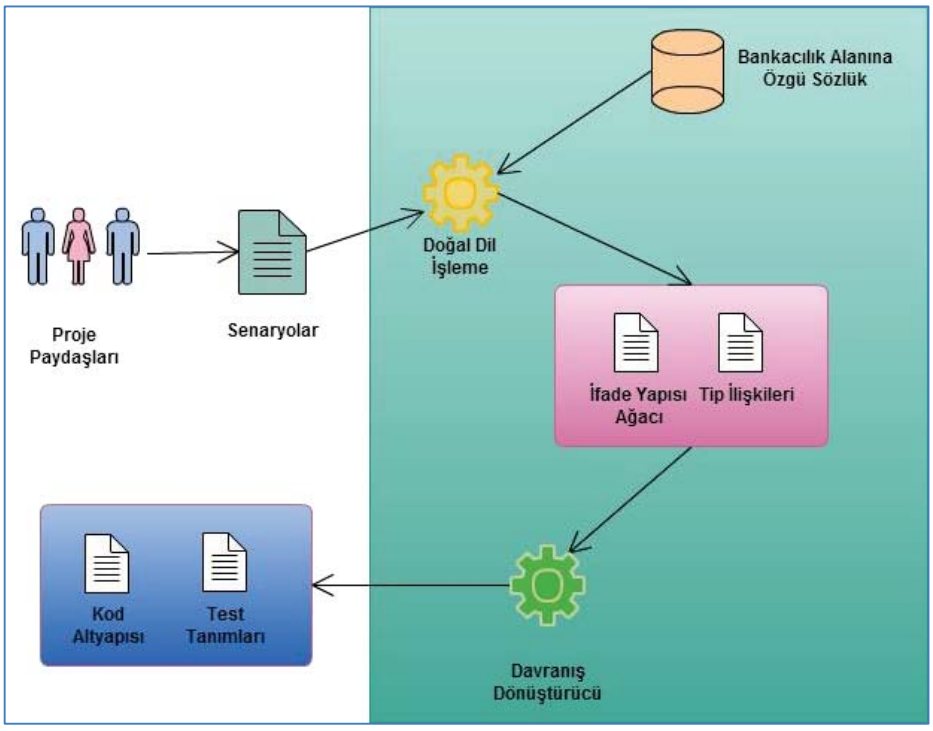
Bankacılık alanının doğal diline özgü kelimeler senaryolarımızda sıkça yer almaktadır; senaryoların kaynak koda dönüşümünde bu kelimelerin doğal dil işleme sürecinde sözdizimsel çözümleyici (parser) tarafından doğru değerlendirilmesi daha doğru sonuçlar elde etmek için önemli bir husustur. Senaryolardan otomatik kaynak kod dönüşümü ile müşteri ve proje katılımcılarının üzerinde mutabık oldukları senaryoları doğrudan kaynak koda dönüştürebilir, geliştirme eforlarını azaltarak üretkenliği arttırabiliriz. Bu işlem sonucunda üretilen kaynak kod, genel olarak tasarlanmak istenen sistemin yapısını karşılamakta ve test senaryolarını içermektedir.

2.3 Sürekli Tümleştirme

Literatürde sürekli tümleştirme kavramı daha çok birim testleri bağlamında çalışılmaktadır. Fakat bu çalışmamızdaki modeli önerirken işlevsel testlerin de gerekli durumlarda otomatik çalıştırılması sonucunda davranışsal sürekli tümleştirmenin gerçeklenebileceğini öngördük. Davranışsal sürekli tümleştirme kapsamında yazılım ürününde bir değişiklik yapıldığında, yeni bir davranış eklendiğinde veya çıkarıldığında otomatik olarak tüm davranışlar test edilecek ve ürünün tüm davranışlarının hala müşterinin belirlediği gereksinimlere uygun olduğu doğrulanacaktır.

3 Model

Bu bölümde senaryo oluşumundan, doğal dil işleme sürecine ve senaryodan otomatik kod dönüşümü kavramına yer verilecektir. Klasik yazılım geliştirme yöntemlerinde, yazılım testi ve senaryo testi adımları yazılım geliştirme işlemiyle birlikte başlatılmaktadır. Geliştirme aşamasında yazılım geliştiricileri tarafından birim testler uygulanmakta, bileşenler ortaya çıktıktan sonra ise, senaryo testleri, test uzmanları tarafından gerçekleştirilmektedir. Günümüzde uygulanmaya başlanan test güdümlü yöntemlerle birlikte test süreçleri, yazılım geliştirme işlemi başlamadan hemen önce başlatılabilmektedir. Bu yöntem, hem yazılım tasarımı hem de ürünün validasyonu anlamında yazılım geliştirme sürecine büyük oranda fayda sağlamaktadır. Çalışmamızda, bankacılık alanında oluşturulmuş senaryoları doğal dil işleme bileşenleriyle çözümleyerek, uygulamanın iskelet kodunu oluşturacak yapıların, otomatik bir şekilde üretilmesi amaçlanmaktadır. Bu yapılar aracılığıyla hem yazılım geliştiriciler bu bileşenleri koda yansıtabilecek, hem de test uzmanları test senaryolarını çıkararak ürünün paydaşlar nezdinde doğrulanmasını sağlamış olacaklardır (Şekil.1.)



Şekil. 1. Bankacılık Alanında Doğal Dil İşleme Destekli Davranış Güdümlü Geliştirme Modeli

3.1 Senaryo Oluşumu

Bankacılık iş birimleri (müşteri) ve bankacılık alan uzmanları aralarında yapılan çalışmalar sonucunda projeye ait senaryolar oluşturulur. DGG sürecinde kabul gören senaryoları karşılayan kod yapıları oluşturulur ve her senaryoyu karşılamak üzere alt test adım tanımları yer alır. Sürecin devamında kırmızı (test senaryolarından hata alınması durumu), yeşil (test senaryolarının başarılı olması durumu) ve başarılı teste ait olan kodun yeniden yapılandırmasına yönelik kod geliştirme faaliyetleri bahsedildiği sıra ile ilerler. Bu süreç yeni senaryoların eklenmesiyle kendini tekrarlayarak devam eder.

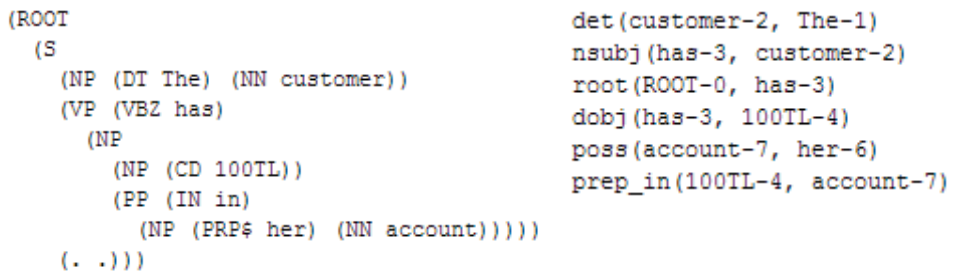
Paydaşların üzerinde anlaştığı senaryolar insan müdahalesini gerektirmeksizin kod dönüşümü sağlayabilecek içeriğe sahiptirler [6]. Bankacılık alanına özgü doğal dil işleme ile senaryodan kod dönüşüm adımı otomatikleştirilerek kod altyapısı ve test adımları üretilebilir.

3.2 Doğal Dil İşleme Desteği

Yukarıda da bahsedildiği üzere klasik Davranış Güdümlü Geliştirme sürecinde doğal dille yazılmış olan senaryolardan yazılım koduna ulaşılmaktadır. Yöntemin detayında doğal diller kullanılarak yazılmış olan senaryodan ara tanımlar çıkarılmakta ve bu tanımlardan kaynak kodun iskeletine, koçanına (stub) ulaşılmaktadır. Yazılım geliştiriciler bu ara çıktı ile yazılım üretmekte ve nihai ürün elde edilmektedir.

Bahsi geçen tüm işlemlerin yazılım geliştiriciler tarafından yapılması, proje çalışanları üzerine büyük bir yük oluşturmakla birlikte hataları da beraberinde getirmektedir. Senaryolardan yazılım kodu iskeletlerine ulaşımın doğal dil işleme yöntemiyle otomatik olarak yapılması, hem yazılım geliştiriciler üzerindeki yükü hafifletecek hem de insan kaynaklı hataların engellenmesini sağlayacaktır. Geliştirmede kullanılacak ara çıktıların, yazılım geliştirme öncesinde, kontrol edilmesi süreç açısından yeterli olacaktır.

Kullanıcılar tarafından doğal dil kullanılarak yazılmış olan senaryolardaki her cümlenin çözümlenmesi, projenin ilk gereklerinden biri olarak karşımıza çıkmıştır. Bu kapsamda, Stanford Natural Language Processing (NLP) grubu tarafından geliştirilen, açık kaynak kodlu Stanford Parser ürünü kullanımına karar verilmiştir. Farklı dillere de adapte edilmiş bu ürün, ilgili dil yapısına göre, cümleleri çözümlemekte ve her cümle için PST (phrase structure tree) ağacı oluşturmaktadır. PST dairesel bir ağaç olmayıp cümlenin yapısına göre bileşenlerin çözümlenmesiyle oluşturulmaktadır. Cümle ilk etapta isim ve fiil parçalarına, bu parçalar da alt parçalar olan sıfat, zamir, isim, fiil, vb. gibi nihai atomik parçalara ayrılmaktadır. Sözbilimsel çözümleyici, cümleyi atomik yapılarına ayırdığı gibi eş zamanlı olarak doğal dil işlemede etkili bir şekilde kullanılan tip bağımlılıklarına da ayırmaktadır. Şekil.2' de Stanford Parser uygulamasının çıktıları gösterilmektedir. [5]



(a). Sözbilimsel çözümleme **(b).** Tip İlişkileri

Şekil. 2. "The customer has 100TL in her account." Cümlesine Stanford Parser'ın Uygulanması

Bankacılık Alanına Özgü Sözlük.
İlgili veri tabanı, bankacılık alanında kullanılan isim, fiil, sıfat, zamir vb. gibi bileşenleri içermektedir. Bilişsel anlamdaşlarına göre gruplanmış olan alandaki her sözcüğe ait bir kayıt yer almakta ve sözbilimsel çözümleyicinin ürettiği atomik kelimeler bu sözlüğe girdi olarak verilerek anlamsal çıkarımların yapılması sağlanmaktadır. Bu şekilde senaryoyu tanımlayan cümlelerden anlam çıkarılması mümkün kılınmaktadır.

Atomik Bileşenler.
Senaryoları oluşturan cümleler sözbilimsel çözümleyiciden geçirilerek cümlenin atomik bileşenlerine ayrılmaktadırlar. Bu cümlelerden ara tanımlar çıkarılması ve bu tanımlardan yazılım kodu iskeletlerine ulaşılması kapsamında ilgili atomik bileşenlerin anlamlandırılması gerekmektedir. İsimler, sistemdeki nesneler ve özneleri, sıfatlar nesneler hakkında ek tanımları, fiiller ise senaryodaki aksiyonları tanımlamaktadır. Nesneler sınıf diyagramlarındaki sınıflara, özneler sıralama diyagramlarındaki aktörlere, sıfatlar sınıfların özelliklerine, fiiller ise sınıflardaki metodlara dönüştürülmektedir. Bu eşleştirme neticesinde senaryodan UML sınıf ve sıralama diyagramları oluşturulması mümkün olabilmekte ve yazılım geliştiricinin hızlı ve doğru bir şekilde kodlamaya başlanması sağlanabilmektedir (Şekil.3.)

```
Scenario: Depositing money into account
* The customer has 100TL in her account
* She deposits 50TL in to his account
* Now, her account balance is 150TL
```

(a). Senaryo

```
Given /^The customer has 100TL in her account$/ do
  @customer = Customer.new
  customer.account.balance
end

Given /^She deposits 50TL in to his account$/ do
  @customer.deposit(50)
end

Given /^Now, her account balance is 150TL$/ do
  @customer.account.balance
end
```

(b). Test Adım Tanımı

```
class Customer
  def deposit(amount)
  end
end

class Account
  attr_accessor :balance
  def initialize(amount)
  @balance = amount
  end
end
```

(b). Kod Altyapısı

```
class Customer
  attr_accessor :account

  def initialize
  @account = Account.new(100)
  end

  def deposit(amount)
  @account.balance += amount
  end
end
```

(b). Uygulama

Şekil. 3. DGG Örneği

4 Sonuç

Yazılım geliştirme yaşam döngüsünde rol alan tüm paydaşların ortak ve yaygın bir dil kullanması ve kodlama aşamasının doğrudan bu ortak ve yaygın dille beslenmesi bir çok sorunu çözecektir. Önerilen modelde yazılım geliştirme sürecinin odak noktası gerçek ihtiyaç sahibi müşterinin nihai yazılım ürününde görmek istedikleri olduğu için nihai ürününde herhangi bir iş değeri (business value) vermeyen davranışların, özelliklerin olması söz konusu olmayacaktır. Gereksinimlerin veya kabul test senaryolarının tümüne davranış ismini verdiğimiz DGG yaklaşımına ek olarak önerdiğimiz doğal dil işlemi desteği sayesinde yazılım ürününün daha kaliteli ve planlanan takvim ve bütçe içinde teslim edileceğini öngörüyoruz.

Daha sağlıklı bir iletişim, yanlış anlaşılmaların azaltılması, daha kaliteli, takvim ve bütçeye uygun teslim gibi önemli kazanımların yanında otomatik işlevsel testler ile davranışsal sürekli tümleştirme sağlanacak ve herhangi bir anda yazılım ürününün davranışsal doğruluğu korunacaktır. Bu sayede klasik anlamda birim testleri ile sağlanan tümleştirme kavramını bir adım öteye taşımış olacağız.

Kaynaklar

1. Haring, R.: Behavior Driven development: Better than test driven development. Java Magazine (2011) ISSN 1571-6236
2. Wyne, M., Hellesoy A.: The pragmatic bookshelf the cucumber book". (2012) ISBN 978-1-93435-680-7
3. Bentivogli, L., Forner, P., Magnini B., Pianta E.: Revising WordNet Domains Hierarchy: Semantics, Coverage, and Balancing. COLING 2004 Workshop on "Multilingual Linguistic Resources", Geneva, Switzerland, August 28, 2004, pp. 101-108.
4. Beck, K.: Extreme Programming Explained. (1999) ISBN 0-201-61641-6.
5. Socher R., Bauer J., Manning C. D., Ng A. Y.: Parsing With Compositional Vector Grammars. Proceedings of ACL 2013
6. Soeken M., Wille R., Drechsler R.: Assisted Behavior Driven Development Using Natural Language Processing. Objects, Models, Components, Patterns Lecture Notes in Computer Science Volume 7304, 2012, pp 269-287