T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NESNE YÖNELİMLİ YAZILIMLARDA KARŞIT KALIPLARIN BELİRLENMESİ

MEHMED TAHA ARAS

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
YRD. DOÇ. DR. YUNUS EMRE SELÇUK

İSTANBUL, 2016

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

NESNE YÖNELİMLİ YAZILIMLARDA KARŞIT KALIPLARIN BELİRLENMESİ

Mehmed Taha ARAS tarafından hazırlanan tez çalışması 26.07.2016 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Yrd. Doç. Dr. Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Yrd. Doç. Dr. Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi _____

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Yrd. Doç. Dr. Tolga OVATMAN

İstanbul Teknik Üniversitesi _____

ÖNSÖZ

Bu çalışma boyunca bana ve çalışmamıza yaptığı rehberlikten ve katkılarından dolayı Danışmanım Yrd. Doç. Dr. Yunus Emre SELÇUK'a çok teşekkür ederim. Geliştirdiğimiz karşıt kalıp tespit sisteminde önerileriyle ve bilgi paylaşımlarıyla destek olan mühendis ve araştırma görevlisi arkadaşlarıma ve desteklerini eksik etmeyen aileme teşekkürlerimi sunarım.

Temmuz, 2016

Mehmed Taha ARAS

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ vi

KISALTMA LİSTESİ vii

ŞEKİL LİSTESİ viii

ÇİZELGE LİSTESİ ix

ÖZET x

ABSTRACT xii

BÖLÜM 1

GİRİŞ 1

1.1 Literatür Özeti 1
1.2 Tezin Amacı 4
1.3 Hipotez 5

BÖLÜM 2

İLGİLİ ÇALIŞMALAR 6

2.1 Tespit Teknikleri 6
2.1.1 Manuel ve Otomatik Tespit 6
2.1.2 Kural Tabanlı Tespit 9
2.1.3 BBN (Bayesian Belief Network) Tabanlı Tespit 10
2.2 Araçlar 10
2.3 Karşılaşılan Problemler 12

BÖLÜM 3

ÖNERİLEN YAKLAŞIM 15

3.1 İncelenen Karşıt Kalıplar 16
3.1.1 Blob 16
3.1.2 Swiss Army Knife 20

3.1.3 Lava Flow 23
3.2 Tespit Mekanizması 26
3.2.1 Adım 1: Metrik Analiz Mekanizması 26
3.2.2 Adım 2: Statik Kod Analiz Mekanizması 27
3.2.3 Adım 3: Filtreleme Mekanizması 27

BÖLÜM 4

BULGULAR 30

4.1 Referans Alınan Veri Setinin Analiz Sonuçları 31
4.2 Uygulanan Yaklaşım Sonuçları 32

BÖLÜM 5

SONUÇ VE ÖNERİLER 39

KAYNAKLAR 41

ÖZGEÇMİŞ 44

SİMGE LİSTESİ

.QL	Kaynak kod analizi için tasarlanmış bir sorgu dili

KISALTMA LİSTESİ

BBN	Bahesian Belief Network
BCEL	The Byte Code Engineering Library
CBO	Coupling Between Object Classes
CK	Chidamber and Kemerer
CKJM	Chidamber and Kemerer Java Metrics
DIT	Depth of Inheritance Tree
DSL	Domain Specific Language
FN	False Negative
FP	False Positive
ICU	Is Class Used
LCOM	Lack of Cohesion in Methods
NADC	Number of Accessed Data Classes
NAM	Number of Attributes and Methods
NOC	Number of Children
OPI	Out of Package Imports
PFS	Percentage of Fields Suspicious
PMS	Percentage of Methods Suspicious
RFC	Response for a Class
SAK	Swiss Army Knife
TN	True Negative
TP	True Positive
TSC	Total Signature Count
WMC	Weighted Method per Class

ŞEKİL LİSTESİ

Sayfa

Şekil 2. 1 Moha ve diğerlerinin oluşturduğu karşıt kalıp – kod taksonomisi 7
Şekil 2. 2 Moha ve diğerlerinin oluşturduğu Spaghetti Code karşıt kalıbı kural kartı... 10
Şekil 3. 1 Örnek Blob sınıf diyagramı..... 17
Şekil 3. 2 Örnek Swiss Army Knife sınıf diyagramı 21
Şekil 3. 3 Lava Flow karşıt kalıp betimlemesi [29]..... 24
Şekil 3. 4 Karşıt kalıp tespit sistemimizin genel işleyiş diyagramı 29
Şekil 4. 1 Blob alt veri kümesinde LCOM değerleri için aykırı değer analizi sonuçları.. 30
Şekil 4. 2 Referans projelerin filtrelenmiş ortalama grafiği 32
Şekil 4. 3 Blob - filtrelenmiş tespit sonuçları grafiği..... 34
Şekil 4. 4 Swiss Army Knife - filtrelenmiş tespit sonuçları grafiği 35
Şekil 4. 5 Lava Flow - filtrelenmiş tespit sonuçları grafiği 36

ÇİZELGE LİSTESİ

Sayfa

Çizelge 2.1 Bazı manuel tespit yaklaşımları tarafından göz önünde bulundurulan problemler 14
Çizelge 2.2 Bazı otomatikleştirilmiş tespit yaklaşımları tarafından göz önünde bulundurulan problemler 14
Çizelge 3.1 Blob karşıt kalıbı tespitinde kullanılan metriklerin sınır değerleri 20
Çizelge 3.2 Swiss Army Knife karşıt kalıbı tespitinde kullanılan metriklerin sınır değerleri 22
Çizelge 3.3 Lava Flow karşıt kalıbı tespitinde kullanılan metriklerin sınır değerleri 26
Çizelge 4.1 Referans projelerin filtrelenmiş ortalama metrik değerleri 31
Çizelge 4.2 Veri setindeki sınıfların dağılımı 32
Çizelge 4.3 Blob karşıt kalıp tespit sonuçları 33
Çizelge 4.4 Swiss Army Knife karşıt kalıp tespit sonuçları 34
Çizelge 4.5 Lava Flow karşıt kalıp tespit sonuçları 35
Çizelge 4.6 Blob karşıt kalıbı karışıklık matrisi 36
Çizelge 4.7 Swiss Army Knife karşıt kalıbı karışıklık matrisi 37
Çizelge 4.8 Lava Flow karşıt kalıbı karışıklık matrisi 37
Çizelge 4.9 İlgili çalışmaların doğruluk sonuçları 38

ÖZET

NESNE YÖNELİMLİ YAZILIMLARDA KARŞIT KALIPLARIN BELİRLENMESİ

Mehmed Taha ARAS

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Yrd. Doç. Dr. Yunus Emre SELÇUK

Kalıplar tasarımı geliştirmek ve bunun sonucunda sürdürülebilirliği, yeniden kullanılabilirliği ve tersine mühendisliği iyileştirmek için kullanılan önemli tekniklerdir. Nesne yönelimli yazılımlarda yaygın olarak meydana gelen problemlere karşı yeniden kullanılabilir olan genel çözümlere ise tasarım kalıpları denmektedir. Tasarım kalıpları yaygın problemleri çözmek için kullanabilecek biçimselleştirilmiş üstün yöntemler şeklindedir. Karşıt kalıplar ise tekrar eden bir probleme karşı yaygın olarak uygulanan, tasarım kalıbı olarak düşünülen fakat aksine genellikle verimsiz ve riskli çözümlerdir. Bazı çalışmalar göstermiştir ki, karşıt kalıpların kullanılması yazılım sistemlerinin sürdürülebilirliği üzerinde olumsuz etkilere sahiptir.

Kod kokusu (kötü koku) zayıf tasarım ve geliştirmenin ve daha derin bir problemin semptomlarıdır. Bazı durumlarda bu semptomlar yazılım geliştiricilerin önemli yamalar geliştirirken acele etmelerinden ya da yerinde olmayan tercihler vermelerinden kaynaklanır. Kod kokuları kodun anlaşılabilirliğini düşürür, aynı zamanda hata ve değişim meyillidirler. Bu yüzden kod kokularının tespit edilmesi ve gerektiği zaman da düzeltme işlemlerinin planlanıp uygulanması gerekmektedir. Kod kokuları aynı zamanda karşıt kalıpların belirtileri olarak da karşımıza çıkmaktadır. Bir karşıt kalıp, bir ya da daha fazla kod kokusunun bir araya gelmesinden, ya da bazı kod metriklerinin belirli sınırları aşmasından meydana gelebilmektedir.

Bu çalışma, Java nesne yönelimli programlama dilinde yazılmış projeleri analiz ederek, bazı kod kokuları ve metrik değerlerinin hesaplanmasını ve bu değerlerin referans aralıklarda olup olmamasına göre karşıt kalıp içeren sınıfların tespit edilmesini

amaçlamaktadır. Karşıt kalıp tespitinde metrik tabanlı analizin yanında kod üzerinde bazı kokuların varlığını gösteren bazı kurallara dayalı doğrulamalar da yapılmaktadır. Varlığının tespit edilmesi amaçlanan karşıt kalıplar Blob, Swiss Army Knife ve Lava Flow karşıt kalıplarıdır. Bu karşıt kalıpların tespit edilmesi için projenin kaynak kod dosyalarını ve binary dosyalarını girdi olarak alan otomatik bir yaklaşım kullanılmaktadır. Sınıflar kendi içlerinde ve birbirleriyle etkileşimli olarak analiz edilip bazı metrikler çıkarılmaktadır. Baz alınan projelerin hesaplanan metrik değerleri üzerinde aykırı değer filtrelemesi yapıldıktan sonra ortaya çıkan ortalamalar ile referans değerler oluşmaktadır. Bu referans değerlerin dışında olan sınıflar potansiyel karşıt kalıp olarak tespit edilmektedir. Apache'nin 3 projesi ve büyük bir havayolu şirketinin 3 projesi referans projeler olarak analiz edilmiş ve filtrelenmiş ortalama değerleri baz alınmıştır. Ayrıca incelenen projenin filtrelenmiş ortalama değerleri ve yaygın olarak bilinen bazı metriklerin kabul edilen sınır değerleri de göz önünde bulundurulmuştur. Karşıt kalıp olarak veya normal olarak işaretlenen sınıflar kontrol edilerek geliştirilen yazılımın ürettiği sonuçların doğruluğu hesaplanmaya çalışılmıştır.

Anahtar Kelimeler: Karşıt kalıp, tasarım kalıbı, kod kokusu, otomatik yaklaşım, metrik tabanlı yaklaşım, kural tabanlı yaklaşım

ABSTRACT

ANTIPATTERN DETECTION IN OBJECT ORIENTED SOFTWARE

Mehmed Taha ARAS

Department of Computer Engineering

MSc. Thesis

Adviser: Asst. Prof. Dr. Yunus Emre SELÇUK

Patterns are significant techniques which improve design and consequently enhance maintainability, reusability and reengineering. Design patterns are common solutions that can be reused against generally occurring problems. Design patterns are used as efficient techniques to common problems. Antipatterns are inefficient and risky solutions which are used against recurring problems, considering that they are design patterns. Some studies have shown that using antipatterns has some negative effects on maintainability of software systems.

Code smells (bad smell) are symptoms of weak design and development or a deeper problem. Sometimes these symptoms are caused from developers who try to finish their tasks as immediate as possible and making wrong decisions. Code smells decreases the understandability of code and they generally are error and change prone. Thus code smells need to be detected carefully and refactored if necessary. At the same time one or more code smells can be symptoms of an antipattern. An antipattern can show up by existence of some code smells or some code metrics which exceed the threshold values that are determined previously.

This study aims to detect antipatterns by calculating some code smells and metric values in projects that were written in Java object oriented programming language. Also some validations are used to detect code smells on the source code depending on some rules. Inspected antipatterns in our study are Blob, Swiss Army Knife and Lava Flow. An automated approach which takes source code files and binary files as input to

detect antipatterns. Classes are analyzed by their own source codes and comparatively with each others. After outlier filtering is applied on some reference projects, averages of metric values show our reference value ranges to detect antipatterns. Classes which are out of these ranges are determined as potential antipatterns. 3 projects of Apache and 3 projects of one of the biggest airline companies are analyzed as our reference projects and filtered average values are used to determine thresholds. Also subject project and accepted values of some known metrics in literature were considered to determine thresholds. Classes that are marked as antipatterns or regular were controlled and accuracy results were tried obtain.

Keywords: Antipattern, design pattern, code smell, automated approach, metric based approach, rule based approach

BÖLÜM 1

GİRİŞ

Bu bölümde çalışma kapsamında taranan literatüre değinilmekte, tezin amacına ve hipotezine vurgu yapılmaktadır. Tezin amacı ileride ayrıntılandırılacağı üzere yazılım kalitesini önemli oranda düşüren karşıt kalıpların belirlenmesidir. Bununla birlikte, yazılım kalitesine değişik yönlerden yaklaşan geniş bir literatür de mevcuttur. Bu nedenle literatür özeti karşıt kalıplara odaklanan çalışmalarla sınırlı tutulmuştur.

1.1 Literatür Özeti

Hızla büyüyen ve gelişen yazılım sektöründe yazılım geliştirme ve bakım süreçleri de gittikçe karmaşık ve maliyetli bir hale gelmektedir. Teknolojinin sürekli ilerlemesi ve erişilebilirliğinin kolaylaşması insanların işlerini yazılımlar vasıtası ile halletmelerine daha çok teşvik etmektedir. Bundan dolayı yazılım kullanan kitle hızla büyümekte ve bu kitle kendi içerisinde de yüksek çeşitlilik göstermektedir. Bu durum özel ve kamu kuruluşlarının devasa ve karmaşık uygulamalara ihtiyaç duymasını beraberinde getirmektedir. Geliştirilecek ve bakımları yapılacak uygulamaların kapsamları büyüdükçe maliyetleri de artmakta ve göz önünde bulundurularak optimize edilmeleri zorunlu hale gelmektedir.

Yazılımların karmaşık hale gelerek geliştirilmelerinin ve bakımlarının imkânsız hale gelmesinin tek sebebi ihtiyaçların çokluğu ya da istenen uygulamanın büyüklüğü olmamaktadır. Nesne yönelimli uygulamalar tasarlanırken ve geliştirilirken doğru olduğu düşünülerek yapılan ancak projenin teknik performansını, okunabilirliğini ve devam ettirilebilirliğini düşüren önemli hataların yapılması da çok büyük bir faktör olabilmektedir. Kalıp olduğu düşünülerek uygulanan bu etkisiz ve problematik

çözümlere karşıt kalıp denmektedir. Karşıt kalıplar konusu kalıp tespitinde de ortaya çıkmaktadır ancak kalıp tespit yöntemlerini kullanarak karşıt kalıplar tespit edilememektedir. Karşıt kalıplar, tasarım ve geliştirme kalıplarının aksine tasarımdaki ve koddaki problemlerden ortaya çıkmaktadır. Karşıt kalıp (Antipattern) kelimesi ilk olarak 1995 yılında Andrew Koenig [1] tarafından kullanılmıştır. Karşıt kalıp kavramı ise 1996 yılında Michael Akroyd [2] tarafından, yaygın olarak yanlış uygulanan kalıplara reaksiyon olarak tanımlanmıştır.

Literatürde şu ana kadar tanımlanmış birçok kod kokusu ve bunlardan oluşan çok sayıda karşıt kalıp bulunmaktadır. Din [3] nesne yönelimli yazılımlarda karşıt kalıp tespitiyle ilgili yaptığı literatür çalışmasında aşağıdaki 22 karşıt kalıbı tanımlamıştır:

Anemic Domain Model: Nesne yönelimli tasarımın temel fikri veri ve sürecin birleştirilerek bir yerde toplanmasıdır ve bunun iyi bir tasarım olduğu düşünülür. Bu karşıt kalıpta ise veriler ve süreçler ayrı objelerde yer almaktadır.

BaseBean: Semantik olarak bir anlam ifade etmemesine rağmen iki sınıf arasında kalıtım uygulandığı takdirde bu karşıt kalıp ortaya çıkmaktadır.

Boat Anchor: Herhangi bir sistemde hiçbir amaca hizmet etmeyen bir yazılım parçasının bulunması bu karşıt kalıbın temel sebebidir. Bu yazılım parçasının bir amacı olmaması ile beraber diğer kullanışlı parçaların da anlaşılabilirliğini düşürmektedir.

Call Super: Bu karşıt kalıp süper sınıfın soyut bir metodunun alt sınıfları tarafından ezilmesi ve sonra da çağrılması durumunda ortaya çıkmaktadır.

Circle-ellipse Problem: Bu problem süper sınıfın, alt sınıflardaki sabitleri geçersiz kılacak şekilde bir objeyi değiştiren metotlara sahip olmasından kaynaklanır.

Circular Dependency: İki nesne veya modülün doğrudan veya dolaylı olarak birbirine bağlı olmasından kaynaklanır.

Constant Interface: Interface'te hiçbir metot olmadan bir sabit tanımlanması gibi interface'in kötü bir kullanımı sonucunda ortaya çıkan bir karşıt kalıptır.

Cut and Paste Programming: Varolan kaynaklardan kodlar kopyalanarak yeni yazılımların geliştirilmesinden kaynaklanır. Yeni yazılımlar önceki yazılımların tabanında

gelişmiş olur ve kontrol edilemeyen hatalara, aşırı büyüyen kaynak kodlarına sebep olur.

Functional Decomposition: Bu karşıt kalıp genellikle nesne yönelimli olmayan dillerde tecrübeli olup, nesne yönelimli dillerde yazılım tasarlayan ve kodlayan geliştiriciler tarafından uygulanır. Kalıtım ve çokbiçimlilik iyi kullanılamamıştır, çok sayıda private alan ve az sayıda metot kullanılmıştır.

God Object: Bu karşıt kalıpta durum bilgisinin saklanması için birçok global değişken kullanılır. Diğer biçok kod parçasında referans verilmiştir ve bu nesneyi kaldırmak başka birçok nesneyi de kullanılmaz hale getirecektir.

Jumble: Yatay ve dikey tasarım elemanları birbirine karışmıştır. Bu durum tekrar kullanılabilirlik ve sağlamlık açısından istikrarsız bir mimari ortaya çıkaracaktır.

Object Cesspool: Nesne örneklendirme maliyeti fazla olduğu durumlarda, her deafsında nesneyi oluşturup yok etmemek için ilklendirilmiş nesnelerden oluşan kümeler kullanılır. Bunlara nesne havuzları denir. Nesne havuzlarından kullanılıp geri dönen nesnelerin durum bilgileri sıfırlanmıyorsa yanlış bir kullanım söz konusudur ve bu karşıt kalıp ortaya çıkmaktadır.

Object Orgy: Kapsüllemenin yer almadığı ya da zayıf olduğu durumlarda ortaya çıkar. Nesnenin iç değişkenlerine doğrudan erişim ve değiştirme söz konusudur.

Poltergeists: Çok kısıtlı sorumluluklara ve role sahip sınıflardan kaynaklanan bir karşıt kalıptır. Genellikle başka sınıfların görevlerini gerçekleştirebilmesi geçişi geçişi sağlarlar ve yaşam döngüleri çok kısadır.

Reinvent the Wheel: Yazılım tekrar kullanımının eksik olmasından kaynaklanır. Önceden kodlanmış olan bir fonksiyonun ya da parçanın kullanılmayarak, tekrar yazılması durumudur.

Sequential Coupling: Bir sınıfın metotlarının belli bir sırayla çağrılması gerekiyorsa bu karşıt kalıp ortaya çıkmaktadır.

Spaghetti Code: Plansız ve düzensiz bir yazılım yapısıyla ortaya çıkan karşıt kalıptır. Yazılımın bakımını ve genişletilmesini zorlaştırmaktadır. Parametresiz uzun metotlar ve süreçlerde kullanılan global değişkenler başlıca belirtileridir.

Stovepipe System: İstenen kalitede olmaya sistemleri simgeler. Sistem bileşenleri plansız bir şekilde birleştirildiğinde ortaya çıkar. Bu karşıt kalıbı içeren sistemlerin anlaşılabilirliği, uyumluluğu, birlikte çalışabilirliği ve tekrar kullanılabilirliği düşük olacaktır.

Swiss Army Knife: Tasarımcı tüm olası ihtiyaçları bir yerde toplamaya çalıştığında bu karşıt kalıp ortaya çıkmaktadır. Kompleks bir interface ve interface'de yüksek sayıda imza belirtilerinden bazılarıdır.

Blob: God Object karşıt kalıbı ile eş niteliktedir. Sorumlulukların çoğu bir sınıfta toplanır ve çok büyük bir sınıf ortaya çıkar.

Vendor Lock-in: Nesne yönelimli bir sistemin dışarıdan sağlanmış başka bir yaklaşıma ya da teknolojiye yüksek ölçüde bağımlı olması durumudur. Yazılımın güncellenmesinde ve bakımında problemler ortaya çıkacaktır.

Yo-yo Problem: Kalıtım hiyerarşileri çok derin olduğunda ortaya çıkan bir karşıt kalıptır. Kodun izlenmesi ve anlaşılması zordur.

Din'in çalışmasında [3] yer almamakla birlikte, tez çalışması kapsamında incelenen Lava Flow karşıt kalıbı şu şekilde tanımlanmıştır:

Lava Flow: Uzun geliştirme sürecine sahip projelerde, geliştirici takımın uyumunun da düşük olması sebebi ile fonksiyonelliğini kaybetmiş ve kullanılmayan kod kümelerinin meydana çıkmasıdır.

1.2 Tezin Amacı

Karşıt kalıplar yazılım tasarımının ve kaynak kodlarının kalitesini önemli ölçüde düşürmektedir. Karşıt kalıplar yazılımın anlaşılabilirliği, devam ettirilebilirliği ve esnekliğini negatif olarak etkilediğinden dolayı, özellikle büyük ölçekli yazılımlarda maliyetleri ciddi miktarda yükseltmektedir. Ayrıca, kodun kalitesizliği gereksiz kaynak kullanımına da yol açacağından yazılım şirketlerine zararları büyük ölçekte olabilmektedir.

Bu çalışmada nesne yönelimli yazılımlarda sık karşılaşılan 3 karşıt kalıp türü incelenerek, kaynak kod üzerinde tespit edilmesi amaçlanmaktadır. Böylece kusurlu

tasarımlar ve kodlar önüne geçilemez bir hal almadan önce üzerlerinde gerekli düzeltme işlemleri uygulanabilecektir. Çalışmamızda kalıp olduğu sanılarak uygulanan 3 karşıt kalıp incelenecektir. Bunlar “Blob” , “Swiss Army Knife” ve “Lava Flow” karşıt kalıplarıdır. Sonuç olarak beklenen ise kaynak kod üzerinde yapılan analizler ile elde ettiğimiz verilerden yola çıkarak karşıt kalıp olmasından şüphe duyulan Java sınıflarının işaretlenmesidir.

1.3 Hipotez

Bu çalışmada karşıt kalıp tespitinde sıkça kullanılan analiz metotlarından olan metrik tabanlı, kural tabanlı yöntemlerin ve süreçlerine göre yaklaşım türlerinden ise otomatikleştirilmiş tespit yönteminin sentezlenmesinden oluşan bir yöntem kullanılmıştır. Bu yönteme göre kaynak kod üzerinden elde edilen önceden belirlenmiş bazı metriklerin değerleri ve sınıfların bazı kurallara göre incelemeye alındığında elde edilen eşleşme değerleri kullanılarak yarı anlamlı sonuç verileri elde edilmektedir. Bu yarı anlamlı sonuç verileri bir filtreleme mekanizmasından geçerek kullanıcının anlayabileceği anlamlı sonuç verilerine dönüşmektedir. Sürece herhangi bir analist müdahalesi olmamakta ve süreç olabildiğince otomatikleştirilmektedir. Karşıt kalıp tespitinde kullanılan metrik ve kural analizlerinin sınır değerleri 3 faktöre göre belirlenmektedir. Bu faktörler; referans olarak alınan bazı büyük projelerin aykırı değerleri elenmiş ortalamaları, incelenen projenin aykırı değerleri elenmiş ortalamaları ve yaygın olarak kullanılan bazı metriklerin literatürde öne sürülmüş sınır değerleridir. Bu yöntemlerin kullanımı ile birlikte hem sürecin otomatikleştirileceği hem de karşıt kalıpların sınır ve sonuç değerlerinin daha isabetli olarak elde edileceği öngörülmüştür.

BÖLÜM 2

İLGİLİ ÇALIŞMALAR

Karşıt kalıp tespitinde çeşitli yaklaşımlar öne sürülmüştür. Bu yaklaşımların tespit modellerine ve süreçlerine göre kendi içlerinde farklı avantajları ve dezavantajları olduğu görülmüştür.

2.1 Tespit Teknikleri

Bu bölümde üç farklı türde karşıt kalıp tespit tekniğinden bahsedilecektir. Bu teknikler; "Manuel ve Otomatik Tespit", "Kural Tabanlı Tespit" ve "BBN Tabanlı Tespit"tir.

2.1.1 Manuel ve Otomatik Tespit

Manuel karşıt kalıp tespit yaklaşımlarının temel geliştirilme sebebi belirsizlik problemini çözmektir. Ancak bu tarz yaklaşımlar büyük sistemlerde başarısız olmakta ve çok fazla zaman ve çaba harcamaktadır. [3]

Otomatikleştirilmiş karşıt kalıp tespit yaklaşımları büyük sistemlerin daha iyi analiz edilebilmesi ve kabul edilebilir zaman ve çaba sonuçlarına ulaşılabilmesi için geliştirilmişlerdir. Manuel yaklaşımların tersine, sonuç olarak elde edilen karşıt kalıp aday listeleri uzun, sıralanmamış ve belirsizdir.

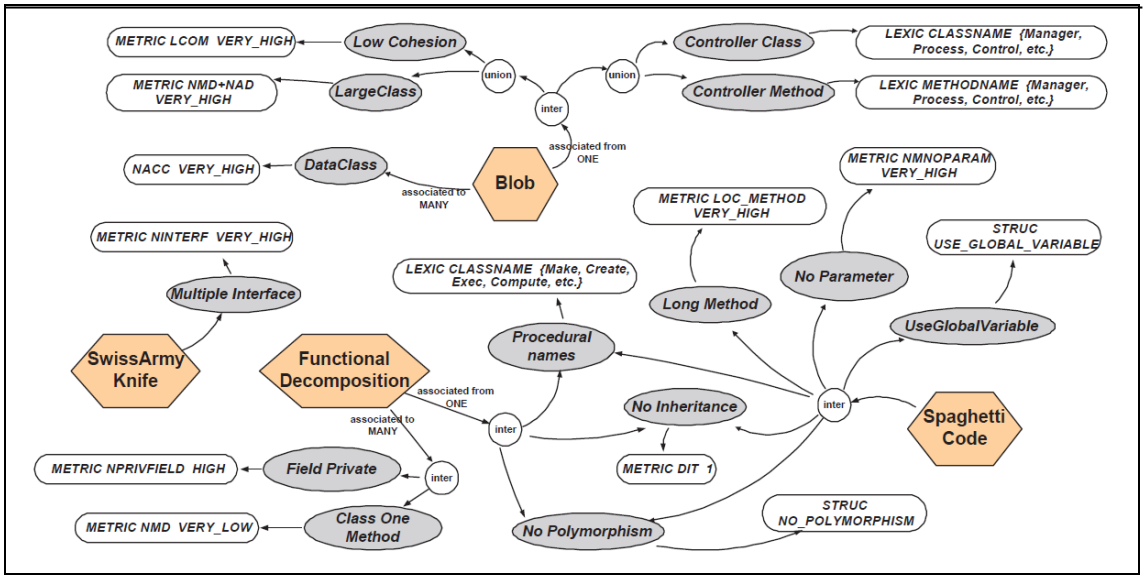
Travassos ve diğerleri [4] manuel olarak değerlendirme ve okuma tekniklerini kullanarak tasarım kokularını tespit etmeyi amaçlayan yaklaşımlarını sunmuşlardır. Teknikleri sadece manuel denetlemeye dayanmaktadır ve kokular açıkça belirtilmemiştir. Bu yüzden bu yaklaşımın büyük çaplı sistemlerde uygulanması uygun

değildir. Ayrıca Munro [5], Alikacem [6], and Ciupke [7] de çoğunlukla manuel değerlendirmelere dayanan karşıt kalıp tespit yöntemleri sunmuşlardır.

Marinescu'nun [8] IPLASMA aracında tasarım kokularını tespit etmek için metrik-tabanlı değerlendirmeye dayanan bir yaklaşım önerilmiştir. Bu yaklaşımda karşıt kalıpların başarıyla tespit edilebilmesi için metrikler hakkında yüksek bilgi birikimi gerekmektedir. Sınır değerleri değiştirmek çok değişik sonuçlara sebep olabilir. Bu yüzden sınır değerlerin belirlenmesi oldukça kritik bir konudur ve hata kabul etmemektedir.

Lanza ve Marinescu [9] , van Emden ve Moonen [10] otomatikleştirilmiş yaklaşımlarını tanıtmışlardır. Bu yaklaşımda belirsizlik ve uzun liste problemine rastlanmaktadır. Ayrıca analist değerlendirmeleri de yok sayılmıştır. Tespit sonuçlarını sunmak için bir çizge editör kullanılarak çizgelerin görselleştirilmesi tekniği uygulanmıştır.

Moha ve diğerleri [11] DECOR-DETEX adını verdikleri çalışmalarında karşıt kalıpları tanımlayan bazı kural kümelerine dayanarak ve bir karşıt kalıp – koku taksonomisi oluşturarak DSL (Domain Specific Language) tabanlı bir yaklaşım sunmuşlardır. Kural kartları tanımlamışlar ve bu kural kartlarını karşıt kalıp tespit algoritmalarına çeviren bir mekanizma oluşturmuşlardır. Belli başlı bazı karşıt kalıplara odaklanıp tanımlamışlar ve bu karşıt kalıpları otomatik üretilmiş algoritmalar ile tespit etmeye çalışmışlardır. Bu çalışmada oluşturulan karşıt kalıp - koku taksonomisi şu şekildedir;



Şekil 2. 1 Moha ve diğerlerinin oluşturduğu karşıt kalıp – kod taksonomisi

Bu taksonomiden de anlaşıldığı gibi bazı kod ve tasarım kokusu kombinasyonları karşıt kalıbın varlığı için tek başına kriter oluşturmaktadır ancak bazı kod kokularının da birlikte bulunması gerekmektedir. Bu yüzden bu taksonomide kokuların birleşim ve kesişimleri gösterilmiştir. Örneğin; bizim de incelediğimiz Blob karşıt kalıbının varlığını tespit etmek için "Büyük Sınıf" ve "Düşük Uyum" kokularının varlıkları birleştirilmiş, aynı şekilde "Kontrol sınıfı" ve "Kontrol metodu" kokularının da varlıkları birleştirilmiştir. Daha sonra ise bu iki koku grubunun içerdiği aday sınıfların kesişimleri alınarak aday liste daraltılmıştır.

Moha ve diğerleri [11]'nin karşıt kalıp taksonomisindeki bizim de incelediğimiz 2 karşıt kalıp bulunmaktadır. Blob için tanımlanmış semptom kokularını şu şekilde açıklayabiliriz:

Düşük Uyum: Bir modüle ait parçaların kendi aralarındaki ilişkilerin zayıf olmasından kaynaklanan bir kokudur. Uyumu yüksek modüller bakım ve yeniden kullanılabilirliği kolaylaştırırken, düşük uyumlu modüller ise tam aksine bu işlemleri zorlaştıracaktır.

Büyük Sınıf: Bir sınıftaki metot ve alan sayılarının toplamının belirlenen bir sınır değerin üstünde olmasıyla oluşan kokudur. Metot ve alan sayıları arttıkça sınıf daha da büyük bir hal almaya başlayacaktır.

Veri Sınıfı: Çok sayıda veriye erişim metodu kullanılan sınıflardır. Diğer sınıfların getter metotlarına ne kadar fazla eriştiğiyle ölçülmektedir.

Kontrol Sınıfı: Sistemin yaptığı işlemin büyük bir çoğunluğunu tekelleştiren, süreçlerin temel merkezi haline gelen sınıflara verilen isimdir. Bu koku gerçekten de Blob karşıt kalıbının semptomu olma konusunda yerinde bir kokudur. Ancak Moha'nın çalışmasında uygun olmayan bir şekilde tespit edilmeye çalışılmıştır. Sınıfın ismine bakarak sözcüksel olarak bir analiz yapıp sınıf isminden kontrol sınıfı olup olmadığını anlamaya çalışmışlardır. Bu çok riskli ve etkisiz bir yöntemdir. Sınıf isimleri geliştiriciden geliştiriciye değişmekle beraber, farklı konuşma dillerinde de farklı sınıf isimleri verilebilecektir.

Kontrol Metodu: Kontrol sınıfında olduğu gibi süreçlerin önemli bir kısmının bir metoda yüklenmesidir. Bunun da tespitinde yanlış bir yol olarak metot isminin sözcüksel analizini kullanmışlardır.

Swiss Army Knife karşıt kalıbında sadece bir koku kullanmışlardır. Bu koku da şu şekildedir:

Çoklu Arayüz: Bir sınıfın birden fazla arayüzü bulunduğunda bu kokunun ortaya çıkacağı öne sürülmüştür. Fakat nesne yönelimli programlama dillerinde çoklu arayüz kullanımı yaygın ve doğru kullanıldığında da etkili bir yöntemdir. Birden fazla arayüze sahip bir sınıfın karşıt kalıp olarak işaretlenmesini doğru bulmadığımız için sonraki bölümlerde anlatacağımız üzere çalışmamızda farklı semptomların kullanımını öne sürdük.

Simon [12] , Rao [13] ve Khomh [14]'un çalışmaları da karşıt kalıp tespiti için yürütülen otomatik yaklaşımlardandır.

2.1.2 Kural Tabanlı Tespit

Bilgilendirici kural tanımlama birçok karşıt kalıp tespit yaklaşımının temel taşını oluşturmaktadır. Bu kurallar analistler tarafından oluşturulur ve kokuların karakteristiklerini tanımlamak için kullanılırlar. Bu karakteristikleri tanımlamak için sayısal, yapısal ve sözcüksel semptomların kombinasyonları kullanılır [15]. Her koku kendi tespit kuralına ve çok kritik bir karar olan doğru sınır değerine ihtiyaç duyar. Bu kuralların kapsamındaki olası karşıt kalıp sayısı çok büyük olabileceğinden, manuel olarak kullanımı etkisiz kalacaktır [3]. Oluşturulan kuralların otomatik bir tespit mekanizmasına yerleştirilmesinin bu sorunu çözeceği öngörülmektedir.

Kural tabanlı tespit yaklaşımlarının sonuçları çok katı olmakta, karşıt kalıptır ya da karşıt kalıp değildir şeklinde sonuçlar ortaya çıkmaktadır. Bu yüzden sınır değerlere yakın olan birçok sınıf yanlış bir şekilde sınırın altında ya da üstünde kalabilir. Bu durum normal sınıfların karşıt kalıp olarak, karşıt kalıpların da normal sınıf olarak tespit edilmesine yol açabilmektedir [3].

Moha [11] çalışmasında her bir karşıt kalıp için kural kartları hazırlamıştır. Sonra bu kural kartlarını karşıt kalıp tespit algoritmalarına çeviren bir mekanizma oluşturmuştur. Kural kartlarında ilgili karşıt kalıbın semptomları ve bu semptomların sınır değerleri yer almaktadır. Ayrıca karşıt kalıbın varlığını tespit edilebilmesi bu semptomların birleştirme işlemine mi yoksa kesişim işlemine mi ihtiyaç duyulduğu da "INTER" ,

"UNION" gibi anahtar kelimelerle belirtilmiştir. Kural tabanlı yaklaşımlarda kural kartları bir karşıt kalıbın tespit edilebilmesi için gerekli tüm ön bilgileri içinde bulundurmaktadır. Bu çalışmada Spaghetti Code karşıt kalıbı için oluşturulan kural kartı örneği şu şekildedir:

```
1  RULE_CARD : SpaghettiCode  {
2    RULE : SpaghettiCode
           { INTER LongMethod NoParamete NoInheritance
           NoPolymorphism ProceduralName UseGlobalVariable };
3    RULE : LongMethod        { METRIC LOC_METHOD VERY_HIGH 10.0 };
4    RULE : NoParameter       { METRIC NMNOPARAM VERY_HIGH 5.0 };
5    RULE : NoInheritance     { METRIC DIT 1 0.0 };
6    RULE : NoPolymorphism    { STRUCT NO_POLYMORPHISM };
7    RULE : ProceduralName    { LEXIC CLASS_NAME
                                    (Make, Create, Exec...) };
8    RULE : UseGlobalVariable { STRUCT USE_GLOBAL_VARIABLE };
9   };
```

Şekil 2. 2 Moha ve diğerlerinin oluşturduğu Spaghetti Code karşıt kalıbı kural kartı

2.1.3 BBN (Bayesian Belief Network) Tabanlı Tespit

BBN bir rastgele değişkenler kümesini ve bunların koşullu bağımlılıklarını yönlendirilmiş çevrimsiz çizge üzerinden temsil eden olasılıksal bir grafik modeldir. BBN tabanlı karşıt kalıp tespit yaklaşımları belirsizlik probleminin önüne geçer ve geçmiş sonuçlar etkinliği artırmak için kullanılabilir [14]. Sınıfların karşıt kalıp olma olasılıkları hesaplanır, fakat bu süreç pahalıdır ve yüksek zaman ve bilgiye ihtiyaç duyar [16].

Bahsedilen süreç uzman kararlarına gereksinim duyar. Geliştiricilerin zaman ve çaba konusundaki etkinlikleri de bu uzmanların yeterince bilgili ve konuya hâkim olup olmamasına bağlıdır.

BBN modelleri yalnızca sundukları içerik ile uyumlu olmaktadır ve farklı içeriklere genelleştirilememektedirler. Bu faktör BBN yaklaşımlarının esnekliğini düşüren ve ilgili konuyu çözmek için ekstra çalışmalar gerektiren bir faktördür [3].

2.2 Araçlar

Kod kokuları, tasarım kusurları ve karşıt kalıplara yönelik akademik ve ticari amaçlı birçok araç geliştirilmiştir. Bu araçlar bağımsız çalışan uygulamalardan, geliştirme

ortamları için yazılmış eklentilere kadar çeşitlilik göstermektedir. Bu araçlar kodu analiz edip kalite tahminlerini gösteren sonuçlar üretmeyi amaçlamaktadır.

Bu araçlardan bazılar şu şekildedir: CheckStyle [17] statik kod analizini destekler ve yapılandırılmış kodlama standartlarını kullanarak kod ihlallerini rapor eder. FindBugs [18] doğruluk ve performansla alakalı bozuklukları statik byte kodunu analiz ederek bulmayı hedefler. JArchitect [19] kod bağımlılıklarını, tanımlanmış belli tasarım kurallarına, etki analizine ve kodun farklı sürümleriyle kıyaslamalara göre görselleştirir ve analiz eder. PMD [20] potansiyel problemleri tanımlamak için Java ve XPath kullanarak geliştiricilere kodu analiz etme imkânı verir. SemmleCode [21] kod kokularını tespit edebilmek için geliştiricilerin .QL olarak adlandırılan dilde nesne yönelimli kod sorgularını çalıştırmasına olanak sağlar.

CKJM Kütüphanesi: Kendi çalışmamızda da kullandığımız CKJM (Chidamber and Kemerer Java Metrics) [22] kütüphanesi derlenmiş Java dosyalarının bytecode'larını kullanarak CK metriklerini hesaplar. CKJM ile hesaplanan metrikler aşağıdaki gibidir:

- WMC (Weighted Method per Class): Sınıf başına düşen ağırlıklı metot sayısının gösteren metriktir.
- DIT (Depth of Inheritance Tree): İlgili projedeki ya da modüldeki kalıtım ağacının derinlik bilgisini barındıran metriktir.
- NOC (Number of Children): Bir ebeveyn sınıfın sahip olduğu çocuk sınıfların sayısıdır.
- CBO (Coupling Between Object Classes): İki obje arasındaki bağlaşımının derecesini gösteren metriktir.
- RFC (Response for a Class): Yine bağlaşım derecesini simgeleyen metriklerden birisidir, bir sınıf içerisinde çağrılan farklı metotlar ya da kurucu metotların sayısıdır.
- LCOM(Lack of Cohesion in Methods): Bir sınıftaki uyum eksikliğinin ne derece olduğunu gösteren bir metriktir. Derecesi 0-1 arası mükemmel seviyede olmakla beraber, arttıkça sınıftaki uyumun düştüğünü göstermektedir.

IPlasma: Marinescu'nun [8] çalışmasında geliştirip kullandığı platformdur. Bu platform 80'den fazla metrik hesaplayabilmektedir. Hesaplanan metrikler aşağıdaki gibi bazı kategorilere ayrılmıştır:

- Büyüklüğe göre
- Karmaşıklığa göre
- Bağlaşım(Coupling) metrikleri
- Uyum(Cohesion) metrikleri

Apache Commons BCEL (The Byte Code Engineering Library): Kullanıcılara binary Java .class dosyalarını analiz etmek, oluşturmak ve değiştirmek için uygun bir yol sağlamaya amaçlayan bir kütüphanedir. BCEL hâlihazırda bazı derleyiciler, eniyileştiriciler, hassas bilgi gizleyiciler, kod üreticiler ve kaynak kod analiz projeleri gibi projelerde kullanılmaktadır.

Sonar: Asıl olarak Java programlama dili için, yazılım geliştiricilerin tasarım hataları, olası buglar ve kod kusurlarından etkisiz tasarımlara, kod tekrarlarına, eksik test kapsamasına ve aşırı karmaşıklığa kadar kod analiz verilerine erişebilmelerini ve takip edebilmelerini sağlayan bir yazılım kalite yönetimi platformudur. [23]

Sonar kaynak kodu toplar ve analiz eder bunun sonucunda kaliteyi ölçmeyi ve projeler için raporlar sağlamayı amaçlar. Statik ve dinamik analiz araçlarını birleştirerek kalite ölçümü yapar. [23]

2.3 Karşılaşılan Problemler

Dhambri ve diğerleri [24] karşıt kalıp tespit süreçlerinde karşılaşılan 6 problem öne sürmüşlerdir ve etkili bir tespit yaklaşımının geliştirilebilmesi için bunları dikkatle göz önünde bulundurulması gerekmektedir [25].

Problem 1: Bir sınıfın karşıt kalıp olup olmadığına karar vermek kesin olmayan bir işlem olabilir. Çünkü tüm karşıt kalıp semptomlarını gösteren bir sınıf normal bir sınıf olabilmektedir. Bu yüzden bu süreçte daha isabetli tespitler yapabilmek için analist görüşlerine ihtiyaç duyulacaktır.

Problem 2: Büyük bir aday sınıflar kümesini listelemek etkisiz bir süreç olacaktır. Sonuçlar çok fazla yanlış pozitif değerler içerir ve bu yüzden tespit tekniği gözden çıkarılabilir. Aday sınıflar listesinin oluştururken analist değerlendirmeleri gerekebilmektedir.

Problem 3: Analist değerlendirmelerini göz önünde bulundurmak karmaşık bir süreç olabilmektedir. Sınıfların güçlü, normal veya zayıf tasarıma sahip olduğuna karar vermek analistlerin bilgi ve tecrübelerine dayanmaktadır.

Problem 4: Karşıt kalıplar tespit edilirken kullanılan çatının göz önünde bulundurulması gerekmektedir. Çoğu analist tarafından iyi olarak nitelendirilen sınıflar, bazı spesifik ortamlarda bazı tasarım konseptlerini ihlal edebilmektedir.

Problem 5: Karşıt kalıp tespitinde kullanılan sınır değerleri belirleme süreci oldukça önemlidir. Farklı analistler tarafından aynı sınır değeri farklı olarak hesaplanabilmektedir.

Problem 6: Semantik verinin kullanılması ve geliştirilmesi sorunlu işlemlerdir. Bazı karşıt kalıp semptomları semantik veri içerebilir ve bunu göz önünde bulundurmak için otomatikleştirilmiş bir tespit tekniği gerekli olacaktır.

Ayrıca Kaur[25] bazı manuel ve otomatikleştirilmiş yaklaşımları Dhambri'nin [24] önerdiği problemlere göre kendi aralarında karşılaştırmıştır. Bu karşılaştırmanın sonuçları aşağıda gösterilmiştir (Çizelge 2.1 ve Çizelge 2.2).

Çizelge 2.1 Bazı manuel tespit yaklaşımları tarafından göz önünde bulundurulan problemler

Problemler	**Manuel Yaklaşımlar**					
	Travassos et al. [4]	**Marinescu [8]**	**Munro [5]**	**Alikacem & Sahraoui [6]**	**Dhambri et.al. [24]**	**Ciupke [7]**
Problem 1	Y					
Problem 2						
Problem 3	Y				Y	Y
Problem 4	Y				Y	Y
Problem 5		Y	Y	Y	Y	
Problem 6	Y	Y	Y			Y

Çizelge 2.2 Bazı otomatikleştirilmiş tespit yaklaşımları tarafından göz önünde bulundurulan problemler

Problemler	**Yarı-Otomatikleştirilmiş Yaklaşımlar**						
	Simon et al. [12]	**van Emden & Moonen[10]**	**Lanza & Marinescu [9]**	**Rao & Reddy [13]**	**Moha [11]**	**F. Khomh et al. [14]**	**Önerilen Yaklaşım**
Problem 1				Y		Y	
Problem 2						Y	Y
Problem 3	Y						Y
Problem 4	Y				Y	Y	
Problem 5	Y	Y	Y		Y	Y	Y
Problem 6					Y	Y	

BÖLÜM 3

ÖNERİLEN YAKLAŞIM

Önceki çalışmalarda farklı karşıt kalıp tespit yaklaşımları önerilmesine rağmen, hepsinin belli başlı güçlü yanları ve karşılaştıkları problemler bulunmaktadır. Bu çalışmalar çeşitli karşıt kalıpları, tasarım kokularını ve kod kokularını tespit etmek için farklı programlama dillerinde ve geliştirme ortamlarında geliştirilmiştir.

Bu çalışmamızda nesne yönelimli yazılım sistemleri için metrik ve kural tabanlı otomatikleştirilmiş bir karşıt kalıp tespit yaklaşımı öneriyoruz. Önerdiğimiz metot şüpheli sınıflardan oluşan nihai sonuçları elde etmek amacıyla 3 ana adımdan oluşmaktadır. Tespit algoritmalarının geliştirilmesinde Java programlama dili kullanılmıştır ve analiz edilecek hedef sınıflar da Java sınıflarıdır. Bu çalışmada, 3 karşıt kalıba odaklanılmıştır. Bu karşıt kalıplar "Blob", "Swiss Army Knife" ve "Lava Flow"dur.

Karşıt kalıp tespitinde kullanılan sınır değerlerinin belirlenmesi çok kritik bir süreç olduğundan dolayı birden fazla sınır belirleme işlemi bir arada kullanılmıştır. Sınır değerler belirlenirken aşağıdaki 3 kriter kullanılmıştır:

- Çalışmamızda kullanılan metriklerin literatürdeki ilgili çalışmalarda kabul edilmiş sınır değerleri
- Referans olarak belirlediğimiz projelerin ortalama değerleri
- Analiz edilen projenin ortalama değerleri

Bu kriterleri kullanmakla hem literatürdeki değerlerden ve bazı büyük projelerin değerlerinden faydalanmış olunacağı hem de projenin ortalama değerleri alındığı için kodlama ve tasarım stillerinin farklılıklarından oluşacak sapmaların engelleneceği öngörülmüştür.

Sınır değerlerin belirlenmesi için analiz edilen referans projeler 6 büyük çaplı projedir. Bunların 3 tanesi Apache'nin açık kaynak projeleri olan BCEL, Commons ve Maven Core projeleridir. Diğer 3 proje ise önemli bir havayolu şirketinin büyük çaptaki projelerindendir.

3.1 İncelenen Karşıt Kalıplar

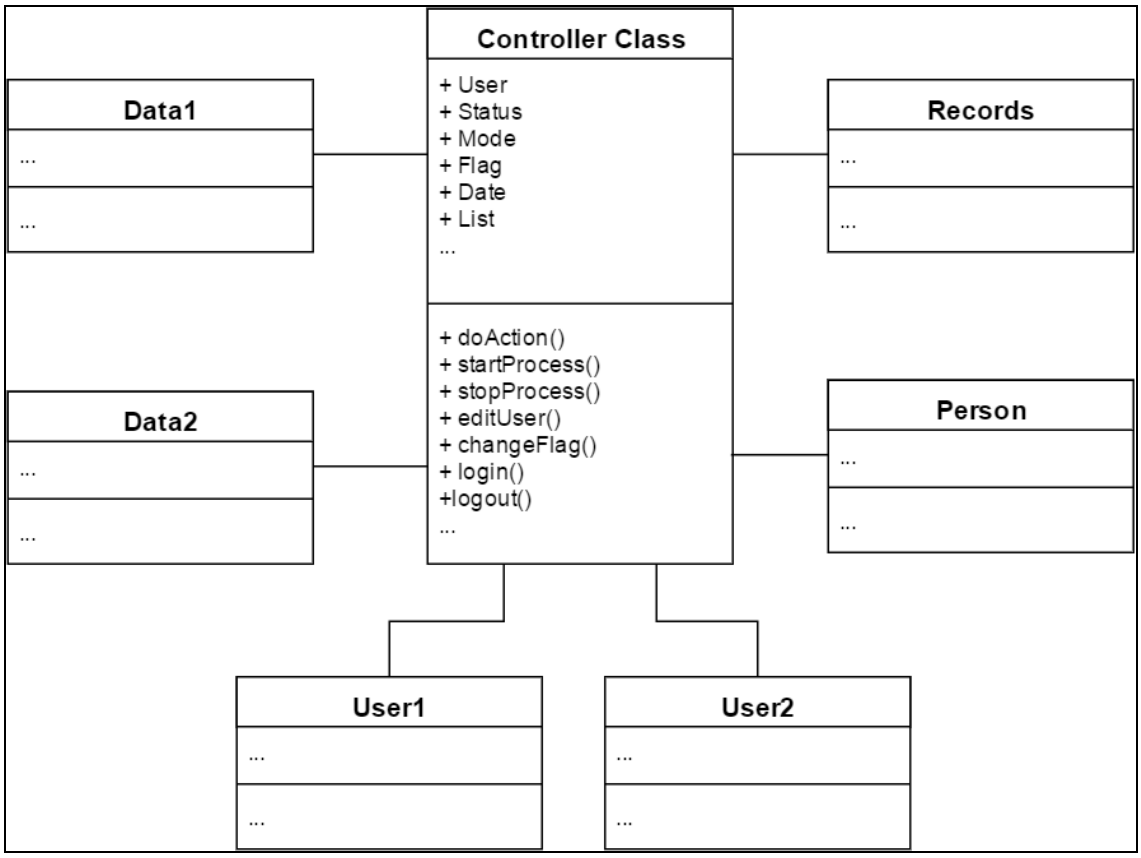
Bu bölümde tez çalışması kapsamında belirlenen karşıt kalıplar tanıtılacaktır.

3.1.1 Blob

"God Class" ya da "God Object" olarak da isimlendirilen, yaygın olarak bilinen bir karşıt kalıptır. Blob çok sayıda ve farklı sorumlulukları olan sınıflardır. Genellikle yüksek sayıda metot ve alanlara sahip olmakla nitelendirilir. Bu sınıflar farklı özellikte işleri üstlenirler ve veri sınıflarıyla çok sayıda ilişkileri vardır [26].

Blob karşıt kalıbı yaygın olarak tüm süreçlerin bir sınıfta tekelleştirildiğinde ortaya çıkmaktadır. Bir sınıf diyagramında, basit veri sınıfları tarafından çevrelenmiş kompleks bir kontrol sınıfı olarak gösterilebilir. Bu veri sınıfları sadece parçalanmış bir biçimde Blob sınıfın ihtiyaç duyacağı verileri sağlamakla görevlidirler ve sürece hiçbir şekilde müdahil olmamaktadırlar. Buradaki temel problem projedeki ya da modüldeki sorumlulukların dengeli bir şekilde dağıtılmamış olması ve tek bir sınıfa yüklenmesidir.

Aslına bakılırsa Blob içeren kod uyarlamaları, her ne kadar nesne yönelimli programlama dillerinde kodlanmış olsalar da prosedürel tasarım yaklaşımının uygulanmaya çalışıldığını göstermektedir. Prosedürel tasarımlarda da aynı Blob karşıt kalıbında olduğu gibi süreçler ve veriler birbirinde tamamen ayrılmaktadır. Ancak nesne yönelimli tasarımlar süreçleri ve verileri farklı parçalar üzerinde birleştirmeyi amaçlar.



Şekil 3. 1 Örnek Blob sınıf diyagramı

Genellikle Blob sınıflar tasarımcıların gereksinimleri uygunsuz bir biçimde dağıtmalarından dolayı ortaya çıkmaktadır. Eğer bir sınıfa yüklenen gereksinimler diğer sınıfların oluşturulma nedenlerini geçersiz kılıyorsa, bundan sonraki tasarım süreci ve hatta ek geliştirmeler bu sınıfa doğru kaymaya başlayacaktır. Bu durum belli bir süreden sonra düzeltmesi çok güçleşen, okunabilirliği düşük ve devam ettirilemez bir sorunlu sınıf ortaya çıkaracaktır.

Blob sınıfların ortaya çıkmalarındaki başka bir neden de tekrarlamalı (iteratif) geliştirme süreçleridir. Bu süreç tipinde sorumlulukların en başından dengeli bir şekilde dağıtılması güç olabilmektedir. Bunun sonucunda her bir iterasyonda sorumluluk dengesi gittikçe bozulabilmekte ve sonunda devasa sınıflar ortaya çıkabilmektedir. Blob sınıflarda bakım ve devam ettirilebilirlik oldukça düşeceğinden olduğu gibi bırakılan gereksiz kod parçaları ileride değineceğimiz Lava Flow karşıt kalıbının oluşmasına da yol açabilmektedir.

Bu karşıt kalıbın engellenmesi ya da tespit edildiği durumlarda düzeltilebilmesi için öncelikle gerekli niteliklerin ve operasyonların doğru şekilde kategorize edilmesi gerekmektedir. Daha sonrasında bunlar alanlar ve metotlar şeklinde uygun sınıflara eklenmeli ya da taşınmalıdır. Gerektiği durumlarda daha yüksek uyuma sahip küçük sınıflar oluşturularak dengeli bir şekilde sorumluluklar dağıtılmalıdır. Sınıflar arasındaki gereksiz bağlantılar ortadan kaldırılmalı ve böylece bağlaşım düşürülmeye çalışılmalıdır. Tüm işlemleri tamamladıktan sonra ortaya yüksek uyumlu ve sorumlulukları dengeli bir şekilde dağıtılmış sınıflar ortaya çıkmalıdır.

Çalışmamızda Blob sınıflarını kaynak kod üzerinde tespit etmek için 4 faktör belirlenmiştir. Bu faktörler şu şekildedir.

- LCOM (Lack of Cohesion in Methods - CK Metrik)
- RFC (Response for a class - CK Metrik)
- NAM (Number of Attributes and Methods)
- NADC (Number of Accessed Data Classes)

LCOM

Bu metrik bir sınıftaki metotlar ve lokal değişkenler arasındaki korelasyonu ölçmektedir. Yüksek uyum yerinde bir sınıf bölümlemesi yapıldığının göstergesidir. Uyum eksikliği anlamına gelen LCOM ise arttıkça bunun tam tersine karmaşıklık artmaktadır. LCOM değeri yüksek olan sınıflar parçalara ayrılarak yüksek uyumlu sınıflara tekrardan sorumluluk dağıtımı yapılabilir. LCOM değeri 0'dan başlamakta ve artarak devam etmektedir. 0 ile 1 arası değere sahip sınıfların mükemmel uyuma sahip olduğu genel olarak kabul edilmiştir. Ancak kötü olarak nitelendirilebilmesi için gerekli sınır farklı çalışmalarda ve projelerde farklı değerler almıştır. Çalışmamızda bu metriğin hesaplanabilmesi için ileride de değineceğimiz CK metriklerini hesaplayan CKJM kütüphanesi kullanılmıştır. LCOM sınıfın kendi içindeki uyuma bağlı olan bir metrik olduğu için, projedeki ya da modüldeki diğer sınıflarla ilişkilendirilerek analiz edilmesine gerek olmamaktadır.

RFC

Bir sınıfa ait bir objenin herhangi bir metodunun çağrılmasıyla tetiklenebilecek farklı metotların sayısı RFC değeri olarak tanımlanmıştır. Öncelikle bir sınıftaki tüm metotların doğrudan ya da dolaylı olarak kaç tane farklı metot tetiklediği hesaplanması gerekir. Bunun için özyinelemeli olarak bir metodun içinde çağrılan metotlar, onların içinde çağrılan metotlar ve onların içinde çağrılan metotların sayıları toplanır ta ki çağrılan farklı bir metot kalmayana kadar. Bu işlem sınıftaki tüm metotlar için yapılacaktır ve elde edilen değerler toplanarak RFC değerine ulaşılacaktır. RFC, sınıfın karmaşıklığının ve diğer sınıflar ile bağlaşımının yüksek olduğunun semptomlarındandır. Çalışmamızda RFC metriğinin hesaplanabilmesi için CKJM kütüphanesi kullanılmıştır.

NAM

Sınıftaki metot sayılarının ve alan sayılarının toplamı olarak ifade edilir. Bu değer yükseldikçe sınıfın karmaşıklığının arttığının ve okunabilirliğinin düştüğünün göstergesidir. Genellikle sorumlulukları iyi dağıtılmamış sınıflarda görülür. İş yükünün ağırlıklı olarak kaydığı sınıfta dolaylı olarak metot ve alan sayısını da yükseltmek gerekecektir. Bu durum NAM metrik değerini olumsuz şekilde artıracaktır.

NADC

Bir sınıfın, diğer veri sınıflarının alanlarına ne kadar erişim yaptığı ile ölçülen bir metriktir. Sınıfın içinde, başka veri sınıflarının getter metotlarının kaç defa çağrıldığı NADC değerini verecektir. Bu değerin yüksek olması ilgili modüldeki süreçlerin bir sınıfta yoğunlaştığını göstermektedir. Genellikle prosedürel dillerin getirdiği alışkanlıkların nesne yönelimli dillerde kullanılmasından kaynaklanır. Veri sınıfları ile süreçleri içeren sınıfın izole edilme yöntemi çoğunlukla prosedürel dillerde kullanılır.

NAM ve NADC sınır değerleri önceden belirlediğimiz 6 referans projedeki metrik sonuçlarının aykırı değerler filtrelendikten sonraki ortalamalarına göre belirlenmiştir. Ayrıca incelenen projenin büyüklüğü ve geliştirme ortamları çok farklılık gösterdiği takdirde dinamik olarak incelenen projenin de ortalamaları göz önünde bulundurulmaktadır. LCOM ve RFC metriklerinin sınır değerleri belirlenirken ise Ferreira [27] ve Jhans'ın [28] çalışmalarında önerilen değerler baz alınmıştır.

Ferreira'nın çalışmasında nesne yönelimli yazılımlarda LCOM için farklı kriterlere göre farklı değerler önerilmiştir. Bu kriterler:

- Genel sınır değerleri
- Uygulamanın çalışma alanına göre sınır değerleri
- Yazılımın tipine göre sınır değerleri
- Yazılımın büyüklüğüne göre sınır değerleri

Biz çalışmamızda genel sınır değerlerini baz alarak LCOM sınırlarımızı belirledik. Blob karşıt kalıbı için belirlediğimiz sınır değerleri aşağıdaki tabloda yer almaktadır:

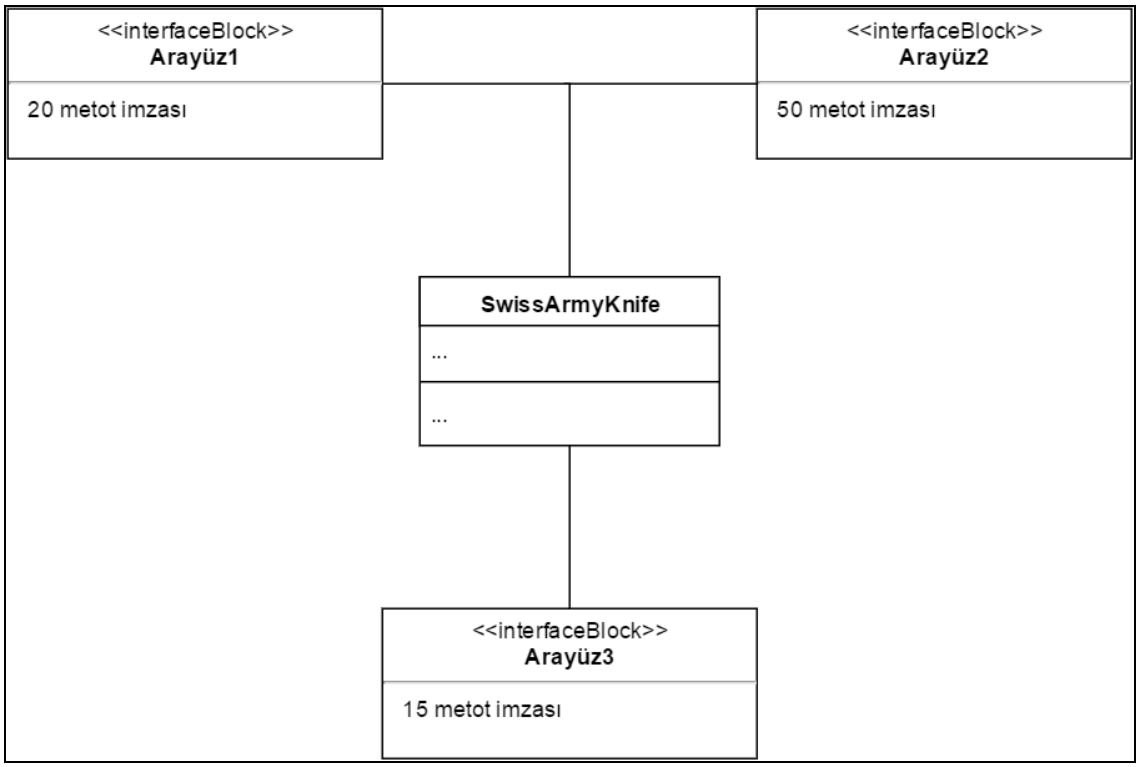
Çizelge 3. 1 Blob karşıt kalıbı tespitinde kullanılan metriklerin sınır değerleri

	İyi	**Sıradan**	**Kötü**
LCOM	0-1	1-20	> 20
RFC	0-50	50-100	> 100
NAM	0-10	10-20	> 20
NADC	0-3	3-6	> 6

3.1.2 Swiss Army Knife

"Kitchen Sink" olarak da adlandırılan bu karşıt kalıp sınıfın karmaşık arayüz yapısına sahip olmasından kaynaklanmaktadır. Swiss Army Knife tasarımsal bir karşıt kalıptır ve tasarımcı tüm olası kullanımları bir sınıfta toplamaya çalıştığında ortaya çıkmaktadır. Tüm ihtiyaçları tek bir sınıfta karşılamaya çalışmak anlamsız bir çabadır ve sınıfın arayüzlerindeki imza sayısının oldukça artmasına neden olmaktadır. Birçok işlev tek bir yerden karşılanmaya çalışıldığı için bu karşıt kalıba İsviçre Çakısı ismi verilmiştir.

Bu karşıt kalıbın sektördeki gerçek örneklerine bakıldığında onlarca metot imzasından binlerceye kadar metot imzasına sahip olan sınıflarla karşılaşılmaktadır. Sağlayıcılar ürünlerinin tüm olası ihtiyaçları karşılamasını istedikleri için ticari yazılımlarda bu karşıt kalıba sıklıkla rastlanmaktadır.



Şekil 3. 2 Örnek Swiss Army Knife sınıf diyagramı

Karmaşıklığı yönetme kaygısı bulunmadığı için Swiss Army Knife karşıt kalıbı problematik bir kullanımdır. Karmaşık arayüz diğer programcılar tarafından anlaşılabilirliği zorlaştıracak ve sınıfların ne amaçla kodlandığının anlaşılmasını engelleyecektir. Bu durum hata ayıklama, belgeleme ve bakım süreçlerini de zor süreçler haline getirecektir.

Swiss Army Knife karşıt kalıbının engellenmesi ya da düzeltilmesi için sınıfların tasarımlarını tekrar gözden geçirmek gerekecektir. İhtiyaç duyulan operasyonlar gruplara ayrılmalı ve bu operasyon gruplarını gerçekleştirecek sınıfların tasarımları birbirinden izole edilmelidir. Arayüzler rahat anlaşılabilir bir hale gelene kadar atomik parçalara ayrılmalı ve gerekmedikçe bir sınıf çok sayıda arayüzü kullanmamalıdır. Bu işlemler ancak iyi bir analiz yapılarak ihtiyaçların ve bunları tatmin etmek için sağlanacak operasyonların yerinde bölümlere ayrılmasıyla olacaktır.

Çoğu çalışmada Swiss Army Knife karşıt kalıbının belirlenmesinde sadece arayüz sayısına ya da arayüzdeki metot imza sayısı göz önünde bulundurulmuştur. Bizim çalışmamızda bu faktöre ek olarak 2 tespit kriteri daha eklenmiştir. Tespit işleminde kullandığımız 3 faktör aşağıdaki gibidir:

- LCOM (Lack of Cohesion in Methods)
- OPI (Out of Package Imports)
- TSC (Total Signature Count)

OPI

Sınıftaki dışarıdan alınan paketlerin arasından Java, Javax, W3 paketleri ve sınıfın ait olduğu kendi paket çıkarıldığında kalan paketlerin sayısı olarak hesaplanır. Sınıfın sorumluluğunda olmayan kaç paketi kullandığı bulunmaya çalışılmaktadır. OPI değerinin yüksek olması sınıfın kendi sorumluluğunda olmayan işleri üzerinde topladığı anlamına gelmektedir.

TSC

Sınıfın kullandığı tüm arayüzlerdeki metot imzalarının toplam sayısını ifade eder. Her bir imza sınıfın gerçekleştirebildiği bir operasyon anlamına geldiğinden dolayı, TSC değeri arttıkça sınıf gereğinden fazla işlemi gerçekleştirebilecek şekilde tasarlanmış anlamına gelecektir.

OPI ve TSC metriklerinin sınır değerleri referans projelerin aykırı değerleri filtrelendikten sonraki ortalamalarına göre belirlenmiştir. Swiss Army Knife karşıt kalıbı için belirlediğimiz sınır değerleri aşağıdaki tabloda yer almaktadır:

Çizelge 3. 2 Swiss Army Knife karşıt kalıbı tespitinde kullanılan metriklerin sınır değerleri

	İyi	**Sıradan**	**Kötü**
LCOM	0-1	1-20	>20
OPI	0-4	4-8	>8
TSC	0-5	5-10	>10

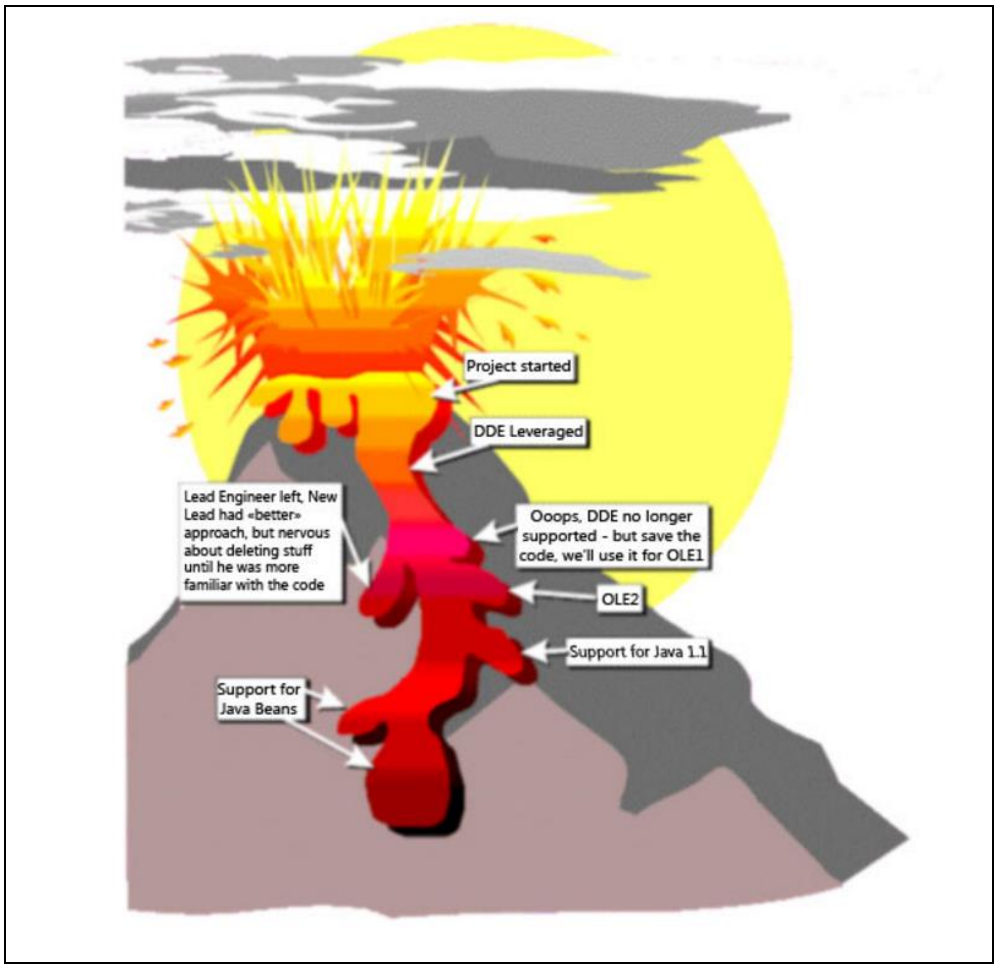
3.1.3 Lava Flow

“Dead Code” olarak da bilinen bu karşıt kalıp genellikle araştırma olarak başlamış fakat üretime kadar devam etmiş yazılım sistemlerinde görülmektedir. Bu karşıt kalıp ana hattan dağılan ve sonunda katılaşıp hareketsiz hale gelen lav akıntılarına benzetilmektedir. Dağılan bu yan akıntılar genellikle bir işlevi olmayan, ne için kodlandığı unutulmuş ölü kod kümeleri olmaktadır.

Bu durum proje henüz araştırma aşamasındayken geliştiricilerin bir hedefi başarabilmek için birkaç yol denemelerinden, kodladıkları bu denemeleri sonradan silmediklerinden ve herhangi bir dokümantasyonun da yapılmamasından kaynaklanır. Bunun sonucunda projede genel işleyişle ilgisi olmayan, okunabilirliği düşüren ve önemli olabileceği düşüncesiyle silinemeyen kod parçaları oluşmaya başlayacaktır. Hakkında çok az şey bilinen bu tarz kod parçalarını, projede uzun yıllar görev yapmış geliştiriciler çözebilseler de kodun üzerinde yapılacak herhangi bir değişiklik ya da kodu devre dışı bırakmak kritik sonuçlar doğurabileceğinden, genellikle kimse bu müdahaleyi yapmak istemeyecektir.

Lava Flow karşıt kalıbı içeren kodların analiz edilmesi, test edilmesi ve doğrulanması pahalı süreçlerdir ve çoğu zaman harcanan efora göre vakit kaybıdır. Bu kod parçalarının belleğe yüklenme işlemi maliyetlidir, önemli derecede kaynak harcayabilirler ve performansı olumsuz olarak etkilerler. Nesne yönelimli programlamanın avantajlarından olan modüllere ayırma ve yeniden kullanma gibi süreçleri imkânsız kılar.

Herhangi bir açıklama yapılmadan yoruma alınmış kod parçaları, kullanılmayan ve öylece bırakılmış kodlar, işlevsiz ve hiçbir yerde kullanılmayan metotlar ve alanlar, dokümante edilmemiş karmaşık ve anlaşılması güç metotlar ya da sınıflar Lava Flow karşıt kalıbının oluşmasında önemli rol oynayan semptomlardandır.



Şekil 3. 3 Lava Flow karşıt kalıp betimlemesi [29]

Lava Flow karşıt kalıbına önlem almanın tek bir kesin yolu vardır. Yazılımdaki mimari değişimler geliştirme safhası devam ederken yapılmamalıdır. Geliştirme devam ederken yapılan mimari değişimler bazı kod parçalarının işlevini kaybetmesine ve ölü kodlar oluşmasına sebep olacaktır. Kod geliştirme işlemi stabil bir mimari üzerine yapılmalıdır. Geliştirme işlemi devam ederken mimari bir değişikliğe ihtiyaç olduğunda, öncelikle geliştirme durdurulmalı ve ertelenmelidir.

Literatürdeki çoğu çalışma Lava Flow karşıt kalıbını incelememiştir ve tespitiyle ilgili herhangi bir aydınlatıcı metrik bilgisine rastlanmamıştır. Çalışmamızda bu karşıt kalıbı tespit etmek için 3 faktör önerdik. Bu faktörler şu şekildedir:

- ICU (Is class used?)
- PMS (Percentage of methods suspicious)
- PFS (Percentage of fields suspicious)

ICU

İncelenen sınıfın kullanılıp kullanılmadığını gösteren boolean bir değerdir. Kullanılmadığından şüphelenilen sınıflar bir karşıt kalıp adayı olarak işaretlenirler. Aynı zamanda ileride de değineceğimiz filtreleme mekanizmasında sonuçlara uygulanacak mantıksal çıkarımlar için de kullanılır. Örneğin: Bir sınıf herhangi bir objesi yoluyla ya da statik olarak kullanıldıysa, en az bir metodu kullanılmış olmalıdır. Bu kuralı çiğneyen tutarsız sonuçlar filtreleme mekanizmasında elenebilmelidir.

PMS

Bir sınıftaki metotlardan yüzde olarak ne kadarının kullanılmadığından şüphelenildiğini gösteren değerdir. Bu değer şu şekilde hesaplanmaktadır:

(Kullanılmadığından şüphelenilen metot sayısı / Toplam metot sayısı) * 100

İncelenen sınıfın bir metodu aşağıdakilerden hiçbirine uymuyorsa şüphelidir:

- Lokal olarak deklare edildiği sınıfta kullanılmıştır.
- Başka bir sınıf içerisinde ait olduğu sınıfın bir objesi vasıtası ile kullanılmıştır.
- Statik olarak “ClassName.methodName()” şeklinde kullanılmıştır.
- Bir veri sınıfının “Setter” veya “Getter” metodudur.
- İlklendirilirken “upcasting” ya da “downcasting” yapılarak oluşturulan obje üzerinden kullanılmıştır.

PFS

Bir sınıftaki alanlardan yüzde olarak ne kadarının kullanılmadığından şüphelenildiğini gösteren değerdir. Bu değer şu şekilde hesaplanmaktadır:

(Kullanılmadığından şüphelenilen alan sayısı / Toplam alan sayısı) * 100

İncelenen sınıfın bir alanının sadece lokal olarak kullanılıp kullanılmadığına bakılmaktadır. Yani bir alan şu tek maddeye uygun değilse şüpheli olarak işaretlenecektir:

- Alan lokal olarak deklare edildiği sınıfta kullanılmıştır.

PMS ve PFS metriklerinin sınır değerleri referans projelerin ve incelenen projenin aykırı değerleri filtrelendikten sonraki ortalamalarına göre belirlenmiştir. Lava Flow karşıt kalıbı için belirlediğimiz sınır değerleri aşağıdaki tabloda yer almaktadır:

Çizelge 3. 3 Lava Flow karşıt kalıbı tespitinde kullanılan metriklerin sınır değerleri

	İyi	**Sıradan**	**Kötü**
PMS	0-10%	10-20%	>20%
PFS	0-6%	6-12%	>12%

3.2 Tespit Mekanizması

Çalışmamızda karşıt kalıp sınıflarının tespit sonuçlarının elde edilmesi için 3 adımdan oluşan bir tespit mekanizması önerilmektedir.

3.2.1 Adım 1: Metrik Analiz Mekanizması

Bu mekanizmanın amacı kaynak kodlarını ve binary dosyalarını analiz ederek bazı kod metriklerini ve CK metriklerini çıkarmak ve bunları anlamlandırmaktır. Çalışmamızda metrik analiz mekanizması Blob ve Swiss Army Knife karşıt kalıpları için kullanılmıştır. LCOM, RFC, NAM, NADC, OPI ve TSC metrikleri bu mekanizma kullanılarak hesaplanmaktadır. Bu metrikler hesaplandıktan sonra önceden belirlenen sınır değerlere göre veriler anlamlandırılır. Ancak bu mekanizmadan çıkan sonuçlar filtreleme mekanizmasına girmeden önce henüz yarı anlamlandırılmış haldedir.

CKJM [22] ve JavaParser [30] kütüphaneleri yukarıda sayılan metriklerin elde edilmesi sürecinde kullanılmıştır. Deklare edilmiş metotlar, alanlar, importlar gibi basit değerlerin elde edilmesinden sonra dönüştürücü algoritmalar çalışarak ham veriyi işlerler ve sonuç olarak yarı anlamlandırılmış veri üretirler. LCOM ve RFC metrikleri için CKJM kütüphanesi kullanılmıştır. NAM, NADC, OPI ve TSC metrikleri için ise JavaParser kütüphanesi ile elde edilen değerler dönüştürücü algoritmalara tabi tutulmuşlardır.

3.2.2 Adım 2: Statik Kod Analiz Mekanizması

Bu kod mekanizması sadece Lava Flow karşıt kalıbı için kullanılmıştır. Tamamının ya da bir kısmının kullanılıp kullanılmadığından şüphelenilen sınıfların tespit etmeyi amaçlamaktadır. Kendi içerisinde bir dosya gezgini süreci bulunmaktadır ve bu süreçte tüm paketlerdeki tüm sınıflar birbirleri ile karşılaştırılır. Ölü kodların tespit edilebilmesi bazı kurallar belirlenmiştir ve bu kurallara uyan eşleşmeler kullanılmış olarak işaretlenecektir. Uymayan kısımlar ise ölü kod adayı olarak ortaya çıkacaktır. Bu süreçlerle teme olarak 3 adım gerçekleştirilmeye çalışılmaktadır.

- Bir sınıfın, proje sınırları içerisinde bir objesi oluşturularak ya da statik olarak kullanılıp kullanılmadığını tespit etmek.
- Deklare edildiği sınıf içerisinde ya da tüm proje içerisinde kullanılmadığından şüphelenilen metotları tespit etmek.
- Bir alanın deklare edildiği kapsam içinde lokal olarak kullanılıp kullanılmadığını tespit etmek.

Bu adımların hepsinin kendi algoritmaları bulunmaktadır ve bu algoritmalar çalışmamızın başında belirlenen bazı kurallara göre çalışmaktadır. Bu kurallar kullanım bilgilerinin elde edilmesini sağlayan bire bir kod eşleşmelerini vermektedir. İncelenen karşıt kalıplar kısmında da belirtildiği gibi ICU, PMS ve PFS metrikleri bu kuralların kaynak koda uygulanması ile elde edilmektedir.

Bu kuralların kaynak koda uygulanması ile edilen veriler hala işlenmemiş halde basit değerlerdir. Bu verilere dönüştürücü bazı algoritmalar uygulanarak yarı anlamlı veriler üretilir.

3.2.3 Adım 3: Filtreleme Mekanizması

1. adım ve 2. adımda üretilen yarı anlamlı veriler bu mekanizmanın girdileri olarak analiz edilirler. Yarı anlamlı verileri işleyerek sınıfların karşıt kalıp olup olmamasına dair ve son kullanıcının anlayabileceği anlamlı verileri üretmektir. Bu mekanizma sayesinde şüpheli karşıt kalıp sınıflarının daha isabetli olarak tespit edilebilmesi sağlanmaktadır. Filtreleme mekanizmasının temel amacı geliştiricilerin kodlama stillerindeki farklılıklar,

incelenen projenin büyüklüğü, kapsamı ve kullanılan programlama dilinin özellikleri gibi tespit sonuçlarını olumsuz etkileyebilecek faktörleri elimine etmektir. Tüm bu faktörler final sonuçlarını yorumlayacak olanları yanlış yönlendirebilir. Tespit işleminde bazı karşıt kalıplar sistem tarafından ıskalanabilir. Aynı şekilde bazı normal sınıflar da karşıt kalıp olarak işaretlenebilir. Bu istenmeyen durumları önlemek ve karşıt kalıp tespit sonuçlarının doğruluğunu artırmak için bu filtreleme mekanizması geliştirilmiştir.

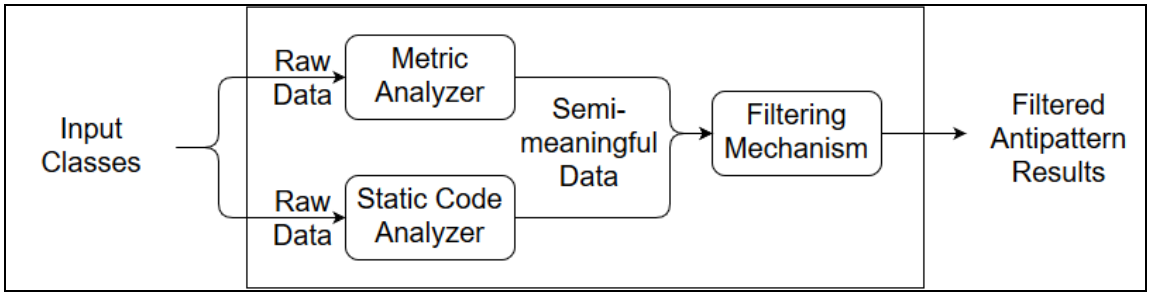
Filtreleme mekanizmasının ilk kısmı işleyiş süresinde dinamik olarak çalışır ve gerektiğinde sürece müdahale eder. Sistemde bir müdahaleye ihtiyaç olup olmadığına karar vermek için bazı mantıksal kurallar belirlenmiştir. Gerektiği takdirde müdahale edilip ilgili sınıf ya da metotlar yok sayılmalıdır veya işareti tersine çevrilmelidir. Bu kurallar şu şekildedir:

- Bir sınıf statik olarak ya da bir objesi vasıtası ile kullanılmış ise, en az bir metodu harici olarak kullanılmış olmalıdır.
- Bir veri sınıfının “setter” ve “getter” metotları doğalarından dolayı şüpheli olarak işaretlenmemelidir.
- Bir sınıf statik olarak ya da bir objesi vasıtası ile hiçbir yerde kullanılmamış ise hiçbir metodu harici olarak kullanılmış olamaz.
- Bir sınıfın içinde dâhili olarak kullanılan metotlar olabilir. Ancak sınıf hiçbir yerde kullanılmamışsa bu metotlar da sınıfın kalanıyla beraber ölü kod hükmündedir.

Filtreleme mekanizmasının ikinci kısmı sonuçlar üretildikten sonra devreye girer. Bu kısım çoğunlukla istatistiksel işlemlere dayanmaktadır. Sonuç listesinde aşağıdaki filtreleme işlemleri yapılmaktadır.

- Box plot yöntemi kullanılarak aykırı değerler tespit edilir ve sonuç listesinden elenir. Bu işlem sonuçların doğruluk oranlarını önemli ölçüde artırmaktadır.
- TSC (arayüzlerdeki toplam imza sayısı) olağandışı bir metriktir. Çünkü arayüz kullanmayan sınıfların imza sayıları da 0 olacaktır. Bu yüzden TSC değeri sadece arayüz kullanan sınıflar arasında analiz edilir.

Mekanizmalarımızın işleyiş şekilleri ve süreçlerine genel bir perspektiften bakacak olursak, sistemimizin şu şekilde bir genel işleyiş mekanizmasına sahip olduğunu söyleyebiliriz:



Şekil 3. 4 Karşıt kalıp tespit sistemimizin genel işleyiş diyagramı

BÖLÜM 4

BULGULAR

Karşıt kalıp tespit yaklaşımımızın sonuçları sınıfların karşıt kalıp olup olmadığına göre işaretlenmesi ve bu işaretlemelerin doğruluk analizlerinin yapılması şeklinde yürütülmüştür. Sistemimiz tarafından üretilen sonuçlar gerek referans projelerden elde edilen baz değerler belirlenirken, gerek incelenen hedef projelerin yada modüllerin değerleri belirlenirken aykırı değer filtrelemesinden geçirilmiştir. Geliştiriciden geliştiriciye değişen kullanım farklılıkları, programlama dilinin geniş çerçevedeki kuralları bazı değerlerin aykırı olarak çıkmasında önemli rol oynamıştır. Bu yüzden aykırı değer analizi ve filtrelemesinin sonuçların doğruluğunu önemli ölçüde artırdığı görülmüştür. Aykırı değer analiz olarak Box Plot analizi kullanılmıştır ve aykırı değer olarak belirlenenler kesinlikle ortalamalara dahil edilmemiştir.

```
|| Blob Karşıt Kalıbı LCOM Aykırı Değer Filtrelemesi ||

Kullanılan teknik = Box Plot

Sıralanmış Liste:
[1.0, 1.0, 3.0, 9.0, 9.0, 11.0, 24.0, 24.0, 25.0, 26.0, 30.0, 845.0, 1439.0]

Median = 24.0
Q1 = 3.0
Q3 = 26.0

------------------------------------------------------------
Aykırı değer çıkartıldı : 845.0
Aykırı değer çıkartıldı : 1439.0
------------------------------------------------------------

Filtrelenmiş Liste:
[1.0, 1.0, 3.0, 9.0, 9.0, 11.0, 24.0, 24.0, 25.0, 26.0, 30.0]
```

Şekil 4. 1 Blob alt veri kümesinde LCOM değerleri için aykırı değer analizi sonuçları

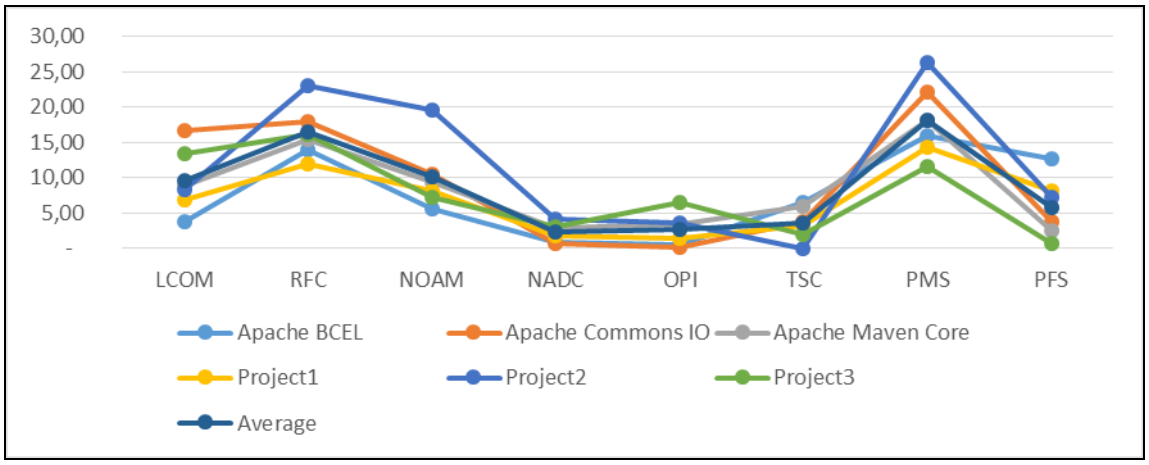
Box Plot analizi için ek bir araç kullanılmamakla birlikte projenin dahilinde hesaplanmaktadır. Örnek olarak Blob alt veri kümesinde LCOM değerleri için uygulandığında Şekil 4.1'deki gibi bir çıktı üretecektir.

4.1 Referans Alınan Veri Setinin Analiz Sonuçları

Karşıt kalıpların tespitinde referans olarak baz aldığımız 6 proje üzerinde önerdiğimiz tespit yaklaşımı uygulanmıştır. Bu uygulamanın sonucunda elde ettiğimiz filtrelenmiş değerler, bu aşamadan sonra sistemin inceleyeceği tüm projeler, modüller veya sınıflar için kontrol değerleri olarak ele alınmıştır. Apache'nin BCEL, Commons IO ve Maven Core projeleri ve büyük bir havayolu şirketinin büyük ve orta çaplı 3 projesi sistemimiz ile analiz edilmiştir. Bu analiz sonucunda elde edilen değerler Çizelge 4.1'deki gibi gösterilmiştir ve Şekil 4.1'de de grafik olarak sunulmuştur.

Çizelge 4. 1 Referans projelerin filtrelenmiş ortalama metrik değerleri

	LCOM	**RFC**	**NAM**	**NADC**	**OPI**	**TSC**	**PMS**	**PFS**
Apache BCEL	3,85	13,94	5,67	0,84	0,45	6,48	15,93	12,76
Apache Commons IO	16,70	17,96	10,59	0,76	0,22	3,86	22,12	3,72
Apache Maven Core	8,81	15,50	9,45	2,89	3,38	5,95	18,12	2,49
Project1	6,93	12,03	8,14	1,83	1,39	3,14	14,32	8,17
Project2	8,28	23,11	19,69	4,19	3,64	0	26,39	7,21
Project3	13,37	16,10	7,20	3,00	6,51	1,97	11,53	0,60
Ortalama	9,65	16,44	10,12	2,25	2,60	3,57	18,07	5,82



Şekil 4. 2 Referans projelerin filtrelenmiş ortalama grafiği

4.2 Uygulanan Yaklaşım Sonuçları

Karşıt kalıp tespit sistemimizi üzerinde uyguladığımız veri seti 36 java sınıfından oluşmaktadır. Bu veri setinde 21 karşıt kalıp olan sınıf ve 15 normal sınıf bulunmaktadır. Her sınıf üzerinde tek bir karşıt kalıbın analizi yapılmıştır. Bu yüzden veri seti her bir karşıt kalıp için bir alt gruba ayrılmıştır. Her alt grup 5 normal sınıf içermektedir. Alt gruplardaki sınıfların dağılımı Çizelge 4.2'de gösterilmiştir:

Çizelge 4. 2 Veri setindeki sınıfların dağılımı

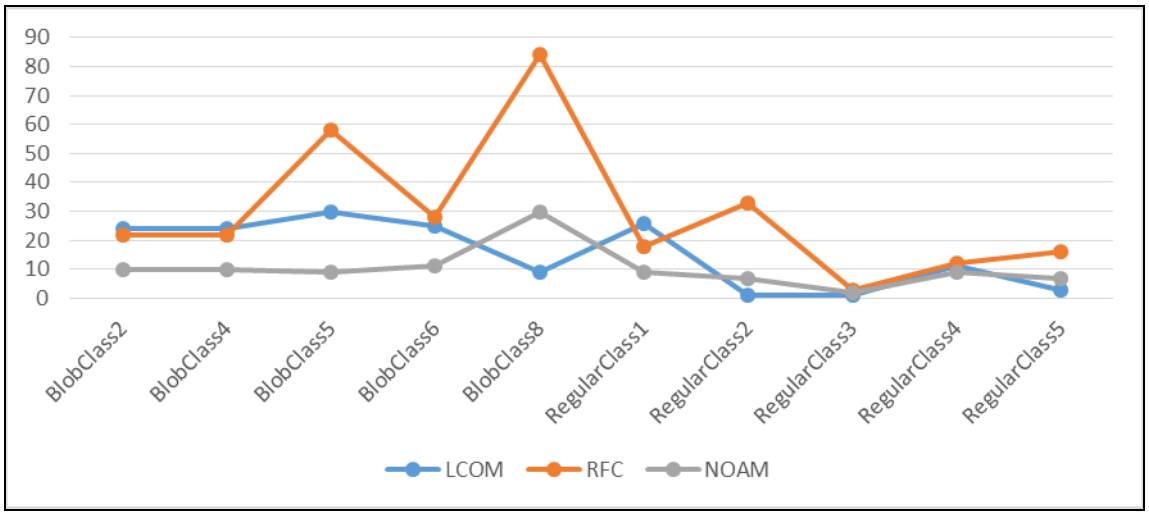
	Number of Classes
Normal	Her karşıt kalıp için 5 sınıf
Blob	8
Swiss Army Knife	6
Lava Flow	7

Tespit sistemimiz tarafından bu 36 sınıf analiz edildikten sonra sınıflar "karşıt kalıptır" ya da "değildir" olarak işaretlenmektedir. Bu karar sistemimizi anlattığımız bölümde de belirtildiği gibi birden fazla kritere göre verilmekte ve hatta daha sonrasında filtreleme yapılarak hata yüzdesi azaltılmaya çalışılmaktadır. En sonunda verilen karar ise sistemin şüphelendiği ve kullanıcının bilgilenmesi için önerdiği sonuçlardır.

İncelenen karşıt kalıpların kendi kriterlerine göre gerçekteki karşıt kalıp kimlikleri ve sistemimiz tarafından ne olarak işaretlendiği Çizelge 4.2, Çizelge 4.3, Çizelge 4.4'deki gibi gösterilmiştir. Bu çizelgelerde her karşıt kalıp tipi için sadece kendi kriteri olan metrikler ele alınmıştır. Ayrıca her bir karşıt kalıbın sonuçları grafik olarak görselleştirilmiştir (Şekil 4.2, Şekil 4.3, Şekil 4.4). Çizelgelerde verilerin tamamı gösterilmiş fakat grafiklerde sonuçların daha iyi gözlenebilmesi için aykırı değer içeren sınıflar dahil edilmemiştir.

Çizelge 4. 3 Blob karşıt kalıp tespit sonuçları

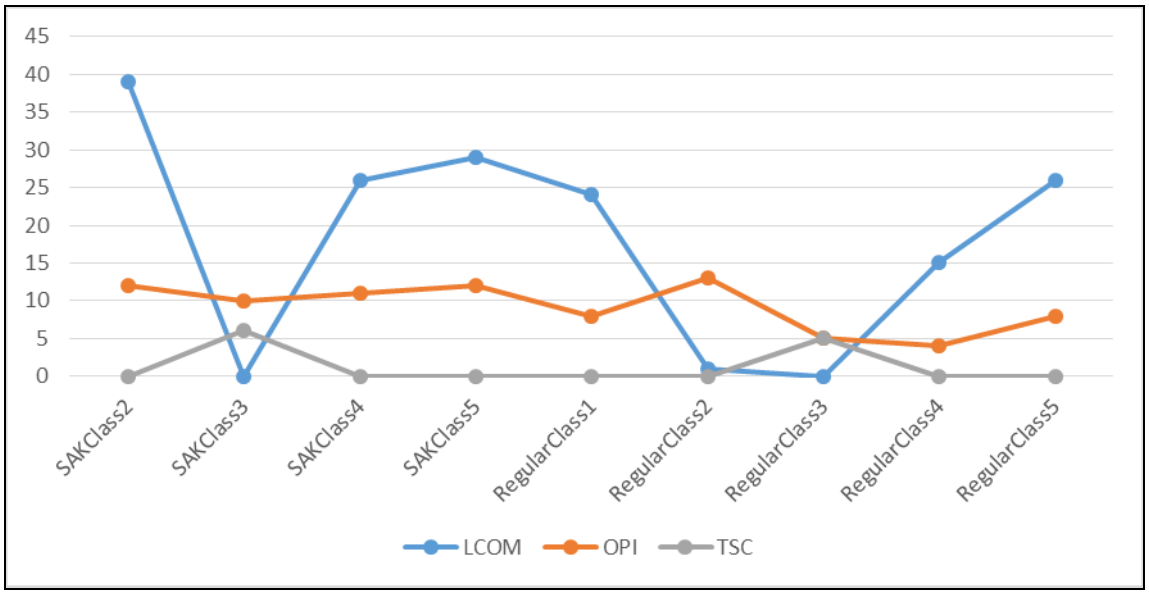
	LCOM	**RFC**	**NAM**	**NADC**	
BlobClass1	1439	105	126	47	Doğru Tespit Edildi
BlobClass2	24	22	10	3	Yanlış Tespit Edildi
BlobClass3	845	228	122	217	Doğru Tespit Edildi
BlobClass4	24	22	10	3	Doğru Tespit Edildi
BlobClass5	30	58	9	46	Doğru Tespit Edildi
BlobClass6	25	28	11	2	Doğru Tespit Edildi
BlobClass7	9	134	89	82	Doğru Tespit Edildi
BlobClass8	9	84	30	26	Doğru Tespit Edildi
RegularClass1	26	18	9	3	Yanlış Tespit Edildi
RegularClass2	1	33	7	16	Doğru Tespit Edildi
RegularClass3	1	3	2	0	Doğru Tespit Edildi
RegularClass4	11	12	9	4	Doğru Tespit Edildi
RegularClass5	3	16	7	5	Doğru Tespit Edildi



Şekil 4. 3 Blob - filtrelenmiş tespit sonuçları grafiği

Çizelge 4. 4 Swiss Army Knife karşıt kalıp tespit sonuçları

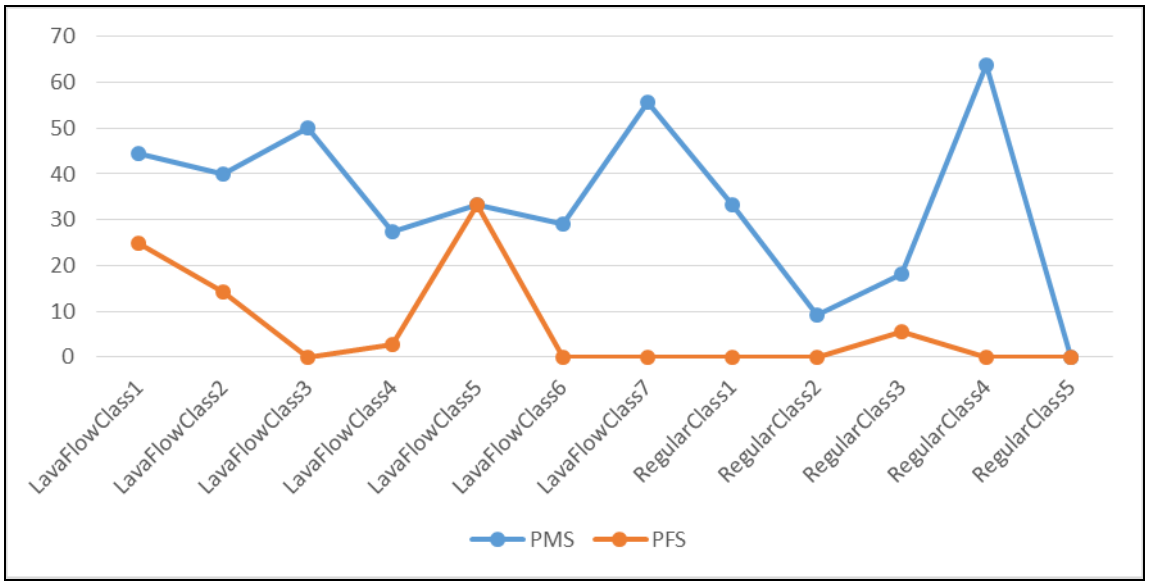
	LCOM	**OPI**	**TSC**	
SAKClass1	209	15	5	Doğru Tespit Edildi
SAKClass2	39	12	0	Doğru Tespit Edildi
SAKClass3	0	10	6	Yanlış Tespit Edildi
SAKClass4	26	11	0	Doğru Tespit Edildi
SAKClass5	29	12	0	Doğru Tespit Edildi
SAKClass6	151	24	0	Doğru Tespit Edildi
RegularClass1	24	8	0	Yanlış Tespit Edildi
RegularClass2	1	13	0	Doğru Tespit Edildi
RegularClass3	0	5	5	Doğru Tespit Edildi
RegularClass4	15	4	0	Doğru Tespit Edildi
RegularClass5	26	8	0	Yanlış Tespit Edildi



Şekil 4. 4 Swiss Army Knife - filtrelenmiş tespit sonuçları grafiği

Çizelge 4. 5 Lava Flow karşıt kalıp tespit sonuçları

	PMS	**PFS**	
LavaFlowClass1	44,44444	25	Doğru Tespit Edildi
LavaFlowClass2	40	14,28571	Doğru Tespit Edildi
LavaFlowClass3	50	0	Doğru Tespit Edildi
LavaFlowClass4	27,27273	2,702703	Yanlış Tespit Edildi
LavaFlowClass5	33,33333	33,33333	Doğru Tespit Edildi
LavaFlowClass6	29,16667	0	Yanlış Tespit Edildi
LavaFlowClass7	55,55556	0	Doğru Tespit Edildi
RegularClass1	33,33333	0	Yanlış Tespit Edildi
RegularClass2	9,090909	0	Doğru Tespit Edildi
RegularClass3	18,18182	5,405405	Doğru Tespit Edildi
RegularClass4	63,88889	0	Yanlış Tespit Edildi
RegularClass5	0	0	Doğru Tespit Edildi



Şekil 4. 5 Lava Flow - filtrelenmiş tespit sonuçları grafiği

İncelediğimiz üç karşıt kalıbın tespit sonuçları Çizelge 4.2, Çizelge 4.3 ve Çizelge 4.4'deki gibi görselleştirilmiştir. Bu çizelgelerde TP (True Positive) gerçekte karşıt kalıp olan ve bizim karşıt kalıp olarak işaretlediklerimiz anlamına gelmektedir. FP (False Positive) gerçekte karşıt kalıp olmayan fakat bizim karşıt kalıp olarak işaretlediklerimiz anlamına gelmektedir. TN (True Negative) gerçekte normal sınıf olan ve bizim normal sınıf olarak işaretlediklerimiz anlamına gelmektedir. FN (False Negative) gerçekte normal sınıf olmayan ancak bizim normal sınıf olarak işaretlediklerimiz anlamına gelmektedir.

Çizelge 4. 6 Blob karşıt kalıbı karışıklık matrisi

	Predicted: YES	**Predicted: NO**	**Total:**
Actual: YES	TP = 7	FN = 1	8
Actual: NO	FP = 1	TN = 4	5
Total:	8	5	

Çizelge 4. 7 Swiss Army Knife karşıt kalıbı karışıklık matrisi

	Predicted: YES	**Predicted: NO**	**Total:**
Actual: YES	TP = 5	FN = 1	6
Actual: NO	FP = 2	TN = 3	5
Total:	7	4	

Çizelge 4. 8 Lava Flow karşıt kalıbı karışıklık matrisi

	Predicted: YES	**Predicted: NO**	**Total:**
Actual: YES	TP = 5	FN = 2	7
Actual: NO	FP = 2	TN = 3	5
Total:	7	5	

Karşıt kalıpların tespit edilmesinde ulaşılan doğruluk yüzdeleri (2.1)'deki gibi hesaplanmaktadır.

$$\text{Doğruluk} = (TP + TN) / (TP + TN + FP + FN) \quad (2.1)$$

(2.1)'de belirtilen eşitliğe göre karşıt kalıp tespit sonuçlarımız aşağıdaki gibidir:

- Blob tespit doğruluk yüzdesi: 11 / 13 (84.6%)
- Swiss Army Knife tespit doğruluk yüzdesi: 8 / 11 (72.7%)
- Lava Flow tespit doğruluk yüzdesi: 8 / 12 (66.6%)

Lava Flow tespit doğruluğu diğerlerine göre göreceli olarak daha düşük bir değer elde etmiştir. Bunun temel sebeplerinden biri olarak metotların, objelerin ve alanların nesne yönelimli programlama dillerindeki çok çeşitli kullanımların ölü kodların tespitini negatif olarak etkilediğini görmekteyiz.

Sistemin ortalama doğruluk yüzdesi 74.6% olarak hesaplanmıştır. Tatmin edici sonuçlardan da görüldüğü gibi tespit sistemimiz verimlilik testlerini başarıyla geçmiştir ve yorumcular için anlamlı veriler üretmeyi başarmıştır. Bu anlamlı sonuçlar problematik sınıfların tespitinde ve düzeltme önerilerinde kabul edilir bir düzeyde kullanılabilmektedir.

Yaklaşımımızın diğer ilgili çalışmalar ile doğruluk karşılaştırması Çizelge 4.5'te gösterilmiştir. Literatürde diğer çalışmaların uygulama ve gereçleri erişilebilir olmadığından doğruluk değerleri bildiri ve makalelerinde belirtilen değerler olarak ele alınmıştır. İncelediğimiz üç karşıt kalıbı bir veya birden fazlasını ele almış olan çalışmalar gösterilmiştir. Lava Flow karşıt kalıbını inceleyerek tespit sonuçlarını paylaşan herhangi bir çalışma gözlenmemiştir ve bu yüzden çalışmamızın Lava Flow tespit bölümü literatürde özgün bir nitelik taşımaktadır. Bu çizelgedeki herhangi iki çalışma inceledikleri karşıt kalıp tiplerinin kesiştiği kısımlarda karşılaştırılabilir olarak göz önünde bulundurulmalıdır.

Çizelge 4. 9 İlgili çalışmaların doğruluk sonuçları

	Blob	Swiss Army Knife	Lava Flow
F. Palomba [26]	76%	N/A	N/A
A. Maiga [31]	94.49%	76.63%	N/A
N. Moha [11]	88.6%	41.1%	N/A
Our Approach	84.6%	72.7%	66.6%

BÖLÜM 5

SONUÇ VE ÖNERİLER

Önerdiğimiz yaklaşım Dhambri'nin [24] tanımladığı şu problemleri göz önünde bulundurmaktadır:

- Karşıt kalıp adaylarından oluşan büyük küme filtreleme mekanizması tarafından filtrelenmekte ve Problem 2'de belirtilen yüksek sayıdaki false positive sorununun önüne geçilmektedir.
- Tespit sistemimiz karşıt kalıpların tespit edilmesi için kural ve metrik doğrulamalarından oluşan bazı ifadelerin birleştirilmesi yöntemini kullanmaktadır. Böylelikle bu aşamada herhangi bir analist değerlendirmesine ihtiyaç duyulmamaktadır. Bu durum da Problem 3'ün çözülmesinin hedeflendiğine işaret etmektedir.
- Çözmeyi amaçladığımız son problem sınır değerlerinin belirlenmesi problemidir. Sistemimizde sınır değerleri üç faktöre göre belirlenmektedir. Bu faktörler; önceden belirlediğimiz referans projelerin filtrelenmiş ortalama değerleri, incelenen projenin aykırı değerleri elendikten sonraki ortalama değerleri ve yaygın olarak bilinen bazı metriklerin kabul edilen sınır değerleri. Böylece Problem 5'te belirtilen sınır belirleme sorunu da göz önünde bulundurulmuştur.

Çalışmamızda önerilen yaklaşım literatürde karşıt kalıp tespit sürecini tamamen otomatikleştirmeyi amaçlayan az sayıdaki yaklaşımlardandır. Tespit sürecinin hiçbir safhasında analist değerlendirmesine ihtiyaç duyulmamaktadır.

Sınır değerlerin belirlenmesi için tek kaynaktan beslenen benzer çalışmaların aksine üç faktör ele alınmıştır. Bu faktörler; referans projelerin ortalamaları, yaygın kullanılan metriklerin literatürdeki kabul edilmiş değerleri ve incelenen projenin ortalama değerleridir.

Çalışmamız literatürde Lava Flow karşıt kalıbını inceleyen ilk çalışmadır. Tespit sürecinin tamamen otomatikleştirilmesine rağmen diğer yarı otomatik çalışmalar ile kıyaslandığında tatmin edici sonuçlar elde edilmiştir.

Çalışmamızın sonraki adımlarında tespit sistemimizin kapsamına yeni karşıt kalıplar eklemeyi planlamaktayız. Eklenecek bu yeni karşıt kalıpların tespit edilebilmesi için yeni kurallar ve metrikler tanımlanacaktır. Ayrıca filtreleme mekanizmasında da bu karşıt kalıplar için gerek duyulan yeni algoritmalar geliştirilecektir.

Tespit sistemimizde uygulanabilecek olası geliştirmelerden biri de kullanıcıya düzeltme önerilerinin sunulmasıdır. Anlamlı sonuçlar göz önünde bulundurularak, sistemimiz kullanıcı ile etkileşime girerek muhtemel düzeltme seçeneklerini önerebilir. Başka bir iyileştirme; kodlama stilindeki farklılıklar gibi tespit sonuçlarını olumsuz etkileyen faktörlerin tamamının elimine edilmesi yönünde olabilir. Ayrıca kod kullanım kurallarımızı genişletmeyi ve hata oranını minimuma indirmeyi amaçlamaktayız.

KAYNAKLAR

[1] Koenig, A., (1995). "Patterns and Antipatterns", Journal of Object Oriented Programming, 8(1): 46-48.

[2] Akroyd, M., (1996). "Anti Patterns Session Notes", Object World West, 18-22 August 1996, San Jose.

[3] Din, J., AL-Badareen, A. B. ve Jusoh, Y. Y., (2012). "Antipattern Detection Approaches in Object-Oriented Design: a Literature Review", Computing and Convergence Technology (ICCCT) 7th International Conference, 3-5 December 2012, Seoul, 926-931.

[4] Travassos, G., Shull, F., Fredericks, M. ve Basili, V. R., (1999). "Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality", Proceedings of the 14th Conference on Object-Oriented Programming, Systems, Languages and Applications, 1-5 November 1999, Denver, Colorado, 47-56.

[5] Munro, M. J., (2005). "Product Metrics for Automatic Identification of Bad Smell Design Problems in Java Source-Code", Proceedings of the IEEE 11th International Software Metrics Symposium, 19-22 September, Como, 2005.

[6] Alikacem, E. ve Sahraoui, H., (2006). "Détection D'anomalies Utilisant un Langage de Description de Règle de Qualité", Actes du 12e Colloque Langages, Modèles, Objets, 22-24 March 2006, Nimes, 185–200.

[7] Ciupke, O., (1999). "Automatic Detection of Design Problems in Object-Oriented Reengineering", Proceeding of 30th Conference on Technology of Object-Oriented Languages and Systems, 1-5 August 1999, Santa Barbara, 18-32.

[8] Marinescu, R., (2004). "Detection Strategies: Metric-Based Rules for Detecting Desing Flaws", Proceedings of the 20th International Conference on Software Maintenance, 11-14 September 2004, Chicago, 350-359.

[9] Lanza, M. ve Marinescu, R., (2006). "Object-Oriented Metrics in Practice", Springer Publishing Co., New York.

[10] Van Emden, E. ve Moonen, L., (2002). "Java Quality Assurance by Detecting Code Smells", Proceedings of the 9th Working Conference on Reverse Engineering (WCRE'02), 28 October – 1 November 2002, Virginia.

[11] Moha, N., Guéhéneuc, Y. G., Duchien, L. ve Meur, A. F. L., (2010). "Decor: A Method for the Specification and Detection of Code and Design Smells", IEEE Transactions on Software Engineering, 36(1): 20-36.

[12] Simon, F., Steinbrückner, F. ve Lewerentz, C., (2001). "Metrics Based Refactoring", Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR '01), 14-16 March 2001, Lisbon, 30-38.

[13] Rao, A. ve Reddy, K. N., (2008). "Detecting Bad Smells in Object Oriented Design Using Design Change Propagation Probability Matrix", Proceedings of the International MultiConference of Engineers and Computer Scientists, 19-21 March 2008, Hong Kong.

[14] Khomh, F., Vaucher, S., Guéhéneuc, Y. G. ve Sahraoui, H., (2011). "Bdtext: A Gqm-Based Bayesian Approach for the Detection of Antipatterns", Journal of Systems and Software, 84(4): 559-572.

[15] Chidamber, S. ve Kemerer, C., (1994). "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, 20(6): 476-493.

[16] Oliveto, R., Khomh, F., Antoniol, G. ve Guéhéneuc, Y. G., (2010). "Numerical Signatures of Antipatterns: An Approach Based on B-splines", 14th European Conference on Software Maintenenance and Reengineering (CSMR), 15-18 March 2010, Madrid, 248-251.

[17] CheckStyle, Overview, http://checkstyle.sourceforge.net, 25 Nisan 2016.

[18] Hovemeyer, D. ve Pugh, W., (2004). "Finding Bugs is Easy", SIGPLAN Notices, 39(12): 92-106.

[19] JArchitect, Features, http://www.jarchitect.com , 12 Mayıs 2016.

[20] PMD, Online Documentation, http://pmd.sourceforge.net ,12 Mayıs 2016.

[21] Semmle, Documentation, http://semmle.com , 15 Mayıs 2016.

[22] Spinellis, D., (2005). "Tool Writing: A Forgotten Art?", IEEE Software, 22(4): 9-11.

[23] Charalampos, S. A., (2012). "Sonar Code Quality Testing Essentials", Pack Publishing Co., Birmingham.

[24] Dhambri, K., Sahraoui, H. ve Poulin, P., (2008). "Visual Detection of Design Anomalies", Proceedings of the 12th European Conference on Software Maintenance and Reengineering, 1-4 April 2008, Athens, 279-283.

[25] Kaur, H. ve Kaur, P. J., (2014). "A Study on Detection of Anti-Patterns in Object Oriented Systems", International Journal of Computer Applications, 93(5): 25-28.

[26] Palomba, F., Bavota, G., Oliveto, R. ve De Lucia, A., (2015). "Antipattern Detection: Methods, Challenges, and Open Issues", Advances in Computes, 95: 201-238.

[27] Ferreira, K. A. M., Bigonha, M. A. S., Bigonha, R. S., Mendes, L. F. O. ve Almeida, H. C., (2012). "Idetifying Thresholds for Object-Oriented Software Metrics", The Journal of Systems and Software, 85: 244-257.

[28] Jhans, I. K. ve Priya, V.K., (2016). "Improved Analysis of Refactoring in Forked Project to Remove the Bugs Present in the System", International Journal of Innovative Research in Science, Engineering and Technology, 5(2): 2231-2238.

[29] Sourcemaking, Lava Flow , https://sourcemaking.com/antipatterns/lava-flow, 6 Mart 2015.

[30] Viswanadha, S. ve Gesser, J. V., (2013). Java Parser, https://github.com/javaparser/javaparser , 22 Ekim 2015.

[31] Maiga, A., Ali, N., Bhattacharya N., Sabane A., Guéhéneuc, Y. G. ve Aimeur E., (2012). "SMURF: A SVM Based Incremental Anti-Pattern Detection Approach", Proceedings of the 19th Working Conf. on Reverse Engineering (WCRE), 15-18 October 2012, Kinston, Ontario, Canada, 466-475.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmed Taha ARAS

Doğum Tarihi ve Yeri : 10.07.1990 , Amasya

Yabancı Dili : İngilizce

E-posta : m.taha.aras@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Y. Lisans	Bilgisayar Mühendisliği	Yıldız Teknik Üniversitesi	-
Lisans	Yazılım Mühendisliği	Bahçeşehir Üniversitesi	2013
Lise	Fen Bilimleri	Kütahya Anadolu Öğretmen Lisesi	2008

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2015-halen	Türk Hava Yolları	Yazılım Mühendisi

YAYINLARI

Bildiri

1. Aras, M. T. ve Selçuk, Y. E., (2016). “Metric and Rule Based Automated Detection of Antipatterns in Object-Oriented Software Systems”, 7th International Conference on Computer Science and Information Technology (CSIT), 13-14 July 2016, Amman.