FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

MUSTAFA KAPDAN

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NESNEYE DAYALI PROGRAMLAMA TABANLI YAZILIMLARDA YAZILIM METRIKLERI KULLANILARAK YAPISAL KOD KLON TESPITI

MUSTAFA KAPDAN

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
YAZILIM PROGRAMI

DANIŞMAN
YRD. DOÇ. DR. MEHMET SIDDIK AKTAŞ

İSTANBUL, 2014

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

NESNEYE DAYALI PROGRAMLAMA TABANLI YAZILIMLARDA YAZILIM METRIKLERI KULLANILARAK YAPISAL KOD KLON TESPITI

Mustafa KAPDAN tarafından hazırlanan tez çalışması 15.09.2014 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. Mehmet Sıddık AKTAŞ

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Yrd. Doç. Dr. Mehmet Sıddık AKTAŞ

Yıldız Teknik Üniversitesi _____

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Yrd. Doç. Dr. Hasan SÖZER

Özyeğin Üniversitesi _____

ÖNSÖZ

Öncelikle bana bu tez ile beraber çalışma fırsatı sunan Sayın Yrd. Doç. Dr. Mehmet Sıddık AKTAŞ hocama teşekkür ederim. Tez süresince yapmış olduğu desteklerinden ve çıkarmış olduğumuz yayınlardaki katkılarından dolayı sonsuz teşekkür ederim.

Yapmış olduğumuz yayınlardaki büyük katkısı ve desteğinden dolayı Doktora adayı arkadaşım Sayın Melike YİĞİT'e teşekkürü borç bilirim. Ayrıca çalışmamda bana yardımcı olan tüm Türk Hava Yolları çalışanı olan uygulama geliştirici arkadaşlarıma teşekkür ederim.

Son olarak, hayatım boyunca benimle kederlenen ve benimle sevinen aileme manevi desteklerinden dolayı çok teşekkür ederim. Bu uzun yolda ailemin sunmuş olduğu desteğin önemini bir kez daha kavramış bulunmaktayım ve bu tezi aileme adıyorum.

Eylül, 2014

Mustafa KAPDAN

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ ix

KISALTMA LİSTESİ x

ŞEKİL LİSTESİ xi

ÇİZELGE LİSTESİ xii

ÖZET xi

ABSTRACT xiii

BÖLÜM 1

GİRİŞ 1

1.1 Literatür Özeti 1
1.1.1 Kod klonlarının ortaya çıkma nedenleri 2
1.1.2 Kod klonlarının yazılım projelerindeki büyüklüğü 3
1.1.3 Kod klonlarının sebep oldukları olumsuzluklar 4
1.2 Tezin Amacı 4
1.2.1 Organizason Yapısı 5
1.3 Hipotez 6

BÖLÜM 2

GENEL BİLGİLER VE LİTERATÜR TARAMASI 7

2.1 Genel Bilgiler 7
2.2.1 Yazılım Kod Klon Analizin Tanımları 8
2.2 Literatür Taraması 10
2.2.1 Basit Klon Tespit Yöntemleri 11
2.2.1.1 Metin Temelli Teknikler 11
2.2.1.2 Jeton Temelli Teknikler 13
2.2.1.3 Ağaç Temelli Teknikler 14

2.2.1.4 Program Bağımlı Çizge Teknikler 14
2.3.1.5 Metrik Temelli Teknikler 15
2.2.2 Yapısal Klon 16
2.2.3 Nesneye Dayalı İlişkiler 19
2.2.4 Yazılım Metrikleri 22
2.2.1.1 Chidamber ve Kemerer Metrik Kümesi 22
2.2.1.2 Lorenz ve Kidd Metrik Kümesi 23
2.2.1.3 MOOD Metrik Kümesi 25
2.3 Tezin Özgün Katkısı 25

BÖLÜM 3

ARAŞTIRMA PROBLEMİNİN TANIMI 27

3.1 Araştırma Soruları 28
3.1.1 Metrik tabanlı yapısal kod klon tespiti yöntemleri yeterli midir? 28
3.1.2 Gerekli metrik kategorileri ve metrikler nelerdir? 28
3.1.3 Kullanılabilecek hesaplama yöntemleri nelerdir? 29
3.1.4 Hangi metrikler daha fazla önemlidir? 29
3.1.5 Diğer yöntemlere göre karşılaştırma sonuçları nasıldır? 29

BÖLÜM 4

ÖNERİLEN ÇÖZÜM YÖNTEMİ 30

4.1 Yapısal Kod Klon Tespit için Önerdiğimiz Yazılım Kalite Metrikleri 31
4.1.1 Bağlaşım (Coupling) Kümesi 32
4.1.2 Karmaşıklık (Complexity) Kümesi 32
4.1.3 Kapsülleme (Encapsulation) Kümesi 33
4.1.4 Çok Biçimlilik (Polymorphism) Kümesi 33
4.1.5 Kalıtım (Inheritance) Kümesi 33
4.1.6 Boyut (Size) Kümesi 34
4.2 Yapısal Kod Klon Tespiti için Önerdiğimiz Metodoloji 34
4.3 Yapısal Kod Klon Tespitinde Metodolojinin Uygulama Adımları 37

BÖLÜM 5

UYGULAMA DETAYLARI 38

5.1 Uygulama Geliştirilirken Kullanılan Teknolojiler 39
5.2 Sonar Geliştirmeleri 39
5.2.1 Ayrıştırıcı (Parser) 40
5.2.2 Dilbilgisi (Grammar) 41
5.2.2.1 Üretici Dilbilgisi 42
5.2.2.2 Çözümlemeli Dilbilgisi 42
5.2.3 Sonar Ayrıştırıcı 42
5.2.3.1 Ayrıştırıcı İfade Dilbilgisi 43
5.2.3.2 Ayrıştırıcı İfade Dilbilgisinin Avantajları 44
5.2.3.3 Ayrıştırıcı İfade Dilbilgisinin Dezavantajları 45
5.2.3.4 Sonar Dilbilgisi Örneği 46

5.2.3.5 Sonar Çıktı Veri Yapısı 47
5.2.4 Ziyaretçi Tasarım Şablonu 48
5.2.5 Sonar Ayrıştırma Metrikleri 48
5.2.6 Sonar Byte Kod Çözümleyici 49
5.2.6.1 ASM Çerçeve Programı 50

BÖLÜM 6

YÖNTEMİN DEĞERLENDİRİLMESİ 52

6.1 Kontrol Veri Kümesi 53
6.2 Test - 1: Önerdiğimiz Metrik Kümelerinin Yapısal Kod Klon Tespitinde Başarı Oranlarının, Ağırlık Derecelerinin ve Eşik Değerlerinin Bulunması 54
6.2.1 Metrik Kümelerinin Ağırlıklarının Belirlenmesi 55
6.2.2 Eşik Değerinin Belirlenmesi 56
6.3 Test – 2: Tüm Metrik Kategorilerinin Beraber Kullanıldığında Başarı Oranlarının Bulunması 60
6.4 Test – 3: Bir Proje için Elde Edilen Eşik Değerlerinin Başka Bir Projede Kullanıldığında, Yapısal Kod Klon Tespitindeki Başarı Oranlarının Belirlenmesi 60
6.5 Test – 4: Yapısal Kod Klon Tespitinde En iyi Sonuçları Veren Metrik Kombinasyonlarının Bulunması 62
6.6 Değerlendirme 64

BÖLÜM 7

SONUÇLAR VE GELECEKTE PLANLANAN ÇALIŞMALAR 66

7.1 Araştırma Sorularına Cevaplar 66
7.1.1 Nesneye dayalı programlama ile geliştirilmiş yazılımlarda metrik tabanlı olarak yapısal klon tespiti yapılabilir mi? Metrik tabanlı yapısal kod klon tespiti yöntemleri yapısal kod klon tespitinde yeterli midir? 67
7.1.2 Nesneye dayalı programlama ile geliştirilmiş yazılımlarda metrik tabanlı yapısal klon tespiti yapılabilmesi için ölçülmesi gereken metrik kategorileri ve bu kategorilerdeki metrikler nelerdir? 67
7.1.3 Metrik ölçüm sonuçlarının sayısal veriler olduğu düşünüldüğünde, yapısal klon tespiti için kullanılabilecek hesaplama yöntemleri nelerdir? 67
7.1.4 Hangi metriklerin yapısal kod klon analizi tespitinde daha fazla önemi vardır? 68
7.1.5 Yapısal kod klon tespitinde kullanılan diğer yöntemlerle metrik tabanlı tespit yöntemleri karşılaştırıldığında nasıl sonuçlar elde edilmektedir? 68
7.2 Gelecekte Planlanan Çalışmalar 68

KAYNAKLAR 69

EK-A

METRİK KATEGORi KÜMELERİ 77
A-1 Bağlaşım Metrik Kümesi 77
A-2 Karmaşıklık Metrik Kümesi 78
A-3 Kapsülleme Metrik Kümesi 79
A-4 Çok Biçimlilik Metrik Kümesi 80
A-5 Kalıtım Metrik Kümesi 81
A-6 Boyut Metrik Kümesi 82
A-7 Sonar Yazılım Metrikleri 84
A-8 Yeni Geliştirilen Metrikler 90
A-9 Erişim Seviyesine Göre Metrikler 94

EK-B

EKRAN GÖRÜNTÜLERİ 96
ÖZGEÇMİŞ 100

KISALTMA LİSTESİ

AST	Soyut Sözdizimsel Ağaç (Abstract Syntax Tree)
JPA	Java Persistence API
PDG	Program Bağımlı Çizge (Program Dependency Graph)
PEG	Ayrıştırıcı İfade Dilbilgisi (Parser Expression Grammar)
UML	Birleşik Modelleme Dili (Unified Modelling Language)

ŞEKİL LİSTESİ

Sayfa

Şekil 2. 1 Genel klon tespit süreçleri 8
Şekil 2. 2 Kod stilinden kaynaklanan hatalı program örneği 11
Şekil 2. 3 İfade ismi değiştirilmiş hatalı program örneği 12
Şekil 2. 4 Tek satır ifadelere parantez ekleme ve çıkarma hatalı program örneği 12
Şekil 2. 5 Nesneye Dayalı Programlar için İlişki Çeşitleri 21
Şekil 4. 1 Önerilen genel çalışma yöntemi 30
Şekil 5. 1 Önerilen yapısal kod klon tespit sistemi mimarisi 38
Şekil 5. 2 Ayrıştırma Süreçleri 41
Şekil 5. 3 Java anahtar sözcükleri 43
Şekil 5. 4 Ayrıştırma karışıklığı 45
Şekil 5. 5 Sonar sınıf tanımlama kuralı 46
Şekil 5. 6 Ayrıştırıcı çıktı veri yapısı 47
Şekil 5. 7 Sonar ziyaretçi tasarım şablonu 48
Şekil 6. 1 SimPack eşik değeri – bulunan klon sayısı ilişkisi 57
Şekil 6. 2 DNSJava eşik değeri – bulunan klon sayısı ilişkisi 57
Şekil 6. 3 SimPack eşik değeri – ortak bulunan klon sayısı ilişkisi 58
Şekil 6. 4 DNSJava eşik değeri – ortak bulunan klon sayısı ilişkisi 58
Şekil 6. 5 SimPack projesi eşik değeri - doğruluk oranı ilişkisi 59
Şekil 6. 6 DNSJava projesi eşik değeri - doğruluk oranı ilişkisi 59

ÇİZELGE LİSTESİ

Sayfa

Çizelge 2. 1 Klon tiplerinin sınıflandırılması 9
Çizelge 2. 2 Klon tespiti için yapılan çalışmalar 18
Çizelge 2. 3 CK metrik kümesi 23
Çizelge 2. 4 Lorenz ve Kidd metrik kümesi 24
Çizelge 2. 5 MOOD metrik kümesi 25
Çizelge 4. 1 Önerilen metrik kümesi 32
Çizelge 5. 1 PEG operatörleri 44
Çizelge 5. 2 Sonar ayrıştırma aşamasındaki metrikleri 49
Çizelge 6. 1 Uygulama geliştirici sonuç değerleri 53
Çizelge 6. 2 Uygulama geliştiricileri arasındaki tutarlılık 54
Çizelge 6. 3 Örnek benzerlik faktörleri ve geliştirici değerlendirmesi 55
Çizelge 6. 4 Metrik kümesine göre en yüksek doğruluk – eşik değeri 56
Çizelge 6. 5 Tüm metrik kümelerinin kullandığı durumda doğruluk 60
Çizelge 6. 6 DNSJava projesinin SimPack kümesinin değerlerine göre doğruluğu 61
Çizelge 6. 7 SimPack projesinin DNSJava kümesinin değerlerine göre doğruluğu 61
Çizelge 6. 8 DNSJava projesinin SimPack tüm değerlerine göre doğruluğu 62
Çizelge 6. 9 SimPack projesinin DNSJava tüm değerlerine göre doğruluğu 62
Çizelge 6. 10 SimPack projesi için en iyi sonucu veren ilk 10 metrik kombinasyonu 63

ÖZET

NESNEYE DAYALI PROGRAMLAMA TABANLI YAZILIMLARDA YAZILIM METRIKLERI KULLANILARAK YAPISAL KOD KLON TESPİTİ

Mustafa KAPDAN

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Yrd. Doç. Dr. Mehmet Sıddık AKTAŞ

Gereksiz tekrarlanmış kodlar (klonlar) iyi dokümante edilmemiş ve bakımı zor olan kodlardır. Bu tip kodlarda, tespit edilen bir hatanın tüm tekrarlarda düzeltilmesi gerekir. Bu durum yazılım bakım maliyetlerini önemli ölçüde artırdığı gibi kodların okunabilirliği ve anlaşılabilirliği için daha fazla çaba sarf edilmesini de gerektirir.

Günümüz literatüründe kod klon problemlerini azaltmak ya da engellemek için birçok teknik önerilmiştir. Bu tekniklerin odağında basit klon ve yapısal klon kod tespiti yer almaktadır. Klon kod'lar iki ana başlık altında incelenmektedir. Yazılım içerisinde kod parçacığının benzerliğinden kaynaklanan kod tekrarlamalarına basit klon adı verilirken, sistem mimarisi içerisinde, aynı yapı ile inşa edilmiş kodlara yapısal klon denmektedir.

Basit klon tespit teknikleri, tekrarlanan kod parçacıklarına geniş bir açıdan bakamadıkları için, tasarım seviyesindeki olası tekrarlamalardan kaynaklanan yapısal kod klonlarını saptayamamaktadır. Buradaki eksikliği gidermeyi amaçlayan yapısal klon tespitleri ise, yazılımdaki üst seviye benzerliklerinin ortaya çıkartılması, yeniden kullanılabilirliğin artırılması ve yazılımın basitleştirilmesine odaklanmaktadır.

Yapısal klon tespit teknikleri literatürde önerilen basit klon tekniklerinin kullanımına dayanmaktadır. Bu tez kapsamında metrik tabanlı olarak yapısal kod klon tespiti için en uygun metrikler ve bu metriklere dayalı olarak klon tespit metodolojisi önerilmektedir.

Ortaya konan yöntembilimi gerçeklenerek, açık kaynaklı Sonar Kalite Ölçüm aracına eklenti olarak geliştirilmektedir. Yöntemin değerlendirilmesi yapılmakta ve sadece metriklere dayalı olarak yapısal kod klon tespitlerinde başarılı sonuçlar alındığı ortaya konulmaktadır.

Anahtar Kelimeler: Kod klon analizi, Yapısal kod klon analizi, Yazılım metrikleri

ABSTRACT

STRUCTURAL CODE CLONE DETECTION ON OBJECT ORIENTED SOFTWARE USING SOFTWARE METRICS

Mustafa KAPDAN

Department of Computer Engineering

MSc. Thesis

Advisor: Assist. Prof. Dr. Mehmet Sıddık AKTAŞ

Unnecessary repeated codes clones have not been well documented and are difficult to maintain. Detected bugs must be fixed in all occurences in these types of codes. Code clones may become an important problem in software development cycle and they must be fixed in all occurrences. This condition increases significantly software maintenance costs and required effort/duration for understanding the code.

Many techniques have been proposed in order to minimize or prevent the code cloning problems in contemporary literature. The main focus of these techniques is on the detection of clones. In such studies, code cloning is studied under two main categories. While the repetition of code fragments arising from the code similarity is called as simple clone in the software, codes built with the same structure are called as structural clone in the system architecture.

Simple clone detection techniques fail to determine the reasons of code repetition whether it is due to design or not, as they do not look at the code from a wider perspective for repetitive code snippets. Structural code clone detection techniques reveal the high level similarities, increase the reusability and simplify the software for filling the deficiency of simple clone detection techniques.

The structural code clone detection techniques are based on existing simple code clone techniques in the literature. In this thesis, the most suitable metrics for metric

based structural code clone detection and detection methodology based on these metrics are proposed. The proposed methodology has been implemented as a plugin for Sonar that is an open source quality management tool. In conclusion, this thesis evaluates the proposed methodology and proves that metric based structural code clone detection gives the successful results.

Keywords: Code clone analysis, Structural code clone analysis, , Software metrics

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Yazılım geliştirme süreçlerinde, var olan bir kodun kopyalanıp küçük değişiklikler yapılarak ya da hiç değiştirilmeden kullanılması sıklıkla karşılaşılan bir durumdur. Bu tarz yeniden kullanım şekline basit kod klon denilmektedir. Sistemi oluşturan mimarinin içerisinde tekrarlanan kod bulunan, soyutlandırmanın birden çok seviyesinde aynı yapı ile inşa edilmiş program yapılarına ise yapısal klon denilmektedir. Yazılım projesi içerisinde klon bulunması, yazılım sisteminin kalitesinin kötü olduğunun bir işareti olarak kabul edilmektedir [1], [2]. Kod kalitesini arttırmak, daha etkin ve kullanışlı kodlar üretebilmek için yazılım projelerinde kod klonlarının tespit edilmesi büyük önem taşımaktadır. Bu amaçla kod klonlarının ortaya çıkma nedenleri, yazılım projelerindeki büyüklükleri ve sebep oldukları olumsuzluklar detaylı olarak irdelenmelidir [3].

1.1.1 Kod klonlarının ortaya çıkma nedenleri

Kod tekrarlamalarının sebepleri çeşitlilik göstermektedir. Bu çeşitlilikler; geliştirme stratejisi, geliştirme sürecindeki çekinceler, kısıtlamaları aşma isteği, bilgi eksikliği ve yanlışlıkla kopyalama olarak 4 ana başlık altında incelenmektedir.

- **Geliştirme stratejisi:** Yazılım sisteminin kendisinden önceki sistemleri desteklemesi, farklı iki sistemin birleştirilmesi, otomatik kod üreten araçların kullanılması gibi sebeplerden dolayı, projeler içerisinde klon kod oluşumları gözlemlenmektedir.

- **Geliştirme sürecindeki çekinceler:** Geliştiriciler, kimi zaman yazacakları yeni kodun hata oluşturması çekincesinden dolayı, var olan kodu kopyala/yapıştır yöntemi ile kullanmaktadırlar. Bunun yanı sıra yazılımın mimarisini bozmamak adına da kod tekrarlamaları yapılabilmektedir. Aynı işi yapan farklı iki kod parçacığı oluşturarak, birinin çökmesi durumunda diğer klon'un cevap vermesi için program içerisinde kasıtlı olarak klon gerçekleştirilmektedir. Programın bir metodu çağırmasının maliyetli olduğu durumlarda, metod çağırılması gereken yerde tekrar kodlanabilmektedir.
- **Kısıtlamaları aşma isteği:** Programlama dilinin yeniden kullanılabilir kod mekanizmasını desteklememesi, ya da yeniden kullanılabilir kodun yazılmasının çok zaman gerektirmesi ve hataya açık olmasından dolayı geliştiriciler kod tekrarları yapabilmektedir. Bunlarla birlikte geliştiricinin sistemi tam olarak anlamaması, işin tamamlanması için zaman verilmesi ve günlük ürettiği kod satırı üzerinden performans ölçümünün yapılması gibi sebepler de proje içerisinde klon kod bulunmasına sebebiyet vermektedir.
- **Bilgi eksikliği:** Geliştiricilerin, nesneye dayalı tasarım, soyutlama ve kalıtım gibi temel yazılım bilgilerinin eksik olmasından dolayı, modüler ve yeniden kullanılabilir kod geliştirememesi ve aynı kodu farklı yerlerde kullanma yönetimini uygulamaları, kod klonlarının oluşmasına sebebiyet vermektedir.
- **Yanlışlıkla kopyalama:** Birbirinden habersiz bir şekilde birden fazla geliştiricinin aynı işi yapan benzer kod kısımlarını geliştirmesi, ya da geliştiricinin önceden karşılaştığı bir sorun için kullandığı çözüm yöntemini tekrar tekrar kullanmasından dolayı kod klonları oluşmaktadır.

1.1.2 Kod klonlarının yazılım projelerindeki büyüklüğü

Son yıllarda yapılan çalışmalar, tekrarlanmış kodlarla, endüstriyel yazılım sistemlerinde yaygın olarak karşılaşıldığını ortaya koymaktadır [4], [5], [6], [7]. Yapılan araştırmalara göre, yazılımın alanı ile alakalı olarak değişkenlik göstermekle birlikte; projelerin önemli bir kısmı klon kodlardan oluşmaktadır [8], [9]. Baker [4] yazılım projelerin %13-%20'sinin klon koddan oluşduğunu bildirmektedir. Lague [10] yazılım projelerinde

sadece fonksiyonel klonları incelemiş ve yaptığı araştırma sonucunda %6-8 arasında klon tespit etmiştir. Baxter [11] yazılım projelerinin %31'ini klon olarak kabul etmektedir. Mayrand [12] endüstriyel yazılımlar üzerinde %5-20 arasında klon tespit etmiştir. Kasper ve Godfrey [13], yazılım sisteminin %10-15'nin klon olduğunu tespit etmiştir. Tüm bu araştırmalar yazılım projelerinde kod klonlanmasının yaygın kullanılan bir yöntem olduğunu göstermektedir.

1.1.3 Kod klonlarının sebep oldukları olumsuzluklar

Büyük yazılım sistemleri içerisinde, birçok nedenden dolayı klon kod bulunmaktadır. Sistemin içerisinde klon kodun bulunması, sistemin çalışmasını etkilemez, fakat sistemin karmaşıklığını artırdığı için olası yeni geliştirilmelerin maliyetinin yükselmesine [14] ve bunun yanında daha birçok hataya sebebiyet vermektedir [4], [5], [6], [7], [12], [15]. Kod klonlarından dolayı ortaya çıkan problemleri aşağıdaki maddelerde özetlenmektedir.

- **Bakım maliyetlerinin artması:** Eğer herhangi bir klonlanmış kod içerisinde bir hata (bug) bulunursa, hatanın düzeltilmesi için öncelikle bu kodun klonu olan tüm kodlar bulunmaktadır. Daha sonra bu hata her bir klon kod üzerinde düzeltimektedir. İçerisinde klon kod bulunduran yazılım sistemlerinin, klon bulundurmayan sistemlere göre bakım maliyetlerinin yüksek olduğunu gösteren birçok çalışma vardır [4], [16]. Yazılım sisteminin tesliminden sonra yapılan herhangi bir değişikliğin maliyeti, yazılımın toplam maliyetinin %40 ile %70'i arasında değişiklik göstermektedir [17].
- **Olası hata sayısının artması:** Kopyalanıp ufak bir değişiklikle ya da değişiklik olmaksızın kullanılan kodlar hataların yayılmasında önemli bir etkiye sahiptir. Eğer kopyalanan kod, içerisinde bir hata barındıyorsa; bu hatanın kodun kopyalandığı her yerde ortaya çıkma ihtimali oluşmakta ve toplamdaki olası hata sayısını artırmaktadır [16].
- **İyileştirme maliyetlerinin artması:** Bir klon kod üzerinde yapılan iyileştirme veya zenginleştirme çalışması da bu kodun her bir klonu üzerinde de yapılmalıdır. Bu da bakım ve güncelleme kapsamındaki maliyetleri arttırmaktadır.

- **Kaynak ihtiyacının artması:** Sistemin boyutu, kopyala/yapıştır yönteminden kaynaklanan kod tekrarlamalarından dolayı artmaktadır. Artan bu boyut kimi alanlarda önemli olmaz iken, mobil sistemler gibi alanlarda yazılım ile birlikte donanımında değişimini zorunlu kılmaktadır.
- **Değişiklik taleplerine geç cevap verilmesi:** Talep edilen değişiklerin gerçekleştirilmesi için gerekli olan düzenleme ve/veya yeni kod geliştirmenin yapılabilmesi için sistemin tamamen anlaşılması gerekmektedir. Fakat klonlanmış kodlar yüzünden, sistemin ve kullanılmış kodların anlaşılması ekstra zaman almaktadır.
- **Kötü sistem mimarisi oluşması:** Nesneye dayalı tasarım, soyutlama ve kalıtım gibi temel yazılım bilgilerinin eksik olmasından dolayı kaynaklanan kod klonları sistem mimarisinin kötü olduğunu göstermektedir. Bu tarz klon kodlar, sistem içerisinde yeniden kullanılabilirliği zorlaştıracağı gibi bakım maliyetini de artırmaktadır.
- **Yeni hata ihtimalinin artması:** Önceden yazılmış olan kodların, yeni sistemlere, yeni programlama dilleri kullanılarak, taşınması aşamasında, çoğu zaman yazılım geliştiriciler tekrarlanmış kodların mantık akışını değiştirmeden kullanmaktadır. Bu kullanım yöntemi hataya açıktır ve sistem içerisinde yeni hatalar oluşaması ihtimalini artırmaktadır.

1.2 Tezin Amacı

Bölüm 1. 3'de bahsedilen olumsuzlukların ortadan kaldırılması için, yazılım sistemleri içersinde klon kodlarının tespiti büyük önem taşımaktadır. Bunun için sözdizimsel ya da işlevsel olarak birbirine benzeyen kod parçalarının ortaya çıkarılması gerekmektedir.

Günlük hayatımızda artan teknoloji kullanımı ile birlikte, bankacılıktan iletişime, iletişimden ulaşıma kadar birbirinden farklı alanlarda yazılım teknolojileri kullanılmaktadır. Artan bu yazılım teknolojileri kullanımı, yazılım kalitesinin önemini artırmaktadır. Bu bağlamda, kaliteli yazılımlar üretebilmek için, kod klonlarının tespit edilmesi ve kaldırılması önemli hale gelmektedir. Ne yazık ki, literatürde önerilen kod klon tespit yöntemleri her tip klonu tespit edememektedir.

Karmaşık, büyük çaplı projelerde, kod klonlarının tespiti; uygulama geliştirme süreci ve kullanım sürecinde önem arz etmektedir. Eğer, kod klonları tespit edilemez ise bir hata oluştuğunda hatanın çözümü, kod klon barındırmayan projelere göre daha maliyetli olmaktadır. Basit klonlar üzerinde yapılan birçok çalışma bulunmasına rağmen, yapısal klon tespiti üzerinde yapılan çalışmalar hala geliştirme aşamasındadır.

Bölüm 1.1'de açıklanan nedenlerden dolayı, yazılım projelerinde klon sayısı artmaktadır. Birçok projede basit klon tespit yöntemleri kullanılmaktadır. Ancak bu yöntemler, yapısal kod klon tespitinde yeterli değildir. Çünkü basit klon tespitleri satır, satır benzerlikleri ararken, yapısal klon tespit yöntemleri kod parçaçıklarından ziyade kodun mimari benzerliklerini araştırmaktadır. Yapılan araştırmalar [18] , uygulama içerisinde basit klonun bulunmasının aslında, var olan yapısal klonlardan kaynaklanabileceğini göstermektedir. Yapısal klonun tespit edilmesi ile, basit klonları oluşturan ana sebebin tespit edilmesi amaçlanmaktadır. Bundan dolayı, yapısal kod klon tespit yöntemlerinin araştırılıp, geliştirilmesi gerekmektedir.

Yukarıda bahsedilen nedenlerden dolayı, yazılım projelerinin kalitesini artırmak için yapısal kod klon tespit tekniklerine ihtiyaç duyulmaktadır. Bu tez çalışması ile yapısal kod klon analizi için metriklere dayalı tespit yöntemi önerilecek ve önerilen yöntem açık kaynak kodlu olarak gerçeklenerek ihtiyaç duyanların hizmetine sunulacaktır.

1.2.1 Organizasyon Yapısı

Bu tez çalışması, yapısal kod klon analizinde metrik tabanlı tekniklerin nasıl uygulanabileceğini göstermektedir. Tezin organizasyon yapısı takip eden şekildedir. Bölüm 1'de yazılım projelerinde kod klonların ortaya çıkma sebepleri, kod klonların görülme sıklığı ve ortaya çıkardığı olumsuzluklar özetlenmektedir. Bölüm 2'de yazılım kod klon analizinde kullanılan tanımlar verilmekte ve literatürde bulunan çalışmalar incelenmektedir. Yapısal kod klon tespitinde çözülmesi gereken problem ve araştırma soruları Bölüm 3'te verilmektedir. Bölüm 4'te önerilen çalışma yöntemi, çalışma yöntemi içerisinde kullanılabilecek metrikler ve hesaplama yöntemleri sunulmaktadır. Önerilen bu çalışma yönteminin gerçekleştirilmesi için uygulanan mimari yapı, kullanılan teknolojiler ve yapılan geliştirmeler Bölüm 5'te anlatılmaktadır. Bölüm 6'da önerilen yöntemi gerçekleştirmek için geliştirilen uygulama kullanılarak, önerilen

yöntemin doğruluk testleri yapılmaktadır. Bölüm 7'de elde edilen sonuçlar ışığında Bölüm 3'de verilen araştırma sorularına cevaplar verilmekte ve gelecekte yapılması planlanan çalışmalar anlatılmaktadır. Bölüm 7'den sonra Ek A ve Ek B olmak üzere iki adet ek bölüm mevcuttur. Ek A'da önerilen yöntemde kullanılan metrik kümelerinin her bir metrik elemanı gösterilirken, Ek B'de ekran görüntüleri gösterilmektedir.

1.3 Hipotez

Yapısal kod klon tespiti için literatürde yapılan çalışmalar, ihtiyaçları karşılayacak kadar olgunluğa ulaşmamıştır. Nesneye dayalı programlama dilleri için nesneler arasındaki ilişkilerden ortaya çıkan metrik kümeleri kullanılarak yapısal kod klon tespiti yapılabilir. Nesneler arasındaki metriklerden yola çıkıldığı için, bilinen diğer yapısal kod klon tespit yöntemlerinin hataya düştüğü durumlarda başarı ile sonuç verebilir. Önerilen bu metrik tabanlı yapısal kod klon tespiti açık kaynak kodlu olarak geliştirilerek, herkesin kullanımına açılabilir.

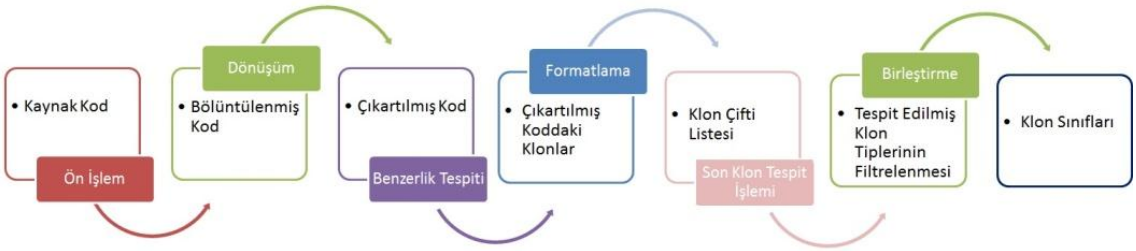
BÖLÜM 2

GENEL BİLGİLER VE LİTERATÜR TARAMASI

2.1 Genel Bilgiler

Kod klonlar üzerinde Yazılım Mühendisliği alanında geniş kapsamlı araştırmalar yapılmaktadır [3], [19], [20], [21]. Literatüründe kod klon problemlerini azaltmak ya da engellemek için birçok teknik önerilmektedir [22], [23], [24], [25]. Bu araştırmalarda özellikle yapısal kod klon tespiti irdelenmektedir. Bu bağlamda, literatür taramasında kod klonlarının tespit yöntemleri üzerine yoğunlaşılmıştır.

Bir yazılım projesini oluşturan kaynak dosyalarının metinleri içerisindeki, birbirine yüksek oranda benzeyen kod parçaları klon algılayıcıları aracılığıyla yakalanabilinmektedir. Klon algılayıcıları, sistemi oluşturan kaynak dosyaları içerisinde birbirine yüksek oranda benzeyen kod parçalarını bulmaya çalışmaktadırlar. Fakat buradaki temel sorun, hangi kod parçacığının birden çok kez kullanıldığının bilinmemesidir. Bundan dolayı her bir kod parçacığı, diğer tüm parçacıklar ile tekrar tekrar karşılaştırılmalıdır. Bu karşılaştırma oldukça yüksek bir hesaplama maliyetine neden olmaktadır. Bu araştırma maliyetini düşürmek için bazı teknikler geliştirilmiştir. Bu tekniklerin temel olarak akış diyagramları Şekil 2. 1'de gösterilmektedir.



Şekil 2. 1 Genel klon tespit süreçleri

2.1.1 Yazılım Kod Klon Analizin Tanımları

Genel yazılım kod klon terminoloji terimleri Çizelge 1.1'de özetlenmekte ve açıklamaları aşağıda verilmektedir. Bu terminoloji Kod Parçacığı, Kod Klon, Klon Çifti ve Klon Sınıfı terimlerini içermektedir.

Kod Parçacığı: Kod parçacığı, içerisinde yorum barındıran veya barındırmayan sıralı kod satırlarıdır. Kod parçacıkları, kod içerisindeki herhangi bir sıralı ifade olabileceği gibi, fonksiyon tanımlaması da olabilir. Kod parçacıkları bulunduğu orijinal dosyanın ismi, dosyadaki başlangıç satırının numarası ve bitiş numarası ile gösterilmektedir.

Kod Klon: Bir kod parçacığı, başka bir kod parçacığı ile önceden tanımlanan benzerlik fonksiyonuna göre benzerlik gösteriyorsa, bu kod parçacığına, benzediği kod parçacığının klonu denilmektedir.

Klon Çifti: Program içerisinde, sadece birbirlerine benzeyen iki kod parçasının oluşturduğu çifte klon çifti denilmektedir. Bu kod parçası, birkaç satırdan oluşabileceği gibi tüm bir sınıf da olabilmektedir.

Klon Sınıfı: Birbirine benzeyen kod parçacıklarının oluşturduğu kümeye kod klon sınıfı denilmektedir. Kümedeki her kod parçacığı, geriye kalan diğer kod parçacıklarına benzemektedir.

Klon Tipleri: Kod Klon Tipleri basit klonlar ve yapısal klonlar ana başlıkları altında incelenmektedir. Basit klon tipinde, kod parçacıkları arasındaki benzerlikler programın metni bazında olabileceği gibi metinden bağımsız olarak, işlevsel bazlı da

olabilmektedir. Bu benzerlikler temel olarak 4 ana başlık altında aşağıda belirtildiği gibi incelenmektedir:

- **Tip-1:** Yorumlar, boş satırlar ve kod düzeni haricinde kalan kod parçacıklarının metinsel olarak birbirinin tamamen aynı olmasıdır.
- **Tip-2:** Tip-2, Tip-1'den farklı olarak, program içerisinde tanımlanan ifadelerin söz dizimsel olarak benzer olup olmadığını değerlendirmektedir. Program içerisinde söz dizimsel olarak aynı olan ifadeleri yakalamaktadır.
- **Tip-3:** Programdan kopyalanan kod parçacığının içerisindeki; ifade isimlerinin ve tiplerinin değiştirilmesinin yanı sıra, söz dizim içerisine fazladan ifade ekleme/çıkarma yapılmasıyla oluşan klon tipidir.
- **Tip-4:** Aynı sonucu dönen fakat metinsel ve söz dizimsel olarak birbirinden tamamen farklı olan iki veya daha fazla kod parçacığı bu klon tipini oluşturmaktadır.

Yukarıda bahsedilen klon tipleri daha çok kod parçacıkları üzerine çalışmaktadır. Fakat bu kod parçacıkları arasındaki benzerlikler, üst seviyedeki mimari benzerlikten dolayı da kaynaklanabilmektedir. Yapısal klon, benzerlik seviyesine göre yukarıda bahsedilen 4 çeşit basit klon tipinden herhangi birini kapsayabilmektedir.

Çizelge 2. 1 Klon tiplerinin sınıflandırılması

Klon Çeşitleri	**Basit Klon**		**Yapısal Klon**
	Metin Bazlı Klon	**İşlevsel Klon**	
TİP-1	✓	×	✓
TİP-2	✓	×	✓
TİP-3	✓	×	✓
TİP-4	×	✓	✓

Çizelge 2. 1'de klon türlerinin hangi klon çeşitlerini içerip hangilerini içermediği gösterilmektedir. Bu bağlamda Yapısal Klon türünün diğer klon türlerinden farklı olarak

var olan tüm klon tiplerini içerdiği ve buna ek olarak programın genel mimarisindeki klonları da kapsadığı görülmektedir.

2.2 Literatür Taraması

Literatürde, kod klonlarının türlerine dayalı olarak birçok tespit yöntemi önerilmektedir. Basit klon tespit teknikleri, kaynak kod gösterimi ve karakteristiğine göre çeşitlilik göstermektedir. Bu çeşitlilikler 5 farklı kategori altında incelenmektedir: Metin Tabanlı, Anahtar Tabanlı, Soyut Senkronize Ağaç Tabanlı, Program Bağımlı Çizge Tabanlı ve Metrik Tabanlı. Bu basit klon tespit teknikleri, tekrarlanan kod parçacıklarına geniş bir açıdan bakmadıkları için, bu tekrarlamaların tasarım seviyesindeki olası tekrarlamalardan kaynaklanıp kaynaklanmadığını saptayamamaktadırlar. Bu nedenle, programın üst seviye benzerliklerinin ortaya çıkarılması; programın anlaşılması, ileriki geliştirilmelerden kaynaklanabilecek risklerin azaltılması, yeniden kullanılabilirliğin artırılması, programının çıktılarını ve işlevlerini değiştirmeden içyapısının yeniden düzenlenerek basitleştirilmesi için yapısal klonun tespit teknikleri gerekmektedir. Bu tezde yapısal kod klon tespit yöntemleri detaylı olarak irdelenerek, özellikle metrik tabanlı yapısal kod klon tespit yöntemleri üzerinde, teknolojideki en son gelişmeleri yansıtan bir değerlendirme yapılmaktadır.

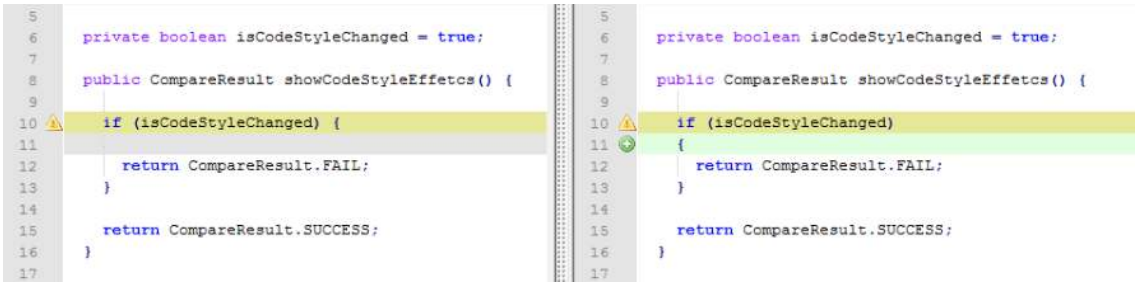
Literatürde birden fazla klon tespit aracı bulunmaktadır. Bunlardan bazıları akademik olduğu gibi, ticari olanları da vardır. Tespit araçları temel olarak veriyi dönüştürme ve benzerlik tespiti olmak üzere 2 aşamadan oluşmaktadırlar. Datayı dönüştürme aşamasında programın kaynak kodu, benzerlik algoritmasının kolay ve verimli bir şekilde çalışabileceği veri formatına çevrilmektedir. Benzerlik tespiti aşamasında ise birbirlerine benzer ya da eşit olan kod parçacıkları tespit eilmektedir. Karşılaştırmayı ve dolayısıyla sonucu etkilediği için veri formatı, tespit araçlarının sınıflandırılmasında önemli bir rol oynamaktadır. Kod parçacıklarının karşılaştırılma seviyesi ve temel olarak alınan klon tespit yöntemine göre bu araçlar birbirinden farklı tipte klon tespit edebilmektedirler.

2.2.1 Basit Klon Tespit Yöntemleri

2.2.1.1 Metin Temelli Teknikler

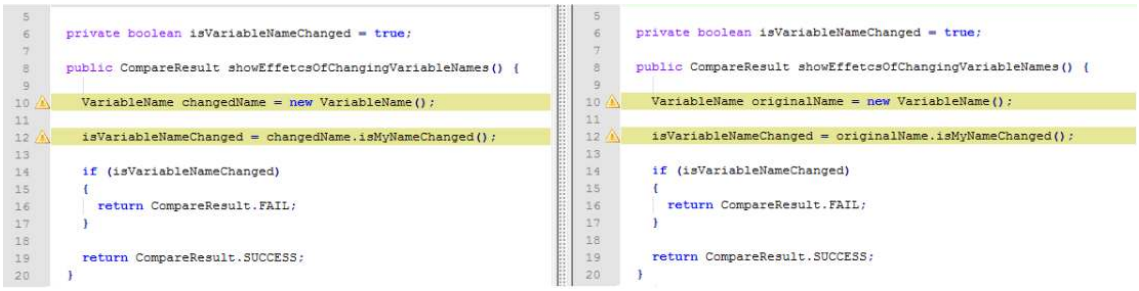
Sadece metin bazlı birçok teknik bulunmaktadır. Bu yaklaşım tekniğin de programın kaynak kodu sıralı cümleler (metinler) halinde düşünülmektedir. Herhangi 2 kod parçacığı birbiri ile karşılaştırılarak aynı metin sırası yakalanmaya çalışılmaktadır. Aynı olan maksimum sayıdaki metin satırı klon olarak belirtilmektedir. Bu teknikte kaynak kod verisi çok fazla dönüşüm işlemine tabi tutulmaz, çoğu zaman kaynak kod doğrudan karşılaştırma algoritmasında kullanılmaktadır. Fakat bazı yaklaşımlarda yorumların ve boşlukların kaldırılması gibi küçük çaplı dönüşüm işlemleri gerçekleştirilmektedir. Satır satır karşılaştırılarak yapılan bu tespit yöntemi bazı hatalara sebep olmaktadır, bu hataların bazıları aşağıda sıralanmaktadır.

- Satır sırasının kod stilinden dolayı yer değiştirmesi hataya yol açmaktadır. Örneğin, Şekil 2. 2'de gösterilen her iki kod parçacığı birbirinin aynısı olmasına rağmen klon olarak değerlendirilmemektedir.



Şekil 2. 2 Kod stilinden kaynaklanan hatalı program örneği

- İfadelerin isimlerinin değiştirilmesi tespit tekniğini hataya düşürmektedir. Örneğin, Şekil 2. 3'te gösterildiği gibi her ne kadar her iki satır aynı işlevi yapmış olsa da bu satırlar klon olarak değerlendirilmemektedir.

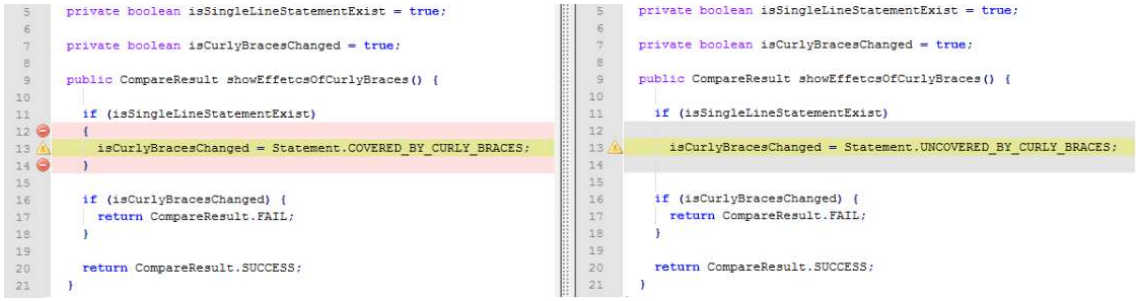


```
private boolean isVariableNameChanged = true;

public CompareResult showEffetcsOfChangingVariableNames() {

  VariableName changedName = new VariableName();

  isVariableNameChanged = changedName.isMyNameChanged();

  if (isVariableNameChanged)
  {
    return CompareResult.FAIL;
  }

  return CompareResult.SUCCESS;
}
```

```
private boolean isVariableNameChanged = true;

public CompareResult showEffetcsOfChangingVariableNames() {

  VariableName originalName = new VariableName();

  isVariableNameChanged = originalName.isMyNameChanged();

  if (isVariableNameChanged)
  {
    return CompareResult.FAIL;
  }

  return CompareResult.SUCCESS;
}
```

Şekil 2. 3 İfade ismi değiştirilmiş hatalı program örneği

- Tek satır ifadelere parantez ekleme ya da çıkarılması sonucu satır satır metin karşılaştıran bu teknik klonu yakalayamamaktadır. Bu durum Şekil 2. 4'te gösterilmektedir.



```
private boolean isSingleLineStatementExist = true;

private boolean isCurlyBracesChanged = true;

public CompareResult showEffetcsOfCurlyBraces() {

  if (isSingleLineStatementExist)
  {
    isCurlyBracesChanged = Statement.COVERED_BY_CURLY_BRACES;
  }

  if (isCurlyBracesChanged) {
    return CompareResult.FAIL;
  }

  return CompareResult.SUCCESS;
}
```

```
private boolean isSingleLineStatementExist = true;

private boolean isCurlyBracesChanged = true;

public CompareResult showEffetcsOfCurlyBraces() {

  if (isSingleLineStatementExist)

    isCurlyBracesChanged = Statement.UNCOVERED_BY_CURLY_BRACES;


  if (isCurlyBracesChanged) {
    return CompareResult.FAIL;
  }

  return CompareResult.SUCCESS;
}
```

Şekil 2. 4 Tek satır ifadelere parantez ekleme ve çıkarma hatalı program örneği

İlk gerçeklenen bazı klon algılayıcıları sözcüksel analiz tabanlıdır. Bu bağlamda Baker [26], Dup adı verilen aracı ile kaynak koddaki satır sıralarını kullanarak, satır satır klonları sözcüksel ve satır bazlı dizgi eşleştirme algoritması ile bulmaktadır. Dup sırasıyla sekmeleri, boşlukları ve yorumları kaldırır; fonksiyonlardaki belirleyicileri, değişkenleri ve tipleri değiştirir; bütün satırları birleştirerek analizin tek metin satırı içerisinde yapılmasını sağlar; her bir satırı karşılaştırma için karıştırır ve son ek ağaç algoritmasını kullanarak en uzun eşleştirilmiş klon çifti kümesini çıkarır. Dup ayrıca parametre eşleştirmelerini de tespit ederekbulunan eşleştirmeleri göstermek için rapor oluştumaktadır. Ancak bu araç kopyalanmış kodlar üzerinde araştırma ve navigasyon yapamamaktadır. Çünkü farklı kod stili ile yazılmış ve farklı değişken ismi kullanılmış kod klonlarını tespit edememektedir. Dup dizgilerdeki karakterleri karşılaştırmak yerine

sözcükleri karşılaştırarak yorumlarda ve boşluklardaki değişikliklerden dolayı oluşan problemleri önlemektedir.

Ducase ve arkadaşları başka bir metin bazlı klon tespit yaklaşımı sunmaktadır [27], [28]. Yaklaşım ayrıştırmaya dayanmamaktadır ve bundan dolayı herhangi bir dile kolayca uyum sağlayabilmektedir. Bu yöntem sırasıyla kaynak kodları okur, satırlardan diziler oluşturur, boşlukları ve yorum satırlarını kaldırır ve dizgi temelli Dinamik Model Eşleme algoritmasını kullanarak eşleşmeleri tespit eder. Çıktı olarak, klon çiftlerinin satır numaralarını ve onların içerisindeki olası silinmiş satırları vermektedir. Ancak Ducase'nin bu yönteminin hesaplama karmaşıklığın boyutlu bir girdi de O (n^2) olduğu için çok masraflıdır.

2.2.1.2 Jeton Temelli Teknikler

Bu yaklaşımda, bütün kaynak kod jeton (token) sırası halinde çevrilmektedir. Daha sonra bu sıralı jeton dizinleri, klon kodları bulmak için taranmaktadır ve orijinal kod üzerinden klonlanmış sıralı alt jeton listeleri ortaya çıkarılmaktadır. Metin tabanlı sistemlerle karşılaştırıldığında, metin tabanlı tekniklerde ortaya çıkan kodlama formatı ve standardından kaynaklanan hatalar, bu teknikle ortadan kaldırılmaktadır.

Jeton bazlı tekniklerden en gelişmişi Kamiya ve arkadaşları [6] tarafından gerçekleştirilmiş CCFinder uygulamasıdır. CCFinder, ilk olarak kaynak dosyaları sözcüksel analiz programı ile jetonlara ayrıştırır ve ayrıştırılan jetonları birleştirerek tek bir jeton sırası oluşturur. Sonrasında bu sıraya, jeton ekleyerek, silerek veya değiştirerek yazılım dilinin kurallarına göre dönüşüm yapar. Her bir tanımlayıcı değişkeni ve tipi özel jetonlarla değiştirilir. Bu tanımlayıcı değişimi ile kod parçacıklarından farklı değişken isimli klon çiftleri oluşturulur ve sonrasında ekleme tabanlı ağaç alt dizgi eşleme algoritması ile dönüşümü yapılmış klon jeton dizisinden benzer alt diziler bulunur ve bu diziler klon çifti / klon sınıfı olarak gruplanır.

CCFinder dışında jeton temelli daha birçok çalışma yapılmıştır. Baker'ın Dup [4], [26], [29] uygulaması jeton tabanlı CCFinder'a benzer şekilde sözcüksel analiz programını kullanıp kaynak kodu jetonlara ayırarak, ekleme tabanlı ağaç alt dizgi eşleme algoritması ile kod klonları bulmaktadır. Dup'ın CCFinder'dan farkı dönüşüm

yapmamasıdır. CP-Miner [8], [30] ise jeton bazlı klon tespiti için yapılmış olan başka bir çalışmadır. Bu çalışmada benzer parçalanmış ifadeler ardı ardına alt dizi madencilik (mining) teknikleri kullanılarak bulunmaktadır.

2.2.1.3 Ağaç Temelli Teknikler

Programın kaynak kodu, programlama diline bağlı olarak soyut sözdizimi ağacına (Abstract Syntax Tree (AST)) veya ayrıştırma ağacına çevrilmektedir. Çevrilen ağaç üzerinde bazı eşleştirme ağacı yöntemleri kullanılarak, alt ağaçlar aranmaktadır. Yakalanan alt ağaçlar sadece klon çifti ya da sınıfı olarak gösterilmektedir.

Soyut sözdizimli ağaç tekniklerinden en önemlisi Baxter ve arkadaşları tarafından yapılmış olan CloneDR uygulamasıdır [11]. CloneDR derleyici kullanarak açıklamalı ayrıştırılmış ağaç (annotated parse tree) oluşturmaktadır ve oluşturulan bu ağacın alt ağaçlarını metrik tabanlı özet fonksiyonlar (hash function) ile karşılaştırarak ağaçları eşleştirmektedir. Benzer ağaçların kaynak kodu klon olarak gruplanmaktadır. Ağaç temelli teknikler için yapılmış başka bir araç da Bauhaus'un ccdiml [31] klon tespit aracıdır. ccdiml CloneDR'ye benzediği gibi benzerlik metrikleri ve özet fonksiyonları kullanmayarak CloneDR'den farklılıklar göstermektedir. Ayrıca, CloneDR eş zamanlı çalışıp, tutarlı yeniden adlandırma kontrolleri yaparken, ccdiml bunları yapamamaktadır ve CloneDR soyut sözdizimi ağaçlarını direk karşılaştırma amaçlı kullanırken, ccdiml'de bu ağaçlar ara dil (intermediate language) olarak temsil edilmektedir.

2.2.1.4 Program Bağımlı Çizge (Program Dependency Graph) Teknikler

Programın akış kontrolü ile birlikte veri kontrolünü de içerisinde barındırmaktadır. Bu sayede diğer tüm tekniklerden bir adım öteye geçerek, programın iş mantığı bilgisini de barındırmaktadır. Programın çizgesi elde edildikten sonra, izomorfik alt çizge eşleme algoritmaları kullanılarak klon alt çizgeler bulunmaktadır.

Komondoor ve Horwitz [32], [33] tarafından yapılan PDG-DUP aracı program bağımlı çizge yaklaşımlarından biridir. PDG-DUP bölme programı (program slicing) ile eş yapılı alt çizgeleri bulmaktadır. Komondoor ve Horwitz ayrıca tanımlanan klonları orijinal

kodun semantiğini bozmadan birlikte gruplamak için bir yaklaşım sunmaktadır. Başka bir program bölme temelli PDG-DUP'a benzer bir çalışma ise Gallagher ve Lucas [34] tarafından gerçekleştirilmiştir. Gallagher ve Luca'nın amacı ayrıştırılan parçacıkların klon olup olmadığı analizini program bölme aracını sistemdeki bütün değişkenlere uygulayarak yapmaktır. Ancak, analizin sonucundan tam emin olamadıklarından dolayı yalnızca avantajlarını ve dezavantajlarını açıklamışlardır. Chen ve arkadaşları [35] ise kodun sözdizimsel ve veri akışını dikkate alarak kod sıkıştırma için program bağımlı çizge tekniği sunmuştur.

2.2.1.5 Metrik Temelli Teknikler

Metrik temelli yaklaşımlar kod parçacıkları için farklı metrikleri toplar ve kodu direk karşılaştırmak yerine bu metrik vektörlerini karşılaştırmaktadır. Benzer kodları tespit etmek için yazılım metriklerini kullanan birçok klon tespit tekniği vardır. İlk olarak, parmak izi fonksiyonları adı verilen yazılım metrik kümesi ile bir veya birden fazla sınıf, fonksiyon veya metod gibi sözdizimsel birimler hesaplanır ve sonrasında bu birimler üzerindeki metrik değerleri karşılaştırılarak klonlar bulunmaktadır. Birçok durumda, kaynak kod bu metriklerin hesaplanması için parçalanarak onun soyut sözdizimi ağacı / program bağımlı çizge temsili oluşturulmaktadır

Patenaude ve arkadaşları [36] gerçekleştirdikleri metrik temelli çalışmada metod içerisinden çağrılma sayısı, ifade sayısı, McCabe'in siklometik karmaşıklık ölçüsü, yerel olmayan değişkenlerdeki kullanılan tanımlama sayısı ve yerel değişken sayısı gibi metod seviyesinde birbirine çok benzer metrikleri benzer metodları bulmak için kullanmışlardır. Bu metrikleri Java programlama dili için tanımlamışlardır ve IBM Datrix aracını yazılım kalite değerlendirmesinde Java'yı desteklemesi için genişletmişlerdir.

Kanika ve arkadaşları [37] verilen Java programları için MCD Finder adı verilen metrik hesaplama aracını sunmuşlardır. MCD Finder, kaynak kodu dönüştürüp metrik hesaplaması yapmak yerine, metrik hesaplaması için Java programlarındaki bayt kodları kullanmaktadır. Bayt kod kullanmasının nedeni ise platformdan bağımsız ve yapısal olarak birleştirilmiş kod elde etmektir.

2.2.2 Yapısal Klon

Yapısal klon analizleri içerisinde, sözdizimsel olduğu kadar mantıksal analiz de yapılmaktadır. Yazılım içersindeki kod kadar, yazılımın tasarım şablonları ve kod yorumları yapısal klon analizinde kullanılmaktadır. Fakat çoğu zaman yazılımın başlangıç aşamasında çizilen tasarım şablonları, yazılımın zaman içerisindeki değişiklikleri ile birlikte geliştiriciler tarafından güncellenmemektedir. Başlangıçta çizilen tasarım şablonları, sistemin yapısını özetlemekten uzak kalmaktadır. Yapısal klonların tespiti, doğrudan alan analizi (domain analysis)'ne bağlıdır. Farklı yapıdaki yapısal klonları tespit etmek için farklı teknikler gerekmektedir. Bazı durumlarda uzman geliştiricilerin yorumlamalarına ihtiyaç duyulmaktadır, bu da tamamen otomatik bir tespit aracının geliştirilmesini zorlaştırmaktadır. Fakat yine de yapısal klon tespit araçları, basit klon tespit araçlarına göre, kullanıcıya mimari anlamda daha fazla bilgi vermektedir.

Yapısal klon tespiti için bazı araçlar geliştirilmiştir. Bu araçlardan bir tanesi Basit ve arkadaşları [38] tarafından gerçekleştirilen Clone Miner'dır. Clone Miner, ilk aşamada basit klonları tespit eder ve daha sonra tespit edilen bu basit klonları dosya ya da sınıf seviyesinde gruplayarak yapısal klon analizi yapmaktadır.

De Lucia ve arkadaşları [39] yapısal klon tekniğini kullanarak web sayfaları üzerinde klon tespit çalışmasını gerçekleştirmişlerdir. Bu çalışmada herhangi bir web sayfası verilen bir eşik değerine göre diğer sayfaya benziyorsa, bu iki sayfanın birbirinin klonu olduğu iddia edilmektedir. Levenshtein uzaklık algoritmasını kullanarak, sayfaların birbirine olan benzerliği ölçülmektedir.

Gemini ve arkadaşları [40] ise yaptıkları yapısal klon çalışmasında sistem içerisindeki dosyaların birbirlerine benzerliğini, dosya içerisinde bulunan basit klonlar üzerinden yapmışlardır. Basit klonların tespiti için CCFinder aracını kullanmışlardır. Çalışmalarında dosyalar arasındaki ikili benzerlik oranını ortaya koymuşlarıdır fakat grup benzerliğini yapamamışlardır. Birbirine benzer olan iki dosyanın birbirlerinin klonu olup olmadığı yönünde karar vermekten kaçınmışlardır.

De Lucia ve arkadaşları [39] yapısal klon tekniğine başka bir yaklaşımda bulunarak web spesifik yapısal klonlarını tespit etmişlerdir. Web sayfalarını klonun sınıfı olarak

değerlendirerek, sayfalar arasında geçişi sağlayan köprüler (hyperlink) klon tespiti için ilişki olarak değerlendirilmiştir. Çalışmalarında çizge tabanlı şablon eşleştirme algoritmalarını kullanmışlardır.

Marcus ve Maletic'in [41] öngördüğü klon tespit yaklaşımı diğerlerinden farklı bir bakış açısına sahiptir. Şimdiye kadar incelenen klon tespit araçları kod içerisinde ki tanımlayıcıların (identifier) benzerliğini göstermeyi hedeflemektedir. Bu çalışmada ters akış mantığı ile kaynak kod içerisindeki tanımlayıcı isimlerinden yola çıkılarak yapısal klon analizi yapılmaktadır. Tanımlayıcı isimleri birbirine benzeyen ve yapısal olarak birbirinin klonu olan iki ayrı sistem için, tanımlayıcı isimlerinin değişmesiyle bu çalışma yapısal klonları tespit etmede başarısız olacaktır. Tüm bu çalışmaların ortak noktası, yapısal kod klon analizi yapılırken, kodların metinsel ve semantik olarak karşılaştırılmasına dayalı olmalarıdır. Ancak, statik kod analizleri sonucunda ortaya çıkmış olan ölçüm sonuçlarından yararlanılmamaktadır. Bu da yapısal kod klon analizi için, yazılımların ayrıca bir değerlendirilmeye tutulmasını gerektirmektedir. Bu çalışma ile, metrik tabanlı yapılan statik kod analizi sonuçlarından yola çıkılarak, karşılaştırma tabanlı ayrı bir değerlendirmeye gerek duyulmadan, yapısal kod klon analizi yapmak hedeflenmiştir.

Yazılım sistemleri içerisindeki sınıfların yapısını oluşturan tasarım şablonlarına benzer olan fakat uygulama seviyesinde soyutlama sağlayan yapılara mikro şablonlar denmektedir [42]. Bu mikro tasarım şablonları üzerinden yapısal klon analizi yapılabilmektedir. Benzer şekilde Shi ve Olsson [43] tarafından gerçekleştirilen Pinot aracında da kaynak kod içerisinde tasarım şablonları aranmıştır. Bu yöntemlerin eksik yanı, sistem içerisinde bilinen şablonları aramalarıdır. Fakat yapısal klon tespiti için bilinmeyen şablonlar üzerinde de arama yapabilmelidir. Bu kategoride yer alan çalışmalarda statik kod analizleri sonucu ortaya çıkan metrik ölçümlerinden yararlanmamaktadır. Yazılımların yapısal kod klon analizi için ayrıca değerlendirilmesini gerektirmektedir. Bu yöntemlerin çalışabilmesi için yazılım sistemlerinin içinde mikro şablonların bulunması da gerekmektedir. Çizelge 2. 2'de klon tespiti için yapılan çalışmalar yapıldığı yıl, klon tespit sınıfı ve klon tespit tekniğine göre incelenmektedir.

Çizelge 2. 2 Klon tespiti için yapılan çalışmalar

Çalışma	**Yıl**	**Klon Tespit Sınıfı**	**Klon Tespit Tekniği**
Baker, Dup [15]	1992	Basit	Metin temelli / Jeton temelli
Baxter ve arkadaşları, CloneDR [10]	1998	Basit	Ağaç temelli
Chen ve arkadaşları [35]	1999	Basit	Program Bağımlı Çizge
Patenaude ve arkadaşları [36]	1999	Basit	Metrik temelli
Marcus ve Maletic [41]	2001	Basit / Yapısal	Metin temelli / Jeton temelli
Komondoor ve Horwitz, PDG-DUP [32]	2001	Basit	Program Bağımlı Çizge
Kamiya ve arkadaşları, CCFinder [6]	2002	Basit	Jeton temelli
Gallagher ve Lucas [34]	2003	Basit	Program Bağımlı Çizge
Ducase ve arkadaşları [28]	2004	Basit	Metin temelli
Basit ve arkadaşları, CloneMiner [18], [38]	2005	Basit / Yapısal	Tüm Basit Klon Teknikleri

Bu tez kapsamında yapılan çalışmanın amacı, her iki kategoride yer alan çalışmalardan farklı olarak, metrik tabanlı bir yaklaşımla, yazılımlardaki yapısal kod klon tespitini gerçekleştirmektir. Bu bağlamdatemel hedef, yaygın kullanım alanından dolayı, nesneye dayalı programlama mantığı ile geliştirilmiş yazılımlarda yapısal kod klon tespitinin yapılmasıdır.

2.2.3 Nesneye Dayalı İlişkiler

Nesneye dayalı programlamada bir nesnenin diğer bir nesneye mesaj gönderebilmesi (kullanabilmesi) için bu iki nesne arasında bir ilişki olmalıdır. Bu ilişki tipleri Sahiplik (Association), Kullanma (Dependency), Toplama (Aggregation), Meydana Gelme (Composition), Kalıtım/Miras Alma (Inheritance) şekillerinde olabilir.

Sahiplik ilişki türünde, kullanan nesne, kullanılan nesne türünden bir üyeye sahiptir. Bu ilişki sayesinde bir nesne diğer nesnenin yeteneklerini kullanabilmektedir. Sahiplik ilişkisi tek yönlü olabildiği gibi, iki yönlü de olabilmektedir. İki yönlü ilişkide her nesne diğerini kullanabilmektedir.

Kullanma ilişkisi, bir nesnenin diğer nesneyi sahiplik olmadan kullandığı durumu ifade etmektedir. Bu durumda bir nesne diğer nesneyi bir mesajın (fonskiyonun) parametresi olarak kullanabilmektedir.

Toplama ilişkisi, parça-bütün ilişkisini simgelemektedir. Bu ilişki türünde bir nesne birden fazla diğer nesne örneğine sahiptir. İlk nesne bütünü simgelerken, diğerleri parçalarını simgelemektedir. Toplama ilişkisi sahiplik ilişkisinden kavramsal anlamda daha güçlüdür. Örneğin, bir otobüs hattı en az iki durağa sahip olmalıdır ve bir hatta yeni duraklar eklemek için uyulması gereken bazı kurallar vardır.

Meydana Gelme ilişkisi daha kuvvetli bir parça-bütün ilişkisini ifade etmektedir. Meydana gelme ilişkisinde, toplamadan kuvvetli olarak, bir parça aynı anda sadece bir tek bütüne dahil olabilir. Takım ve oyuncu nesneleri ile buna örnek verilebilir. Bir oyuncu bir anda tek bir takıma üye olabilirken, bir takım çok sayıda oyuncuya sahiptir. **Kalıtım/Miras Alma ilişkisi** mevcut bir sınıftan yeni bir sınıf türetmenin yoludur. Kalıtım yolu ile üst sınıftan alt sınıfa hem üye alanlar hem de üye metodlar aktarılmaktadır.

Tanımlanan nesneye dayalı ilişkiler iki temel kategori altında toplanmaktadır. Bunlar IS-A ve HAS-A olarak tanımlanmaktadır. Bir nesnenin başka bir nesnenin alttürü olduğu durumlar nesneler arasında IS-A ilişki kategorisini oluşturmaktadır. Yukarıda tanımlanan Kalıtım/Miras Alma ilişkisi buna örnek olarak verilebilir. Bir nesnenin içerisinde başka bir nesneyi barındırdığı durumlar HAS-A ilişki kategorisini oluşturmaktadır. Bu kategoride, yukarıda tanımlanan, Sahiplik, Kullanma, Toplama ve Meydana Gelme ilişkileri yer almaktadır.

Nesneye dayalı programlarda sınıflar iki temel ana bloktan oluşur: Somut (Concrete) ve Soyut (Abstract). Bunların yanında interface sınıfları da bulunmaktadır. Bu sınıflar soyut sınıflardır fakat içerlerinde hiç uygulama (implementation) barındırmazlar. Nesneye dayalı programlama dillerinde nesneler Şekil 2. 5'de gösterildiği gibi birbirleriyle ilişkili olabilmektedirler. İlişkiler tanımlanırken nesneye dayalı programlama dillerinde yaygın olarak kullanılan anahtar kelimelerden faydalanılır. Örneğin, nesneye dayalı bir dil olan Java programlama dili için IS-A ilişkileri "extends (genişletmek)" ve "implements (uygula)" anahtar kelimeleri kullanılarak oluşturulur.

Genişletme (Extends) Var olan bir sınıfın, bir başka sınıfı miras alma yoluyla geliştirmesi durumunda kulllanılan anahtar kelimedir. Extend eden sınıf somut bir sınıf olabileceği gibi, soyut bir sınıf da olabilmektedir [44].

Gerçekleme (Implements) Bir sınıfın soyut olan başka bir sınıfı miras alma yoluyla geliştirilmesi durumunda kullanılan anahtar kelimedir. Bu noktada miras alan sınıf somut olabileceği gibi, soyut da olabilmektedir.

Şekil 2. 5 Nesneye Dayalı Programlar için İlişki Çeşitleri

Nesneler arasındaki ilişkilerin gösteriminde Birleşik Modelleme Dili (Unified Modelling Language (UML)) adındaki gösterim dili kullanılmaktadır. Bu gösterim dilinin getirmiş olduğu notasyonlar sayesinde, IS-A ve/veya HAS-A ilişkisi içerisinde bulunan nesnelerin arasındaki ilişkiler anlatılabilmektedir. Bu notasyonlar Şekil 2. 5' de gösterilmektedir.

Bu notasyonlardan biri olan gidiş gelişe uygunluk simgesi (navigability), küçük bir ok işareti ile gösterilmektedir. Bu ok işareti, hangi nesnenin hangi nesneyi kullandığı, uyguladığı veya genişlettiğini göstermektedir. Ok işaretinin çıktığı yer etkilenen nesneyi, ok işaretinin ulaştığı yer kimden etkilendiğini göstermektedir. Şekil 2. 5' de

gösterilen ilk ilişkide somut sınıf, arayüz sınıfını gerçekleştirmektedir. Somut sınıf, arayüz sınıfının soyut metodu varsa gerçekleştirmek zorundadır ve bu ilişkiden etkilenen nesne olmuştur. Gidiş gelişe uygunluk simgesi haricinde, nesneler arasındaki kullanımlar da detaylandırılabilmektedir. Birleşik Modelleme Dili dili, HAS-A ilişkisini yukarıda tanımladığımız toplama (aggregation) ve birleştirme (composition) gibi ilişkileride ifade edebilmektedir. Toplama ilişkisinde bağımlı olan nesne bağlandığı nesne hiç olmasa bile varlığını korurken, birleştirme ilişkisinde bağımlı olan nesne bağlandığı nesne olmadan varlığını devam ettirememektedir [44]. Örnek olarak, arabayı oluşturan aynalar o araba için toplama ilişkisinde bulunurken, arabanın tekerleri o araba için birleştirme ilişkisi içerisindedir. Bu ilişkiler Şekil 2. 5' de gösterilmektedir.

Bu tez kapsamında yaptığımız çalışma, nesneye dayalı ilişkilerden yola çıkarak elde edilen yazılım metrikleri ile yapısal kod klon tespiti yapmaktır. Bu bağlamda, yapısal klon tespit aşamasında nesneye dayalı ilişki ağı büyük önem taşımaktadır. İlişki ağından dolayı ortaya çıkan metrikler sayesinde, yapısal klon tespit aşamasında değerlendirilen herhangi bir kod parçası (sınıf) için, benzerlik karşılaştırmasında kullanılacak bilgiler elde edilmektedir.

2.2.4 Yazılım Metrikleri

Yazılım metriklerinin, yazılım kalitesine olan etkisi üzerine literatürde yapılan birçok çalışma mevcuttur [45], [46]. Yazılımın analiz edilmesi, tasarlanması, kodlanması ve test edilmesinde bu metriklerden yararlanılmaktadır. Birbirinden bağımsız olarak birden farklı sayıda metrik kümesi kalite ölçümleri için önerilmiştir. Chimber ve Kemerer [47], MOOD [48], [49] ve Lorenz ile Kidd'in [50] önerdiği metrik kümeleri yaygın olarak kullanılmaktadır..

2.2.4.1 Chidamber ve Kemerer Metrik Kümesi

Nesneye dayalı yazılım mühendisliği için, 1994 yılında Chidamber ve Kemerer tarafından yazılım tasarım metrikleri sunulmuştur. Literatürde bu metriklerin kullanımı ve geçerliliği üzerine yapılan çalışmalar mevcuttur [47], [51]. Nesneye dayalı

programların kalitesini ölçebilmek için birbirinden farklı özellikleri olan 6 adet metrik önermişlerdir. Çizelge 2. 3'de önerilen bu metrikler ve açıklamaları mevcuttur.

Çizelge 2. 3 CK metrik kümesi

Metrik Adı	**Tanımı**
Sınıf başına düşen fonksiyon sayısı (Weighted Methods per Class)	Sınıf içerisindeki toplam metod sayısı ya da sınıf içerisindeki metodların karmaşıklarının toplamı olarak tanımlanmaktadır. Bu değer yüksek olduğunda okunabilirlik ve yeniden kullanım azalmaktadır [52].
Cevap vermek için yapılan işlem sayısı (Response For a Class)	Sisteme gelen talep için, cevap verene kadar işletilen toplam metod sayısıdır. Ne kadar çok sayıda metod ile işlem yapıyorsa, karmaşıklık o kadar yüksek çıkmaktadır [52].
Metodlar arası etkileşim eksikliği (Lack of Cohesion of Methods)	Sınıf içerisinde, birbirinden bağımsız iki ayrı metodun bulunmasıdır. Bu gibi durumlarda sınıf iki ayrı parçaya bölünebilmektedir. Fakat metriğin ölçüm değeri ve kalitesi açısından tartışmalar bulunmaktadır [53], [54], [55].
Hiyerarşideki sırası (Depth of Inheritance Tree)	Sınıf hiyerarşisi içerisinde, herhangi bir nesne ağaçtaki konumu bildirmektedir. Yeniden kullanılabilirliği artırmaktadır.
Sahip olduğu çocuk sayısı (Number Of Children)	Herhangi bir sınıftan türeyen toplam sınıf sayısını vermektedir. Yüksek derecede türetilmiş sınıf bulunması kötü bir tasarım olarak değerlendirilmektedir [56].
Nesneler arasındaki ilişki sayısı (Coupling Between Object classes)	Diğer sınıflar ile arasındaki toplam ilişki sayısını ölçmektedir [47].

2.2.4.2 Lorenz ve Kidd Metrik Kümesi

Yazılım kalitesi için birden çok yazılım metriği önermişlerdir [50]. 1994 yılındaki çalışmalarında sınıf diyagramı üzerinde gösterilebilecek olan 11 adet metriği; sınıf boyutu (class size) metrikleri, sınıf kalıtım (class inheritance size) metrikleri ve sınıfa

özgü metrikler (class internal size) olarak 3 farklı kategoride sunmuşlardır. Çizelge 2. 4'de metrikler açıklanmaktadır.

Çizelge 2. 4 Lorenz ve Kidd metrik kümesi

Kategori	**Metrik Adı**	**Tanımı**
Sınıf boyutu metrikleri	Public metod sayısı (Number of Public Methods)	Sınıf içerisindeki toplam ortak erişim seviyeli metod sayısını bulmaktadır.
	Metod sayısı (Number of Methods)	Sınıf içerisindeki toplam metod sayısını bulmaktadır.
	Sınıfın ortak değişken sayısı (Number of public variables)	Sınıf içerisindeki toplam ortak erişim seviyeli değişken sayısını bulmaktadır.
	Sınıfın toplam değişken sayısı (Number of Variables per class)	Sınıf içerisindeki toplam değişken sayısını bulmaktadır.
	Toplam değişken sayısı (Number of Class Variables)	Yazılımdaki tanımlanan toplam değişken sayısını vermektedir.
	Toplam metod sayısı (Number of Class Methods)	Yazılımdaki tanımlanan toplam metod sayısını vermektedir.
Sınıf kalıtım metrikleri	Miras alınan metod sayısı (Number of Methods Inherited)	Kalıtım yoluyla geçen toplam metod sayısını vermektedir.
	Üzerine yazılan metod sayısı (Number of Methods Overridden)	Kalıtım yoluyla elde edilen ve üzerine yazılan (override) edilen toplam metod sayısın vermektedir.
	Yeni tanımlanan metod (Number of New Methods)	Yeni tanımlanan toplam metod sayısını vermektedir.
Sınıfa özgü metrikler	Gönderilen ortalama parametre sayısı (Average param per method)	Sınıf içerisindeki toplam parametre sayısının, toplam metod sayısına bölümü ile elde edilir.
	Özel Indeks (Specialization Index)	Üzerine yazılan toplam metod sayısının, hiyerarşideki yeri - ile çarpılıp, toplam metod sayısına bölünmesi ile elde edilir. Çoçuk sınıfların, miras aldıkları sınıfları ne ölçüde override ettiklerini ölçer.

2.2.4.3 MOOD Metrik Kümesi

Fernando Brito ve Rogerio Carpura tarafından 1994 yılında nesneye dayalı tasarım metrikleri sunulmuştur. Daha sonraki çalışmalarında, 1998 yılında bu metriklerin yeni sürümü MOOD2'yi ve otomatik metrik toplama aracı olan MOODKIT'i sunmuşlardır. MOOD metrik kümesi içerisinde 6 farklı metrik bulunmaktadır. Metrik değerleri 0 ile 1 arasında değişkenlik göstermektedir. Çizelge 2. 5'de metrikler açıklanmaktadır.

Çizelge 2. 5 MOOD metrik kümesi

Metrik Adı	**Tanımı**
Metod saklama faktörü (Method Hiding Factor)	Toplam görülmez metod sayısının, toplam sınıf sayısına bölünmesi ile ortaya çıkan metriktir.
Değişken saklama faktörü (Attribute Hiding Factor)	Toplam saklanmış olan değişken sayısının, toplam sınıf sayısına bölünmesi ile ortaya çıkan metriktir.
Metrik kalıtım faktörü (Metric Inheritance Factor)	Toplam miras alınan metod sayısının, toplam sınıf sayısına bölünmesi ile ortaya çıkan metriktir.
Değişken kalıtım faktörü (Attribute Inheritance Factor)	Toplam miras alınan değişken sayısının, toplam sınıf sayısına bölünmesi ile ortaya çıkan metriktir.
Çok biçimlilik faktörü (Polymorphism Factor)	Toplam üzerine yazılan metod sayısının, toplam miras edilen metod sayısına bölünmesi ile hesaplanmaktadır.
İlişki faktörü (Coupling Factor)	En yüksek ilişki değerine göre, herhangi bir sınıfın ilişki değerinin oranıdır.

2.2.5 Tezin Özgün Katkısı

Bu çalışmada nesneye dayalı programlama ile geliştirilmiş yazılımlar için kod klon analizinin yapılması hedeflenmektedir. Bu araştırmanın özgün katkısı, yazılım kalite metrikleri kullanılarak nesneye dayalı programlama dilleri ile yazılmış projelerde yapısal kod klonlarının tespit edilebilmesi için bir yöntem sunmaktır ve geliştirmektir. Önerilen

yöntemin açık kaynaklı bir yazılım kalite ölçüm aracına kolayca eklenebilir olması yazılım kalitesi bilimsel araştırma alanına katkı sağlamaktır. Yapılan literatür taraması çerçevesinde yapısal kod klon tespitinde metrik tabanlı yaklaşıma ait herhangi bir çözüme rastlanılmamıştır.

BÖLÜM 3

ARAŞTIRMA PROBLEMİNİN TANIMI

Bölüm 2'de verilen literatür taraması basit klon tespit yöntemleri kullanılarak, yapısal klonların tespit edilemediğini göstermektedir [18], [37], [38], [39], [41]. Yapısal klon tespit aşamalarında, basit klondan farklı olarak kod parçaçıkları yerine yapının bütününe bakılmaya çalışılmaktadır. Yapısal klon tespit edildiğinde, basit klonlar gibi her bir klon parçaçığında sorunu çözmek yerine, sorun en temelinden çözülmeye çalışılmaktadır. Ayrıca yapısal klon tespiti ile büyük ölçekli kod parçaları arasındaki benzerliklerin çıkartılması ile, programın anlaşılması kolaylaşır, yeniden kullanılabilirliği artar, güncelleme durumunda oluşabilecek olası hata riski düşürülür ve yeniden düzenleme (reengineering) yöntemleri ile projenin sadeleştirilmesi sağlanır.

Günümüzde en yaygın olarak kullanılan programlama yaklaşımı nesneye dayalı programlamadır [57]. Bu programlama yaklaşımında her şey bir nesnedir [58]. Bu yaklaşım sayesinde yazılım kaynak kodları birimlere (module) ayrıştırılmaktadır. Birimlere ayrıştırılmış olan yazılım projelerinde, Bölüm 1. 1'de bahsedilen nedenlerden dolayı kod klonlara rastlanılmaktadır.

Bu tezde nesneye dayalı programlama ile geliştirilmiş yazılımlar için yapısal kod klon analizinin nasıl yapılacağı problemi üzerinde durulmaktadır. Tez kapsamında belirlenen problem ile ilgili araştırma soruları Bölüm 3. 1'de detaylı olarak anlatılmaktadır. Bu araştırmanın özgün katkısı, yazılım kalite metriklerini kullanarak nesneye dayalı programlama dilleri ile yazılmış projelerde yapısal kod klonlarının tespit edilmesi için açık kaynaklı yazılım kalite ölçüm aracına kolayca eklenebilir bir yöntem geliştirmek ve yazılım kalitesi bilimsel araştırma alanına katkı sağlamaktır.

3.1 Araştırma Soruları

Metrik tabanlı yapısal kod klon tespit yönteminin gerçekleştirilmesi için takip eden araştırma sorularına cevapların aranması gerekmektedir.

3.1.1 Nesneye dayalı programlama ile geliştirilmiş yazılımlarda metrik tabanlı olarak yapısal klon tesbiti yapılabilir mi? Metrik tabanlı yapısal kod klon tespiti yöntemleri yapısal kod klon tespitinde yeterli midir?

Yazılım kalitesi araştırma alanı, yazılımların statik ve dinamik kod analizlerinin yapılabilmesi için birçok yazılım metriği ortaya koymaktadır. Metrikler kullanılarak yazılımlar hakkında önemli bilgiler edinilebilmektedir. Ancak, metrikler kullanılarak yazılımlarda yapısal kod klonlarının tespiti üzerinde yapılmış araştırmaların eksikliği görülmektedir. Bu araştırma ile yapısal kod klon analizinde yazılım metriklerinin kullanımının başarılı olup olmayacağı ortaya çıkarılmaktadır. Günümüzdeki yaygın kullanımı nedeniyle nesneye dayalı programlama ile geliştirilmiş yazılımlarda kod klon tespiti yapılmaktadır. Bu araştırmayla metrik tabanlı yöntemin yapısal kod klonları tespit etmede yeterli olup olmadığı irdelenmektedir..

3.1.2 Nesneye dayalı programlama ile geliştirilmiş yazılımlarda metrik tabanlı yapısal klon tesbiti yapılabilmesi için ölçülmesi gereken metrik kategorileri ve bu kategorilerdeki metrikler nelerdir?

Yazılım metrikleri bir yazılımın karmaşıklığını, bakım yapılabilirliğini, nesneye dayalı programlamaya uygunluğunu, büyüklüğünü ortaya koyabilmektedir. Yapısal kod klon tespiti yapılırken hangi metrik kategorilerilerinin kullanılabileceği, bu kategorilerde hangi metriklerin yer alabileceği bilinmemektedir. Bu araştırma ile nesneye dayalı programlarda yapısal kod klon analizlerinde kullanılabilecek metrikler deneysel yolla araştırılarak ortaya konulmaktadır.

3.1.3 Metrik ölçüm sonuçlarının sayısal veriler olduğu düşünüldüğünde, yapısal klon tesbiti için kullanılabilecek hesaplama yöntemleri nelerdir?

Yazılım metrik ölçüm değerleri, ölçümü alınan her bir sınıf için sayısal değerlerden oluşan bir vektör şeklinde düşünülebilir. Yapılan araştırmalar farklı benzerlik hesaplama

yöntemlerinin, farklı doğrulukta sonuçlar ortaya koyduğunu göstermektedir [59], [60]. Yazılım birimlerininin, yapısal kod klon tespiti için, yazılım metrik ölçüm değerlerine dayalı olarak, karşılaştırıldığında hangi benzerlik hesaplama yöntemlerinin daha doğru sonuçlar ortaya koyduğu, bu araştırma ile irdelenmektedir.

3.1.4 Hangi metriklerin yapısal kod klon analizi tespitinde daha fazla önemi vardır?

Yapısal kod klon tespitinde bazı metrik kategorileri diğerlerine göre daha önemli olabilmektedir. Bir başka deyişle, yapısal kod klon tespiti yapılırken bazı metrikler daha yüksek doğruluk ile sonuçlar verebilirken, bazı metrikler ise daha az etkin olabilmektedir. Bu çalışmayla hangi metrik kategorilerinin veya metriklerin yapısal kod klon tespitinde daha önemli olduğu ortaya konulmaktadır.

3.1.5 Yapısal kod klon tespitinde kullanılan diğer yöntemlerle metrik tabanlı tespit yöntemleri karşılaştırıldığında nasıl sonuçlar elde edilmektedir?

Yapısal kod klon tespit yöntemi ile basit klon ve kümeleme analiz sonuçlarını kullanan diğer tespit yöntemleri [18], [38] karşılaştırıldığında nasıl sonuçlar verdiği araştırılacaktır.

BÖLÜM 4

ÖNERİLEN ÇÖZÜM YÖNTEMİ

Yazılım sistemlerinde basit klonların yoğun olarak bulunması, tasarım çözümlerinin kopyalanması sonucu oluşan yüksek seviyede benzerliklerin bulunduğunu göstermektedir. Bu tasarım çözümlerinin kopyalanmasının sebepleri aşağıdaki gibi sıralanmaktadır.

- Aynı analiz şablonlarının kullanılması [61],
- Aynı tasarım şablonlarının kullanılması [62],
- Mimari seviyede kullanılan parçaların tasarımlarının benzemesi,
- Benzer sorunu çözmek için, geliştiricinin aynı mimariyi kullanması [18]

Literatür taraması incelendiğinde yapısal klonların tespitinde metrik tabanlı çalışmaların kullanılmadığı ve bu alanda bir eksiklik olduğu görülmektedir. Önerilen yöntem yapısal klonların, yazılım metrik ölçüm değerlerine dayalı olarak tespitinde kullanılmaktadır. Bu bağlamda Şekil 4. 1'de, yapısal kod klonların tespitinde önerilen çalışma yöntemi gösterilmektedir.

Şekil 4. 1 Önerilen genel çalışma yöntemi

Yapısal klon tespit yönteminde kullanılacak olan metrikler, sonucu doğrudan etkileyen faktörlerden biridir. Bu sebeple tespit yönteminde kullanılması önerilen metrikler Bölüm 4. 1'de verilmektedir.

4.1 Yapısal Kod Klon Tespit için Önerdiğimiz Yazılım Kalite Metrikleri

Yazılım projelerinin kalite ve yönetimi aşamasında boyut, tasarım, karmaşıklık gibi birbirinden farklı alanlarda tanımlanmış metrikler kullanılmaktadır. Bu metrikler arasındaki farklılıklar, kalite yönetimi yapılmak istenilen yazılım dilinin özelliğine göre değişkenlik gösterebilmektedir. Yazılım programlama yaklaşımları kendi içlerinde fonksiyonel, yordamsal (procedural), nesneye dayalı gibi tanımlamalarla alt alanlara ayrılmaktadır.

Nesneye dayalı programlama dilleri, 1980'li yıllardan itibaren ortaya çıkan bir programlama yaklaşımıdır. İlk nesneye dayalı programlama dili Smalltalk [63] dilidir. Bu programlama yaklaşımında her şey bir nesnedir [58]. Bu yaklaşım sayesinde yazılım kaynak kodları birimlere (module) ayrıştırılmaktadır. Bu birimlere ayrıştırılabilme yeteneğinden dolayı aynı anda birden fazla uygulama geliştirici birbirini beklemeden ve etkilemeden farklı parçaları geliştirebilmektedir. Günümüzde ihtiyaç duyulan kurumsal yazılım çözümlerinin hızlı olarak geliştirilmesinde yoğun olarak kullanılmaktadırlar [57]. Nesneye dayalı programlama yaklaşımının uygulama alanları ağ teknolojileri [64], internet uygulamaları [65], gömülü sistemler [66], cep telefonu uygulamaları [67] gibi birbirinden farklı alanlar olabilmektedir.

Bu tezde önerilen yöntemde nesneye dayalı yazılım kalite metrikleri kullanılmaktadır. Yazılım projelerinden bu metriklerin toplanması için yazılım kalite ve yönetim araçlarından yararlanılmaktadır. Günümüzde kullanılan birbirinden farklı yazılım kalite ve yönetim aracı bulunmaktadır [68], [69], [70], [71]. Bu araçlar arasından açık kaynaklı olması ve yaygın olarak kullanılmasından dolayı Sonar [71] aracı bu tez çalışmasında tercih edilmiştir.. Metrik ölçüm aracı olan Sonar'ın üretmiş olduğu değerlerden yola çıkarak sınıf seviyesindeki benzerlikler ortaya çıkarılmıştır ve yapısal klonlar ortaya konmaya çalışılmıştır.

Önerilen yapısal klon tespit yöntemi ile bir veya birden fazla yazılım projesi içerisinde, yapısal olarak birbirine benzeyen sınıfların tespiti yapılmaktadır. Bu tespit yönteminin gerçekleştirilmesinde 6 farklı kategoride metrik kümesi kullanılmaktadır. Çizelge 4. 1'de kullanılan metrik kümeleri ve açıklamaları gösterilmektedir.

Çizelge 4. 1 Önerilen metrik kümesi

Metrik Kümesi	**Açıklama**
Bağlaşım	Herhangi bir sınıfın başka sınıfı kullanması
Karmaşıklık	Sınıf içerisindeki karmaşıklık
Kapsülleme	Sınıf içerisindeki verinin dışarıdan erişme karşı korunması
Çok biçimlilik	Sınıfın tür değiştirebilme yeteneği
Kalıtım	Sınıfın bir başka sınıftan miras alması
Boyut	Sınıf ile alakalı olan, genel boyut bilgileri

4.1.1 Bağlaşım (Coupling) Kümesi

Herhangi iki nesne arasındaki bağımlılık ölçümüne Bağlaşım denilmektedir. Eğer A nesnesi, B nesnesinin bir methodunu veya değişkenini çağırıyor ve kendi içerisinde kullanıyorsa, A nesnesi ile B nesnesi birbirine bağımlıdır. B nesnesine bağımlı olan A nesnesi yapısal olarak klonlandığı zaman, klonlanan yeni nesne de B nesnesine bağımlı olmaktadır. Bundan dolayı nesneler arasındaki bağımlılık değeri yapısal klon araştırmalarında önem arz etmektedir. Ek A-1'de bağlaşım metrikler gösterilmektedir.

4.1.2 Karmaşıklık (Complexity) Kümesi

Karmaşıklık metrik kümesi Cyclomatic karmaşıklık olarak da adlandırılmaktadır. Program akışının fonksiyonel olarak çatallaştığı zamanlarda, değeri 1 artmaktadır. Her bir metodun minimum karmaşıklık değeri 1'dir. Herhangi bir nesnesin fazla sayıda

metodu bulunması, o nesneyi daha karmaşık hale ve hataya açık hale getirir. Ayrıca çok fazla sayıda metod bulundurması olası çoçuklarının üzerinde de etki yapar. Çok fazla sayıda metodu bulunan nesneler, yeniden kullanılabilirliği azaltmaktadır. Yapısal olarak klonlanan herhangi iki nesnenin karmaşıklık değerleri birbirlerine yakın olabilmektedir. Ek A-2'de karmaşıklık kümesinin metrikleri gösterilmektedir.

4.1.3 Kapsülleme (Encapsulation) Kümesi

Nesne içerisindeki değişken ve metodlara erişimin, doğrudan dışarıdan erişime kapatılması ve erişimin sadece ilgili nesne üzeirnden yapılması işlemine kapsülleme denilmektedir [72]. Kapsülleme yapısını kullanan büyük çaplı projelerin, kullanmayanlara göre 4 kat daha kolay değiştirilebilir ve anlaşılabilir olduğu tespit edilmiştir [73]. Yapısal olarak kopyalanan nesnelerin kapsülleme değerleri arasında ilişki bulunabilmektedir. Herhangi bir nesne hiçbir şekilde değiştirilmeden doğrudan kopyalanıp yapıştırılarak kullanıldığında, içerisinde kapsüllenen değişken ve metod sayısı aynı olduğundan değerleri de aynı olmaktadır. Samra kümesi içerisinde kullanılan tüm metrikler Ek A-3'de gösterilmektedir.

4.1.4 Çok Biçimlilik (Polymorphism) Kümesi

Çok biçimlilik, herhangi bir nesnenin birden farklı nesne çeşidine dönüşebilme yeteneğidir. A nesnesi içerisinde bulunan metodların, A'dan türeyen B nesnesi içerisinde gerçekleştirilmesine -olanak vermektedir. Örneğin, çalışanlara ödeme yapan bir sistem düşünüldüğünde, ödeme yapan sistemin öde metodu; çalışanın durumuna göre saatlik, haftalık ve aylık olarak değişik şekillerde gerçekleştirilebilir. Ödeme yapan sistem tek olmasına rağmen, her bir çalışan tipi için farklı özellikler gösterebilmektedir. Herhangi iki nesnenin çok biçimlilik değerlerinin birbirine yakın olması, iki nesnenin de ortak atadan türemiş olma ihtimalini ortaya koymaktadır. Ek A-4'de tüm metrikler gösterilmektedir.

4.1.5 Kalıtım (Inheritance) Kümesi

Nesneye dayalı yazılım dillerinin en önemli özelliklerinden birisi kalıtım hiyerarşisinin olmasıdır [74]. Kalıtım sayesinde yazılım projelerinin yeniden kullanılabilirliği

artmaktadır. Kolay anlaşılabilirliği artırdığı için yazılım karmaşıklığını azaltmaktadır. Kalıtımın kullanılmadığı durumlarda benzer metod ve değişkenlere ihtiyaç duyan her nesne kendi içerisinde tekrar yazılmaktadır. A nesnesi, B nesnesinden türetildiğinde, B nesnesi içerisindeki, A nesnesine görünür (visible) olan tüm değişken ve metodlar A nesnesine miras olarak alınmaktadır. Herhangi bir nesnenin bu miras alma sonucu ortaya çıkan ağaç yapısındaki derinliğine hiyerarşideki yeri denilmektedir ve en önemli kalıtım metriklerinden biridir. Bir diğer önemli kalıtım metriği ise herhangi bir metriğin kaç farklı nesne tarafından miras alındığının tesbitidir. Bu tespit sonucu ortaya çıkan metriğe literatürde sahip olduğu çocuk sayısı denilmektedir. Kalıtım kümesi içerisinde kullanılabilecek tüm metrikler Ek A-5'de gösterilmektedir. Yapısal olarak kopyalanan nesnelerin kalıtım metrik kümesine göre çıkan sonuçlarının birbirine yakın olması beklenmektedir.

4.1.6 Boyut (Size) Kümesi

Yapısal kod klon analizinde en düşük derecede öneme sahip olan metrik kümelerinden biridir. Uygulamanın diğer nesneler ile aralarındaki ilişkilerden çok, nesnenin sahip olduğu öz değerlerle ilgili olabilmektedir. Tüm boyut metrikleri Ek A-6'da gösterilmektedir. Kopyalanıp hiç değiştirilmeden yapıştırılarak kullanılan nesnelerin analizinde, yapısal klon tespitine yardımcı olmaktadır.

4.2 Yapısal Kod Klon Tespiti için Önerdiğimiz Metodoloji

Yapısal kod klon tespit değerlendirilmesi yapılacak her bir sınıf için Bölüm 4. 1'de önerilen tüm metrik kümelerinin ölçüm değerlerinin bulunması gerekmektedir. Bulunan bu ölçüm değerleri, her bir sınıf için vektör uzay modelinde saklanmaktadır.

$$\mathrm{C_{v1}} = (\mathrm{m_1}, \mathrm{m_2}, \mathrm{m_3}, \ldots\ldots \mathrm{m_k}) \ldots\ldots\ldots \mathrm{C_{vn}} = (\mathrm{m_1}, \mathrm{m_2}, \mathrm{m_3}, \ldots\ldots \mathrm{m_k}) \qquad (4.1)$$

Örneğin, (4.1) 'de C_{v1} ile 1'inci sınıf için ortaya çıkan vektör uzay modeli gösterilmektedir. m_1, 1.nci metrik değerinin ölçüm sonucunu temsil ederken, m_{k} ise ilgili metrik kümesindeki k.ncı metriğin ölçüm değerini temsil etmektedir. C_{vn} ise yapısal kod klon değerlendirilmesi yapılan n'inci sınıfa ait vektör uzay modelini temsil etmektedir. Elde edilen bu ölçüm değerleri benzerlik hesaplama yöntemleri

kullanılarak aralarındaki benzerlikler tespit edilmektedir. Bu tezde literatürde yaygın olarak kullanılan hesaplama yöntemleri kullanılmaktadır. Bu amaçla, Coisine Benzerlik (Similarity) ve Jaccard İlişki (Correlation) hesaplama yöntemleri seçilmiştir.

Bölüm 2.1.1'de belirtildiği üzere, değerlendirme sonunda birbirine benzer olan iki sınıf birbirlerinin klon çifti olarak adlandırılmaktadır.

Cosine Benzerliği: Verilen iki vektörü birbiri ile sayısal olarak çarparak -1 ile 1 arasında bir değer dönmektedir. Kullanılan hesaplama yöntemi (4.2) 'de gösterilmektedir. Sonuç olarak verilmiş olan iki sınıf arasındaki benzerliği geri dönmektedir.

- Sonuç 1 olduğunda verilen iki vektör birbirinin aynısıdır.
- Sonuç -1 olduğunda verien iki vektör birbirinin 180 derece tersi fakat aynısıdır.
- Sonuç 0 olduğunda verilen iki vektor arasında bir benzerlik yoktur.

$$coisineBenzerliği(A,B) = \cos(\theta) = \frac{\mathrm{A.B}}{\|\mathrm{A}\|\|\mathrm{B}\|} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{Ai}\times\mathrm{Bi}}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{Ai})^2}\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}}(\mathrm{Bi})^2}} \quad (4.2)$$

Burada A ile 1 numaralı sınıfın ölçüm değerlerinden oluşan vektör gösterilirken, B ile gösterilen 2 numaralı sınıfın ölçüm değerlerinden oluşan vektörü gösterilmektedir.

Jaccard İlişkisi: Coisine benzerliği gibi verilen iki farklı vektör arasındaki ilişkiyi çıkarmak için kullanılmaktadır. Verilen vektörlerin kesişimlerinin birleşimlerine bölünmesi ile elde edilmektedir. Jaccard yönteminin hesaplanmasında kullanılan yöntem (4.3) 'de gösterilmektedir.

$$\mathrm{jaccardBenzerliği} = \mathrm{J(X,Y)} = \frac{|\mathrm{X}\cap\mathrm{Y}|}{|\mathrm{X}\cup\mathrm{Y}|} \quad (4.3)$$

(4.3) 'de, X ile 1 numaralı sınıfın vektör değerleri gösterilirken, 2 numaralı sınıfın vektör değerleri Y notasyonuyla gösterilmektedir. X ile Y vektörünün kesişim kümesinin, birleşim vektörüne bölünmesiyle sonuç bulunmaktadır. Sonuç olarak verilen iki vektörün arasındaki benzerlik 0 ile +1 arasında değerler almaktadır ve aşağıdaki gibi ifade edilmektedir.

- Sonuç 1 olduğunda verilen iki vektör birbirinin aynısıdır
- Sonuç 0 olduğunda aralarında herhangi bir benzerlik yoktur.

Benzerlik Hesaplama Yöntemi Yaklaşımı: Yapısal kod klon tespit yönteminde, ortaya konulan tespit sonucunun doğruluğunu artırmak için, değerlendirmesi yapılacak olan her bir sınıfa coisine ve jaccard hesaplama yöntemleri ayrı ayrı uygulanmaktadır. Coisine ve Jaccard hesaplama yöntemlerinin her ikisinin de yapısal klon olarak tespit ettiği sınıflar, klon çifti olarak kabul edilmektedir. Kullanılan yöntem (4.4) 'de gösterilmektedir.

$$KlonKümesi_{\substack{1<i<n \\ 1<j<n}}\left(B(cv_i, cv_j)\right) = \begin{cases} KlonKümesi \cup B(cv_i, cv_j)\ ; & B(cv_i, cv_j) = var \\ İşlem\ yapılmaz\ ; & B(cv_i, cv_j) = yok \end{cases} \tag{4.4}$$

$$Benzerlik = B(cv_i, cv_j) = \begin{cases} yok\ ; & i = j \\ var; & coisineBenzerliği(cv_i, cv_j) > EşikDeğeri \ \ Ve \\ & jaccardBenzerliği(cv_i, cv_j) > EşikDeğeri \\ yok\ ; & diğer\ durumlarda \end{cases}$$

(4.4) 'de n ile gösterilen değer, değerlendirilmesi yapılan projedeki sınıf sayısı iken, i ve j değerleri sınıfların indeks değerini göstermektedir. Değerlendirilmesi yapılan projenin, her bir sınıfı oluşturulan vektör uzay modelleri cv ile gösterilmektedir ve bu vektör değerleri birbirleri ile karşılaştırılmaktadır. Coisine ve jaccard karşılaştırma yönteminin her ikisine göre ortaya çıkan sonuç, önceden belirlenmiş olan eşik değerinin (threshold) üzerinde ise karşılaştırılan ikili, klon olarak tutulmaktadır. Karşılaştırılması için verilen i ve j indislerinin aynı olması durumunda, sınıf kendisi ile karşılaştırılmak istenmektedir ve benzerlik yok olarak cevap dönülmektedir. Coisine ve jaccard hesaplama yöntemlerinden çıkan sonuçlardan her ikisinin de eşik değerinin üzerinde bulunmadığı durumlarda sonuç olarak benzerlik yok kabul edilmektedir.

4.3 Yapısal Kod Klon Tespitinde Metodolojinin Uygulama Adımları

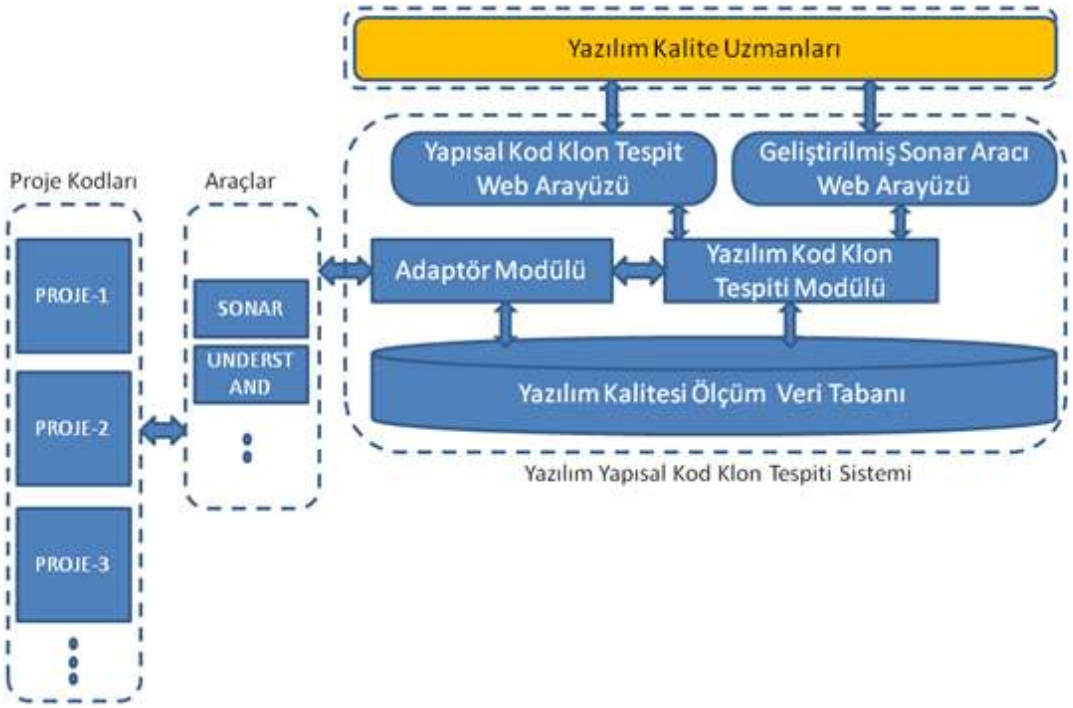
Nesneye yönelik programlama mantığına dayalı olarak geliştirilmiş yazılımlarda yapısal kod klon tespitinin gerçekleştirilmesi için iki farklı yöntem kullanılmaktadır. Bu yöntemlerden ilki bu tez kapsamında önerilen yöntem bilimi diğeri ise yazılım geliştiricilerden oluşan denekler kullanılarak, göz ile yapılan tespit yöntemidir. Sonar ile analizi yapılan herhangi bir proje üzerinde yapısal klon tespiti için, web tabanlı bir uygulama geliştirilmiştir. Bunun yanı sıra, Sonar üzerinde yapılan ek geliştirmelerle, yeni önerilen metriklerin ölçümleri yapılmaktadır. Geliştirilen Sonar sürümünün hesaplamış olduğu metrikler kullanarak yapısal klon tespiti yapılmaktadır. Geliştirmiş olunan uygulama, tespit aşamasında şu adımları takip etmektedir.

1) Aynı proje içerisinde mi, farklı projeler arasında mı klon tesbpti yapılacağının kararının verilmesi
2) Sonar üzerinde değerlendirilen projelerin her bir sınıfı için metrik değerlerinin toplanması
3) Yapısal klon tespit yönteminde kullanılacak olan ve Bölüm 4. 1 ‘de anlatılan metrik kümelerine göre, olası klon adaylarının belirlenmesi
4) Tez kapsamında belirlenen metrik kategorilerinin kullanılarak, Bölüm 4. 2’de verilen hesaplama yöntemlerinin sonuçlarına göre benzerlik ölçümlerinin bulunması
5) Tez kapsamında yapılan deneysel yöntem ile tespit edilen eşik değeri kullanılarak, her bir sınıfın yapısal klonlarının tespit edilmesi

BÖLÜM 5

UYGULAMA DETAYLARI

Bu tez çalışması kapsamında nesneye dayalı programlama dillerinde metrik tabanlı yapısal kod klon analizini gerçekleştirmek için önerilen sistemin mimarisi Şekil 5. 1 ‘de gösterilmektedir.



Şekil 5. 1 Önerilen yapısal kod klon tespit sistemi mimarisi

Şekil 5. 1’de gösterildiği gibi bir veya birden fazlı proje, yazılım kalite ve yönetim araçlarında değerlendirilmektedir. Çıkan ölçüm sonuçları adaptör modülü sayesinde

önerilen yazılım yapısl kod klon tespit sistemi içerisine alınmaktadır. Alınan bu ölçüm sonuçları yazılım kalitesi ölçüm veri tabanında tutularakyapısal kod klon tespit aşamasında yazılım kod klon tespit modülü tarafından kullanılmaktadır. Yazılım kod klon tespit modülü almış olduğu bilgileri işleyerek ortaya çıkardığı yapısal klonları veritabanına kaydetmektedir. Aynı zamanda bulunan yapısal klonlar kullanılan yazılım kalite ve yönetim aracının arayüzünde gösterilmektedir. Yapısal kod klon tespiti sırasında elde edilen sonuçların daha detaylı açıklanması için geliştirilmiş olunan yapısal kod klon tespit web arayüzü kullanılmaktadır. Bu sayede yazılım kalite uzmanları, yüzeysel bilgileri yazılım ve kalite aracı arayüzünde görebilirken, daha detaylı bilgileri geliştirilmiş olunan arayüzden takip edebilmektedirler.

5.1 Uygulama Geliştirilirken Kullanılan Teknolojiler

Önerilen bu sistemin gerçekleştirilmesinde, yazılım kalite ve yönetim aracı olarak Sonar[71] aracı kullanılmaktadır. Bu aracın seçilmesindeki en önemli faktörler açık kaynaklı olması ve yaygın olarak kullanılmasıdır. Yazılım kalitesi ölçüm veri tabanı ise MySQL [75] veri tabanıdır. Herhangi bir lisans ücreti olmadığındanve Java programlama dilleri ile yazılmış web tabanlı projeler için yaygın olarak kullanıldığından dolayı [76] , bu veri tabanı kullanılmaktadır. Java [77] programlama dili geliştirme dili olarak kullanılmaktadır. İşletim sistemine bağımlı olmaması [78], web tabanlı uygulamalarda yaygın olarak kullanılması [79] ve herhangi bir lisans ücretinin olmamasındandolayı tercih edilmiştir. Yazılım tespit modülü ile adaptör modülü için gerekli olan veri tabanına yazma, okuma, silme ve güncelleme gibi temel işlemlerin gerçekleştirilmesinde Hibernate [80] ve Java Persistence API (JPA) [81] kullanılmıştır. Geliştirilmiş olan uygulamanın hem veri tabanı hem de arayüz ile iletişim içerisinde olmasından dolayı, katmanlı bir mimari yapısı tercih edilmiştir. Bu katmanlı mimari yapısının gerçekleştirilmesinde Spring Çerçevesi (Framework) [82] kullanılmıştır. Sonar aracının arayüz geliştirmeleri sırasında gömülü (embedded) Ruby [83] kullanılmıştır. Geliştirilmiş olan yazılım yapısal kod klon tespit arayüzünde ise Java dili için hazırlanmış olan ve web tabanlı uygulama geliştirmek için hazır parçaları (component) bulunan Java Server Faces [84], [85] kullanılmıştır. Java Server Faces içerisinde bulunan parçalarızenginleştiren açık kaynaklı kütüphane olan PrimeFaces [86] kullanılmıştır.

Tüm bu parçaların bir arada düzenli bir şekilde çalışmasını sağlamak ve olası güncellemeleri takip edebilmek için Maven [87], [88] proje yönetim aracı kullanılmıştır. Geliştirilmiş olan yapısal kod klon tespit yazılımının yayınlanması aşamasında, uygulama sunucusu olan Tomcat [89], [90] tercih edilmiştir. Lisans ücreti olmadan kullanılabilmesi ve Java tabanlı web uygulamaları ile uyumlu olduğu için Tomcat tercih edilmiştir.

Yapılan inceleme sonucunda, Sonar aracının Ek A-7'de gösterilen metrikleri sağladığıve önermiş olunan sistem için daha fazla metriğe ihtiyaç olduğu fark edilmiştirBu sebepten dolayı Sonar içerisinde yaptığımız geliştirmeler ile Ek A-8 'de gösterilen metrikleri oluşturduk.

5.2 Sonar Geliştirmeleri

Sonar, yazılım projelerini değerlendirirken kendi içerisinde 2 farklı yöntem kullanmaktadır. Bu yöntemler kaynak kodun parse edilerek değerlendirilmesi ve kaynak kodun byte kodu üzerinden yapılan değerlendirmelerdir. Bu iki farklı değerlendirme yönteminin sonucu olarak toplamış olduğu metrikler sayesinde yazılım projelerinin kalitelerinekarar verebilmektedir.

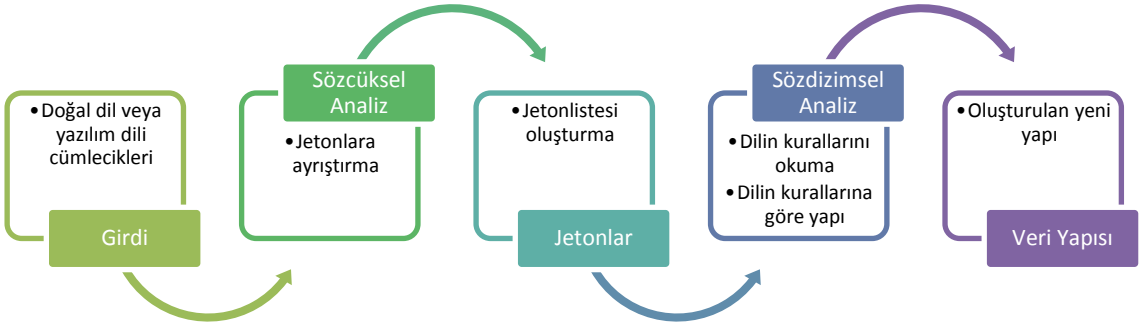
5.2.1 Ayrıştırıcı (Parser)

Ayrıştırıcı, kendisine girdi (input) olarak verilmiş olan doğal ya da yazılım diline ait cümlecikleri o dilin tanımlamış olduğu kurallara bağlı kalarak değerlendiren ve ortaya yeni bir veri yapısı oluşturan bir yazılım aracıdır. Şekil 5. 2'de gösterildiği gibi girdi verisi parser içerisindeki sözcüksel (lexical), jeton (tokens) ve söz dizimi analiz aşamalarından geçerek istenilen veri yapısına çevirilmektedir.

Girdi: Ayrıştırıcıya verilecek olan doğal ya da yazılım dilinin formatıdır. Genel olarak cümle halinde verilmekle birlikte yazılım dilleri için XML, HTML gibi işaret dili formatında da verilebilmektedir.

Sözcüksel Analiz: Girdi olarak verilmiş olan verinin, ayrıştırma işlemi için anlamlı olabilecek küçük parçalara ayırma işlemidir. Buradaki ayırma işleminde girdi verisinin

her bir parçası tek tek değerlendirilmektedir ve uygun olan küçük parçalara çevrilmektedir.



Şekil 5. 2 Ayrıştırma Süreçleri

Jeton: Sözcüksel analiz içerisinde çevrimi yapılan anlamlı olabilecek küçük parçalara jeton denilmektedir. Sözcüksel analiz içerisinde tek tek değerlendirme sonucunda ortaya bir jeton listesi çıkmaktadır.

Sözdizimsel Analiz / Ayrıştırıcı: Bu aşamada jetonlarına ayrılmış olan girdi verisi, talep edilen anlamlı veri yapısına çevrilmektedir. Bu noktada verilen doğal ya da yazılım dili girdisinin kuralları kullanılmaktadır. Dilin kurallarına göre, jetonlar kullanılarak anlamlı bir veri yapısı oluşturulmaktadır [91].

Veri yapısı: Ayrıştırma işlemi sonrasındaki işlemlerde kullanılmak üzere, oluşturulan veri yapısı tipidir. Bu veri yapısı soyut sözdizimi ağacı, ayrıştırılmış ağaç veya bir başka hiyerarşik veri yapısı olabilmektedir.

5.2.2 Dilbilgisi (Grammar)

Ayrıştırma aşamasında bahsi geçen kurallar bütününe o dilin dilbilgisi denilmektedir. Doğal ya da yazılım dili fark etmeksizin her dilin oluşmasını sağlayan kurallar bulunmaktadır. Örneğin doğal dil olarak ingilizce dili ele alınırsa, İngilizce dilbilgisi kuralına göre özne önce gelir onu yüklem takip eder ve son olarak nesne gelmelidir. Bu kurallar dizini kendi içerisinde iki farklı gruba ayrılmaktadır. Bu gruplar çözümlemeli (analytic) ve üretici (generative) olarak adlandırılmaktadır.

5.2.2.1 Üretici Dilbilgisi

Dilbilgisi çeşitleri içerisinde en yaygın olan türdür. Sadece bir sembolden başlayarak, o dil içerisinde anlamlı olan olası tüm kelimelerin oluşturulmasına dayanan kurallar bütünüdür. Üretici dilbilgisinin sahip olduğu algoritma tanımlandığı dil içerisindeki kelimeleri üretmektedir. Noam Chomsky [92] çalışmasında üretici dilbilgilerini 4 farklı ana başlık altında toplamaktadır.

- Type -0 : Sınırsız (unrestricted)
- Type-1 : İçeriğe duyarlı (context-sensetive)
- Type-2 : İçeriğe duyarsız (context-free)
- Type-3 : Düzenli dilbilgisi (regular grammars)

Bunların haricinde özyinelemeli (recursive) dilbilgisi bulunmaktadır ve, özyinelemeli dilleri ayrıştırma işlemlerinde kullanılmaktadır.

5.2.2.2 Çözümlemeli Dilbilgisi

Üretici dilbilgisinden farklı olarak, kendisine girdi olarak verilen kelimenin, o dile ait bir kelime olup olmadığını inceleyerek karar veren bir kurallar bütünüdür. Özet olarak, çözümlemeli dilbilgisi nasıl okunulacağını belirtirken, üretici dilbilgisi daha çok nasıl yazılması gerektiğini belirlemektedir. Yukarıdan-aşağıya (top-down) dilbilgisi ve bu yukarıdan aşağıya dilbilgisinin derleyici (compiler) gereksinimlerine göre yeniden düzenlenmesi ile elde edilen ayrıştırıcı dilbilgisi ve ifade (expression) çözümlemeli dilbilgisi olarak en yaygın kullanılan dilbilgileridir.

5.2.3 Sonar Ayrıştırıcı

Sonar [93], [94] içerisindeki ayrışrıtıcı aşamasında, değerlendirilmesi için programrama tanıtılan her bir program dosyası öncelikle teker teker sözcüksel analiz işlemine alınmaktadır. Bu aşamada Sonar tarafından tanımlanmış olan sözcüksel analiz değerlerine göre, programın kaynak kodu ayrıştırma işlemi için anlamlı olabilecek küçük parçalara bölünmektedir. Bu aşamada programlama dilinin anahtar kelimeleri de

göz önüne alınarak, yazılımın kaynak kodu jetonlarına ayrıştırılmaktadır. Şekil 5. 3'te ayrıştırma aşamasında dikkate alınan Java anahtar sözcükleri gösterilmektedir.

```
private static final String[] JAVA_KEYWORDS = new String[] { "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char",
    "class", "const", "continue", "default",
"do", "double", "else", "enum", "extends", "false", "final", "finally", "float", "for",
    "goto", "if", "implements", "import", "instanceof",
"int", "interface", "long", "native", "new", "null", "package", "private",
    "protected", "public", "return", "short", "static", "strictfp",
    "super", "switch", "synchronized", "this", "throw", "throws", "transient", "true", "try", "void", "volatile", "while"};
```

Şekil 5. 3 Java anahtar sözcükleri

Yazılım dilinin anahtar sözcüklerine göre jetonlarına ayrılan yazılım kaynak kodu, bir sonraki aşama olan ayrıştırma aşamasında dilin kurallarınagöre değerlendirilmektedir. Sonar yazılım dillerini ayrıştırma aşamasında çözümlemeli dilbilgisi çeşitlerinden olan ayrıştırıcı ifade dilbilgisini (Parser Expression Grammar (PEG)) kullanmaktadır.

5.2.3.1 Ayrıştırıcı İfade Dilbilgisi

Biçimsel dilleri tanımlamakta kullanılan dilbilgisi çeşidir. 1970'li yıllarda tanımlanmış olan yukarıdan-aşağıya ayrıştımaya benzer olarak Bryan Ford tarafından 2004 yılında tanıtılmıştır. Yukarıdan-aşağıya ayrıştırmaya olan benzerliği, her ikisinde de dil içerisinde tanımlanmış olan dilbilgisi kurallarının kolaylıkla okunabilir ve anlaşılabilir olmasından kaynaklanmaktadır.

PEG içerisinde sıralı bir seçim mevcuttur, bu sıralamaya göre ilk sırada gelen alternatif öncelikle değerlendirilecektir. Eğer ilk alternatif başarılı bir şekilde parse edildi ise ikincisine geçilecektir fakat ilk başarısız olduysa, ilk konuma geri dönülecek ve parse işlemine ikinci aşamadan devam edilecektir. Bundan dolayı PEG ile oluşturulmuş bir parser hiçbir zaman kararsız duruma düşmemektedir.

PEG içerisinde aritmetik işlemler için gerekli olan kurallar Çizelge 5. 1'deki gösterime göre hazırlanmaktadır. Eğer girdi olarak verilen kelime Çizelge 5. 1'deki değerlere göre hazırlanan kurallardan birine uyuyorsa başarılı, hiç birine uymuyorsa başarısız olarak cevap dönmektedir.

Çizelge 5. 1 PEG operatörleri

Sembol	Açıklaması
E*	Sıfır veya çok: Mümkün olan en çok sayıda E kuralı kabul edilir. Her zaman başarılı sonuç döner.
E?	Sıfır veya bir: E kuralı ile eşleşme arar eğer bulursa, girdi verisini tüketmiş olur ve başarılı sonuç döner.
!E	E kuralı ile karşılaşmamayı dener, fakat girdi verisini tüketmez. Eğer E kuralı ile karşılaşırsa hata döner, yoksa başarılı sonuç döner.
&E	E kuralı ile karşılaşmayı dener, fakat girdi verisini tüketmez. Eğer E kuralı ile karşılaşmaz ise hata döner.
E+	Bir veya çok: E kuralı ile karşılaşmaya çalışır ve girdi verisini tüketir. E kuralı ile bir veya birden fazla kez karşılaşırsa başarılı sonuç döner, yoksa hatalı sonuç döner.
A B C D	Her bir kuralı verilen sırada denetler ve her bir kural için girdi verisini tüketir. Eğer herhangi bir kural hataya düşerse başlangıç pozisyonuna geri döner ve hata mesajı döner.
A / B / C / D	Sıralı verilen her bir kuralı dener, eğer herhangi biri başarılı sonuç verirse girdi verisini tüketir. Eğer herhangi biri başarılı dönmezse hata verir.
[abcd]	Verilmiş olan ‘a’, ‘b’, ‘c’, veya ‘d’ karakterleri ile eşleşmeyi aramaktadır. Eğer herhangi biri eşleşirse girdi verisini tüketir ve başarılı sonuç döner, yoksa hata döner.
[c1-c2]	Verilmiş olan c1 ile c2 karakterleri arasındaki herhangi bir karakter ile karşılaşmayı arar. Eğer karşılaşma olursa girdi verisini tüketir ve başarılı döner, yoksa hata döner.
‘Verilen metin’	Verilen metin ile karşılaşmayı arar. Eğer karşılaşma olursa başarılı sonuç döner, yoksa hata döner.
.	Herhangi bir karakter arar, var olan tüm karakterler için başarılı sonuç döner. Girdi verisi bittiğinde hata döner.

5.2.3.2 Ayrıştırıcı İfade Dilbilgisinin Avantajları

PEG‘in en önemli avantajı hiçbir zaman kararsızlığa düşmemesidir. Örnek olarak kimi programlama dillerinde bulunan iç içe geçmiş koşullu ifadelerin yorumlanması

verilebilmektedir.

```
IF firstCondition THEN
   IF secondCondition THEN
      #do something
ELSE
    #do somethingElse
```

```
IF firstCondition THEN
   IF secondCondition THEN
       #do something
   ELSE
       #do somethingElse
```

Şekil 5. 4 Ayrıştırma karışıklığı

Şekil 5. 4'deki kod program parçaçığının yorumlanması içeriğe duyarsız dilbilgileri için karışıklığa yol açmaktadır. Şekil 5. 4'de gösterildiği gibi bu durum 2 farklı şekilde yorumlanabilmektedir. Fakat bu kod parçaçığı PEG kullanan bir ayrıştırıcı tarafından ayrıştırıldığında sıralı ayrıştırma seçimi olduğundan herhangi bir karmaşıklık olmamaktadır.

Bir diğer avantajı ise, içeriğe duyarsız dilbilgisi gibi dilbilgileri tarafından ayrıştırılması mümkün olmayan dilleri de ayrıştırabilmesidir. Bu özelliğinin temel kaynağı ayrıştırırken kestirim yöntemlerini kullanarak sınırsız boyutta ileriyi kestirebilmesidir [95].

Geliştirmeye ve değiştirilmeye açık olması bir diğer avantajı olarak öne çıkmaktadır. Birden fazla kural ve operatör, eğer içerlerinde kullandıkları sembol isimler birbirleri ile çakışmıyorsa tek bir kural altında toplanabilmektedir. Bu birleştirme işlemi Rats ayrıştırıcıyı baz alan Fotress yazılım dilinde yoğun olarak kullanılmaktadır [96].

5.2.3.3 Ayrıştırıcı İfade Dilbilgisinin Dezavantajları

Öz yinelemeli ifadeler sonsuz döngüye girebilmektedir [94]. Her ne kadar bu sonsuz döngülerin çözümlendiği iddia edilse de [93] bu konu hala belirsizliğini korumaktadır.

```
private static void classDeclaration(LexerlessGrammarBuilder b) {
  b.rule(CLASS_DECLARATION).is(CLASS, IDENTIFIER, b.optional(TYPE_PARAMETERS), b.optional(EXTENDS, CLASS_TYPE), b.optional(IMPLEMENTS, CLASS_TYPE_LIST),
      CLASS_BODY);

  b.rule(CLASS_BODY).is(LWING, b.zeroOrMore(CLASS_BODY_DECLARATION), RWING);
  b.rule(CLASS_BODY_DECLARATION).is(b.firstOf(
      SEMI,
      CLASS_INIT_DECLARATION,
      b.sequence(b.zeroOrMore(MODIFIER), MEMBER_DECL)));
  b.rule(CLASS_INIT_DECLARATION).is(b.optional(STATIC), BLOCK);
  b.rule(MEMBER_DECL).is(b.firstOf(
      b.sequence(TYPE_PARAMETERS, GENERIC_METHOD_OR_CONSTRUCTOR_REST),
      b.sequence(TYPE, IDENTIFIER, METHOD_DECLARATOR_REST),
      FIELD_DECLARATION,
      b.sequence(VOID, IDENTIFIER, VOID_METHOD_DECLARATOR_REST),
      b.sequence(IDENTIFIER, CONSTRUCTOR_DECLARATOR_REST),
      INTERFACE_DECLARATION,
      CLASS_DECLARATION,
      ENUM_DECLARATION,
      ANNOTATION_TYPE_DECLARATION));
  b.rule(FIELD_DECLARATION).is(TYPE, VARIABLE_DECLARATORS, SEMI);
  b.rule(GENERIC_METHOD_OR_CONSTRUCTOR_REST).is(b.firstOf(
      b.sequence(b.firstOf(TYPE, VOID), IDENTIFIER, METHOD_DECLARATOR_REST),
      b.sequence(IDENTIFIER, CONSTRUCTOR_DECLARATOR_REST)));
  b.rule(METHOD_DECLARATOR_REST).is(FORMAL_PARAMETERS, b.zeroOrMore(DIM), b.optional(THROWS, QUALIFIED_IDENTIFIER_LIST), b.firstOf(METHOD_BODY, SEMI));
  b.rule(VOID_METHOD_DECLARATOR_REST).is(FORMAL_PARAMETERS, b.optional(THROWS, QUALIFIED_IDENTIFIER_LIST), b.firstOf(METHOD_BODY, SEMI));
  b.rule(CONSTRUCTOR_DECLARATOR_REST).is(FORMAL_PARAMETERS, b.optional(THROWS, QUALIFIED_IDENTIFIER_LIST), METHOD_BODY);
  b.rule(METHOD_BODY).is(BLOCK);
}
```

Şekil 5. 5 Sonar sınıf tanımlama kuralı

5.2.3.4 Sonar Dilbilgisi Örneği

Doğal ya da yazılım dillerinin değerlendirilme aşamasında dilbilgisi kuralları kullanılmaktadır. Örneğin, İngilizce dili ile yazılmış bir cümlenin başarılı bir şekilde ayrıştırılması için İngilizce dilbilgisi kuralları kullanılmaktadır. İngilizce dil kuralına göre, bir cümlenin geçerli olması için aşağıdaki şablonu izlemesi gerekmektedir. Burada isteğe bağlı olan eleman nesnedir. Nesne olmadan da herhangi bir cümle kurulabilmektedir.

$$Subjcet + Verb + Object_{(optional)}$$

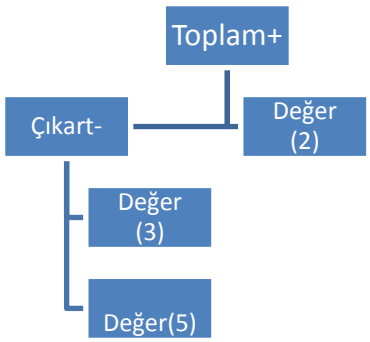
Doğal dillerde olduğu gibi yazılım dillerinde de dilbilgileri kullanılmaktadır. Sonar içerisinde PEG kullanılmaktadır. Çizelge 5. 1'de verilen kurallara göre, yazılım dillinin kaynak kodu üzerinden ayrıştırma işlemi yapılmaktadır. Örneğin, Sonar içerisinde Java dili için kullanılan sınıf tanımlama kuralları Şekil 5. 5'de gösterilmektedir.

```
b.rule (CLASS_DECLARATION).is (CLASS, IDENTIFIER, b.optional
(TYPE_PARAMETERS), b.optional (EXTENDS, CLASS_TYPE), b.optional
(IMPLEMENTS, CLASS_TYPE_LIST),
        CLASS_BODY);
```

Burada kullanılan kurala göre, verilmiş olan jeton listesinin bir sınıf tanımlaması olabilmesi için, sınıf ile başlamış olması, sınıf adının verilmiş olması ve bir gövdesinin (body) olması gerekmektedir.

5.2.3.5 Sonar Çıktı Veri Yapısı

Ayrıştırma sonucunda yazılımın kaynak kodu ileriki aşamalarda kullanılmak üzere hiyerarşik veri yapılarına dönüştürülmektedir. Sonar içerisinde, bu hiyerarşik yapılardan soyut sözdizim ağacı kullanılmaktadır. Yaprak düğümlerin (leaf node) değerler ve ara notların operatörler olduğu bir ağaç yapısı yazılım tasarım şablonlarından olan bileşim tasarım şablonu kullanılarak oluşturulmaktadır.

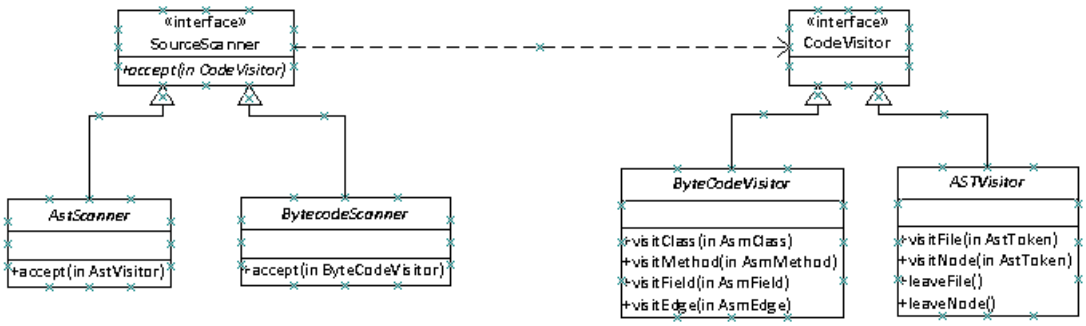


Şekil 5. 6 Ayrıştırıcı çıktı veri yapısı

Bu ağaç yapısında operatörler ve değerler ifade olarak değerlendirilmektedir. Her bir ifadenin hesaplama süreçlerinde bir işlem özelliği bulunmaktadır. Şekil 5. 6 'da değer olarak gösterilen ifadeler kendi değerlerini geri dönerken, operatör ifadeleri ile gösterilen ifadeler dönen değerler üzerinden sahip oldukları operatör işlemlerini yapmaktadırlar. İşlem özellikleri ile verinin taşınmasında iki farklı yöntem öne çıkmaktadır. Bunlardan birincisi veri merkezli yönetim biçimidir ve veri ile birlikte operatörün gezdirilmesi gerekmektedir. Diğer yöntem ise, veri ile işlem yapan yapıları birbirinden ayırmaktadır. Bu yapıya operasyon merkezli yapı denmektedir. Operasyon merkezli yapılarda veri yapısı, yapacağı operasyon hakkında bilgi tutmamalıdır, bunun sağlanabilmesi için ziyaretçi (visitor) tasarım şablonu kullanılmaktadır. Sonar içerisinde ayrıştırma aşamasından sonra oluşturulan soyut sözdizim ağacı üzerinde işlemleri yapabilmek için bu tasarım şablonu kullanılmaktadır.

5.2.4 Ziyaretçi Tasarım Şablonu

Herhangi bir nesne yapısını oluşturan sınıflar üzerinde yapılacak olan bazı işlemleri tanımlamak için ziyaretçi tasarım şablonu kullanılmaktadır. Sonar içerisindeki ayrıştırma işlemi sonucu yazılım kaynak kodları birbirinden farklı türlerdeki jetonlara ayrıştırılmıştır. Ayrıştırılan jetonlar üzerinden gerekli metrikleri toplamak amacıyla bu tasarım şablonu kullanılmaktadır. Şekil 5. 7'de gösterildiği gibi iki farklı metrik toplama yöntemi (bytecode, abstract syntax tree) için SourceScanner sınıfı gerçekleştirilmiştir. Bu tarayıcı sınıflar, taramak istedikleri nesnenin yapısına göre byte kod ya da jeton ziyaretçi sınıflarını kabul etmektedirler.



Şekil 5. 7 Sonar ziyaretçi tasarım şablonu

5.2.5 Sonar Ayrıştırma Metrikleri

Ayrıştırma aşamasından sonra oluşan Soyut Sözdizim Ağaç yapısının Şekil 5. 7'deki gibi bir ziyaretçi tasarım şablonu ile gezilmesi sonucu sonar tarafından toplanan metriklerin bazıları Çizelge 5. 2 'de gösterilmektedir. Java dili için ayrıştırma tekniğiyle metrik toplamak için kullanılan tüm ziyaretçi sınıfları Ek B-1'de gösterilmektedir.

Sonar içerisinde yapılan katkılar sonucu MethodLocalVariableVisitor ve ClassLevelVariableVisitor sınıfları eklenmiştir. Bu ziyaretçi sınıfları kullanılarak, sınıf içerisinde tanımlanmış toplam değişken sayısı ve herhangi bir sınıf içerisinde bulunan metodlar içerisinde tanımlanmış toplam yerel değişken sayısı bulunmaktadır.

Çizelge 5. 2 Sonar ayrıştırma aşamasındaki metrikleri

Ziyaretçi Adı	**Ölçülen Metrik Adı**	**Domain**
PackageVisitor	Paket sayısı	Boyut
FileVisitor	Dosya sayısı	Boyut
ClassVisitor	Sınıf sayısı	Boyut
MethodVisitor	Metot sayısı	Boyut
AccessorVisitor	Erişim sağlayıcı metot sayısı	Boyut
PublicApiVisitor	Erişilebilir doküman (public_doc)	Boyut
PublicApiVisitor	Erişilebilir doküman program arayüzü	Dokümantasyon
LinesVisitor	Satır sayısı	Boyut
LinesOfCodeVisitor	Kod satır sayısı	Boyut
CommentLinesVisitor	Başlıksız yorum satır sayısı	Dokümantasyon
CounterVisitor	İfade sayısı	Boyut
MethodLocalVariableVisitor	Metot içindeki yerel değişken sayısı	Boyut
ClassLevelVariableVisitor	Sınıf değişken sayısı	Boyut
ComplexityVisitor	Karmaşıklık	Karmaşıklık

5.2.6 Sonar Byte Kod Çözümleyici

Program içerisindeki potansiyel hataların bulunması, kullanılmayan kodların tespit edilmesi ve uygulamanın performansının izlenmesi gibi birden çok kullanım amacı için yazılım projelerini anlaşılabilir küçük komut (instruction) kümelerine dönüştürerek analiz etmektedir [97], [98]. Sonar, Java ile yazılmış olan uygulamalardan metrikleri toplamak için bayt kodlar (bytecode) üzerinden işlem yapmaktadır. Literatürde birden fazla bayt kod çevirici uygulama bulunmaktadır [99], [100]. Bu uygulamalar içerisinden ASM bayt kod çerçevesi tercih edilmiştir.

5.2.6.1 ASM Çerçeve Programı

Farklı sistemlere uyumlu olarak Java sınıflarının dinamik olarak yaratılması ve işletilmesi için gelişitirilmiş bir araçtır. BCEL [99], Serp [101] and JOIE [100] gibi var olan çerçeve programları ile karşılaştırıldığında, performans ve hafıza kullanımı açısından daha iyi sonuçlar vermektedir [102].

Diğer çerçeve programları bayt kod üzerinden nesneleri oluşturmaktadırlar. Bu da onların işlemlerinde çok fazla zaman ve hafıza kaybına yol açmaktadır. İşlemleri sırasında Şekil 5. 7'de görüldüğü gibi ziyaretçi tasarım şablonunu kullanmaktadır.

ASM çerçeve programı, sınıf içerisinde bulunan metod, değişken ve anatasyon gibi sınıf yapısını oluşturan yapıları otomatik olarak yöntebilmektedir. Ayrıca bayt kodları oluştururken onları birer etiket ile sabitlemektedir, bu yöntem sayesinde istenildiği zaman var olan bayt kodlar arasına kolaylıkla ekleme yapılabilmektedir. Sonar içerisinde, Şekil 5. 7'de bulunan visitor tasarım şablonu kullanılarak gerçekleştirilen örnek bir bayt kod çözümleyici şu şekildedir.

```
1. BytecodeScanner scanner = new BytecodeScanner();
2. MethodByteVisitor methodVis = new MethodByteVisitor ();
3. scanner.accept(methodVis);
4. scanner.scan(bytecode);
```

Burada öncelikle bayt kod üzerinde analiz işlemini gerçekleştirecek olan ByteCodeScanner sınıfına uygulamanın bayt kodları verilmektedir. 2 numaralı adımda tasarım metirklerinden olan metod içerisindeki özel (private), genel (public), korunmuş (protected) sayılarının hesaplanması için bir ziyaretçi tasarlanmaktadır. 3 numaralı adımda uygulamayı tarayacak sınıfa metod ziyaretçisi eklenmektedir. Son aşama olarak, verilmiş olan bayt kod dosyası üzerinde, metdoların erişim seviyesine göre ölçüm sonuçları taranmaktadır.

ASM çerçeve programı tarafından bayt koda çevrilmiş olan herhangi bir sınıfın değerlerinin onaltılık (hexadecimal) değerleri bulunmaktadır. Bu onaltılık değerler

kullanılarak, Ek A-9 ‘da gösterilen erişim seviyesine göre tüm metrikler bayt kod üzerinden yapılan analizler sonucunda eklenmektedir.

BÖLÜM 6

YÖNTEMİN DEĞERLENDİRİLMESİ

Önerilen yöntemin deney aşamasında, Zürih üniversitesindeki Dynamic and Distributed Information Systems Group tarafından yazılmış olan ve ontolojiler arasındaki benzerlik tespitini amaçlayan SimPack [103] ile Brian Wellington tarafından Java diliyle yazılmış DNSJava [104] adlı açık kaynak kodlu yazılımlar kullanılmıştır. Seçilen SimPack projesi içerisinde 128 adet sınıf vardır, DNSJava projesinde ise 165 adet sınıf bulunmaktadır.

Gerçekleştirilen yazılım projesinin sonuçlarının değerlendirilmesi aşamasında, uygulama geliştiricilerin tecrübelerinden yararlanılmaktadır. Nesneye dayalı olarak geliştirilmiş bir uygulamanın tüm kaynak kodları geliştiricilere karşılıklı olarak çiftler halinde gösterilmiştir. Uygulama geliştiricilerin vermiş olduğu karara göre gösterilen çiftin klon olup olmadığı kayıt altında tutulmuştur. Uygulama geliştiricilerden değerlendirme aşamasında aşağıdaki maddelere dikkat edilmesi talep edilmiştir.

1) Herhangi bir nesnenin başka bir nesneyi geniştletmesiyada gerçeklemesi
2) Herhangi bir nesnenin içerisinde bir başka nesneyi kullanması

Bu iki maddeden herhangi biri gerçekleşmişse, gösterilen kod çifti klon olarak değerlendirilmektedir.

6.1 Kontrol Veri Kümesi

Uygulama için geliştirilmiş olan arayüz sayesinde, 10 farklı uygulama geliştiriciden birbirlerinden bağımsız olarak uygulama içerisindeki olası yapısal klonları tespit etmeleri talep edilmektedir. Uygulama geliştiricileri, arayüz üzerinde ikili çiftler halinde sunulan sınıfların yapısal klon olup olmadığına karar vermektedirler. Geliştirilmiş olan uygulamanın arayüzleri Ek-B-2 'de gösterilmektedir. Kod klon tespiti için önerilen yöntemin değerlendirilmesinde, literatürde bulunan [105] ve [106] çalışmaları ile aynı yöntem izlenmektedir. Bütün uygulama geliştiricileri bilgisayar mühendisliği bölümlerinden mezun ve nesneye dayalı programlama dillerinin tasarımı ile kodlanmasında deneyimli kullanıcılar arasından seçilmiştir. Uygulama geliştiricilerin yapmış oldukları analizlerden elde edilen sonuçların toplanmasından sonra her bir uygulama geliştiricinin bulmuş olduğu sonuçların karşılaştırılması Çizelge 6. 1'de gösterilmektedir.

Çizelge 6. 1 Uygulama geliştirici sonuç değerleri

Uygulama Geliştirici Değerlendirmesi	**SimPack Projesi**		**DNSjava Projesi**	
	Bulunan Klon Çifti Sayısı	**Zaman**	**Bulunan Klon Çifti Sayısı**	**Zaman**
1	670	3saat	1154	4saat 15Dakika
2	908	4saat 30Dakika	916	3saat 30Dakika
3	740	3saat 15Dakika	1098	4saat 15Dakika
4	698	3saat	1070	4saat
5	1050	4saat 15Dakika	1062	3saat 45Dakika
6	1034	4saat	994	3saat 30Dakika
7	1044	3saat 45Dakika	852	3saat 15Dakika
8	720	3saat 15Dakika	960	3saat 30Dakika
9	820	3saat 15Dakika	1170	4saat 15Dakika
10	922	3saat 30Dakika	1142	4saat
Ortalama	718	3saat 40Dakika	950	3saat 50Dakika

Çizelge 6. 1'de gösterildiği üzere tüm geliştiricilerin SimPack projesi için ortalama değerlendirme süresi 3 saat 40 dakika ve bu sürede ortalama 718 adet olası klon çifti bulunmaktadır. DNSJava projesinin değerlendirilme süresi yaklaşık olarak ortalama 3 saat 50 dakikada gerçekleştirilmiştir. DNSjava projesi için ortalama 950 adet olası klon çifti tespit edilmiştir.

Eğer 10 uygulama geliştiriciden 7'si gösterilen çifti klon olarak değerlendiriyorsa, bu çift geliştiricilerin değerlendirmesini oluşturan kümeye alınmaktadır. Burada kullanılmış olan %70 benzerlik oranı, Kim ve arkadaşları [106] tarafında yapılan çalışmaya dayanarak kullanılmaktadır.

$$d(A,B) = \left(\frac{|A \cap B|}{\frac{|A|+|B|}{2}} \right) \times 100 \qquad (6.1)$$

Uygulama geliştiricilerin arasında yapılan doğruluk değerlendirmesi için (6.1) kullanılmaktadır. (6.1) 'de, A karşılaştırma yapılan herhangi iki uygulama geliştiricilerden ilkini gösterirken, B ise karşılaştırılan diğer uygulama geliştiriciyi göstermektedir. Buna göre A ile B geliştiricisinin ortak bulduğu yapısal kod klon adayların, her ikisinin buldukları ortalama yapısal kod klon sayısına olan oranı sonucu vermektedir. Yapılan değerlendirme sonuçlarındaki ortak görüşleri gösterebilmek adına, 1'inci uygulama geliştirici ile diğer geliştiriciler arasındaki tutarlılık yüzdesel olarak Çizelge 6. 2'de gösterilmektedir.

Çizelge 6. 2 Uygulama geliştiricileri arasındaki tutarlılık

	#2	#3	#4	#5	#6	#7	#8	#9	#10
SimPack	92,64	90,14	97,60	94,81	82,86	83,58	83,10	95,47	89,40
DNSJava	89,62	90,82	90,30	94,10	92,77	94,05	94,43	97,73	90,3

6.2 Test - 1: Önerdiğimiz Metrik Kümelerinin Yapısal Kod Klon Tespitinde Başarı Oranlarının, Ağırlık Derecelerinin ve Eşik Değerlerinin Bulunması

Geliştirilen uygulama içerisindeki metrikler, herhangi iki sınıfın birbiri arasındaki yapısal klon tespitini yapmak için (4.2) ve (4.3) 'de verilen coisine ve jaccard benzerlik yöntemleri ile kullanılmaktadır. Bu yönteme göre herhangi iki nesnenin Sonar

tarafından ölçülen ölçüm sonuçları iki farklı vektör içerisinde tutulmaktadır. Bu iki vektör, Jaccard ve Coisine benzerlik yöntemine göre ayrı ayrı karşılaştırılmaktadır ve sonuç her ikisinde de belirli bir eşik değerinin üzerinde ise karşılaştırılan nesneler yapısal klon çifti olarak kaydedilmektedir. Bu hesaplama yöntemi, Bölüm 4. 2'de detaylı olarak anlatılmaktadır.

6.2.1 Metrik Kümelerinin Ağırlıklarının Belirlenmesi

Önerilen her bir metrik kümesi için, yöntemin verdiği sonuçlar gözle değerlendirme kümesi ile karşılaştırılarak her metrik kümesinin ağırlığı belirlenmektedir. Bu başarı oranları metrik kümelerinin ağırlıklarının belirlenmesinde kullanılmaktadır. Metrik kümelerinin ağırlıklarının belirlenmesi için seçilen 6 farklı klon çifti adayı, ve bu adayların herbir metrik kümesi için benzerlik değerleri ile geliştirici değerlendirme kümesindeki değeri Çizelge 6. 3'de gösterilmektedir.

Çizelge 6. 3 Örnek benzerlik faktörleri ve geliştirici değerlendirmesi

Aday No	**Kalıtım**	**Bağlaşım**	**Karmaşık**	**Kapsülleme**	**Çok Biçimlilik**	**Boyut**	**Göz ile seçilmiş**
1	0,6	0,82	0,98	0,00	1,00	1,00	1
2	0,73	1,00	1,00	0,98	1,00	0,99	1
3	0,76	0,99	1,00	0,57	1,00	1,00	1
4	0,78	1,00	0,80	0,41	1,00	1,00	1
5	0,85	0,98	1,00	0,50	1,00	0,99	1
6	0,91	0,89	0,98	0,32	1,00	1,00	1
Toplam	1	0,75	0,35	0,43	0,48	0,32	

Metrik kümeleri ile geliştirici değerlendirme kümesi arasındaki doğruluğun hesaplanması için (6.1) kullanılmaktadır. Eğer bir metrik kümesi sonucu geliştirici değerlendirme kümesine yakınsa, yakın olan bu metrik kümesinin önemli olduğu öngörülmektedir. Bu öngörüye dayalı olarak her bir metriğin hesaplamalardaki ağırlığı,

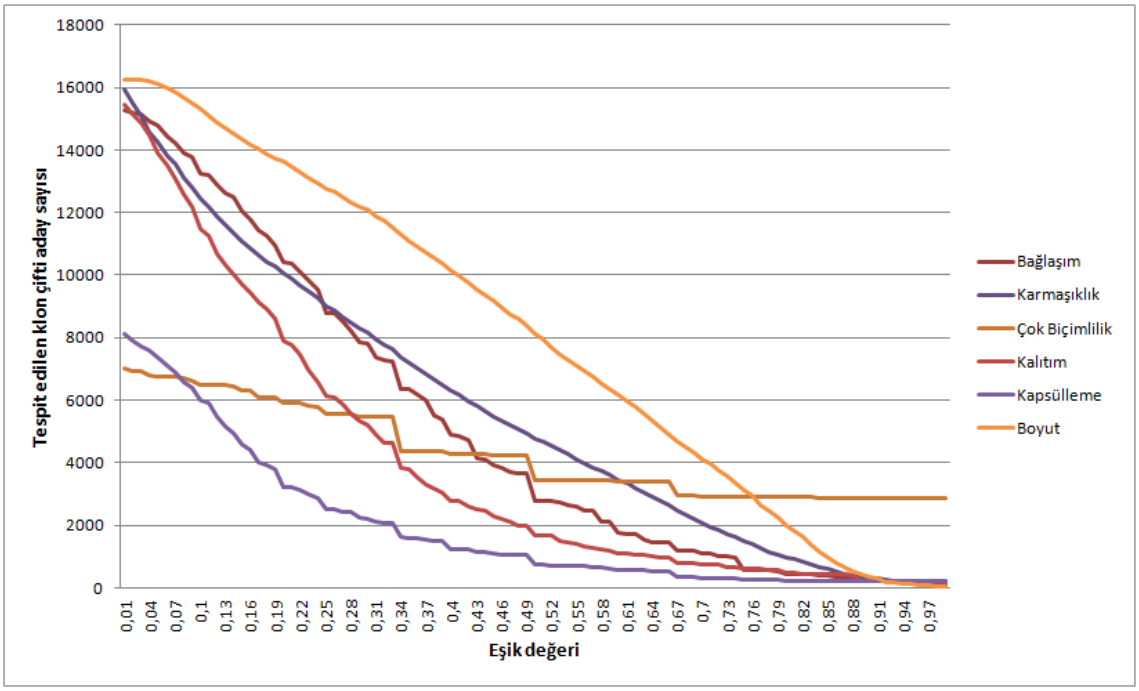
başarı oranına göre belirlenmektedir. Bu durum Çizelge 6. 2'de toplam değerleri alınarak gösterilmektedir. Herbir metrik kümesinin doğruluğu, o kümenin önemini göstermektedir. Metrik kümelerinin doğrulukları ilgili eşik değerleri ile Çizelge 6. 4'de gösterilmektedir. Eğer bir metrik kümesinin doğruluğu yüksek çıkıyorsa, bu küme toplam benzerliği bulmada daha fazla katkı sağlamaktadır. Metrik kümelerindeki en iyi benzerliği elde etmek için ise eşik değeri 0,1'den 0,99'a kadar 0.01 artırılarak değiştirilmektedir.

Çizelge 6. 4 Metrik kümesine göre en yüksek doğruluk – eşik değeri

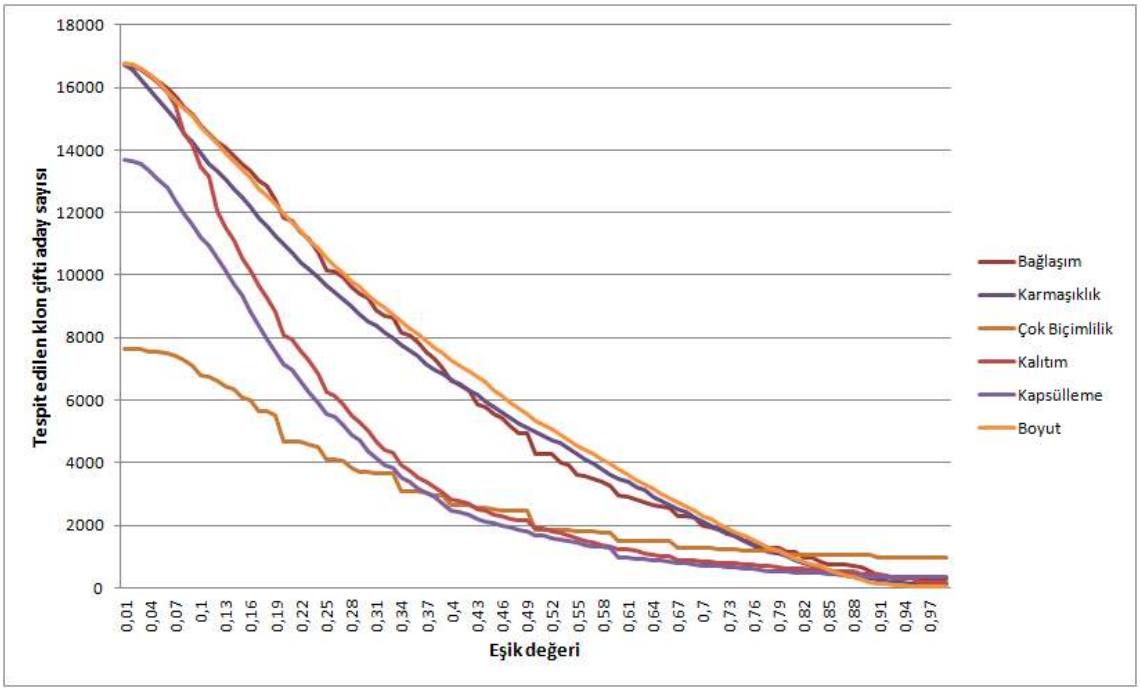
	SimPack Ağırlıkları		**DNSjava Ağırlıkları**	
Metrik Kümesi	**Eşik Değeri**	**Doğruluk**	**Eşik Değeri**	**Doğruluk**
Bağlaşım	0,69	54,37302424	0,81	66,2835249
Karmaşıklık	0,75	25,27272727	0,75	33,14332248
Çok Biçimlilik	0,89	35,13062813	0,91	59,05759162
Kalıtım	0,72	72,62723521	0,67	66,30552546
Boyut	0,9	23,59550562	0,79	33,08128544
Kapsülleme	0,5	31,21546961	0,62	56,75583864

6.2.2 Eşik Değerinin Belirlenmesi

Birbirinden farklı metrik kümeleri için en iyi sonucu veren eşik değerlerinin farklı olabileceği düşünülerek 0.01'den 0.99'a kadar 99 adet eşik değer için Bölüm 4. 2'de anlatılan (4.4)'e göre ölçüm sonuçları alınmıştır. Eşik değeri 0.01'den 0.99'a doğru ilerledikçe bulunan yapısal klon aday çiftinin sayısı metrik kümelerine göre farklı oranlarda azalış göstermektedir. Bu durum SimPack [103] projesi için Şekil 6. 1'de gösterilirken, DNSJava projesi için Şekil 6. 2'deki gibidir.

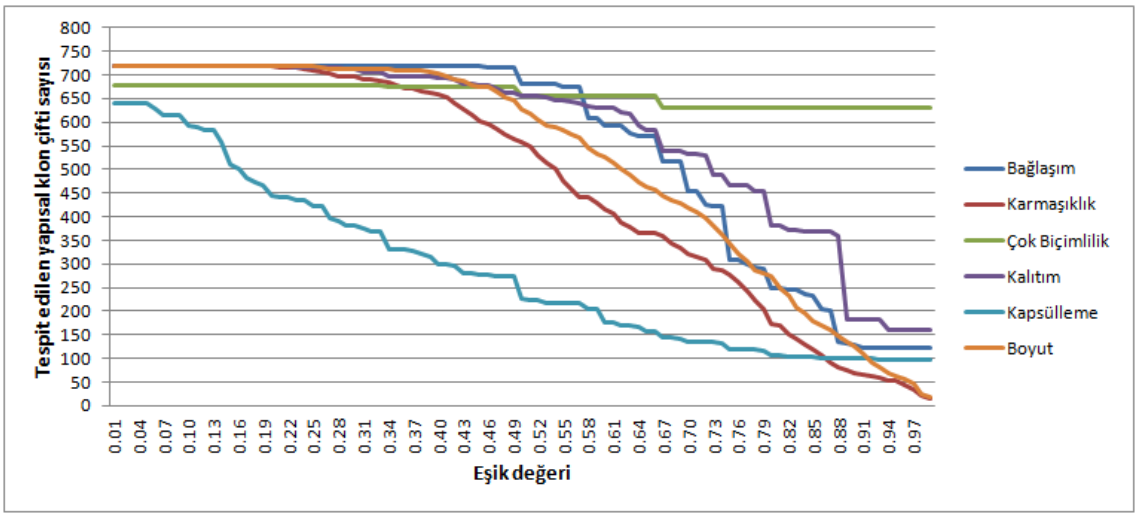


Şekil 6. 1 SimPack eşik değeri – bulunan klon sayısı ilişkisi

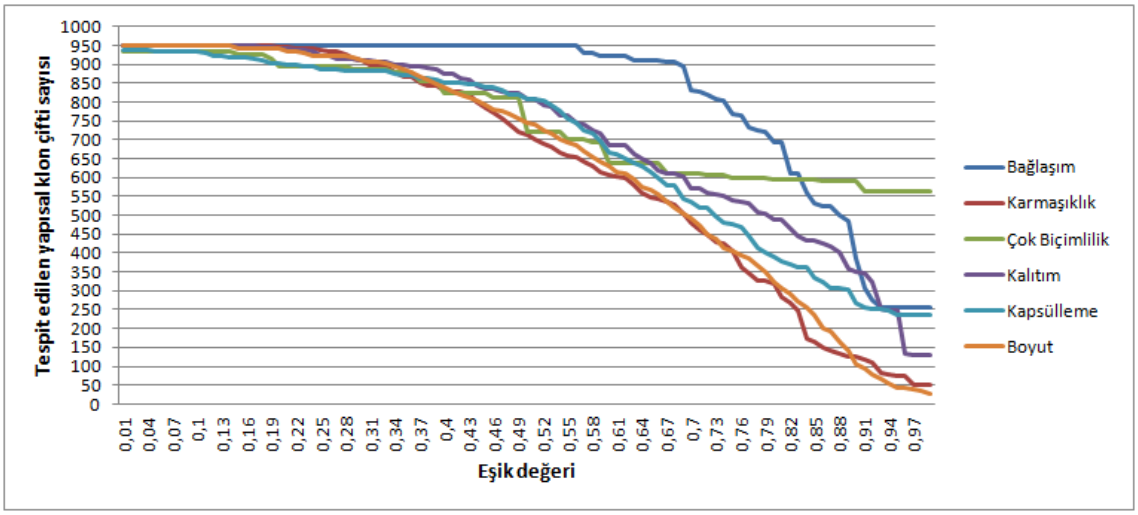


Şekil 6. 2 DNSJava eşik değeri – bulunan klon sayısı ilişkisi

Her bir metrik kümesinin, 0.01'den 0.99'a kadar olan her bir eşik değerine göre bulduğu yapısal klon aday çiftleri (6.1) kullanılarak, uygulama geliştiriciler tarafından bulunan sonuçlarla karşılaştırılmaktadır. Verilen eşik değerine göre yazılımın bulduğu yapısal kod klon çiftlerinin sayısı A ile gösterilirken, B gözle tespit yönteminin bulduğu aday sayısını vermektedir. $A \cap B$ ise A ile B'nin bulduğu ortak klon sayısını vermektedir. Ortaya çıkan sonuç ise, verilen eşik değeri için doğruluk oranını vermektedir.



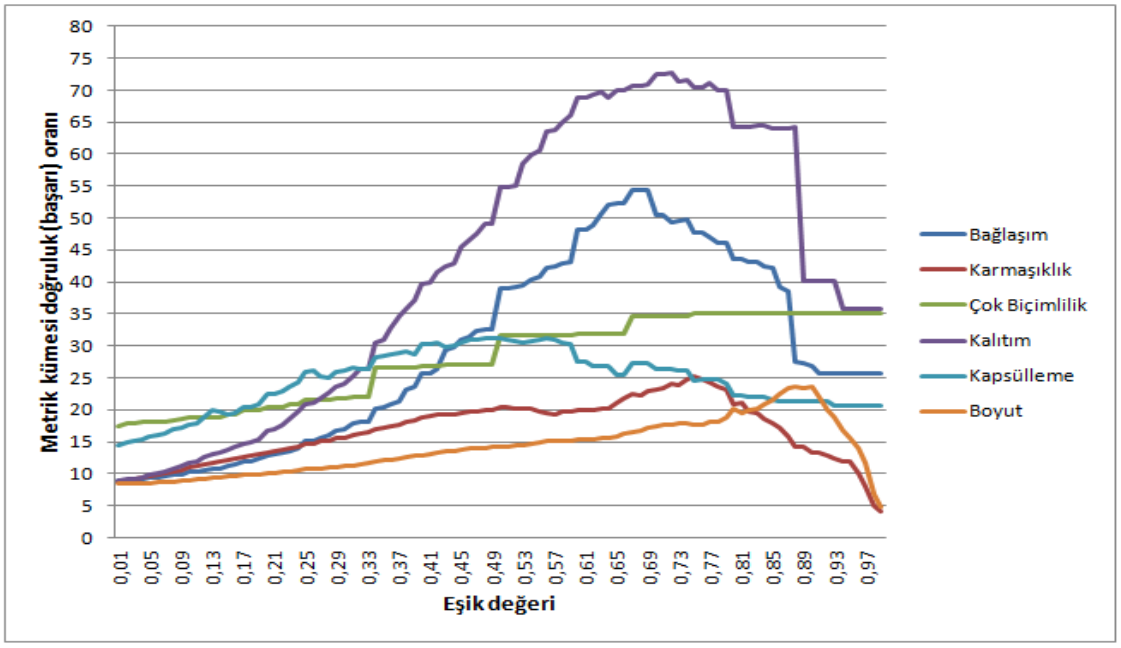
Şekil 6. 3 SimPack eşik değeri – ortak bulunan klon sayısı ilişkisi



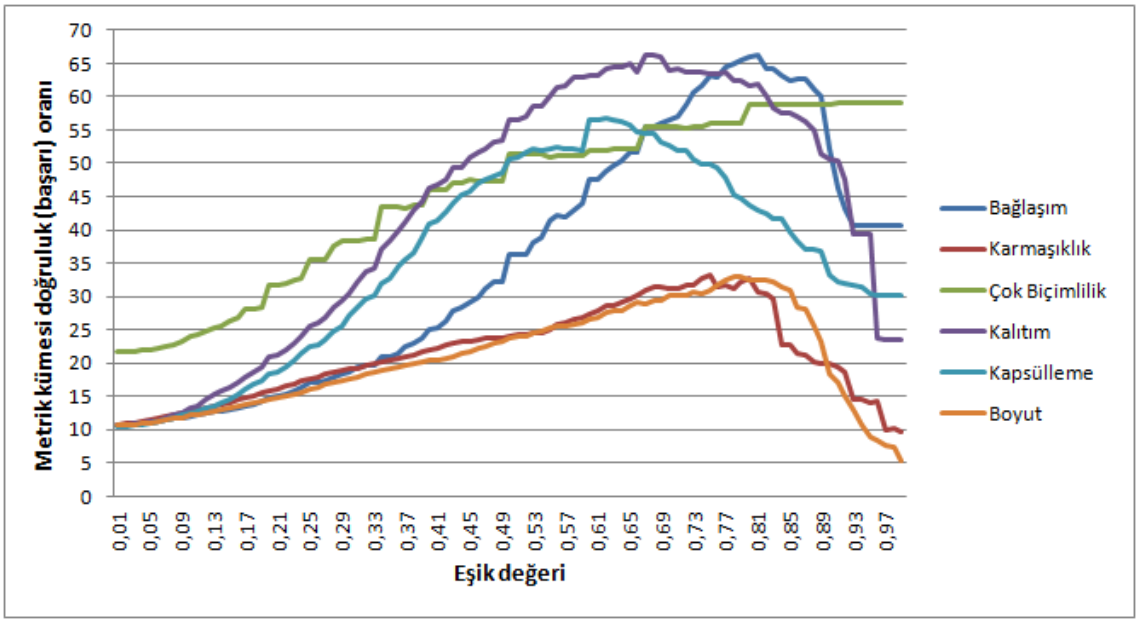
Şekil 6. 4 DNSJava eşik değeri – ortak bulunan klon sayısı ilişkisi

Farklı eşik değerine göre program ve geliştiricilerin ortak olarak bulduğu klon çiftinin sayıları arasındaki ilişki SimPack projesi için Şekil 6. 3'te gösterilirken, DNSJava projesi

için Şekil 6. 4 ‘de gösterilmektedir. Kalıtım ve Bağlaşım metrik kümelerinin daha iyi ortak sonuçlar bulduğu gözlenmektedir. Bu sonuçlar ışığında ortaya çıkan metrik kümesi bazlı eşik değeri ve doğruluk ilişkisi SimPack projesi için Şekil 6. 5’te ve DNSJava projesi için Şekil 6. 6’da gösterilmektedir.



Şekil 6. 5 SimPack projesi eşik değeri - doğruluk oranı ilişkisi



Şekil 6. 6 DNSJava projesi eşik değeri - doğruluk oranı ilişkisi

Eşik değerinin 0.60 ile 0.80 arasında olduğu durumlarda kalıtım metrik kümesinin en yüksek doğruluk oranına ulaştığı gözlemlenmektedir. Bu değeri takip eden en iyi metrik kümesi ise Bağlaşım metrik kümesidir.

6.3 Test – 2: Tüm Metrik Kategorilerinin Beraber Kullanıldığında Başarı Oranlarının Bulunması

Bu bölümde, önerilen yöntemin doğruluğu gözle tespit yönteminde bulunan sonuçlarla karşılaştırarak test edilmektedir. İlk olarak iki farklı proje için, SimPack ve DNSJava, doğrulukları ve eşik değerleri hesaplanmaktadır. Sonrasında Çizelge 6. 4'de gösterilen metrik kümeleri için belirlenen ağırlık değerlerine göre, her iki proje içinde tüm metrikler bir arada kullanılarak doğruluk değerleri hesaplanmaktadır. Hesaplanan bu değerler Çizelge 6. 5'de gösterilmektedir.

Çizelge 6. 5 Tüm metrik kümelerinin kullandığı durumda doğruluk

Project	**SimPack**	**DNSJava**
Doğruluk	72,71489362	71,94961952

6.4 Test – 3: Bir Proje için Elde Edilen Eşik Değerlerinin Başka Bir Projede Kullanıldığında, Yapısal Kod Klon Tespitindeki Başarı Oranlarının Belirlenmesi

Bölüm 6. 2'de yapılan test sonucu elde edilen metrik kümesi ağırlıklarının, değerlendirilmesi yapılan projeye bağımlı olup olmadığı test edilmektedir. Bu amaçla değerlendirmesini yaptığımız SimPack projesinden elde edilen doğruluk ve eşik değerleri kullanılarak DNSJava projesi değerlendirilmektedir. Yapılan test sonucunda ortaya çıkan herbir metrik kümesine göre çıkan doğruluk oranları Çizelge 6. 6'da gösterilmektedir.

SimPack projesinden elde edilen metrik ağırlıklarına göre değerlendirilen DNSJava projesi için ortaya çıkan metrik kümelerinin doğruluklarında en iyi sonucu veren Kalıtım metrik kümesi olmaktadır. Bölüm 6. 2'de bulunan metrik ağırlık ve eşik değerlerinin geçerliliğini kanıtlamak amacıyla, DNSJava projesinden elde edilen değerler

kullanılarak SimPack projesi üzerinde ikinci bir test yapılmaktadır. Yapılan bu testin sonuçları Çizelge 6. 7'de gösterilmektedir.

Çizelge 6. 6 DNSJava projesinin SimPack metrik kümesinin değerlerine göre doğruluğu

	DNSjava Sonuçları
Metrik Kümesi	**Doğruluk**
Bağlaşım	56,03502189
Karmaşıklık	33,14332248
Çok biçimlilik	58,94105894
Kalıtım	63,78132118
Boyut	18,27768014
Kapsülleme	50,66921899

Çizelge 6. 7 SimPack projesinin DNSJava metrik kümesinin değerlerine göre doğruluğu

	SimPack Sonuçları
Metrik Kümesi	**Doğruluk**
Bağlaşım	43,55400697
Karmaşıklık	25,27272727
Çok biçimlilik	35,13062813
Kalıtım	70,68062827
Boyut	26,77165354
Kapsülleme	18,8172043

SimPack projesi için elde edilen metrik kümesi ve ağrılıklarının hepsinin bir arada DNSJava projesi için de kullanıldığında doğruluk oranının nasıl değiştiği Çizelge 6. 8'de gösterilmektedir. Çıkan sonuca göre DNSJava projesi için kullanılan metrikler ve ağırlıkları ile SimPack projesi için kullanılan değerler birbiri ile tutarlı bulunmaktadır. DNSJava projesi kendi metrik kümesi ve ağırlıkları ile çalıştırıldığında doğruluk oranı

71,94961952 iken (Çizelge 6. 5'de gösterildiği gibi) SimPack değerleri ile çalıştırıldığında 71,51118934'dir.

Çizelge 6. 8 DNSJava projesinin SimPack tüm değerlerine göre doğruluğu

Project	**DNSJava (Kendi Değerleri)**	**DNSJava (SimPack Değerleri)**
Doğruluk	71,94961952	71,51118934

Elde edilen sonuçların doğruluğunu pekiştirmek için DNSJava projesinin değerlendirilmesinden elde edilen ve Çizelge 6. 4'de gösterilen değerler kullanılarak, SimPack projesi değerlendirilmektedir.

Çizelge 6. 9 SimPack projesinin DNSJava tüm değerlerine göre doğruluğu

Project	**SimPack (Kendi Değerleri)**	**SimPack (DNSJava Değerleri)**
Doğruluk	72,71489362	72,44040473

Çizelge 6. 9'da gösterilen sonuçlara göre, DNSJava projesi için elde edilen değerlere göre değerlendirilen SimPack projesi ile kendi değerlerine göre değerlendirilen SimPack projesinin sonuçları tutarlılık göstermektedir.

6.5 Test – 4: Yapısal Kod Klon Tespitinde En iyi Sonuçları Veren Metrik Kombinasyonlarının Bulunması

Bölüm 4. 1'de yapısal kod klon için kullanılabilecek birden çok metrik kümesi tanıtılmaktadır. Bölüm 6. 2'de yapılan testlerin sonucunda, kalıtım ve bağlaşım metrik kümelerinin kullanılması ile yapısal kod klon tespitinin başarı oranları diğer metriklere göre yüksek olmaktadır. Tüm metrik kümelerinin beraber kullanılması ile elde edilen başarı oranı ise, tek tek kullanımdan elde edilen başarı oranından daha düşüktür. Bu sonuçtan yola çıkılarak, kalıtım ve bağlaşım metrik kümelerinin yanında daha fazla metrik kümesinin kullanılmasının doğruluğu artırıp artırmadığı araştırılmaktadır. Bu bağlamda, farklı farklı metrik kümesi kombinasyonları kullanılarak doğruluk sonuçları elde edilmektedir. SimPack projesi için en iyi sonucu veren ilk 10 kombinasyon ve toplam benzerlik (sıra 57) Çizelge 6. 10'da gösterilmektedir.

Çizelge 6. 10 SimPack projesi için en iyi sonucu veren ilk 10 metrik kombinasyonu

Sıra	**Metrik Kombinasyonu**	**Doğruluk**
1	Kalıtım, Çok biçimlilik	90,32258065
2	Kalıtım, Bağlaşım	86,89839572
3	Kalıtım, Bağlaşım, Çok biçimlilik	86,1663286
4	Bağlaşım, Çok biçimlilik, Kapsülleme	80,08998875
5	Kalıtım, Çok biçimlilik, Kapsülleme	78,14930016
6	Kalıtım, Çok biçimlilik, Kapsülleme, Bağlaşım	77,79295275
7	Kalıtım, Bağlaşım, Kapsülleme	77,69443348
8	Kalıtım, Bağlaşım, Çok biçimlilik, Karmaşıklık	76,89383655
9	Kalıtım, Bağlaşım, Çok biçimlilik, Boyut	75,41944075
10	Kalıtım, Bağlaşım, Kapsülleme, Karmaşıklık	75,11737089
57	Kalıtım, Bağlaşım, Çok biçimlilik, Kapsülleme, Karmaşıklık, Boyut	72,71489362

DNJSJava projesi için birbirinden farklı metrik kümelerinin bir arada kullanılması sonucu elde edilen en iyi ilk 10 metrik kümesi kombinasyonu Çizelge 6. 11'de gösterilmektedir.

Çizelge 6. 11 DNSJava projesi için en iyi sonucu veren ilk 10 metrik kombinasyonu

Sıra	**Metrik Kombinasyonu**	**Doğruluk**
1	Bağlaşım, Kapsülleme	85,68486097
2	Kalıtım, Bağlaşım	85,1445663
3	Bağlaşım, Çok biçimlilik	84,10661402
4	Kalıtım, Çok biçimlilik, Bağlaşım	81,58379374
5	Kalıtım, Kapsülleme, Bağlaşım	80,91926217
6	Çok biçimlilik, Kapsülleme, Bağlaşım	80,07279345
7	Kalıtım, Bağlaşım, Çok biçimlilik, Kapsülleme	78,46556234
8	Kapsülleme, Bağlaşım, Boyut	77,37804878
9	Kalıtım, Karmaşıklık, Kapsülleme	76,46528404
10	Kalıtım, Bağlaşım, Çok biçimlilik, Karmaşıklık	75,20232413
57	Kalıtım, Bağlaşım, Çok biçimlilik, Kapsülleme, Karmaşıklık, Boyut	71,94961952

Doğruluk sonuçlarında da belirtildiği gibi tüm metrik kümelerinin birarada kullanılması en iyi doğruluğu vermemektedir (sıra 57). En iyi doğruluk oranı toplam doğruluk oranından daha yüksektir. Önem derecesi kalıtım ve Bağlaşımdan sonra gelen çok biçimlilik ve kapsülleme, en iyi doğruluk veren kombinasyonlarda üst sıralarda yer almaktadır.

6.6 Değerlendirme

Bu bölümde, yapısal kod klon tespitinde kullanılması önerilen metrik kümelerinin ne oranda başarılı olduğu irdelenmektedir. Değerlendirme aşamasında insan deneklerle yapılan değerlendirme sonuçlarıyla uygulamanın bulmuş olduğu sonuçlar karşılaştırılarak, başarı oranları tespit edilmektedir. Bölüm 6. 2'de yapılan ilk testin sonuçlarına göre test edilen her iki proje için kalıtım metrik kümesi en yüksek doğruluk değerini vermektedir. Kalıtım metrik kümesini takip eden en iyi metrik kümesi ise Çizelge 6. 4'de gösterildiği gibi Bağlaşım metrik kümesidir. Her bir metrik kümesinin farklı eşik değerlerde en iyi doğruluk sonucunu verdiği Şekil 6. 5'de ve Şekil 6. 6 'da gösterilmektedir. Bölüm 6. 3'de yapılan ikinci test ile tüm bu metrik kümelerinin beraber kullanıldığında ortaya çıkan doğruluk oranı araştırılmaktadır. SimPack ve DNSJava projesi için tüm metrik kümeleri hep beraber kullanıldıklarında ortaya çıkan doğrulukları Çizelge 6. 5'de gösterilmektedir. Tüm metrikler kullanıldığında SimPack ve DNSJava için yapısal kod klon tespitinde yaklaşık olarak %70 oranında başarı sağlanmaktadır. Bölüm 6. 4'de yapılan üçüncü testte önerilen metrik kümelerinin değerlendirilmesi yapılan proje ile olan bağımlılığı araştırılmaktadır. SimPack projesinden elde edilen metrik ağırlıkları ve eşik değerleri DNSJava projesi için kullanılırken, DNSJava projesinden elde edilen değerler SimPack projesi için kullanılmaktadır. Yapılmış olunan üçüncü testin sonucuna göre DNSJava ve SimPack projelerinin doğruluk oranları %70 civarındadır ve Çizelge 6. 8 ile Çizelge 6. 9'da gösterilmektedir. Bu sonuçlara göre elde edilen bu değerler farklı projeler için de tutarlı görünmektedir. Bölüm 6. 5'de yapılan dördüncü test ile hangi metrik kümelerinin beraber kullanıldıklarında en başarılı sonuçları verdiği araştırılmaktadır. Çizelge 6. 10'da gösterildiği üzere SimPack projesi için kalıtım ve çok biçimlilik metrik kümelerinin beraber kullanılması en iyi sonucu vermektedir. DNSJava projesi için en iyi sonucu

veren kombinasyon Bağlaşım ve Kapsülleme metrik kümesinden oluşmaktadır ve Çizelge 6. 11'de gösterilmektedir. Sonuç olarak; kalıtım, Bağlaşım, çok biçimlilik ve Kapsülleme metrik kümelerinden oluşan kombinasyonların kullanılmasıyla ortaya çıkan doğruluk oranları, tüm metriklerin bir arada kullanılmasıyla ortaya çıkan doğruluk oranından yüksektir. Yapılan tüm bu testlerin sonucunda, değerlendirilmesi yapılan projeden bağımsız olarak belirlenen metrik ağırlıkları ve eşik değerlerinin kullanımı ile yapısal kod klon tespitinde %70 oranında başarı sağlanmaktadır.

BÖLÜM 7

SONUÇLAR VE GELECEKTE PLANLANAN ÇALIŞMALAR

Bu tez çalışmasında, yazılım projelerinde sıklıkla karşılaşılan kod klonlar incelenmiştir ve bir kod klon çeşidi olan yapısal kod klonları tespit etmek için metrik tabanlı yenilikçi bir çözüm getirilmiştir. Önerilen metrik tabanlı yapısal kod klon tespiti sayesinde yazılım projelerinde bulunan yapısal klonlar tespit edilmektedir.

Önermiş olduğumuz yöntemi gerçekleştirmek için Sonar açık kaynaklı kalite ölçüm aracına bütünleşmiş (entegre) edilen bir yazılım geliştirilmiştir [107]. Geliştirilen uygulama sayesinde, yazılım kalite kontrol çalışanları/ yöneticileri yapısal kod klon analizi yapabilmektedir.

Geliştirilen uygulamanın doğruluğunu değerlendirmek için yapılan testlerde, uygulamanın yapısal kod klon tespit işlemlerinde % 70'in üzerinde başarı sağladığı gözlemlenmiştir.

7.1 Araştırma Sorularına Cevaplar

Bölüm 3. 1'de ortaya konulan araştırma soruları, Bölüm 6'da yapılan testler sonucunda bulunan bulgular ile takip eden kısımlarda yanıtlanmaktadır.

7.1.1 Nesneye dayalı programlama ile geliştirilmiş yazılımlarda metrik tabanlı olarak yapısal klon tespiti yapılabilir mi? Metrik tabanlı yapısal kod klon tespiti yöntemleri yapısal kod klon tespitinde yeterli midir?

Geliştirilen yazılım projesi ile yapılan testler sonucunda, yapısal kod klon tespiti için metrik tabanlı yöntemlerin yeterli olacağı kanıtlanmaktadır. Bölüm 6. 4'de Test-4

sonucunda kalıtım, çok biçimlilik ve Bağlaşım metrik kümesinin %70'in üzerinde doğruluk oranı ile en başarılı sonuçları verdiği görülmektedir.

7.1.2 Nesneye dayalı programlama ile geliştirilmiş yazılımlarda metrik tabanlı yapısal klon tespiti yapılabilmesi için ölçülmesi gereken metrik kategorileri ve bu kategorilerdeki metrikler nelerdir?

Bu tez kapsamında yapısal kod klon tespiti için 6 farklı metrik kategorisi önerilmektedir. Bunlar: Bağlaşım, karmaşıklık, çok biçimlilik, kalıtım, kapsülleme ve boyuttur. Bu metrik kümeleri ve her bir kümede hangi metrikler olduğu Bölüm 4. 1'de anlatılmaktadır. Bölüm 6. 2'de verilen testler sonucunda seçilmiş olan metriklerin %70'in üzerinde doğruluk oranına ulaştığı gösterilmektedir.

7.1.3 Metrik ölçüm sonuçlarının sayısal veriler olduğu düşünüldüğünde, yapısal klon tespiti için kullanılabilecek hesaplama yöntemleri nelerdir?

Sonar gibi kalite yönetim araçlarının ölçmüş olduğu metrik sonuçları sayısal verilerden oluşmaktadır. Statik kod analizi sonucu oluşan ölçüm değerleri, ölçümün alındığı yazılım parçaçığını yapısal mimarisi açısından açıklayıcı değere sahiptir. Bu tez kapsamında yapısal kod klon tespiti için benzerlik hesaplama yöntemlerinden Jaccard ve Coisine benzerlik yöntemleri birlikte kullanılmıştır. Yapısal kod klon tespiti için kullanılan hesaplama yöntemi bu iki hesaplama yöntemine dayanmaktadır ve Bölüm 4.2.1'de detayları ile anlatılmaktadır. Bölüm 6'da detayları verilen testler sonucu kullanılan klon tespit hesaplama yönteminin kabul edilebilir doğruluk yüzdelerine ulaştığı gösterilmektedir.

7.1.4 Hangi metriklerin yapısal kod klon analizi tespitinde daha fazla önemi vardır?

Gerçekleştirilen uygulama sayesinde, Bölüm 4.1.4 'de önerilen metrikler ayrı ayrı olarak test edilmiştir. Yapılan testlerin güvenilirliğini artırmak için bu testler SimPack [103] ve DNSJava [104] adındaki iki farklı proje üzerinde de gerçekleştirilmiştir. Bölüm 7'de bu testler ile ilgili aşamalar anlatılmaktadır. Yapılan testler sonucunda, Çizelge 6. 4 'de gösterildiği gibi Kalıtım ve Bağlaşım metrik kümelerinin diğer metrik kümelerine göre daha iyi sonuç verdiği tespit edilmiştir.

7.1.5 Yapısal kod klon tespitinde kullanılan diğer yöntemlerle metrik tabanlı tespit yöntemleri karşılaştırıldığında nasıl sonuçlar elde edilmektedir?

Yapısal kod klon tespitinde, metrik tabanlı yöntemden farklı olarak veri madenciliğine dayanan bir yöntem mevcuttur [18], [38]. Önerilen bu yöntem, öncelikle basit klonların tespit edilmesine dayanmaktadır. Tespit edilen basit klonlar, veri madenciliği tekniği kullanılarak gruplanmakta ve bu gruplar yapısal kod klonun tespiti için kullanılmaktadır. Yapısal klon tespit aşamasında verilen sayıda basit klon grubunu barındıran nesneler yapısal klon olarak adlandırılmaktadır. Önerilen metrik tabanlı yönteme göre eksiği, basit klonlara dayanmış olmasıdır. Bölüm 2.2.1'de anlatıldığı üzere birbirinden farklı basit klon bulunmaktadır ve [18], [38]'da önerilen yöntem bunlardan sadece ilk ikisini kapsamaktadır. Önerilen yöntem metriklere dayandığı için, bütün basit klonları kapsayabilmektedir. Bunun yanı sıra veri madenciliğine dayalı yöntemler, kaynak kodların karşılaştırılmasına dayanmaktadır. Bu da tespit yöntemlerinin daha fazla sürede sonuca ulaşmasına sebep olmaktadır. Bu tezde önerilen yöntem yazılımlardan doğru metrikler için alınan ölçüm değerlerinin karşılaştırılması sounucunda yapısal kod klonların tespitini daha kısa sürede sağlamaktadır.

7.2 Gelecekte Planlanan Çalışmalar

Bu tez ile ortaya konulan, nesneye dayalı programlama dilleri için metrik tabanlı yapısal kod klon analizinin sadece sınıflar arasında değil, yazılım paketleri arasında da yapılması planlanmaktadır. Yapısal kod klon analizi için yapılmış olan veri madenciliği tabanlı çalışmalarla, metrik tabanlı yapısal kod klon analizi çalışmasının karşılaştırılmalı olarak incelenmesi planlanmaktadır.

KAYNAKLAR

[1] Fowler, M., (1999). Refactoring: improving the design of existing code, Addison-Wesley Professional, New York.

[2] Du Bois, B. Demeyer, S. ve Verelst, J., (2004). "Refactoring-improving coupling and cohesion of existing code", Reverse Engineering, 2004. Proceedings. 11th Working Conference on, 8-12 Kasım 2004, Delft.

[3] Roy, C.K. ve Cordy, J.R., (2007). A survey on software clone detection research: Technical Report 541, Queen's University at Kingston.

[4] Baker, B.S., (1995). "On finding duplication and near-duplication in large software systems", Reverse Engineering, 2004. Proceedings. 11th Working Conference on, 14-16 Temmuz 1995, Toronto.

[5] Casazza, G. Antoniol, G. Villano, U. Merlo, E. ve Di Penta, M., (2001). "Identifying clones in the linux kernel", Source Code Analysis and Manipulation, 2001 Proceedings First IEEE International Workshop on, 2001, Florence.

[6] Kamiya, T. Kusumoto, S. ve Inoue, K., (2002). "CCFinder: a multilinguistic token-based code clone detection system for large scale source code", Software Engineering, IEEE Transactions on, 28: 654-670.

[7] Kontogiannis, K., (1997). "Evaluation experiments on the detection of programming patterns using software metrics", Reverse Engineering, 2004. Proceedings. Fourth Working Conference on, 6-8 Ekim 1997, Amsterdam.

[8] Li, Z. Lu, S. Myagmar, S. ve Zhou, Y., (2006). "CP-Miner: Finding copy-paste and related bugs in large-scale software code", Software Engineering, IEEE Transactions on, 32: 176-192.

[9] Jiang, L. Misherghi, G. Su, Z. ve Glondu, S., (2007). "Deckard: Scalable and accurate tree-based detection of code clones", Proceedings of the 29th international conference on Software Engineering, 20-26 May 2007, Minneapolis.

[10] Lague, B. Proulx, D. Mayrand, J. Merlo, E.M. ve Hudepohl, J., (1997). "Assessing the benefits of incorporating function clone detection in a

development process" Software Maintenance, 1997 Proceedings, International Conference on, 1-3 Ekim 1997, Bari.

[11] Baxter, I.D. Yahin, A. Moura, L. Sant'Anna, M. ve Bier, L., (1998). "Clone detection using abstract syntax trees", Software Maintenance, 1998 Proceedings, International Conference on, 16-20 Kasım 1998, Bethesda .

[12] Mayrand, J. Leblanc, C. ve Merlo, E.M., (1996). "Experiment on the automatic detection of function clones in a software system using metrics", Software Maintenance 1996, Proceedings, International Conference on, 4-8 Kasım 1996, Monterey.

[13] Kapser, C.J. ve Godfrey, M.W., (2006). "Supporting the analysis of clones in software systems", Journal of Software Maintenance and Evolution: Research and Practice, 18: 61-82.

[14] Rysselberghe, F.V. ve Demeyer, S., (2004). "Evaluating clone detection techniques from a refactoring perspective", Proceedings of the 19th IEEE international conference on Automated software engineering, 20-24 Eylül 2004.

[15] Antoniol, G. Villano, U. Merlo, E. ve Di Penta, M., (2002). "Analyzing cloning evolution in the linux kernel", Information and Software Technology, 44: 755-765.

[16] Johnson, J.H., (1993). "Identifying redundancy in source code using fingerprints", Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering-Volume 1; 1993: IBM Press.

[17] Grubb, P. ve Takang, A.A., (2003). Software maintenance: concepts and practice: World Scientific.

[18] Basit, H.A. Ali, U. ve Jarzabek, S., (2011). "Viewing simple clones from structural clones' perspective", Proceedings of the 5th International Workshop on Software Clones; 23 Mayıs 2011, Honolulu.

[19] Roy, C.K. Cordy, J.R. ve Koschke, R., (2009). "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach", Science of Computer Programming, 74: 470-495.

[20] Koschke, R., (2007). Survey of research on software clones: Internat. Begegnungs-und Forschungszentrum für Informatik.

[21] Astekin, M. ve Sozer, H., (2012). "Utilizing Clone Detection for Domain Analysis of Simulation Systems", Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on, 20-24 Agustos 2012, Helsinki.

[22] Smith, R. ve Horwitz, S., (2009). "Detecting and measuring similarity in code clones", Proceedings of the International Workshop on Software Clones (IWSC), 24 Mart 2009, Kaiserslautern.

[23] Juergens, E. Deissenboeck, F. ve Hummel, B., (2009). "CloneDetective-A workbench for clone detection research", Proceedings of the 31st International Conference on Software Engineering, 16-24 Mayıs 2009, Vancouver.

[24] Kodhai, E. Kanmani, S. Kamatchi, A. Radhika, R. ve Vijaya Saranya, B., (2010). "Detection of Type-1 and Type-2 code clones using textual analysis and metrics", Recent Trends in Information, Telecommunication and Computing (ITC), 2010 International Conference on, 12-13 March 2010, Kerala.

[25] Bari, M.A. ve Ahamad, D.S., (2011). "Code Cloning: The Analysis, Detection and Removal", International Journal of Computer Applications (0975–8887) Volume.

[26] Baker, B.S., (1993). "A program for identifying duplicated code", Computing Science and Statistics: 49-49.

[27] Ducasse, S. Nierstrasz, O. ve Rieger, M., (2004). "Lightweight detection of duplicated codea language-independent approach", Institute for Applied Mathematics and Computer Science, University of Berne, Switzerland: 0.

[28] Ducasse, S. Rieger, M. ve Demeyer, S., (1999). "A language independent approach for detecting duplicated code", Software Maintenance, 1999(ICSM'99) Proceedings IEEE International Conference on, 30 Aug.- 3 Sep 1999, Oxford.

[29] Baker, B.S., (1993). "On finding duplication in strings and software", submitted for publication.

[30] Li, Z. Lu, S. Myagmar, S. ve Zhou, Y., (2004). "CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code", OSDI,2004, Urbana.

[31] Raza, A. Vogel, G. ve Plödereder, E., (2006). Bauhaus–a tool suite for program analysis and reverse engineering, ed. Reliable Software Technologies–Ada-Europe 2006. Springer, 71-82.

[32] Komondoor, R. ve Horwitz, S., (2001). Using slicing to identify duplication in source code, ed. Static Analysis. Springer, 40-56.

[33] Komondoor, R.V., (2003). Automated duplicated-code detection and procedure extraction, UNIVERSITY OF WISCONSIN.

[34] Gallagher, K. ve Layman, L., (2003). "Are decomposition slices clones?", Program Comprehension, 2003 11th IEEE International Workshop on, 10-11 Mayıs 2003, Portland.

[35] Chen, W.-K. Li, B. ve Gupta, R., (2003). Code compaction of matching single-entry multiple-exit regions, ed. Static Analysis. Springer, 401-417.

[36] Patenaude, J.-F. Merlo, E. Dagenais, M. ve Laguë, B., (1999). "Extending software quality assessment techniques to java systems", Program Comprehension, 1999 Proceedings Seventh International Workshop on, 1999, Pittsburgh.

[37] Raheja, K. ve Tekchandani, R., (2013). "An Emerging Approach towards Code Clone Detection:Metric Based Approach on Byte Code", International Journal of Advanced Research in Computer Science and Software Engineering, 3.

[38] Basit, H.A. ve Jarzabek, S., (2005). "Detecting higher-level similarity patterns in programs", SIGSOFT Software Engineering Notes, 2005, ACM.

[39] De Lucia, A. Francese, R. Scanniello, G. ve Tortora, G., (2004). "Reengineering web applications based on cloned pattern analysis", Program Comprehension, 2004 Proceedings 12th IEEE International Workshop on, 24-26 Haziran 2004, Bari.

[40] Ueda, Y. Kamiya, T. Kusumoto, S. ve Inoue, K., (2002). "Gemini: Maintenance support environment based on code clone analysis", Software Metrics, 2002 Proceedings Eighth IEEE Symposium on, 4-7 Haziran 2002, Ottawa.

[41] Marcus, A. ve Maletic, J.I., (2001). "Identification of high-level concept clones in source code", Automated Software Engineering, 2001(ASE 2001) Proceedings 16th Annual International Conference on, 26-29 Kasım 2001, Chicago.

[42] Gil, J.Y. ve Maman, I., (2005). "Micro patterns in Java code", SIGPLAN Notices, Ekim 2005, New York.

[43] Shi, N. ve Olsson, R.A., (2006). "Reverse engineering of design patterns from java source code", Automated Software Engineering, 2006. ASE '06. 21st IEEE/ACM International Conference on, 18-22 Eylül 2006, Tokyo.

[44] Wren, A., (2007). "Relationships for object-oriented programming languages", University of Cambridge, Computer Laboratory, Technical Report.

[45] Pressman, R.S. ve Jawadekar, W.S., (1987). "Software engineering", New York 1992.

[46] Fenton, N.E. ve Pfleeger, S.L., (1998). Software metrics: a rigorous and practical approach: PWS Publishing Co.

[47] Chidamber, S.R. ve Kemerer, C.F., (1994). "A metrics suite for object oriented design", Software Engineering, IEEE Transactions on, 20: 476-493.

[48] Harrison, R. Counsell, S.J. ve Nithi, R.V., (1998). "An evaluation of the MOOD set of object-oriented software metrics", Software Engineering, IEEE Transactions on, 24: 491-496.

[49] Abreu, F.B. Goulão, M. ve Esteves, R., (1995). "Toward the design quality evaluation of object-oriented software systems", Proceedings of the 5th International Conference on Software Quality, 1995, Austin.

[50] Lorenz, M. ve Kidd, J., (1994). Object-oriented software metrics: a practical guide: Prentice-Hall, Inc.

[51] Succi, G. Pedrycz, W. Stefanovic, M. ve Miller, J., (2003). "Practical assessment of the models for identification of defect-prone classes in object-

oriented commercial systems using design metrics", Journal of Systems and Software, 65: 1-12.

[52] Singh, G. Singh, D. ve Singh, V., (2011). "A Study of Software metrics", IJCEM International Journal of Computational Engineering & Management, 11: 22-27.

[53] Basili, V.R. Briand, L.C. ve Melo, W.L., (1996). "A validation of object-oriented design metrics as quality indicators", Software Engineering, IEEE Transactions on, 22: 751-761.

[54] Briand, L.C. Wüst, J. Daly, J.W. ve Victor Porter, D., (2000). "Exploring the relationships between design measures and software quality in object-oriented systems", Journal of Systems and Software, 51: 245-273.

[55] Li, W., (1998). "Another metric suite for object-oriented programming", Journal of Systems and Software, 44: 155-162.

[56] Chatzigeorgiou, A., (2003). "Mathematical assessment of object-oriented design quality", IEEE Transactions on Software Engineering, 29: 1050-1053.

[57] TIOBE, "TIOBE Index for August 2014", http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, 25 Ağustos 2014.

[58] Budd, T. ve Design, R.D., (1997). Introduction to Object Oriented Programming 2E: Addison-Wesley.

[59] Huang, A., (2008). "Similarity measures for text document clustering", Proceedings of the sixth new zealand computer science research student conference, 2008, Christchurch, New Zealand.

[60] Strehl, A. Ghosh, J. ve Mooney, R., (2000). "Impact of similarity measures on web-page clustering", Workshop on Artificial Intelligence for Web Search, 2000.

[61] Xue, Y., (2011). "Reengineering legacy software products into software product line based on automatic variability analysis", Proceedings of the 33rd International Conference on Software Engineering, 21-28 Mayıs 2011, Honolulu.

[62] Bakota, T. Ferenc, R. ve Gyimothy, T., (2007). "Clone smells in software evolution", Software Maintenance, 2007 ICSM 2007 IEEE International Conference on, 2-5 Ekim 2007, Paris.

[63] Goldberg, A. ve Robson, D., (1983). Smalltalk-80: the language and its implementation: Addison-Wesley Longman Publishing Co., Inc.

[64] Van Cutsem, T. Mostinckx, S. Gonzalez Boix, E. Dedecker, J. ve De Meuter, W., (2007). "Ambienttalk: Object-oriented event-driven programming in mobile ad hoc networks", Chilean Society of Computer Science, 2007 SCCC'07 XXVI International Conference of the, 8-9 Kasım 2007, Iquique.

[65] Gellersen, H.-W. ve Gaedke, M., (1999). "Object-oriented web application development", IEEE Internet Computing, 3: 60-68.

[66] Huang, C.-C. ve Liang, W.Y., (2004). "Object-oriented development of the embedded system based on Petri-nets", Computer standards & interfaces, 26: 187-203.

[67] Bettini, L. Bono, V. ve Venneri, B., (2002). "Coordinating mobile object-oriented code", Coordination Models and Languages, 14 Mart 2002, Springer.

[68] Bergel, A. Denier, S. Ducasse, S. Laval, J. Bellingard, F. Vaillergues, P. Balmas, F. ve Mordal-Manet, K., (2009). "Squale–software quality enhancement", Software Maintenance and Reengineering, 2009 CSMR'09 13th European Conference on, 24-27 Mart 2009, Kaiserslautern.

[69] Coverity, (2003). "Coverity: Software Testing and Static Analysis Tools", http://www.coverity.com/, 24 Ağustos 2014.

[70] Cast, "Cast: Improve Code Quality", http://www.castsoftware.com/solutions/improve-code-quality, 24 Ağustos 2014.

[71] Campbell, G. ve Papapetrou, P.P., (2014). SonarQube in action: Manning Publ.

[72] Jacobson, I., (1992). Object oriented software engineering: a use case driven approach, Addison-Wesley Professional; 1 edition (June 30, 1992), New York.

[73] Korson, T.D. ve Vaishnavi, V.K., (1986). "An empirical study of the effects of modularity on program modifiability": Ablex Publishing Corp.

[74] Lenzerini, M. Nardi, D. ve Simi, M., (1991). Inheritance hierarchies in knowledge representation and programming languages: John Wiley and Sons Ltd.

[75] AB, M., (1995). "MySQL is a open-source relational database management system (RDBMS)", www.mysql.com, 15 Ağustos 2014.

[76] Matthews, M. Cole, J. ve Gradecki, J.D., (2003). MySQL and Java developer's guide: John Wiley & Sons.

[77] Dietel, P., (2011). Java how to program, Prentice Hall; 9th edition, New York.

[78] Curtin, M., (1998). "Write once, run anywhere: Why it matters", Technical Article, Oracle, California.

[79] Sarmenta, L.F. ve Hirano, S., (1999). "Bayanihan: Building and studying web-based volunteer computing systems using Java", Future Generation Computer Systems, 15: 675-686.

[80] Bauer, C. ve King, G., (2004). Hibernate in action, Manning Publications (August 1, 2004), New York.

[81] Böck, H., (2011). Java persistence api, ed. The Definitive Guide to NetBeans™ Platform 7. Springer, 315-320.

[82] Johnson, R. Hoeller, J. ve Arendsen, A., (2005). "The Spring Framework", Reference Documentation, Version, 2.

[83] LGPL, G.a., (2013). "eRuby (Embedded Ruby) is a templating system that embeds Ruby into a text document", http://www.startuprocket.com/blog/a-quick-introduction-to-embedded-ruby-erb-eruby, 15 Ağustos 2004.

[84] Burns, E. Schalk, C. ve Griffin, N., (2009). JavaServer Faces 2.0: McGraw-Hill.

[85] Holmes, J. ve Schalk, C., (2006). JavaServer faces: the complete reference: McGraw-Hill, Inc.

[86] Varaksin, O., (2013). PrimeFaces Cookbook: Packt Publishing Ltd.

[87] Goldman, M. ve Katz, S., (2007). MAVEN: Modular aspect verification, ed. Tools and Algorithms for the Construction and Analysis of Systems. Springer, 308-322.

[88] Massol, V. ve O'Brien, T.M., (2005). Maven: A Developer's Notebook: A Developer's Notebook: " O'Reilly Media, Inc.".

[89] Brittain, J. ve Darwin, I.F., (2007). Tomcat: The Definitive Guide: The Definitive Guide: " O'Reilly Media, Inc.".

[90] Kurniawan, B. ve Deck, P., (2004). How Tomcat Works, BrainySoftware (April 1, 2004), New York.

[91] Aho, A.V. ve Ullman, J.D., (1972). The theory of parsing, translation, and compiling: Prentice-Hall, Inc.

[92] Chomsky, N., (1956). "Three models for the description of language", Information Theory, IRE Transactions on, 2: 113-124.

[93] Warth, A. Douglass, J.R. ve Millstein, T.D., (2008). "Packrat parsers can support left recursion", PEPM, 8: 103-110.

[94] Tratt, L., (2010). Direct left-recursive parsing expression grammars, ed^eds.: Tech. Rep. EIS-10-01, School of Engineering and Information Sciences, Middlesex University (October 2010).

[95] Ford, B., (2004). "Parsing expression grammars: a recognition-based syntactic foundation", SIGPLAN Notices, 1 Ocak 2004, New York.

[96] Grimm, R., (2006). "Better extensibility through modular syntax", ACM SIGPLAN Notices; 6 Haziran 2006, New York.

[97] Bruneton, E. Lenglet, R. ve Coupaye, T., (2002). "ASM: a code manipulation tool to implement adaptable systems", Adaptable and extensible component systems, 30.

[98] Aarniala, J., (2005). "Instrumenting java bytecode", Seminar work for the Compilers course, Department of Computer Science, University of Helsinki, Finland, 2005.

[99] Dahm, M., (1999). Byte code engineering, ed. JIT'99. Springer, 267-277.

[100] Cohen, G.A. ve Kaminsky, D., (1998). Automatic Program Transformation with JOIE, USENIX Annual Technical Conference.

[101] White, A., (2014). "Serp is an open source framework for manipulating Java bytecode", http://serp.sourceforge.net/, 15 Ağustos 2014.

[102] Kuleshov, E., (2007). "Using the asm framework to implement common java bytecode transformation patterns", Aspect-Oriented Software Development.

[103] Dynamic and Distributed Information Systems Group, U.o.Z., (2014). "SimPack", https://files.ifi.uzh.ch/ddis/oldweb/ddis/research/simpack/index.html, 15 Ağustos 2014.

[104] Wellington, B., (1999). "DNS Java", http://www.dnsjava.org/, 15 Ağustos 2014.

[105] Walenstein, A. Jyoti, N. Li, J. Yang, Y. ve Lakhotia, A., (2003). "Problems creating task-relevant clone detection reference data", 10th Working Conference on Reverse Engineering, 13-16 Kasım 2003, Victoria.

[106] Kim, S. Pan, K. ve Whitehead, E.J., (2005). "When functions change their names: Automatic detection of origin relationships", Reverse Engineering, 12th Working Conference on, 7-11 Kasım 2005, Pittsburgh.

[107] mkapdan, (2014). "Mustafa Kapdan Repositories", https://github.com/mkapdan?tab=repositories, 15 Ağustos 2014.

EK-A

METRİK KATEGORİ KÜMELERİ

Önermiş olduğumuz yapısal klon tespit yönteminin gerçekleştirilmesinde Bölüm 4.1.4'de anlatıldığı üzere 6 farklı kategoride metrik kümesi kullanılmıştır.

A-1 Bağlaşım Metrik Kümesi

Değerlendirilmesi yapılan proje içerisinde bağımlı sınıfların arasındaki metriklerdir.

Çizelge Ek-A. 1 Bağlaşım metrik kümesi

Sıra	Adı	Tanımı
1	Dosya bağımlılık sayısı	Bağımlılığı koparmak için gerekli minimum sayı
2	Başka sınıfı kullanma	Sınıfın toplamda kaç farklı başka sınıfı kullandığı
3	Paket döngüsü	Dizinler arasındaki istenmeyen döngüleri (cycle) tespit etmek için gereken minimum sayıdaki döngü
4	Paket döngü oranı	Paketler arası döngü (cycle) yüzdesini verir.
5	En az paket döngü kırıcı	Paketler arasındaki bağlantıyı kesmek için gerekli kesilecek minimum dosya bağlantı sayısı
6	Paket bağımlılık ağırlığı	Paketler arası dosya bağımlılığının sayısını verir.
7	Dosya döngüsü	İstenmeyen tüm döngüleri tespit edebilmek için gerekli olan en az döngü sayısı
8	Paket döngü oranı	2*(Dosya açısı (File tangle)/Dosya köşe ağırlığı (File edge weight))*100
9	Kullanılma sayısı	Sınıfın toplamda kaç farklı başka sınıf tarafından kullandığı bilgisini vermektedir.
10	Dosya bağımlılık ağırlığı	Paket içerisindeki dosya bağımlılık sayısını verir.

A-2 Karmaşıklık Metrik Kümesi

Programın karmaşıklığını ölçen metrikleri bulunduran kümedir. Bu küme içerisinde bulunan metriklerin değerleri program akışının fonksiyonel olarak çatallaştığı zamanlarda 1 artmaktadır.

Çizelge Ek-A. 2 Karmaşıklık metrik kümesi

Sıra	**Adı**	**Tanımı**
1	Sınıf karmaşıklığı	Sınıf içerisindeki ortalama karmaşıklık.
2	Karmaşıklık	Toplam karmaşıklık
3	Metot karmaşıklığı	Metodların ortalama karmaşıklığı
4	Cevap için gerekli süreç sayısı (RFC)	Sınıf için talebe karşılık vermek (Response for Class), herhangi bir talebe karşılık cevap vermek için geçilen metod sayısı
5	Sınıf içi eksik etkileşim (lcom4)	Sınıf içerisinde birbirinden bağımsız olan metod sayısı
6	Dosya karmaşıklığı	Dosyanın karmaşıklığı
7	Sınıf karmaşıklığı	Sınıfın karmaşıklık değerleri
8	Metot karmaşıklığı	Metodun karmaşıklık değeri
9	Sınıf karmaşıklık dağılımı	Sınıf karmaşıklık değerinin dağılımı

A-3 Kapsülleme Metrik Kümesi

Nesneye dayalı programlarda nesneye ait olan değişken ve metodlara doğrudan dışarıdan erişimin kapatılması ve erişimin sadece ilgili nesne üzerinden yapılması işlemine kapsülleme denilmektedir.

Çizelge Ek-A. 3 Kapsülleme metrik kümesi

No	Adı	Tanımı
1	Private fonksiyonlar	Gizli fonksiyon sayısı
2	Protected fonksiyonlar	Korunmuş fonksiyon sayısı
3	Erişimi tanımsız fonksiyonlar	Erişim tanımlanmamış fonksiyon sayısı
4	Sınıf seviyesi ifade	Sınıf içerisindeki ifade sayısı
5	Sınıf seviyesi gizli ifade	Sınıf içerisindeki gizli ifade sayısı
6	Sınıf seviyesi erişimi tanımsız ifade	Sınıf içerisindeki erişim tanımlanmamış ifade sayısı
7	Sınıf seviyesi korunmuş ifade	Sınıf içerisindeki korunmuş ifade sayısı
8	Erişimi tanımsız sınıf	Erişim tipi belirtilmemiş sınıf sayısı
9	Erişimi tanımsız arayüz	Erişim tipi belirtilmemiş arayüz sınıf sayısı
10	Erişimi tanımsız soyut sınıf	Erişim tipi belirtilmemiş soyut sınıf sayısı
11	Erişimi tanımsız değişmez sınıf	Erişim tipi belirtilmemiş final sınıf sayısı
12	Erişimi tanımsız enum sınıf	Erişim tipi belirtilmemiş enum sınıf sayısı
13	Erişimi tanımsız üzerine yazılan fonksiyon	Üzerine yazılan toplam metod sayısını vermektedir.
14	Erişimi tanımsız soyut fonksiyon	Sınıfta bulunan toplam soyut fonksiyon sayısı
15	Erişimi tanımsız sabit ifade	Sınıftaki değişmez (static) erişim tanımlanmamış ifade sayısı
16	Erişimi tanımsız değişmez ifade	Sınıf içerisindeki final erişim tanımlanmamış ifade sayısı
17	Erişimi tanımsız sabit değişmez ifade	Sınıf içerisindeki final değişmez erişim tanımlanmamış ifade sayısı
18	Özel sabit değişmez ifade	Sınıf içerisindeki final özel ifade sayısı
19	Sınıf seviyesi özel değişmez ifade	Sınıf içerisindeki final değişmez özel ifade sayısı
20	Sınıf seviyesi özel sabit ifade	Sınıf içerisindeki değişmez özel ifade sayısı

Çizelge Ek-A. 3 Kapsülleme metrik kümesi (devamı)

No	Adı	Tanımı
21	Soyut korunmuş fonksiyon	Toplam soyut fonksiyon sayısı
22	Sınıf seviyesi korunmuş değişmez ifade	Sınıf içerisindeki final korunmuş ifade sayısı
23	Sınıf seviyesi korunmuş sabit değişmez ifade	Sınıf içerisindeki final değişmeyen korunmuş ifade sayısı
24	Sınıf seviyesi korunmuş ifade	Sınıf içerisindeki korunmuş ifade sayısı
25	Sınıf seviyesi korunmuş sabit ifade	Sınıf içerisindeki değişmeyen korumuş ifade sayısı

A-4 Çok Biçimlilik Metrik Kümesi

Nesneye dayalı programlama dillerinin en önemli özelliklerinden biri olan çok biçimlilik, herhangi bir nesnenin fonksiyonlarının bir başka nesne tarafından yeniden yazılma yeteneğidir. Bu sayede nesneler sadece gerekli fonksiyonu değiştirerek birbirinden ayrışabilirler.

Çizelge Ek-A. 4 Çok biçimlilik metrik kümesi

Sıra	Adı	Tanımı
1	Toplam üzerine yazılan metot	Nesnenin ilişkiden dolayı üzerine yazılan toplam metot sayısını vermektedir.
2	Genel üzerine yazılan metot	Nesnenin ilişkiden dolayı üzerine yazılantoplam genel metod sayısını vermektedir.
3	Korunmuş üzerine yazılan metot	Nesnenin ilişkiden dolayı üzerine yazılantoplam korunmuş metot sayısını vermektedir.
4	Erişimi tanımsız üzerine yazılan metot	Nesnenin ilişkiden dolayı üzerine yazılantoplam erişim tanımlanmamış metot sayısını vermektedir.
5	Sabit genel metot	Sistem içerisinde bulunan değişmeyen fonksiyon sayını belirtmektedir. İlişki durumlarında üzerine yazılamayan fonksiyonlardır
6	Değişmez sabit metot	Sistem içerisinde bulunan final değişmeyen fonksiyon sayını belirtmektedir. İlişki durumlarında üzerine yazılamayan fonksiyonlardır
7	Değişmez metot	Sistem içerisinde bulunan final fonksiyon sayını belirtmektedir. İlişki durumlarında üzerine yazılan yazılamayan fonksiyonlardır.

A-5 Kalıtım Metrik Kümesi

Herhangi bir nesnenin başka bir nesnenin özelliklerini miras almasına kalıtım denmektedir. Bu yetenek, nesneye dayalı programlamanın en önemli özelliklerindendir.

Çizelge Ek-A. 5 Kalıtım metrik kümesi

Sıra	**Adı**	**Tanımı**
1	Toplam gerçeklenen arayüz sayısı	Nesneyi oluştururken gerçeklenen toplam soyut sayısını vermektedir.
2	Toplam soyut metot	Nesne içerisinde bulunan toplam soyut fonksiyon sayısını vermektedir.
3	Ağaçtaki derinliği	Nesnenin ağaç içerisindeki derinliğini verir.
4	Çoçuk sayısı	Nesneyi gerçekleyen/genişleten eden çoçuk sayısının toplamını vermektedir.
5	Genel soyut metot	Nesne içerisindeki ortak soyut fonksiyon sayısını verir.
6	Korunmuş soyut metot	Nesne içerisindeki korunmuş soyut fonksiyon sayısını verir.
7	Erişimi tanımsız soyut metot	Nesne içerisindeki erişim tanımlanmamış soyut fonksiyon sayısını verir.
8	Erişimi tanımsız sınıf	Özel, genel, korunmuş gibi değişken olmayan erişimi tanımlanmamış sınıf sayısını vermektedir.
9	Erişimi tanımsız arayüz sınıf	Özel, genel, korunmuş gibi değişken erişimi tanımlanmamış arayüz sayısını vermektedir.
10	Erişimi tanımsız soyut sınıf	Özel, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış soyut sınıf sayısın vermektedir.
11	Erişimi tanımsız değişmeyen sınıf	Özel, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış final sınıf sayısın vermektedir.
12	Erişimi tanımsız enum sınıf	Sistem içerisinde bulunan Özel, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış sayım (enum) sınıf sayısını vermektedir.
13	Üzerine yazılan tanımsız erişim metot	Nesnenin ilişkiden dolayı üzerine yazılan toplam erişimi tanımlanmamış metodsayısını vermektedir.
14	Üzerine yazılan korunmuş metot	Nesnenin ilişkiden dolayı üzerine yazılan toplam korunmuş metodmetod sayısını vermektedir.

Çizelge Ek-A. 5 Kalıtım metrik kümesi (devamı)

No	Adı	Tanımı
15	Üzerine yazılan genel metot	Nesnenin ilişkiden dolayı üzerine yazılan toplam genel metod sayısını vermektedir.
16	Üzerine yazılan metot	Nesnenin ilişkiden dolayı üzerine yazılan toplam metod sayısını vermektedir.
17	Arayüz sınıfı	Nesne içerisinde bulunan toplam arayüz sayısını vermektedir.
18	Değişmez sınıf	Java gibi bazı nesneye dayalı programlama dillerinde Final sınıf miras alınıp genişletilemez. Bu nedenle önemlidir.
19	Enum sınıf	Sistem içerisinde bulunan sayım sınıf sayısını verir.
20	Soyut sınıf	Nesne içerisinde bulunan toplam soyut sınıf sayısını vermektedir.

A-6 Boyut Metrik Kümesi

Nesneye dayalı programlama dili olup olmadığına bakmaksızın, her bir programlama dili için ölçülebilecek olan metrikler bulunmaktadır. Bu metrikler daha çok programın büyüklüğü ile ilgilidirler.

Çizelge Ek-A. 6 Boyut metrik kümesi

Sıra	Adı	Tanımı
1	Sınıf	Toplam sınıf sayısı
2	Dosya	Toplam dosya sayısı
3	Boş (void) dönen toplam metod sayısı	Boş (void) dönen toplam metod sayısı
4	boolean dönen toplam metod sayısı	boolean dönen toplam metod sayısı
5	Karakter (char) dönen toplam metod sayısı	Karakter (char) dönen toplam metod sayısı
6	bayt dönen toplam metod sayısı	bayt dönen toplam metod sayısı
7	Kısa (short) dönen toplam metod sayısı	Kısa (short) dönen toplam metod sayısı

Çizelge Ek-A. 6 Boyut metrik kümesi (devamı)

Sıra	**Adı**	**Tanımı**
8	Tamsayı (int) dönen toplam metod sayısı	Tamsayı (int) dönen toplam metod sayısı
9	Kesirli (float) dönen toplam metod sayısı	Kesirli (float) dönen toplam metod sayısı
10	Uzun (long) dönen toplam metod sayısı	Uzun (long) dönen toplam metod sayısı
11	Kesirli (double) dönen toplam metod sayısı	Kesirli (double) dönen toplam metod sayısı
12	Dizi (array) dönen toplam metod sayısı	Dizi (array) dönen toplam metod sayısı
13	Nesne dönen toplam metod sayısı	Nesne dönen toplam metod sayısı
14	Erişim metot sayısı	Toplam alıcı (getter) ve değer atayıcı (setter) sayısı
15	Sınıf seviyesi sabit ifade	Toplam değişmeyen değişken sayısı
16	Sınıf seviyesi değişmez toplam ifade	Toplam final değişken sayısı
17	Sınıf seviyesi değişmez ifade	Toplam final değişmeyen değişken sayısı
18	Sınıf seviyesi değişken	Toplam değişken sayısı
19	satır	Toplam satır sayısı
20	Üretilmiş satır	Toplam işlevsel kod satır sayısı

A-7 Sonar Yazılım Metrikleri

Sonar yazılım kalite ve yönetim aracı içerisinde birbirinden farklı alanlarda kullanılmak üzere 120 adet metrik tanımlanmıştır. Bu metriklerin listesi aşağıda gösterildiği gibidir.

Çizelge Ek-A. 7 Sonar yazılım metrikleri

Sıra	Adı	Tanımı
1	Satır Sayısı (lines)	Satır sayısı verir
2	Üretilmiş satır sayısı (generated_lines)	Üretilmiş satır sayısı verir
3	Kod satır sayısı (Ncloc)	Yorum barındırmayan satır sayısı verir
4	Üretilmiş kod satır sayısı (generated_ncloc)	Üretilmiş yorum barındırmayan satır sayısı verir
5	Sınıf sayısı	Sınıf sayısı verir
6	Dosya sayısı	Dosya sayısı verir
7	Klasör sayısı	Klasör sayısı verir
8	Paket sayısı	Paket sayısı verir
9	Fonksiyon sayısı	Fonksiyon sayısı verir
10	Erişim fonksiyonu sayısı	Erişim fonksiyonu sayısı verir
11	Paragraf sayısı	Paragraf sayısı verir
12	Durum sayısı	Durum sayısı verir
13	Erişilebilir doküman	Erişilebilir doküman sayısı verir
14	Yorum satır sayısı	Yorum sayısı verir
15	Yorum yoğunluğu	Yorum yoğunluğu verir
16	Boş yorum sayısı	Boş yorum sayısı verir
17	Erişilebilir doküman yoğunluğu	Dokümante edilmiş sınıf, fonksiyon ve değişken sayısı verir
18	Dokümante edilmemiş erişilebilir kaynak sayısı	Dokümante edilmemiş sınıf, fonksiyon ve değişken sayısı verir
19	Yoruma alınmış kod satır sayısı	Yoruma alınmış kod satır sayısı verir
20	Karmaşıklık	Karmaşıklık
21	Ortalama sınıf karmaşıklığı	Ortalama sınıf karmaşıklığı verir

Çizelge Ek-A. 7 Sonar metrikleri (devamı)

Sıra	Adı	Tanımı
22	Ortalama fonksiyon karmaşıklığı	Ortalama fonksiyon karmaşıklığı verir
23	Ortalama dosya karmaşıklığı	Ortalama dosya karmaşıklığı verir
24	Ortalama paragraf karmaşıklığı	Ortalama paragraf karmaşıklığı verir
25	Sınıf karmaşıklığı dağılımı	Sınıf karmaşıklığı dağılımı verir
26	Fonksiyon karmaşıklığı dağılımı	Fonksiyon karmaşıklığı dağılımı verir
27	Dosya karmaşıklığı dağılımı	Dosya karmaşıklığı dağılımı verir
28	Paragraf karmaşıklığı dağılımı	Paragraf karmaşıklığı dağılımı verir
29	Birim test sayısı	Birim test sayısı verir
30	Birim test süresi	Birim test süresi verir
31	Birim test hata sayısı	Birim test hata sayısı verir
32	Atlanılan birim test sayısı	Atlanılan birim test sayısı verir
33	Birim test başarısız sayısı	Birim test başarısız sayısı verir
34	Birim test başarı sayısı	Birim test başarı sayısı verir
35	Birim test detayları	Birim test detayları verir
36	Birim test kapsama oranı	Birim test kapsama oranı verir
37	Yeni kodun eskiyi kapsama oranı	Yeni kodun eskiyi kapsama oranı verir
38	Toplam kapsama sayısı	Toplam kapsama sayısı verir
39	Toplam yeni kod kapsama sayısı	Toplam yeni kod kapsama sayısı verir
40	Toplam kapsanmayan satır sayısı	Toplam kapsanmayan satır sayısı verir
41	Toplam kapsanmayan yeni satır sayısı	Toplam kapsanmayan yeni satır sayısı verir
42	Satır kapsamı	Satır kapsamı sayısı verir
43	Eklenen satır kapsamı / Değiştirilen kod	Eklenen satır kapsamı / Değiştirilen kod sayısı verir
44	Kapsanan satırlardaki veri	Kapsanan satırlardaki veri sayısı verir

Çizelge Ek-A. 7 Sonar metrikleri (devamı)

Sıra	Adı	Tanımı
45	Kapsanan durumlar	Kapsanan durumlar sayısı verir
46	Yeni kapsanan durumlar	Yeni kapsanan durumlar sayısı verir
47	Kapsanmayan durumlar	Kapsanmayan durumlar sayısı verir
48	Yeni kapsanmayan durumlar	Yeni kapsanmayan durumlar sayısı verir
49	Dal kapsamı	Dal kapsamı sayısı verir
50	Yeni veya değişitirilmiş kodun dal kapsamı	Yeni veya değişitirilmiş kodun dal kapsamı sayısı verir
51	Kapsanan dallardaki veri	Kapsanan dallardaki veri sayısı verir
52	Satır için durumlar	Satır için durumlar sayısı verir
53	Satır için kapsanan durumlar	Satır için kapsanan durumlar sayısı verir
54	Entegrasyon test kapsamı	Entegrasyon testlerinde kapsam sayısı verir
55	Entegrasyon testlerinde yeni veya değiştirilen kod	Entegrasyon testlerinde yeni veya değiştirilen kod sayısı verir
56	Entegrasyon testlerinde kapsanan satırlar	Entegrasyon testlerinde kapsanan satır sayısı verir
57	Entegrasyon testlerinde kapsanan yeni satırlar	Entegrasyon testlerinde kapsanan yeni satır sayısı verir
58	Kapsanmayan satırlar	Kapsanmayan satır sayısı verir
59	Entegrasyon testlerinde yeni kapsanmayan satırlar	Entegrasyon testlerinde yeni kapsanmayan satır sayısı verir
60	Kapsanan satır	Kapsanan satır sayısı verir
61	Entegrasyon testlerinde eklenen veya değiştirilen kodun satır kapsamı	Entegrasyon testlerinde eklenen veya değiştirilen kodun satır kapsamı sayısı verir
62	Kod kapsamında bulunan veri	Kod kapsamında bulunan veri sayısı verir
63	Kapsanan durumlar	Kapsanan durumlar sayısı verir
64	Entegrasyon testlerinde kapsanan yeni durumlar	Entegrasyon testlerinde kapsanan yeni durumlar sayısı verir
65	Kapsanmayan durumlar	Kapsanmayan durumlar sayısı verir
66	Entegrasyon testlerinde yeni kapsanmayan durum	Entegrasyon testlerinde yeni kapsanmayan durumlar sayısı verir

Çizelge Ek-A. 7 Sonar metrikleri (devamı)

Sıra	Adı	Tanımı
67	Dal kapsamı	Dal kapsamı sayısı verir
68	Entegrasyon testlerinde yeni veya değiştirilen koddaki dal kapsamı	Entegrasyon testlerinde yeni veya değiştirilen koddaki dal kapsamı sayısı verir
69	Satırdaki durumlar	Satırdaki durumlar sayısı verir
70	Satırdaki durumların kapsamı	Satırdaki durumların kapsamı sayısı verir
71	Tekrarlanan satırlar	Tekrarlanan satırlar sayısı verir
72	Tekrarlanan bloklar	Tekrarlanan bloklar sayısı verir
73	Tekrarlanan dosyalar	Tekrarlanan dosyalar sayısı verir
74	İfadelerle dengelenmeiş tekralanan satırlar	İfadelerle dengelenmeiş tekralanan satır sayısı verir
75	Tekrarlanan detaylar	Tekrarlanan detaylar sayısı verir
76	Ağırlıklı uyumsuzluklar	Ağırlıklı uyumsuzluklar sayısı verir
77	Kurallara uyma	Kurallara uyma sayısı verir
78	Uyumsuzluklar	Uyumsuzluklar sayısı verir
79	Engelleyen uyumsuzluklar	Engelleyen uyumsuzluklar sayısı verir
80	Kritik uyumsuzluklar	Kritik uyumsuzluklar sayısı verir
81	Önemli uyumsuzluklar	Önemli uyumsuzluklar sayısı verir
82	Önemsiz uyumsuzluklar	Önemsiz uyumsuzluklar sayısı verir
83	Bilgi verme uyumsuzlukları	Bilgi verme uyumsuzlukları sayısı verir
84	Yeni uyumsuzluklar	Yeni uyumsuzluklar sayısı verir
85	Yeni engelleyen uyumsuzluklar	Yeni engelleyen uyumsuzluklar sayısı verir
86	Yeni kritik uyumsuzluklar	Yeni kritik uyumsuzluklar sayısı verir
87	Yeni önemli uyumsuzluklar	Yeni önemli uyumsuzluklar sayısı verir
88	Yeni önemsiz uyumsuzluklar	Yeni önemsiz uyumsuzluklar sayısı verir

Çizelge Ek-A. 7 Sonar metrikleri (devamı)

Sıra	Adı	Tanımı
89	Yeni bilgi uyumsuzluklar	Yeni bilgi uyumsuzluklar sayısı verir
90	Soyutluk	Soyutluk sayısı verir
91	Kararsızlık	Kararsızlık sayısı verir
92	Uzaklık	Uzaklık sayısı verir
93	Kalıtım ağacındaki yeri(Depth in Inheritance Tree)	Kalıtım ağacındaki yeri sayısı verir
94	Çocuk sayısı (Number of Children)	Çocuk sayısı sayısı verir
95	Sınıf cevabı (Response for Class)	Sınıf cevabı sayısı verir
96	Sınıf dağılımı (Class distribution /RFC)	Sınıf dağılımı sayısı verir
97	Metodların uyum eksikliği (Lack of Cohesion of Methods)	Metodların uyum eksikliği sayısı verir
98	Metodların ve blokların uyum eksikliği (LCOM4 blocks)	Metodların ve blokların uyum eksikliği sayısı verir
99	Sınıf dağılımı	Sınıf dağılımı sayısı verir
100	LCOM4 değeri 1'den büyük olan sınıfların yoğunluğu	LCOM4 değeri 1'den büyük olan sınıfların yoğunluğu sayısı verir
101	İçeri bağlantılar (Afferent couplings)	İçeri bağlantılar sayısı verir
102	Dışarı bağlantılar (Efferent couplings)	Dışarı bağlantılar sayısı verir
103	Bağlılık matriksi (Dependency Matrix)	Bağlılık matriksi sayısı verir
104	Paket döngüleri (Package cycles)	Paket döngüleri sayısı verir
105	Paket indeks açısı (Package tangle index)	Paket indeks açısı sayısı verir
106	Kesilen dosya bağlantıları	Kesilen dosya bağlantıları sayısı verir

Çizelge Ek-A. 7 Sonar metrikleri (devamı)

Sıra	Adı	Tanımı
107	Kesilen paket bağlantıları (Package dependencies to cut)	Kesilen paket bağlantıları sayısı verir
108	Paket köşe ağırlıkları (Package edges weight)	Paket köşe ağırlıkları sayısı verir
109	Dosya döngüleri (File cycles)	Dosya döngüleri sayısı verir
110	Dosya indeks açısı (File tangle index)	Dosya indeks açısı sayısı verir
111	Dosya açıları (Files tangles)	Dosya açıları sayısı verir
112	Sakıncalı dosya bağlantıları (Suspect file dependencies)	Sakıncalı dosya bağlantıları sayısı verir
113	Dosya köşe ağırlıkları (File edges weight)	Dosya köşe ağırlıkları sayısı verir
114	İşlemler	İşlemler sayısı verir
115	Son yapılan işlem tarihi	Son yapılan işlem tarihi verir
116	Revizyon	Revizyon sayısı verir
117	Satır yazarları	Satır yazarları sayısı verir
118	Revizyon yazarları	Revizyon yazarları sayısı verir
119	Satırda son yapılan değişiklik tarih ve zamanı	Satırda son yapılan değişiklik tarih ve zamanı sayısı verir
120	Uyarı	Uyarı

A-8 Yeni Geliştirilen Yazılım Metrikleri

Çizelge Ek-A. 8 Yeni geliştirilen yazılım metrikleri

Sıra	**Adı**	**Tanımı**
1	private metot sayısı	Erişilemez metot sayısı
2	private metot sayısı	Dışarıdan erişime kapalı, kalıtıma açık korunmuş metodlar
3	Erişimi tanımsız metotlar	Erişimi tanımlanmamış değişken metodlar
4	Final Methodlar	Değiştirilemez metotlar
5	Değer dönmeyen metod sayısı	Değer dönmeyen metodların toplam sayısın verir
6	Boolean değer dönen metodlar	Boolean değer dönen metodların toplam sayısın verir
7	Karakter dönen metodlar	Karakter dönen metodların toplam sayısın verir
8	Bayt dönen metodlar	Bayt dönen metodların toplam sayısın verir
9	Kısa değer dönen metodlar	Kısa değer dönen metodların toplam sayısın verir
10	Tamsayı dönen metodlar	Tamsayı dönen metodların toplam sayısın verir
11	Kesirli sayı dönen metodlar	Kesirli sayı dönen metodların toplam sayısın verir
12	Uzun değer dönen metodlar	Uzun değer dönen metodların toplam sayısın verir
13	Kesirli sayı dönen metodlar	Kesirli sayı dönen metodların toplam sayısın verir
14	Dizgi dönen metodlar	Dizgi dönen metodların toplam sayısın verir
15	Nesne dönen metodlar	Nesne dönen metodların toplam sayısın verir
16	Metoddaki yerel değişken sayısı	Metoddaki yerel değişken toplam sayısın verir
17	Sınıftaki değişken sayısı	Sınıftaki değişken toplam sayısın verir
18	Sınıftaki özel değişken sayısı	Sınıftaki özel değişken toplam sayısın verir
19	Sınıfdaki erişimi tanımlanmamış değişken sayısı	Sınıfdaki erişimi tanımlanmamış değişken toplam sayısın verir
20	Sınıftaki korunmuş değişken sayısı	Sınıftaki korunmuş değişken toplam sayısın verir

Çizelge Ek-A. 8 Yeni geliştirilen yazılım metrikleri (devamı)

Sıra	Adı	Tanımı
21	Sınıftaki genel değişken sayısı	Sınıftaki genel değişken toplam sayısın verir
22	Sınıftaki değişmeyen değişken sayısı	Sınıftaki değişmeyen değişken toplam sayısın verir
23	Sınıftaki final değişken sayısı	Sınıftaki final değişken toplam sayısın verir
24	Sınıftaki final değişmeyen değişken sayısı	Sınıftaki final değişmeyen değişken toplam sayısın verir
25	Sınıftaki değişmeyen özel değişken sayısı	Sınıftaki değişmeyen özel değişken toplam sayısın verir
26	Sınıftaki değişmeyen erişimi tanımlanmamış değişken sayısı	Sınıftaki değişmeyen erişimi tanımlanmamış değişken toplam sayısın verir
27	Sınıftaki değişmeyen korunmuş değişken sayısı	Sınıftaki değişmeyen korunmuş değişken toplam sayısın verir
28	Sınıftaki değişmeyen genel değişken sayısı	Sınıftaki değişmeyen genel değişken toplam sayısın verir
29	Sınıftaki final özel değişken sayısı	Sınıftaki final özel değişken toplam sayısın verir
30	Sınıftaki final erişimi tanımlanmamış değişken sayısı	Sınıftaki final erişimi tanımlanmamış değişken toplam sayısın verir
31	Sınıftaki final korunmuş değişken sayısı	Sınıftaki final korunmuş değişken toplam sayısın verir
32	Sınıftaki final genel değişken sayısı	Sınıftaki final genel değişken toplam sayısın verir
33	Sınıftaki final değişmeyen değişken sayısı	Sınıftaki final değişmeyen değişken toplam sayısın verir
34	Sınıftaki final değişmeyen değişken sayısı	Sınıftaki final değişmeyen değişken toplam sayısın verir
35	Sınıftaki değişmeyen korunmuş değişken sayısı	Sınıftaki değişmeyen korunmuş değişken toplam sayısın verir

Çizelge Ek-A. 8 Yeni geliştirilen yazılım metrikleri (devamı)

Sıra	**Adı**	**Tanımı**
36	Sınıftaki final değişmeyen genel değişken sayısı	Sınıftaki final değişmeyen genel değişken toplam sayısın verir
37	Üzerine yazılmış genel metotlar	Üzerine yazılmış genel metotların toplam sayısın verir
38	Üzerine yazılmış protected metotlar	ÜZerine yzılmış korunmuş metotların toplam sayısın verir
39	Üzerine yazılmış erişimi tanımlanmamış metotlar	Üzerine yazılmış erişimi tanımlanmamış metotların toplam sayısın verir
40	Soyut genel metotlar	Soyut genel metotların toplam sayısın verir
41	Soyut korunmuş metot	Soyut korunmuş metotların toplam sayısın verir
42	Soyut erişimi tanımlanmamış metodlar	Soyut erişimi tanımlanmamış metodların toplam sayısın verir
43	Toplam sınıf sayısı	Özle, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış sınıf sayısını vermektedir.
44	Toplam arayüz sayısı	Sınıf içerisinde bulunan toplam arayüz sayısını vermektedir.
45	Erişimi tanımlanmamış arayüz	Özle, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış arayüz sayısını vermektedir.
46	Soyut sınıf	Sınıf içerisinde bulunan toplam soyut sınıf sayısını vermektedir.
47	Erişimi tanımlanmamış soyut sınıf	Özle, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış soyut sınıf sayısını vermektedir.
48	Değiştirilemez sınıf sayısı	Java gibi bazı nesneye dayalı programlama dillerinde Final sınıf miras alınıp genişletilemez. Bu nedenle önemlidir.
49	Erişimi tanımlanmamış sınıf sayısı	Özle, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış final sınıf sayısını vermektedir.
50	Enum sınıfı	Sayım sınıfları geniştletilemes çünkü ilişkileri etkilemektedirler. Sistem içerisinde bulunan sayım sınıf sayısını vermektedir.
51	Erişime açık enum sınıfı	Sistemdeki genel sayım sınıf sayısını vermektedir.

Çizelge Ek-A. 8 Yeni geliştirilen yazılım metrikleri (devamı)

Sıra	**Adı**	**Tanımı**
52	Erişimi tanımsız Enum sınıfı	Sistem içerisinde bulunan özel, genel, korunmuş gibi değişkeni olmayan erişimi tanımlanmamış sayım sınıf sayısını vermektedir.
53	Gerçeklenen arayüz sayısı	Sınıfın her bir seviyesinde gerçekleştirilen toplam arayüz sayısını vermektedir.
54	Üzerine yazılan metot sayısı	Sınıfın her bir seviyesinde ilişkiden dolayı üzerine yazılan toplam metod sayısını vermektedir.
55	Toplam soyut metot	Sınıftaki toplam soyut sayısını metot verir
56	Sabit metot sayısı (static)	Sistem içerisinde bulunan değişmeyen fonksiyon sayını belirtmektedir. İlişki durumlarında üzerine yazılamayan fonksiyonlardır

A-9 Erişim Seviyesine Göre Yazılım Metrikleri

Çizelge Ek-A. 9 Erişim seviyesine göre yazılım metrikleri

Sıra	**Adı**	**Özel**	**Değişkeni Olmayan**	**Korunmuş**	**Genel**
1	Sınıf		Erişimi tanımlanmamış sınıflar		
2	Arayüz		Erişimi tanımlanmamış arayüz sınıfları		
4	Soyut sınıf		Erişimi tanımlanmamış soyut sınıflar		
5	Değiştirilemez Sınıf		Erişimi tanımlanmamış final sınıflar		
6	Enum sınıfı		Erişimi tanımlanmamış sayım sınıfları		Genel sayım sınıfları
7	Metot	Özel fonksiyonlar	Erişimi tanımlanmamış fonksiyonlar	Korunmuş fonksiyonlar	
8	Üzerine yazılan toplam fonksiyonlar	Üzerine yazılmamış toplam özel fonksiyonlar	Overridden total default functions	Üzerine yazılmamış korunmuş toplam fonksiyonlar	
9	Soyut fonksiyonlar	Syout toplam özel fonksiyonlar	Soyut toplam erişimi tanımlanmamış fonksiyonlar	Soyut toplam korunmuş fonksiyonlar	

Çizelge Ek-A. 9 Erişim seviyesine göre yazılım metrikleri (devamı)

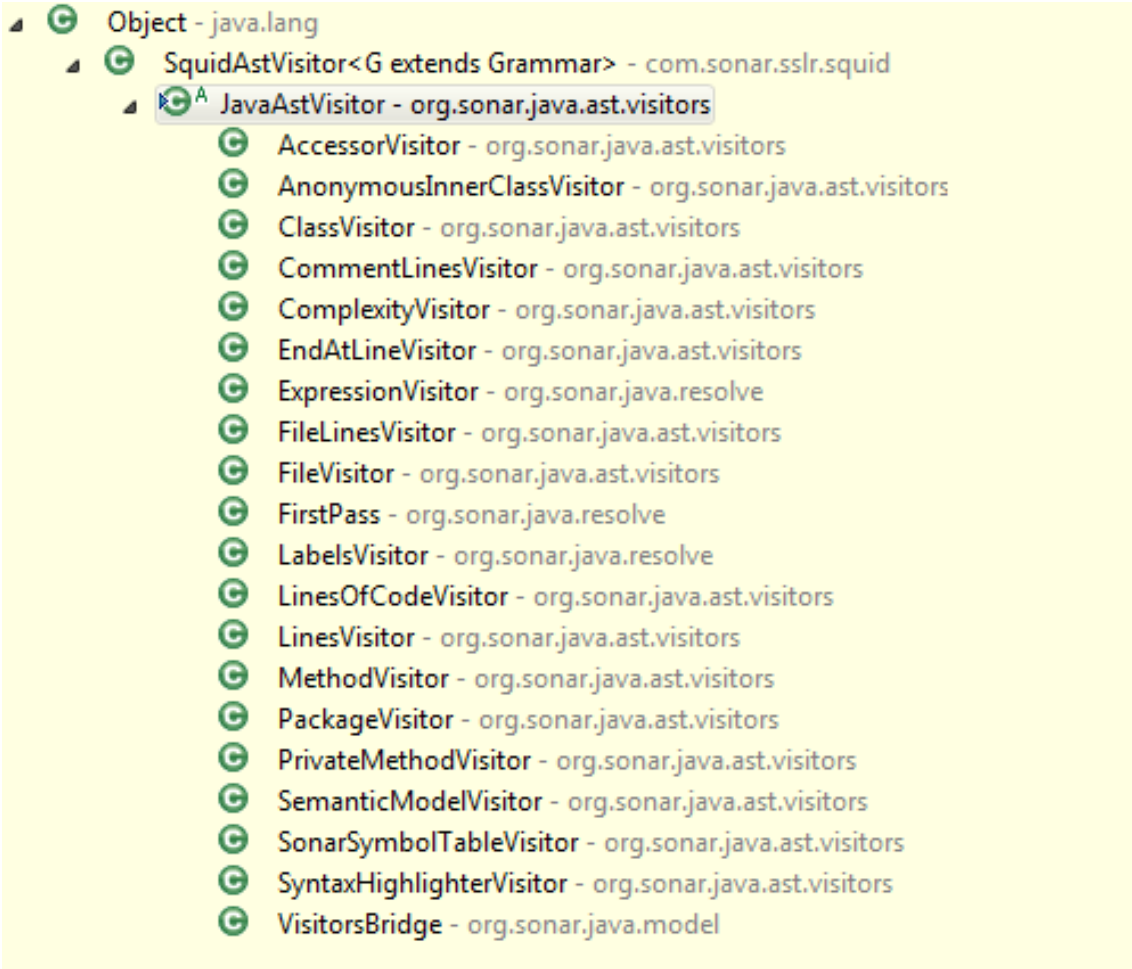
Sıra	**Adı**	**Özel**	**Değişkeni Olmayan**	**Korunmuş**	**Genel**
10	Sınıf düzeyinde değişkenler	Sınıf düzeyinde özel değişkenler	Sınıf düzeyinde erişimi tanımlanmamış değişkenler	Sınıf düzeyinde korunmuş değişkenler	Sınıf düzeyinde genel değişkenler
11	Sınıf düzeyinde değişmeyen değerler	Sınıf düzeyinde değişmeyen özel değerler	Sınıf düzeyinde değişmeyen erişimi tanımlanmamış değerler	Sınıf düzeyinde korunmuş özel değerler	Sınıf düzeyinde değişmeyen genel değerler
12	Sınıf düzeyinde değiştirilemez ifadeler	Sınıf düzeyinde final özel değişkenler	Sınıf düzeyinde final erişimi tanımlanmamış değişkenler	Sınıf düzeyinde final korunmuş değişkenler	Sınıf düzeyinde final genel değişkenler
13	Sınıf düzeyinde değiştirilemez sabit değerler	Sınıf düzeyinde final değişmeyen özel değerler	Sınıf düzeyinde değişmeyen erişimi tanımlan değer	Sınıf düzeyinde final değişmeyen korunmuş değerler	Sınıf düzeyinde final değişmeyen genel değerler

EK-B

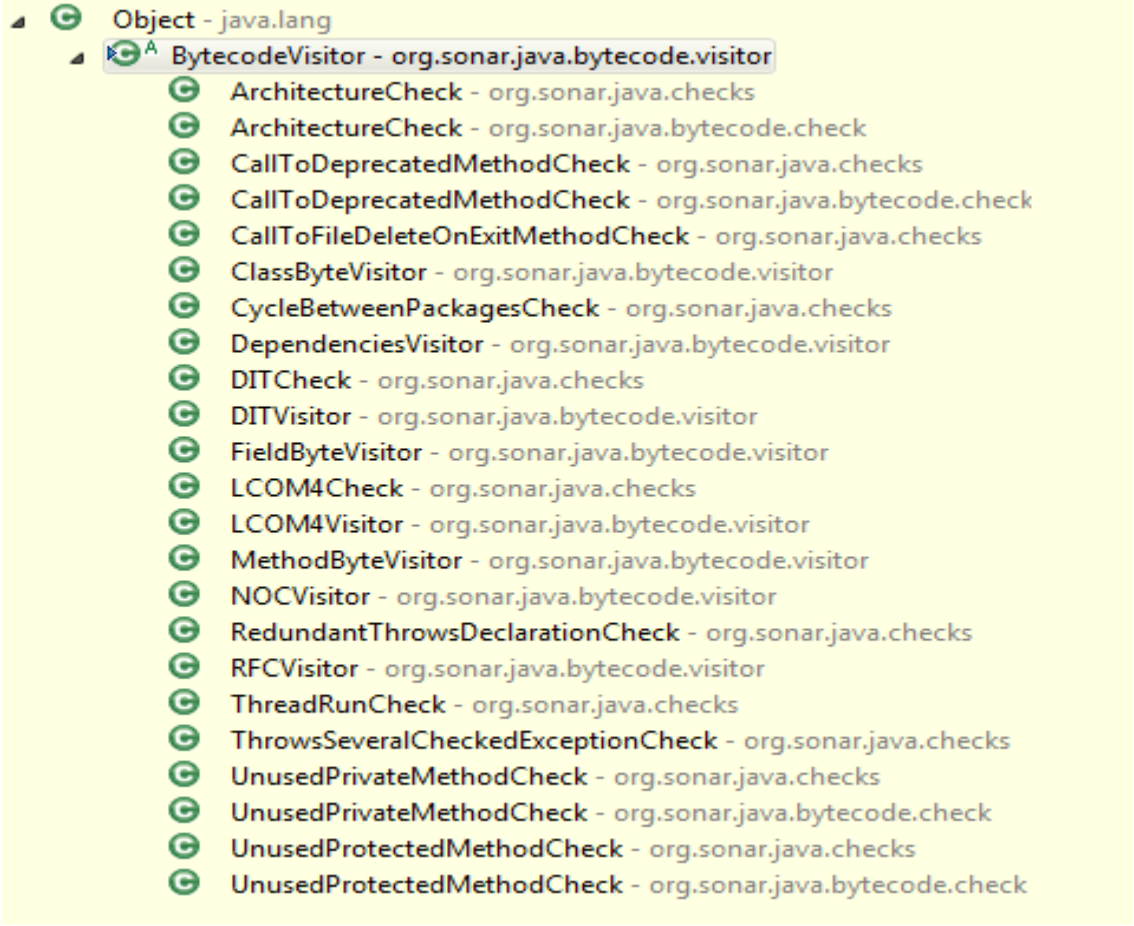
EKRAN GÖRÜNTÜLERİ

Bu kısımda, uygulama içerisinden alınmış ekran görüntüleri gösterilecektir.

B-1 SonarAyrıştırıcı Ziyaretçi ve Bayt Kod Ziyaretçi Sınıfları

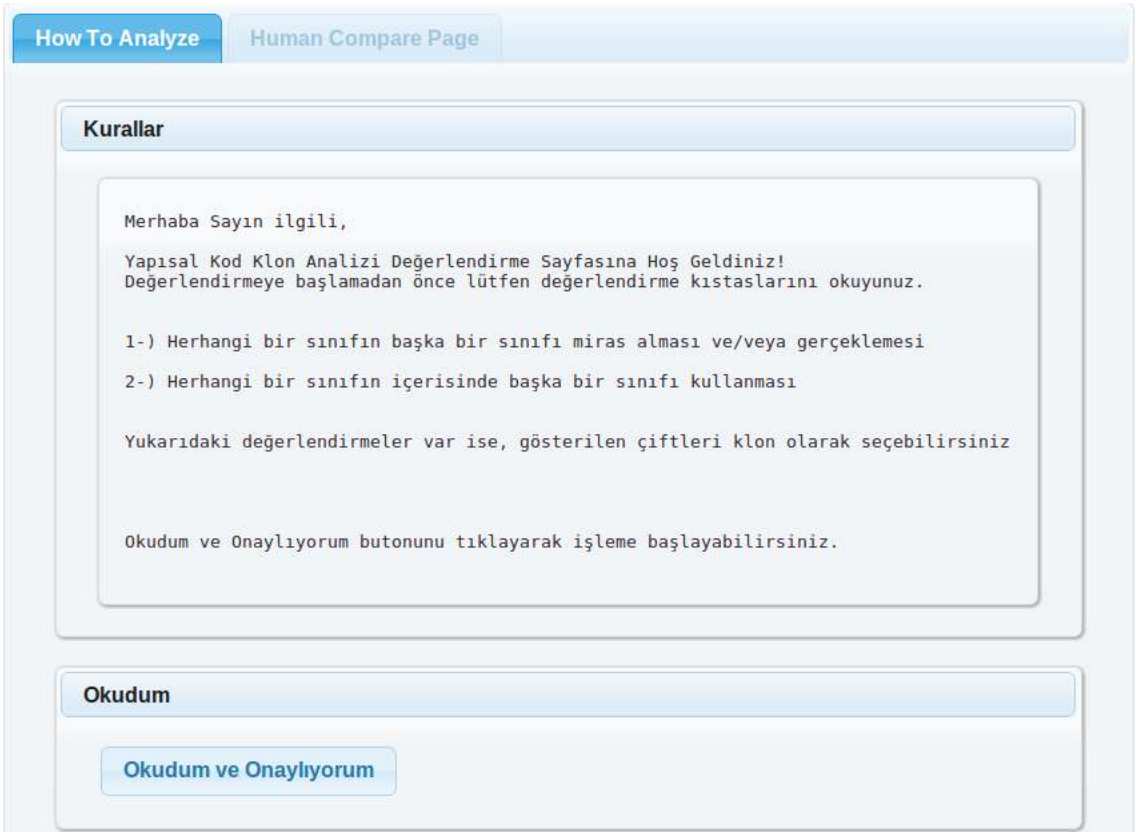


Şekil Ek B. 1 Sonarın parse ederken kullandığı sınıflar

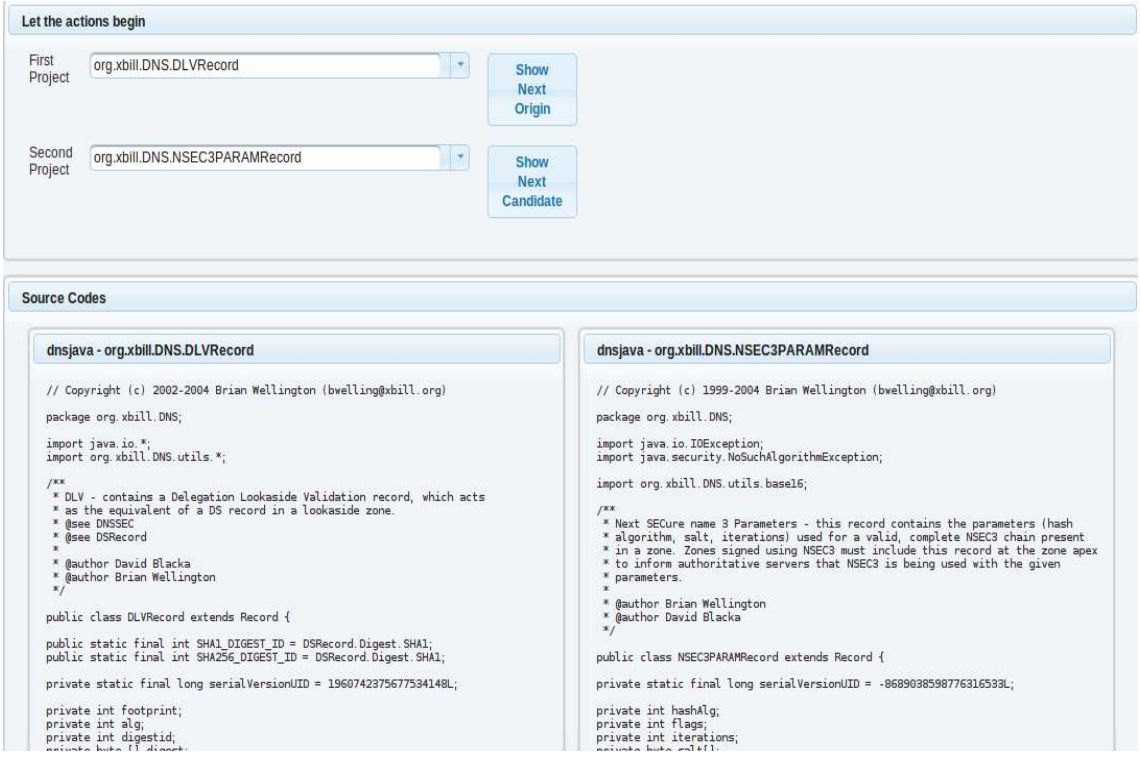


Şekil Ek B. 2 Sonarın bayt kod analiz ederken kullandığı sınıflar

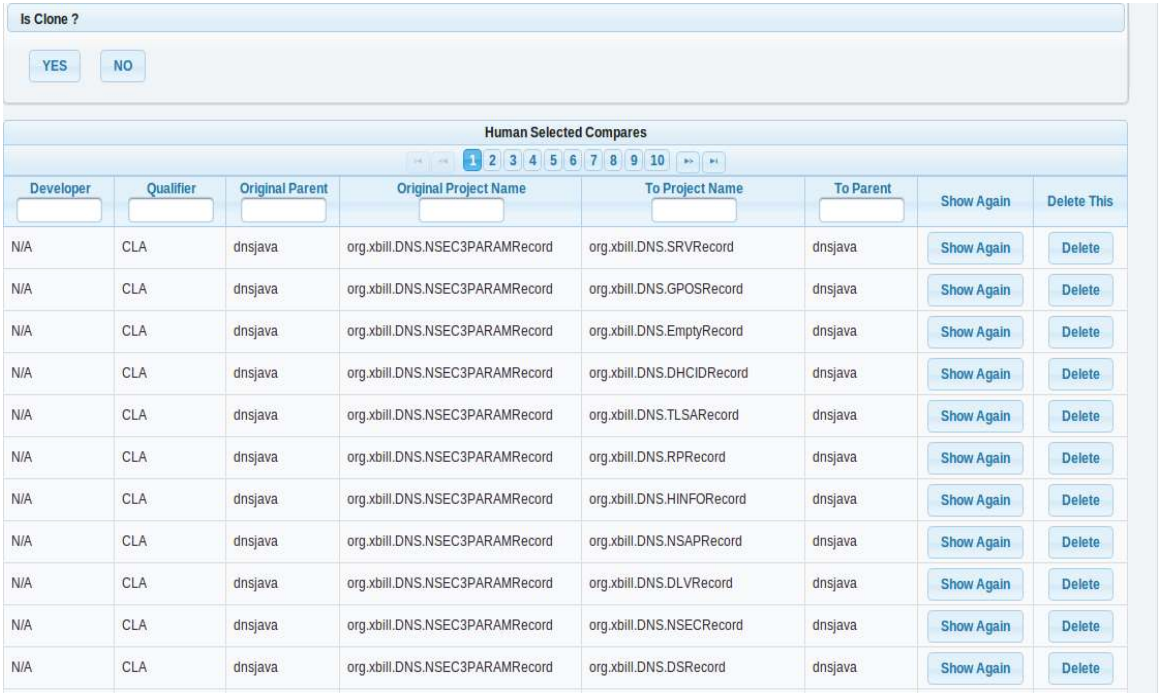
B-2 Uygulama Geliştiriciler için Yapısal Kod Klon Tespit Ekranı



Şekil Ek B. 3 Ana ekran



Şekil Ek B. 4 Karşılaştırma ekranı



Şekil Ek B. 5 Karar ekranı

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mustafa KAPDAN

Doğum Tarihi ve Yeri : 27/04/1987 - Eskişehir

Yabancı Dili : İngilizce

E-posta : mustafakapdan@gmail.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Lisans	Bilgisayar Mühendisliği	Eskişehir Osmangazi Üniversitesi	2011
Lise	Fen Bilimleri	Salih Zeki Anadolu Lisesi	2005

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2013	Türk Hava Yolları	Yazılım Mühendisi
2012	Huawei Telekomünikasyon	Yazılım Mühendisi
2011	Bahçeşehir Üniversitesi	Araştırma Görevlisi

YAYINLARI

Bildiri

1. Kapdan, M. Aktas, M. ve Yigit, M., "Yapısal Kod Klon Analizinde Metrik Tabanlı Teknikler" Ulusal Yazılım Mühendisliği Sempozyumu, UYMS, 2013

2. Kapdan, M. Aktas, M. ve Yigit, M., (2014). On the Structural Code Clone Detection Problem: A Survey and Software Metric Based Approach, ed. Computational Science and Its Applications–ICCSA 2014. Springer, 492-507.