# Yapısal Kod Klon Analizinde Metrik Tabanlı Teknikler

Mustafa Kapdan[1], Mehmet Aktaş[2], Melike Yiğit[3]

[1, 3]Kurumsal Gelişim ve Bilgi Teknolojileri Başkanlığı, Türk Hava Yolları, İstanbul
[1, 2]Bilgisayar Mühendisliği Bölümü, Yıldız Teknik Üniversitesi, İstanbul
[3]Bilgisayar Mühendisliği Bölümü, Bahçeşehir Üniversitesi, İstanbul

[1]mkapdan@thy.com
[2]aktas@yildiz.edu.tr
[3]melikeyigit@thy.com

**Özet.** Gereksiz tekrarlanmış kodlar (klonlar) iyi dokumante 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 klon kod tespiti yer almaktadır. Klon kod'lar iki ana başlık altında incelenmektedir. Yazılım içerisinde kod parçacığının benzerliğinden kaynaklanan tekrarlamalara basit klon adı verilirken, sistem mimarisi içerisinde, soyutlandırmanın birden çok seviyesinde, aynı yapı ile inşa edilmiş program yapılarına yapısal klon denmektedir. Basit klon tespit teknikleri, tekrarlanan kod parçacıklarına geniş bir açıdan bakamadıkları için, bunların tasarım seviyesindeki olası tekrarlamalardan kaynaklanıp kaynaklanmadığını saptayamamaktadır. Buradaki eksikliği gidermeyi amaçlayan yapısal klon tespitleri ise, yazılımdaki üst seviye benzerliklerinin ortaya çıkartılması, yeniden kullanılabilirliğin artırılması ve yazılımın basitleştirilmesine odaklanan yöntemlerden oluşmaktadır. Son yıllarda, yapısal klon tespit teknikleri için, birbirinden farklı öneriler yapılmıştır. Bu çalışmada yapısal klonların tespitinde literatürde yapılmış olan çalışmaları analiz ederek, yazılım metriklerine dayalı olarak yapısal kod klon tespitinin nasıl yapılabileceği konusunda bir değerlendirme yapacağız.

**Anahtar Kelimeler.** Kod Klon Analizi, Yapısal Kod Klon Analizi, Yazılım Metrikleri, Nesneye Dayalı Yazılım Metrikleri

## 1 Giriş

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 denmektedir. Sistemi oluşturan mimarinin içerisinde tekrarlanan kod bulunan, soyutlandırmanın birden çok seviyesinde aynı yapı ile inşa edilmiş program yapılarına ise yapısal klon denmektedir. Yazılım projesi

içerisinde klon bulunması, yazılım sisteminin kalitesinin kötü olduğunun bir işareti olarak kabul edilmektedir [1]. 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. Roy ve Cordy [2], yaptıkları literatür taramasında bu konuları detaylı olarak incelemektedir. Aşağıda bu literatür çalışmasını takip eden bir özet sunulmaktadır.

**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 incelenebilir.

- **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 gerçekleştirilir. Programın bir metodu çağırmasının maliyetli olduğu durumlarda, metot çağırılması gereken yerde tekrar kodlanabilmektedir.
- **Kısıtlamaları aşma isteği:** Programlama dilinin yeniden kullanılabilir kod mekanizmasını desteklememesi, ya da yeniden kullanılabilir kodun yazılma-sının çok zaman gerektirmesi ve hataya açık olmasından dolayı geliştiriciler kod tekrarları yapmaktadırlar. Bunlarla birlikte geliştiricinin sistemi tam ola-rak 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 yö-netimini uygulamaları, kod klonlarının oluşmasına sebebiyet vermektedir.
- **Yanlışlıkla kopyalama:** Birbirinden habersiz bir şekilde birden fazla gelişti-ricinin aynı işi yapan benzer kod kısımlarını geliştirmesi, ya da geliştiricinin önceden karşılaştığı bir sorun için kullandığı çözüm yöntemini tekrar tekrar kullanmasından kaynaklanan klon kodlardır.

**Kod klonlarının yazılım projelerindeki büyüklüğü:** Son yıllarda yapılan çalışmalar, tekrarlanmış kodlarla, endüstriyel yazılım sistemlerinde yaygın olarak karşılaşıldığını ortaya koymaktadır [3,4,5,6].Yapılan araştırmalara göre, yazılımın alanı ile alakalı olarak değişkenlik göstermekle birlikte; projelerin önemli bir kısmı klon kod-

lardan oluşmaktadır [7,8]. Baker [3] yazılım projelerin %13-%20'sinin klon koddan oluşduğunu bildirmektedir. Lague [9] yazılım projelerinde sadece fonksiyonel klonları incelemiş ve yaptığı araştırma sonucunda %6-8 arasında klon tespit etmiştir. Baxter [10] yazılım projelerinin %31'ini klon olarak kabul etmektedir. Mayrand [11] endüstriyel yazılımlar üzerinde %5-20 arasında klon tespit etmiştir. Kasper ve Godfrey [12] yazılım sisteminin %10-15'nin klon olduğunu tespit etmiştir. Tüm bu araştırmalar yazılım projelerinde kod klonlanmasının yaygın kullanılan bir yöntem olduğunu göstermektedir.

**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 [13] ve bunun yanında daha birçok hataya sebebiyet vermektedir [3,4,5,6,11,16]. Kod klonlarından dolayı ortaya çıkan problemleri aşağıdaki maddelerde özetleyebiliriz.

- **Bakım maliyetlerinin artması:** Eğer herhangi bir klonlanmış kod içerisinde bir hata (bug) bulunursa, hatanın düzeltilmesi için öncelikle bu kodun klonu olan tüm kodlar bulunmalıdır. Daha sonra bu hata her bir klon kod üzerinde düzeltilmelidir. İçerisinde klon kod bulunduran yazılım sistemlerinin, klon bulundurmayan sistemlere göre bakım maliyetlerinin yüksek olduğunu gös-teren birçok çalışma vardır [3,14]. Yazılım sisteminin teslimin-den sonra ya-pılan herhangi bir değişikliğin maliyeti, yazılımın toplam maliyetinin %40 ile %70'i arasında değişiklik göstermektedir [15].
- **Olası hata sayısının artması:** Kopyalanıp ufak bir değişiklikle ya da değiş-iklik 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 toplamda-ki olası hata sayısı artırmaktadır [14].
- **İyileştirme maliyetlerinin artması:** Bir klon kod üzerinde yapılan iyileştirme veya zenginleştirme çalışması da bu kodun her bir klonu üzerinde yapılmalıdır. Buda bakım ve güncelleme kapsamındaki maliyetleri arttırmaktadır.
- **Kaynak ihtiyacının artması:** Sistemin boyutu, kopyala/yapıştır yöntemin-den 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 klon-lanmış kodlar yüzünden, sistemin ve kullanılmış kodların anlaşılması ekstra zaman almaktadır.
- **Kötü sistem mimarisi oluşması:** Nesneye dayalı tasarım, soyutlama ve kalı-tım gibi temel yazılım bilgilerinin eksik olmasından dolayı kaynaklanan kod klonları sistem mimarisinin kötü olduğunu göstermektedir. Bu tarz klon kod-lar, sistem

içerisinde yeniden kullanılabilirliği zorlaştıracağı gibi bakım ma-liyetini de artıracaktır.

- **Yeni hata ihtimalinin artması:** Önceden yazılmış olan kodların, yeni sis-temlere, yeni programlama dilleri kullanılarak, taşınması aşamasında, çoğu zaman yazılım geliştiriciler tekrarlanmış kodların mantık akışını değiştirme-den kullanırlar. Bu kullanım yöntemi hataya açıktır ve sistem içerisinde yeni hatalar oluşaması ihtima-lini yaratmaktadır.

Yukarıda bahsedilen olumsuzlukların ortadan kaldırılması için, yazılım sistemleri içersindeki klon kodların tespiti büyük öneme sahiptir. Bunun için sözdizimsel ya da işlevsel olarak birbirine benzeyen kod parçalarının ortaya çıkarılması gerekmektedir. Literatürde, kod klonlarının türlerine dayalı olarak birçok tespit yöntemi önerilmiştir. Basit klon tespit teknikleri, kaynak kod gösterimi ve karakteristiğine göre çeşitlilik göstermektedir. Bu çeşitlilikler 5 farklı kategori altında incelenebilir: 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 tekrarla-malardan kaynaklanıp kaynaklanmadığını saptayamamaktadırlar. Bu nedenle, prog-ramın 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 makale yapısal kod klon tespit yöntemlerini detaylı olarak irdeleyerek, özellikle met-rik tabanlı yapısal kod klon tespit yöntemleri üzerinde, teknolojideki en son gelişme-leri yansıtan bir değerlendirme yapmaktadır.

Bu makalenin organizayon yapışı şu şekildedir. Birinci bölümde yazılım pro-jelerinde kod klonlarının ortaya çıkma sebepleri, kod klonlarının görülme sıklığı ve ortaya çıkardıkları olumsuzluklar özetlenmiştir. İkinci bölümde yazılım kod klon analiz ta-nımları verilmektedir. Üçüncü bölüm kod klon tespit yöntemleri üzerinde literatüre taramasını vermektedir. Dördüncü bölüm yapısal klon tespiti için önerilen metriklere dayalı mimari yapıyı anlatmaktadır. Sonuç bölümü araştırmalarımızı özet-leyerek ma-kaleyi sonlandırılmaktadır.

## 2 Yazılım Kod Klon Analiz Tanımları

Genel yazılım kod klon terminoloji terimleri Tablo 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 parçacığının oluşturduğu çifte klon çifti denilmektedir.
**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 incelenebilir. Basit klon tipinde, kod parçacıkları arasında 2 çeşit temel benzerlik bulunmaktadır. Bu benzerlikler, programın metni bazında olabileceği gibi metinden bağımsız olarak, işlevsel bazlı da olabilir. Bu benzerlikler temel olarak 4 ana başlık altında aşağıda belirtildiği gibi incelenmektedir:

- ***Tip I:*** Yorumlar, boş satırlar ve kod düzeni haricinde kalan kod parçacıkları-nın metinsel olarak birbirinin tamamen aynı olmasıdır.
- ***Tip II:*** Tip II, Tip 1'den farklı olarak, program içerisinde tanımlanan ifadele-rin söz dizimsel olarak benzer olup olmadığını değerlendirmektedir. Program içerisinde tanımlanan ifade metinsel olarak birbirine benzemese de, söz di-zimsel olarak aynı olan ifadeleri yakalamaktadır. Örneğin, int a=3, ile int b=3 ifadesi her ne kadar ifade (a, b) ismi metinsel olarak birbirine benzemese de işlevsel olarak her ikisi de aynıdır.
- ***Tip III:*** Programdan kopyalanan kod parçacığının içerisindeki; ifade isimle-rinin ve tiplerinin değiştirilmesinin yanı sıra, söz dizim içerisine fazladan ifade ekleme/çıkarma yapılmasıyla oluşan klon tipidir.
- ***Tip IV:*** 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. Mimari yapının birbirine benzemesi sonucu oluşan bu klon tipine ise yapısal klon denmektedir. Yapısal klon, benzerlik seviyesine göre yukarıda bahsedilen 4 çeşit basit klon tipinden herhangi birini kapsayabilmektedir.

**Tablo 1.** Klon tiplerinin sınıflandırılması

| **Klon Çeşitleri** | **Basit Klon** | | **Yapısal Klon** |
|---|---|---|---|
| | **Metin Bazlı Klon** | **İşlevsel Klon** | |
| **Tip I** | ✓ | × | ✓ |
| **Tip II** | ✓ | × | ✓ |
| **Tip III** | ✓ | × | ✓ |
| **Tip IV** | × | ✓ | ✓ |

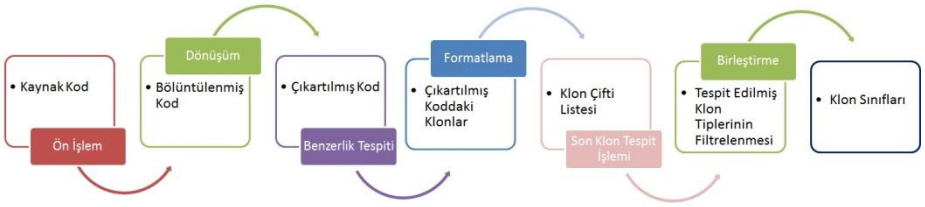
Tablo 1'de klon türlerinin hangi klon çeşitlerini içerip hangilerini içermediği gösterilmektedir. Bu bağlamda Yapısal Klon türünün diğer klon türlerinden farklı olarak

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

## 3 Tespit Yöntemleri Literatür Taraması

Kod klonlar üzerinde Yazılım Mühendisliği alanında geniş kapsamlı araştırmalar yapılmıştır. Literatüründe kod klon problemlerini azaltmak ya da engellemek için birçok teknik önerilmiştir. Bu araştırmaların odağında ise kod klon tespiti yer almaktadır. Bu açıdan, bu literatür taramasında kod klonlarının tespit yöntemleri üzerinde 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 maliyeti doğurmaktadır. Bu araştırma maliyetini düşürmek için bazı teknikler geliştirilmiştir. Bu tekniklerin temel olarak akış diyagramları Şekil 1'de gösterilmektedir.




**Şekil 1.** Genel Klon Tespit Süreçleri

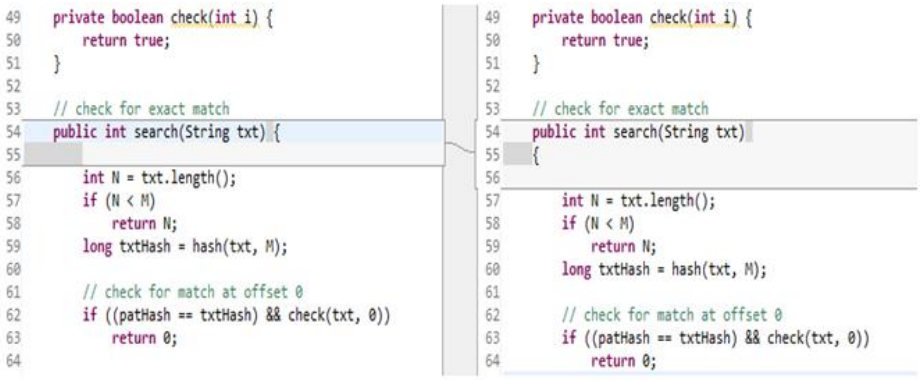
Literatürde birden fazla klon tespit aracı bulunmaktadır. Bunlardan bazıları akademik olduğu gibi, ticari olanları da vardır. Tespit araçları temel olarak veriyi dönüştürme ve benzerlik tespiti olmak üzere 2 aşamadan oluşmaktadırlar. Datayı dönüştürme aşamasında programın kaynak kodu, benzerlik algoritmasının kolay ve verimli bir şekilde çalışabileceği veri formatına çevrilir. Benzerlik tespiti aşamasında ise birbirlerine benzer ya da eşit olan kod parçacıkları tespit edilir. 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. Bu sebepten dolayı, tespit araçlarının kullandığı farklı teknikler aşağıdaki gibi ayrıştırılmaktadır:

**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 belir-

tilir. 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'de gösterilen her iki kod parçacığı birbirinin aynısı olmasına rağmen klon olarak değerlendirilmemektedir.
- İfadelerin isimlerinin değiştirilmesi tespit tekniğini hataya düşürmektedir. Örneğin, Şekil 3'te gösterildiği gibi her ne kadar her iki satır aynı işlevi yap-mış olsa da bu satırlar klon olarak değerlendirilmemektedir.
- Tek satır ifadelere parantez ekleme ya da çıkarılması sonucu satır satır metin karşılaştıran bu teknik klonu yakalayamamaktadır. Bu durum Şekil 4'te gösterilmektedir.

İlk gerçeklenen bazı klon algılayıcıları sözcüksel analiz tabanlıdır. Bu bağlamda Baker [17], Dup adı verilen aracı ile kaynak koddaki satır sıralarını kullanarak, satır satır klonları sözcüksel ve satır bazlı dizgi eşleştirme algoritması ile bulmaktadır. Dup 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 eder ve bulunan eşleştirmeleri göstermek için rapor oluşturur. 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.



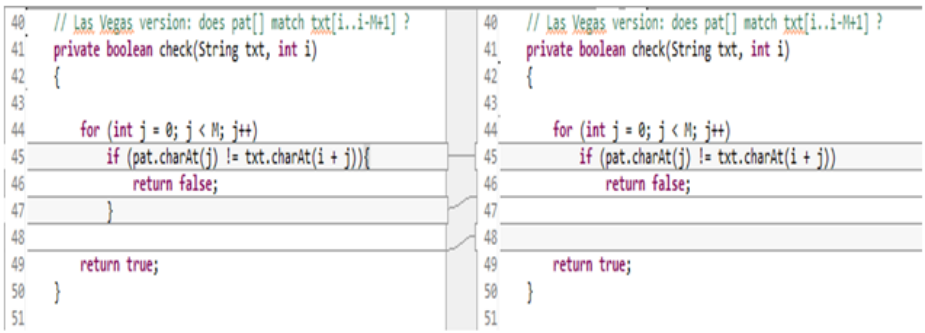

**Şekil 2.** Kod stilinden kaynaklanan hatalı program örneği

```
// a random 31-bit prime
private static long longRandomPrime() {

    BigInteger changedCode = new BigInteger(31, new Random());

    return changedCode.longValue();

}
```

```
// a random 31-bit prime
private static long longRandomPrime() {

    BigInteger originalCode = new BigInteger(31, new Random());

    return originalCode.longValue();

}
```

**Şekil 3.** İfade ismi değiştirilmiş hatalı program örneği

Ducase ve arkadaşları başka bir metin bazlı klon tespit yaklaşımı sunmaktadır [18,19]. Yaklaşım ayrıştırmaya dayanmamaktadır ve bundan dolayı herhangi bir dile kolayca uyum sağlayabilmektedir. Bu yöntem 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 verir. Ancak Ducase'nin bu yönteminin hesaplama karmaşıklığın boyutlu bir girdi de O (n2) olduğu için çok masraflıdır.



```
// Las Vegas version: does pat[] match txt[i..i-M+1] ?
private boolean check(String txt, int i)
{

    for (int j = 0; j < M; j++)
        if (pat.charAt(j) != txt.charAt(i + j)){
            return false;
        }

    return true;
}
```

```
// Las Vegas version: does pat[] match txt[i..i-M+1] ?
private boolean check(String txt, int i)
{

    for (int j = 0; j < M; j++)
        if (pat.charAt(j) != txt.charAt(i + j))
            return false;


    return true;
}
```

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

**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 [5] tarafından gerçekleşti-rilmiş 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 al-goritması 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 [3,17, 20] uygulaması jeton tabanlı CCFinder'a benzer şekilde sözcüksel analiz programını kullanıp kaynak kodu jetonlara ayırarak, ekleme tabanlı ağaç alt dizgi eşleme algorit-ması ile kod klonları bulmaktadır. Dup'ın CCFinder'dan farkı dönüşüm yapmaması-dır. CP-Miner [7,21] ise jeton bazlı klon tespiti için yapılmış olan başka bir çalışma-dır. Bu çalışmada benzer parçalanmış ifadeler ardı ardına alt dizi madencilik (mining) teknikleri kullanılarak bulunmaktadır.

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 [10]. CloneDR derleyici kullanarak açıklamalı ayrıştırılmış ağaç (annotated parse tree) oluşturur ve oluşturulan bu ağacın alt ağaçlarını metrik tabanlı özet fonksiyonlar (hash function) ile karşılaştırarak ağaçları eşleştirir. Benzer ağaçların kaynak kodu klon olarak gruplanır. Ağaç temelli teknikler için yapılmış başka bir araç da Bauhaus'un ccdiml [22] 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.

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 [23,24] 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 sunmakta-dır. Başka bir program bölme temelli PDG-DUP'a benzer bir çalışma ise Gallagher ve Lucas [25] 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 [26] ise kodun sözdizimsel ve veri akışını dikkate alarak kod sıkıştırma için program bağımlı çizge tekniği sunmuştur.

Metrik Temelli Teknikler: Metrik temelli yaklaşımlar kod parçacıkları için farklı metrikleri toplar ve kodu direk karşılaştırmak yerine bu metrik vektörlerini karşılaştırır. 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 metot gibi sözdizimsel birimler hesaplanır ve sonrasında bu birimler üzerindeki metrik değerleri karşılaştırılarak klonlar bulunur. Birçok durumda, kaynak kod bu metriklerin hesaplanması için parçalanarak onun soyut sözdizimi ağacı / program bağımlı çizge temsili oluşturulur.

Patenaude ve arkadaşları [27] gerçekleştirdikleri metrik temelli çalışmada metot içeri-sinden çağrılma sayısı, ifade sayısı, McCabe'in siklometik karmaşıklık ölçüsü, yerel olmayan değişkenlerdeki kullanılan tanımlama sayısı ve yerel değişken sayısı gibi metot seviyesinde birbirine çok benzer metrikleri benzer metotları bulmak için kul-lanmışlardır. Bu metrikleri Java programlama dili için tanımlamışlardır ve IBM Datrix aracını yazılım kalite değerlendirmesinde Java'yı desteklemesi için genişletmişlerdir.

Kanika ve arkadaşları [28] 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.

Yapısal klon: Yapısal klon analizleri içinde, sözdizimsel olduğu kadar mantıksal analiz de yapılmaktadır. Yazılım içersindeki kod kadar, yazılımın tasarım şablonları ve kod yorumları yapısal klon analizinde kullanılmaktadır. Fakat çoğu zaman yazılı-mın 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 duymaktadır ki bu da tamamen otoma-tik 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 [29] tarafından gerçekleştirilen Clone Miner'dır. Clone Miner, ilk aşama-da 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 [30] 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 [31] ise yaptıkları yapısal klon çalışmasında sistem içerisindeki dosyaların birbirlerine benzerliğini, dosya içerisinde bulunan basit klonlar üzerinden yapmışlardır. Basit klonların tespiti için CCFinder aracını kullanmışlardır. Çalışmala-rında dosyalar arasındaki ikili benzerlik oranını ortaya koymuşlarıdır fakat

grup ben-zerliğini yapamamışlardır. Birbirine benzer olan iki dosyanın birbirlerinin klonu olup olmadığı yönünde karar vermekten kaçınmışlardır.

De Lucia ve arkadaşları [30] yapısal klon tekniğine başka bir yaklaşımda bulunarak web spesifik yapısal klonlarını tespit etmişlerdir. Web sayfalarını klonun elemanı olarak değerlendirerek, sayfalar arasında geçişi sağlayan köprüler (hyperlink) klon tespiti için ilişki olarak değerlendirilmiştir. Çalışmalarında çizge tabanlı şablon eşleş-tirme algoritmalarını kullanmışlardır.

Marcus ve Maletic'in [32] 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.

Yazılım sistemleri içerisindeki sınıfların yapısını oluşturan tasarım şablonlarına ben-zer olan fakat uygulama seviyesinde soyutlama sağlayan yapılara mikro şablonlar denmektedir [33]. Bu mikro tasarım şablonları üzerinden yapısal klon analizi yapılabilmektedir. Benzer şekilde Shi ve Olsson [34] tarafından gerçekleştirilen Pinot aracında da kaynak kod içerisinde tasarım şablonları aranmıştır. Bu yöntemlerin eksik yanı, sistem içerisinde bilinen şablonları aramalarıdır. Fakat yapısal klon tespiti bilinmeyen şablonlar üzerinde de arama yapabilmelidir.

Bütün bu klon tespit teknikleri için yapılan çalışmalar Tablo 2'de klon tespit sınıfına ve kronolojik olarak gerçekleştirilme yılına göre gösterilmektedir.

**Tablo 2.** Klon tiplerinin sınıflandırılması

| **Çalışma** | **Yıl** | **Klon Tespit Sınıfı** | **Klon Tespit Tekniği** |
|---|---|---|---|
| Baker, *Dup* [16] | 1992 | Basit | Metin Temelli / Jeton Temelli |
| Baxter ve arkadaşları, *CloneDR.* [9] | 1998 | Basit | Ağaç Temelli |
| Chen ve arkadaşları [20] | 1999 | Basit | Program Bağımlı Çizge |
| Patenaude ve arkadaşları [31] | 1999 | Basit | Metrik Temelli |
| Marcus ve Maletic [22] | 2001 | Basit / Yapısal | Metin Temelli / Jeton Temelli |
| Komondoor ve Horwitz, *PDG-DUP* [26] | 2001 | Basit | Program Bağımlı Çizge |
| Kamiya ve arkadaşları, *CCFinder* [4] | 2002 | Basit /Yapısal | Jeton Temelli |
| Gallagher ve Lucas [28] | 2003 | Basit | Program Bağımlı Çizge |
| Ducase ve arkadaşları [19] | 2004 | Basit | Metin Temelli |

| | | | |
|---|---|---|---|
| De Lucia ve arkadaşları [39] | 2004 | Yapısal | Program Bağımlı Çizge |
| Basit ve arkadaşları, *Clone Miner* [38] | 2005 | Basit / Yapısal | Tüm Basit Klon Teknikleri |
| Li ve arkadaşları, *CP-Miner* [6] | 2006 | Basit | Jeton Temelli |
| Bauhaus, *ccdiml* [25] | 2006 | Basit | Ağaç Temelli |
| Shi ve Olsson, *Pinot* [42] | 2006 | Yapısal | Tüm Basit Klon Teknikleri |
| Kanika ve arkadaşları [47], *MCD Finder* | 2013 | Basit | Metrik Temelli |

## 4 Yapısal Klon Analizinde Metriklerin Kullanılması

Yazılım sistemlerinde basit klonların yoğun olarak bulunması, tasarım çözümlerinin kopyalanması sonucu oluşan yüksek seviyede benzerliklerin bulunduğuna işaret etmektedir. Bu tasarım çözümlerinin kopyalanmasının sebepleri aşağıdaki gibi sıralanabilmektedir.

- Aynı analiz şablonlarının kullanılması [3],
- Aynı tasarım şablonlarının kullanılması [35],
- Mimari seviyede kullanılan parçaların tasarımlarının benzemesi,
- Benzer sorunu çözmek için, geliştiricinin aynı mimariyi kullanması [10].

Literatür taraması incelendiğinde yapısal klonların tespitinde metrik tabanlı çalışmaların kullanılmadığı ve bu alanda bir eksiklik olduğu görülmektedir. Yapısal klonların, yazılım metrik ölçüm değerlerine dayalı olarak tespit edilip edilemeyeceği, ne oranda başarılı tespitler yapılabileceği irdelenmesi gereken bir araştırma sorusudur. Bu bağlamda Şekil 5'te, önerilen çalışma yöntemi gösterilmektedir.


**Şekil 5.** Genel önerilen çalışma yöntemi

Bu makale ile yapısal kod klonlarının tespiti ile ilgili teknolojideki en son ürünler ile ilgili bir durum tespiti yapıyoruz ve Tablo 3'te gösterilen metriklerin yapısal kod klon tespitinde kullanılmasını öneriyoruz. Bu amaçla henüz öncelikli olarak Yapısal Kod Klonlarının tespiti için odaklanmış ve açık kaynaklı Sonar Metik Ölçüm sistemine [36] entegre edilecek bir sistem üzerinde çalışıyoruz. Önerdiğimiz sistem mimarisi Şekil 6' da verilmektedir. Bu mimari tasarımda da görüleceği gibi, hedefimiz farklı yazılım kalite metrik ölçüm araçları kullanılarak oluşturulmuş bir yazılım metrik ölçüm veri tabanı üzerinde çalışacak olan bir eklenti ile yapısal klon analizine olanak sağlamaktır.

**Tablo 3.** Yapısal kod klon analizi için metrik değerlendirmesi

| Metrik Kategori | Metrik Ismi | Metrik Açiklamasi | Metrik Yapısal Kod Klona Uygun? | Kullanım Amacı |
|---|---|---|---|---|
| Satır Sayısı Metrikleri | Satır sayısı | Projedeki toplam kod satır sayısı | Hayır | - |
| | Yorum sayısı | Projedeki toplam yorum sayısı | Hayır | - |
| | Yoruma alınmış kod satır sayısı | Projedeki toplam yoruma alınmış kod satır sayısı | Hayır | - |
| | Yoruma alınmış boş satır sayısı | Projedeki toplam boş yorum satır sayısı | Hayır | - |
| | Yoruma alınmamış kod satır sayısı | Projedeki toplam işlevsel kod satır sayısı | Hayır | - |
| | Dönüştürülmüş kod satır sayısı | Projedeki jetonlarına ayrılmış toplam kod satır sayısı | Hayır | - |

| | | | | |
|---|---|---|---|---|
| | Toplam ifade sayısı | Projedeki jeton-larına ayrılmış top-lam ifade sayısı | Hayır | - |
| **Karmaşıklık Met-rikleri** | Sınıf karmaşıklığı | Sınıf içerisindeki ortalama karmaşıklık | **Evet** | İçerisinde karmaşık-lığı birbirine benzer sınıf barındıran pro-jeler, yapısal olarak da birbirine benzeyebilir. |
| | Metot karmaşıklığı | Metot içerisinde gerçekleştirilen ve akışı değiştiren kontrollerin toplam sayısı | **Evet** | İçerisinde karmaşık-lığı birbirine benzer metot barındıran pro-jeler, yapısal olarak da birbirine benzeyebilir. |
| | Dosya karmaşıklığı | Dosya içeris-indeki ortalama karmaşıklık | **Evet** | İçerisindeki karma-şıklığı birbirine ben-zer dosyalar barındı-ran projeler, yapısal olarak da birbirine benzeyebilir. |
| | Dosya bağımlılığı | Paketler arasın-daki bağımlılığı oluşturan dosyaların sayısı | **Evet** | Paketler arasındaki dosya bağımlılığı birbirine benzer olan projeler, yapısal olarak da birbirine benzeye-bilir. |
| **Nesneye Dayalı Metrikler** | Ağaç derinliği | Sınıfın, temel sınıfa göre kalıtım sayısı | **Evet** | İçerisinde ağaç derin-liği birbirine benzer sınıflar barındıran projeler, yapısal olarak da birbirine ben-zeyeblir. |
| | Sınıf içerisinde referans alınan sınıf sayısı | Bir sınıf içeris-indeki referans alınan top-lam sınıf sayısı | **Evet** | Sınıf içerisinde refe-rans alınan sınıf sayısı birbirine ben-zer olan projeler, yapısal olarak da birbirine benzeyebi-lir. |
| | Türetilmiş alt sınıf sayısı | Sınıftan doğrudan ya da dolaylı yoldan türetilmiş toplam sınıf sayısı | **Evet** | İçerisindeki türetilen toplam sınıf sayısı birbirine benzer olan projeler, yapısal olarak da birbirine benzeye-bilir. |

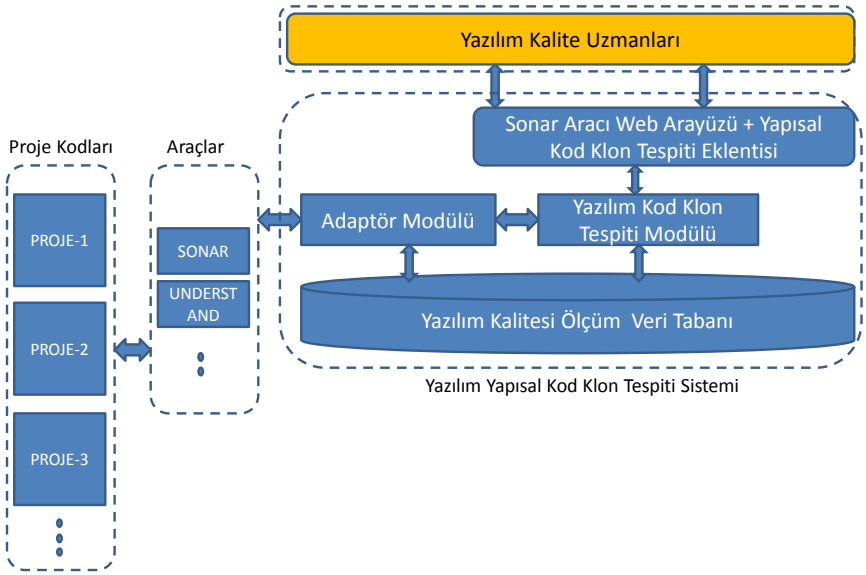
| | | | | |
|---|---|---|---|---|
| | Sınıfın başka sınıflarda referans verilme sayısı | Bir sınıfın toplam referans verilme sayısı | **Evet** | Sınıfın başka sınıflarda referans verilme sayısı birbirine ben-zer olan projeler, yapısal olarak da birbirine benzeyebilir. |
| | Metoda verilen parametre sayısı | Metoda verilen toplam girdi sayısı | **Evet** | Proje içerisindeki metoda verilen toplam girdi sayısı birbirine benzer olan projeler, yapısal olarak da birbirine benzeyebilir. |
| | Metodun döndüğü parametre sayısı | Metodun sonuç ola-rak döndüğü toplam çıktı sayısı | **Evet** | Proje içerisindeki metotların döndüğü toplam çıktı sayısı birbirine benzer olan projeler, yapısal olarak da birbirine benzeyebilir. |
| | Metot içerisinde başka metotları çağırma sayısı | Bir metot içerisinde çağrılan toplam metot sayısı | **Evet** | Proje içerisindeki metotların başka metotları çağırma sayısı birbirine ben-zer olan projeler, yapısal olarak da birbirine benzeyebilir. |

Programlama diline bağımlı olarak, yapısal klon analizinde kullanılacak metrikler değişkenlik gösterebilmektedir. Bu metrikler; ifadeler seviyesinde olabileceği gibi sınıf seviyesinde de olabilmektedir. Örnek olarak, Java programlama dili için yapısal klon analizinde kullanılabilecek metriklerden bazıları; sınıf tanımlamaları, metot tanımlamaları, koşullu veya döngülü ifadeler ve blok aralığı tanımlayıcılarıdır.

Bu önerimizi takiben gerçekleştireceğimiz çalışmamızda, bu amaçla, Tablo 3'te de gösterildiği gibi Karmaşıklık metrikleri (sınıf karmaşıklığı, metot karmaşıklığı, dosya bağımlılığı gibi), Satır sayısı metrikleri (kod satır sayısı, yorum satır sayısı, boş satır sayısı gibi) ve Nesneye Yönelik Metrikler (ağaç derinliği, türetilmiş alt sınıf sayısı, bir sınıf için çağrı sayısı, sınıf içindeki referans alınan sınıf sayısı gibi) incelenecek ve yapısal kod tespiti ve analizinde kullanılabilecek metrikler belirlenecektir.

Önerdiğimiz sistemde kullanmayı planladığımız araç Sonar [36] kalite metrik ölçüm aracıdır. Sonar, projelerin teknik kalitesini ölçmek ve analiz etmek için projenin bütü-nünden metotlarına kadar değişkenlik gösteren seviyelerde yönetim sunan açık kay-naklı bir yazılım kalite yönetim sistemidir. Sonar aracı, 2.11 nolu sürümüne kadar PMD-CPD [37] aracını kullanarak klon tespiti yapmıştır. Bu PMD-CPD tespit aracı kaynak kod içerisindeki jetonları (token) kullanarak Tip I-II ve kısmen de olsa Tip III olan basit klonları tespit edebilmektedir. Sonar içerisindeki bu klon tespit yöntemi hesaplama yapmak için yüksek miktarda hafızaya ihtiyaç duyması, tek proje içerisin-de klon tespiti yapabilmesi, java harici farklı dillere destek vermemesinden

dolayı Sonar-CPD adında yeni bir kütüphane kullanmaktadırlar. Bu kütüphane Google Kod Yazı 2011 kapsamında geliştirilmiştir. Bir önceki sürümüne göre farkları, birden fazla proje içerisinde klon tespiti yapabilmesi ve daha az hafıza kullanmasıdır. Fakat yine basit klon tiplerinden Tip I-II ve kısmen de olsa Tip III desteklemektedir. Sonar, kali-te ölçüm ve değerlendirilmelerinde kullanmak üzere karmaşıklık, tasarım, doküman-tasyon ve boyut gibi farklı alanlarda toplamda 125 adet metrik barındırmaktadır.




**Şekil 6.** Yapısal Kod Klon Tespiti Sismei Genel Mimarisi

Hedeflenen yapı, basit klon tespiti yapan teknikler içersinde, hafızayı etkin bir şekilde kullanma yöntemi olan metrik temelli tespit tekniği kullanarak yapısal klon analizi yapmaktır. Bu noktada Sonar'ın kullanmış olduğu metriklerin yanında yapısal klon için kullanılabilecek gerekli ekstra parametreler de Sonar'a tanıtılacaktır. Sonar'ın metrik ölçümlerinden çıkan sonuçlar ışığında Yapısal klon analizi yapılması hedef-lenmektedir.

## 5 Sonuç ve Gelecek için Planlar

Klon tespiti aktif olan araştırma alanlarından biridir ve literatürde klonları yazılım sistemlerinden kaldırmak için birçok çalışma yapılmıştır. Ayrıca yazılım sistemleri-nin gelişim yaşam döngüsünde klonlardan korunmak için de çalışmalar yapılmaktadır.

Bu çalışmada, yazılım klon tespit araştırma alanında kullanılan klon tipleri, onların tespit mekanizmaları ve ilgili araçları vurgulanarak, yapısal kod klon tekniği ile metrik temelli kod klon tespit tekniği birleştirilerek klon tespiti için karma bir yapı önerilmektedir. Önerilen bu yöntem klon tespit tekniğini kullanan potansiyel kullanıcılara doğru klon tespit aracını seçmelerine yardımcı olmaktır.

Gelecekte yapılacak araştırmamızı, bu makele ile önerdiğimiz metodolojinin gelişti-rilmesi, değerlendirilmesi şeklinde planladık. Bu sistemin gerçeklenmesi sonucunda ortaya çıkacak eklenti, Sonar açık kaynaklı aracı ile entegre kullanılabilecek ve yapısal kod klon analizinde kullanılabilecek bir yazılım ortaya çıkartacaktır.

## Kaynaklar


1. M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 2000
2. C. K. Roy and J. R. Cordy. A survey on software clone detection research. Technical Report 541, Queen's University at Kingston, 2007
3. Brenda Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), pp. 86-95, Toronto, Ontario, Canada, July 1995.
4. Gerardo Casazza, Giuliano Antoniol, Umberto Villano, Ettore Merlo, Massimiliano Di Penta. Identifying Clones in the Linux Kernel. In Proceedings of the 1st IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'01), pp, 90-97, Florence, Italy, November 2001.
5. Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. Transactions on Softwa-re Engineering, Vol. 28(7): 654- 670, July 2002.
6. Kostas Kontogiannis. Evaluation Experiments on the Detection of Programming Patterns using Software Metrics. In Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE'97), pp. 44-54, Amsterdam, The Netherlands, October 1997.
7. Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. In IEEE Transactions on Software Engineering, Vol. 32(3): 176-192, March 2006.
8. Lingxiao Jiang, GhassanMisherghi, Zhendong Su, and Stephane Glondu. DECKARD: Scalable and Accurate Tree-based Detection of Code Clones. In Proceedings of the 29th International Conference on Software Engineering (ICSE'07), pp. 96-105, Minnesota, USA, May 2007.
9. Bruno Lague, Daniel Proulx, Jean Mayrand, Ettore M. Merlo and John Hudepohl. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. In Proceedings of the 13th International Conference on Software Maintenance (ICSM'97), pp. 314-321, Bari, Italy, October 1997.
10. Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna. Clone Detection Using Abstract Syntax Trees. In Proceedings of the 14th International Conference on Software Maintenance (ICSM'98), pp. 368-377, Bethesda, Maryland, November 1998.
11. Jean Mayrand, Claude Leblanc, Ettore Merlo. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In Proceedings of the 12th International Conference on Software Maintenance (ICSM'96), pp. 244-253, Monterey, CA, USA, November 1996.

12. Cory J. Kapser and Michael W. Godfrey. Supporting the Analysis of Clones in Software Systems: A Case Study. Journal of Software Maintenance and Evolution: Research and Practice, Vol. 18(2): 61-82, March 2006.
13. Filip Van Rysselberghe, Serge Demeyer. Evaluating Clone Detection Techniques. In Proceedings of the International Workshop on Evolution of Large Scale Industrial Applications (ELISA'03), 12pp., Amsterdam, The Netherlands, September 2003.
14. J Howard Johnson. Identifying Redundancy in Source Code Using Fingerprints. In Proceeding of the 1993 Conference of the Centre for Advanced Studies Conference (CASCON'93), pp. 171-183, Toronto, Canada, October 1993.
15. Penny Grubb, and Armstrong A Takang. Software Maintenance Concepts and Practice. 2nd edn. World Scientific (2003).
16. G. Antoniol, U. Villano, E. Merlo, and M.D. Penta. Analyzing cloning evolution in the linux kernel. Information and Software Technology, 44 (13):755-765, 2002.
17. Brenda S. Baker. A Program for Identifying Duplicated Code. In Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, March 1992.
18. Stephane Ducasse, Oscar Nierstrasz, and Matthias Rieger. Lightweight detection of duplicated codea language-independent approach. Technical report, University of Bern, Institute of Computer Science and Applied Mathematics, Bern, Switzerland, February 2004.
19. Stephane Ducasse, Matthias Rieger, Serge Demeyer. A Language Independent Approach for Detecting Duplicated Code. In Proceedings of the 15th International Conference on Software Maintenance (ICSM'99), pp. 109-118, Oxford, England, September 1999.
20. Brenda S. Baker. On Finding Duplication in Strings and Software. Journal of Algorithms, 1993.
21. Zhenmin Li, Shan Lu, Suvda Myagmar, Yuanyuan Zhou. CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04), pp. 289-302, San Francisco, CA, USA, December 2004.
22. Aoun Raza, Gunther Vogel, Erhard Pl¨odereder. Bauhaus–A Tool Suite for Program Analysis and Reverse Engineering. In Proceedings of the 11th Ada-Europe International Conference on Reliable Software Technologies, LNCS 4006, pp. 71-82, Porto, Portugal, June 2006.
23. Raghavan Komondoor and Susan Horwitz. Using Slicing to Identify Duplication in Source Code. In Proceedings of the 8th International Symposium on Static Analysis (SAS'01), Vol. LNCS 2126, pp. 40-56, Paris, France, July 2001.
24. Raghavan Komondoor. Automated Duplicated-Code Detection and Procedure Extraction. *Ph.D. Thesis, 2003.*
25. Keith Gallagher, Lucas Layman. Are Decomposition Slices Clones? In Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03), pp.251-256 Portland, Oregon, USA, May 2003.
26. W-K. Chen, B. Li, and R. Gupta. Code Compaction of Matching Single-Entry Multiple-Exit Regions. In Proceedings of the 10th Annual International Static Analysis Symposium ( SAS'03 ), pp. 401-417, San Diego, CA, USA, June 2003.
27. J.-F. Patenaude, E. Merlo, M. Dagenais, and B. Lague. Extending software quality assessment techniques to java systems. In Proceedings of the 7th International Workshop on Program Comprehension (IWPC'99), pp. 4956, Pittsburgh, PA, USA, May 1999.
28. Raheja K., Tekchandani R., "An Emerging Approach towards Code Clone Detection: Metric Based Approach on Byte Code", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, issue 5, May, 2013.

29. Basit, H. A., Jarzabek, S. Detecting Higher-level Similarity Patterns in Programs. Accepted for *ESEC-FSE'05, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2005, Lisbon.
30. De Lucia, A., Francese. R., Scanniello, G., and Tortora, G. Reengineering Web Applications Based on Cloned Pattern Analysis. In Proceedings of the 12th Intl. Workshop on Program Comprehension (IWPC '04). pp. 132-141, 2004.
31. Ueda, Y., Kamiya, T., Kusumoto, S., and Inoue, K. Gemini: Maintenance Support Environment Based on Code Clone Analysis. In Proceedings of the 8th IEEE Symposium on Software Metrics, pp. 67-76, 2002.
32. Andrian Marcus and Jonathan I. Maletic. Identification of high-level concept clones in source code. In Proceedings of the 16th IEEE International Conference on Automated Soft-ware Engineering (ASE'01), pp. 107-114, San Diego, CA, USA, November 2001.
33. J.Y. Gil and I. Maman, "Micro Patterns in Java Code," Proc. 20th Object Oriented Programming Systems Languages and Applications, pp. 97-116, 2005.
34. N. Shi and R.A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code," Proc. 21st IEEE/ACM Int'l Conf. Automated Software Eng., pp. 123-134, Sept. 2006.
35. Tibor Bakota, Rudolf Ference and Tibor Gyimothy. Clone Smells in Software Evolution. In Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM'07), 10pp., Paris, France, October 2007.
36. Sonar, URL http://docs.codehaus.org/display/SONAR/Documentation Last accessed Jun 2013.
37. PMD's CPD. URL http://pmd.sourceforge.net/cpd.html Last accessed Jun 2013.