Journal of Quantum Technologies and Informatics Research
JQTAIR

KUANTUM TEKNOLOJİLERİ VE ENFORMATİK ARAŞTIRMALARI
Volume / Cilt: 2, Issue / Sayı: 1, 2024, pp. 15-29
E - ISSN: 3023-4735
URL: https://journals.gen.tr/index.php/jqtair/
DOI: https://doi.org/10.70447/ktve.2301

Received / Geliş: 11.01.2024
Acccepted / Kabul: 25.01.2024

RESEARCH ARTICLE / ARAŞTIRMA MAKALESİ

Otomatik Birim Test Oluşturmak İçin Opcode Ayrıştırma Yaklaşımının Geliştirilmesi

Enhancing The Opcode Parsing Approach for Automated Unit Test Generation

Sevdanur Genç

Kastamonu Üniversitesi, Taşköprü Meslek Yüksekokulu, Bilgisayar Teknolojileri Bölümü, Türkiye, e-mail: sgenc@kastamonu.edu.tr

Öz

Yazılım geliştirme süreçlerinde doğruluk ve güvenilirlik, birim testlerin etkin bir şekilde oluşturulmasıyla doğrudan ilişkilidir. Bu bağlamda, birim test üretimi ve yazılım testi süreçleri, geliştiricilerin ve yazılım mühendislerinin önemli bir odak noktası haline gelmiştir. Bu çalışma, otomatik birim test oluşturma sürecindeki gelişmeler üzerinde durarak, geliştirilen Opcode ayrıştırma yönteminin Java Agent teknolojisiyle entegrasyonunu incelemekte ve bu entegrasyonun yazılım testi alanındaki potansiyel etkilerini değerlendirmektedir. Java bytecode seviyesindeki opcode'ları analiz ederek, Java Agent'ların dinamik kod manipülasyonu kabiliyetini kullanarak, otomatik test senaryolarının oluşturulması hedeflenmiştir. Bu yöntem, yazılım geliştirme süreçlerinde test kapsamını artırarak, kod kalitesini iyileştirmeyi ve yazılımın doğruluğunu sağlamayı amaçlamaktadır. Ayrıca, çalışma, Java Agent teknolojisinin opcode analiziyle birleştirilmesinin, otomatik birim test oluşturma sürecindeki etkisini deneysel verilerle destekleyerek, bu teknik entegrasyonun pratik uygulanabilirliğini değerlendirmektedir. Elde edilen sonuçlar, Java Agent'ların opcode analiziyle birleşerek otomatik test oluşturma sürecindeki potansiyelini vurgulayarak, yazılım mühendisliği alanına önemli bir katkı sağlamaktadır.

Anahtar Kelimeler: Opcode Ayrıştırma Yöntemi, Birim Test Üretimi, Bytecode, Java Agent, Yazılım Testi.

Abstract

In software development processes, accuracy and reliability are directly associated with the effective creation of unit tests. In this context, the production of unit tests and software testing processes have become a significant focal point for developers and software engineers. This study examines the developments in automated unit test generation processes, focusing on the integration of the developed Opcode parsing method with Java Agent technology, and evaluates the potential impacts of this integration in the field of software testing. By analyzing opcodes at the Java bytecode level and leveraging the dynamic code manipulation capabilities of Java Agents, the aim is to generate automated test scenarios. This method aims to enhance test coverage in software development

Citation/Atıf: GENÇ, S. (2024). Otomatik Birim Test Oluşturmak İçin Opcode Ayrıştırma Yaklaşımının Geliştirilmesi. *Kuantum Teknolojileri ve Enformatik Araştırmaları*. 2(1): 15-29, DOI: 10.70447/ktve.2301

Corresponding Author/ Sorumlu Yazar:
Sevdanur Genç
E-mail: sgenc@kastamonu.edu.tr

processes, improve code quality, and ensure software accuracy. Additionally, the study assesses the practical applicability of this technical integration by supporting it with experimental data, highlighting the impact of merging Java Agent technology with opcode analysis in the process of automated unit test generation. The results underscore the potential of Java Agents combined with opcode analysis in automating the test generation process, offering a significant contribution to the field of software engineering.

Keywords*:* The Opcode Parsing Method, Unit Test Generation, Bytecode, Java Agent, Software Testing.

1. GİRİŞ

Sun Microsystems tarafından geliştirilen ve 1995 yılında piyasaya sürülen Java programlama dilinin ana amacı, taşınabilir, öğrenmesi kolay, genel amaçlı, platformdan bağımsız ve nesne odaklı bir programlama dilinin oluşturulmasıydı. Java derleyicisi, kaynak kodunu platformdan bağımsız bir ara dil olan Java bytecode'a dönüştürür. Bu kod daha sonra her platformda Java sanal makinesi üzerinden işlenir ve çalıştırılır. Java ajanları, JVM tarafından çalışma zamanında yürütülen uygulama mantığını esnek bir şekilde değiştirmek için kullanılır. Bir Java Agent, özel olarak tasarlanmış bir jar dosyasıdır. Bu dosya, mevcut JVM'de yüklenmiş olan bytecode'ları değiştirmek için Instrumentation API'yi kullanır. Java Agent araçlarının en önemli özelliği, çalışma zamanında sınıfları yeniden tanımlayabilme veya değiştirebilme yetenekleridir. Metot gövdelerini sabit ve değişken özelliklerini yeniden tanımlayarak değiştirebilirler. Ayrıca, metotların nesne veya kalıtım özelliklerini değiştirebilirler.

Yazılım kalitesi ve üretkenlik ihtiyaçları yazılım geliştirme sürecinde bir arada değerlendirilir. Bu nedenle, akıcı bir algoritma ve güçlü risk yönetimi gibi faktörler mevcut yazılımın bu kriterlere uygunluğunu test etmek için gereklidir. Bu faktörleri yerine getirmek için test alanında ciddi sorumluluklar bulunmaktadır. Hızlı testler ve sonuçların yüksek doğruluğu, yazılım geliştirme sürecinde fark yaratan faktörlerdir. Bu faktörü gerçekleştirmek için yazılım test otomasyonu gereklidir. Yazılım projelerindeki odak noktası genellikle yazılım test süreçleridir. Başarılı bir test sürecinin sonunda, hataları en az olan ve yüksek doğrulukta bir yazılım elde edilir. Türkiye'deki yazılım kalitesi üzerine yapılan çalışmalar, test odaklı yazılım geliştirme süreci ve test araçlarına olan ihtiyacın ülkemizde artmakta olduğunu göstermektedir [1]. Test odaklı yazılım geliştirmede hedef, gerekli işi yapacak kod yazmadan önce, öncelikle test edilebilir bir kod yazmak ve bu koda ait senaryoyu oluşturmaktır. Farklı yazılım geliştirme prensipleri bu test edilebilir kodu tasarlar; eğer yüksek doğrulukta bir sonuç alınırsa, yazılım başarılı bir şekilde testi geçmiş olur. Test sonuçları başarısız olursa, başa dönülür, kod incelenir ve sorun düzeltilmeye çalışılır. Yazılım projelerinde, her birim (sınıf ve metot) hatasız çalıştığını kanıtlamak adına birim testleri denilen testler yazılır. Birim testleri yazılım geliştirme sürecini kolaylaştırır, hızlandırır ve her sınıf ve metodun doğru şekilde çalıştığından emin olunmasını sağlar.

Java platformunda en yaygın kullanılan iki temel test çerçevesi bulunmaktadır: JUnit ve TestNG [2]. Her ikisi de karmaşık test durumlarında istenilen kod parçaları üzerinde test yapmaya imkan tanıyacak kadar güçlüdür. JUnit, yinelenebilir testler yazmak ve çalıştırmak için açık kaynak kodlu bir framework'tür. Çeşitli test durumlarıyla test verilerini çalıştırarak programdan beklenen sonuçları test eder. TestNG, JUnit'den daha işlevseldir ve kullanımı daha kolaydır. TestNG, test thread'lerinde test durumlarını paralel olarak çalıştırmayı destekler. Ayrıca esnek test yapılandırmaları, detaylı hata mesajlarının analizi, gelişmiş arşivleme ve editörler için eklenti desteği gibi birçok özelliğe sahiptir.

Geliştirici tüm bu birim testlerini otomatik olmayan sistemlerle yani elle yazarak kodlar. Testleri otomatikleştirmek, zamanı kaliteli bir şekilde kullanmayı ve iş trafiğini hızlandırmayı önerir. Bu nedenle, test araçlarının hızı ve doğruluğu önemlidir. Örneğin, anahtar kelime tabanlı kodlar önemli üstünlüklere sahiptir.

Bu yaklaşımda, test edilen yazılımın boyutu önemlidir, test sayısı değil. Bu, kod bakım maliyetini büyük ölçüde azaltır ve otomatik testlerin uygulanmasını hızlandırır. Aynı zamanda, test otomasyonunda başarı elde etmek için kodların ve verilerin yeniden kullanılabilir olması gereklidir. Bu, tekrarlanan görevleri ortadan kaldırır ve yeni testlerin uygulanmasını hızlandırır. Bununla birlikte, otomatik testler, elle yazılan testlere kıyasla yazım hatalarının da önlenmesine yardımcı olabilir [3].

Bu araştırma, çalışma zamanında birim testlerin otomatik olarak oluşturulabilmesi için bir çözüm olarak opcode ayrıştırma yöntemini önermektedir. Bu yazılım, bir masaüstü uygulaması olarak geliştirilmiş olup belirtilen Java sınıflarında birim test dönüşümleri gerçekleştirebilmektedir. Tüm bu dönüşümler, Java Agent ve bytecode temellidir. Çalışılacak örnek Java sınıflarının nesneleri, metotları ve değişkenleri ile ilgili tüm bilgi çalışma zamanında veriye dönüştürülmüştür. Bu dönüşüm sırasında bilgi hem veritabanına kaydedilmiş hem de kaydedilen verilerle birlikte bir şablon motoru aracılığıyla otomatik birim testleri haline getirtilmiştir.

Bu çalışmada, otomatik birim testi üretimi için çeşitli yaklaşımlar sunulmuştur. Çalışmanın literatüre katkıları şunlardır:

1. Bytecode dönüşümleri sırasında çalışacak olan bir opcode ayrıştırma yöntemi, Java string fonksiyonlarından yararlanarak geliştirilmiştir. Bu yöntemle, her bir opcode'a karşı nesneler, değişkenler ve giriş-çıkış parametreleri gibi değerler ayrılmış ve bu veriler JSON formatında listelenmiştir. Geliştirilen opcode ayrıştırma yöntemi, açık kaynak kodlu bir yaklaşım olduğu için gelecekte farklı ihtiyaçlara göre geliştirilmeye açıktır. Literatürdeki çalışmalarda, Bytecode API'nin sınırlı hazır fonksiyonları kullanılmış ve yine bunlarla birlikte farklı yaklaşımlar sunulmuştur.

2. Bir Java sınıfındaki her bir nesne, NoSql veritabanı koleksiyonları kullanılarak JSON formatında saklanır. Bu şekilde, değişkenlerin ve giriş-çıkış parametrelerinin değerleri, oluşturulacak birim testlerde yeniden kullanılabilir veya bu değerlere benzer rastgele değerler atanabilir. Bu veriler aynı zamanda sistemde bir arşiv dosyasında da saklanır. Literatürde, XML ve Oracle gibi farklı veri depolama ortamları kullanılmıştır.

3. JUnit standartlarına göre oluşturulacak her bir birim test için gerekli olan doğrulama yapısı, FTL (FreeMarker template) şablon motoru kullanılarak hazırlanmıştır. FTL şablon motorunun literatürdeki otomatik birim testi üretimi ürünlerinde kullanılmadığı görülmüştür. Bunun yerine farklı yöntemler geliştirilmiştir.

2. İLGİLİ ÇALIŞMALAR

Yazılım test uygulamalarında birim test üretimi üzerine son 20 yılda yayınlanan farklı çalışmalar incelenmiş olup Çizelge 1'de bu çalışmalar özetlenmiştir.

3. MATERYAL VE YÖNTEM

3.1. Otomatik Birim Test

Yazılım test otomasyonlarının mantığı, elle yazılan testleri otomatikleştirmek ve bir araca dönüştürmektir. Otomasyon, test senaryolarının sürekli tekrarlanmasını ifade eder. Bu durumda, test senaryolarını hazırlayan çalışanlar, kodları çalıştırabilen otomatik bir yazılıma ihtiyaç duyar ve bu yazılımlara yazılım test otomasyonu denir. Yazılım test otomasyonunda, test edilebilir her türlü algoritma, metot ve sınıf yapıları bulunmaktadır. Bunları ayrı ayrı test edebilmek için de birim test yapıları ortaya çıkmıştır. Birim testlerde kullanılan test senaryoları öncelikle geliştirilir ve yazılım bu senaryoların sonuçlarına göre kodlanır. Amaç, test sonuçlarında doğruları ve hataları aramaktır. Test kodu geliştiricileri tarafından bulunan tüm hatalar, düzeltilebiliyorsa çalışma zamanında düzeltilir. Aksi takdirde, test sonuçlarının raporlarını ilgili birimlere yönlendirerek yardımcı olurlar. Doğru kabul edilen her bir birimin testi, ürün yazılımına ait kodunun yazılmasıyla devam ettirilir, böylece her modül tamamlandıktan sonra ürün birleştirilir ve tamamlanır.

Bu çalışmada, birim testlerin üretilebilmesi için otomatik yazılım test otomasyonlarında kullanılabilecek bir opcode ayrıştıma yöntemi tasarlanmıştır. Bu yöntemin tasarım aşamasında, java bytecode, java agent ve javassist gibi önemli yapılar kullanılmıştır.

3.2. Java Agent, Java Bytecode ve Javassist

Java Agent: Java Instrumentation API'sını kullanarak JVM üzerinde çalışan uygulamalara müdahale ederek bytecode manipüle edebilen özel bir Java kütüphanesidir. Genellikle bir jar dosyası olarak hazırlanır. Java agent'ı temsil eden sınıflar, Java API kütüphanesinde bulunan diğer sınıflardan farklı değildir. Ancak onları özel kılan şey, Java kodunu JVM'de çalışan diğer herhangi bir uygulamayı engellemesine izin veren belirli bir kuralı takip etmeleridir. Buradaki tek amaç, sadece bytecode'ları sorgulayan veya değiştiren ajanlar oluşturmaktır. Bu güçlü özellik, normalde bir Java programının yapabileceğinden daha fazlasını sağlar. Bir bakıma, bir programa

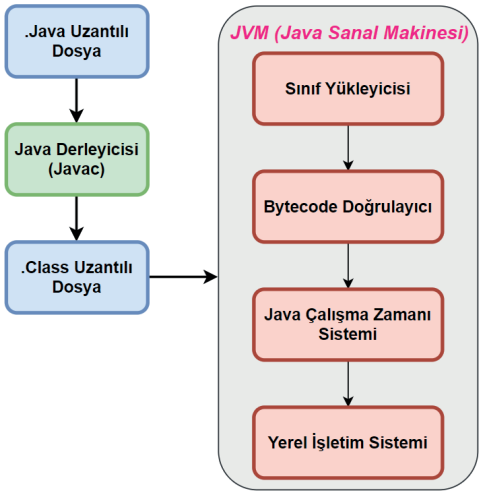
Çizelge-1: Birim test üretimine ilişkin önceki araştırmalar.

Yazar	Çalışma	Bulguları
Csallner ve çalışma arkadaşları [4]	JCrasher adlı otomatik bir test üretim aracı geliştirilmiş. Bu araç, Eclipse IDE ile entegre bir şekilde çalışabilir. Test için JUnit kullanılmıştır.	Test üretimi için verilen örnek Java sınıfının bilgilerini inceledikten sonra rastgele verilerle test edilir. Test aşamasında oluşan hataları da tespit edebilir.
Pacheco ve çalışma arkadaşları [5,6]	JUnit kullanarak Randoop adlı otomatik bir birim testi üretim aracı geliştirilmiş.	Nesne odaklı programlar için geribeslemeli birim testi oluşturabilir ve oluşan hataları yakalayabilir. Rastgele test verileri oluşturmak ve test sonuçlarını birleştirmek için geliştirilmiştir.
Simons ve çalışma arkadaşları [7]	JWalk adlı bir Java birim testi aracı geliştirilmiş.	Örnek bir sınıfın gelişmiş özelliklerini ortaya çıkardıktan sonra birim testlerin sistemli bir şekilde oluşturur. Java test sınıflarının durumu hakkında bilgi sağlayabilir ve JUnit gibi uzmanlaşmış birim testi uygulamalarıyla karşılaştırılmıştır.
Sen [8]	Java ortamında Cute adında bir uygulama geliştirmiş olup, C programlama dilinde yazılmış kodları test etmek etmektedir. Otomatik ve rastgele test mantığını birleştirerek çalışır.	Sembolik kod yürütme kullanarak özel girişler ve kısıtlayıcı çözümleri aşmaya yardımcı olur. Ancak, sistem çağrılarını analiz etme ve doğrusal olmayan tamsayı denklemlerini çözme gibi iyileştirilmesi gereken alanları bulunmaktadır.
Charreteur ve çalışma arkadaşları [9]	Java bytecode programları için otomatik test girdisi elde etmek amacıyla kısıt tabanlı bir akıl yürütme yaklaşımı kullanmışlardır.	Her bir bytecode için geriye doğru arama yapılmasına ve bellek üzerindeki karmaşık kısıtları çözmeye izin veren bir kısıt tabanlı model geliştirilmiştir. JAUT adını verdikleri bu çalışma, Cute, JTEST ve PEX gibi çalışmalar için öncülük etmektedir.
Fraser ve çalışma arkadaşları [10]	EvoSuite adlı bir test oluşturma aracı geliştirilmişlerdir. Java'da yazılmış olan bu test oluşturma aracı geniş özelliklere sahiptir.	Bu araçla gerçekleştirilen tüm testler istenen kriterlerle karşılaştırılabilir. Karşılaştırma sonucunda analiz ve optimizasyon işlemleri gerçekleştirilir.
Sakti ve çalışma arkadaşları [11]	JTExpert adında bir otomatik test oluşturma aracı geliştirilmiş. Java programlama dilinde kullanılabilen bu araç, çalıştırılabilir bir jar dosyasıdır.	JTExpert aracı, test edilen her bir java sınıfı için JUnit formatında otomatik olarak bir test veri paketi oluşturur.
Tanno ve çalışma arkadaşları [12]	CATG adında bir hibrit birim test aracı geliştirilmiş.	Sembolik ve somut girişleri dinamik olarak gerçekleştiren bir kavram olan konkolik testi kullandılar.
Brill ve çalışma arkadaşları [13]	TACKLETEST adında açık kaynaklı bir araç geliştirilmiş. Java uygulamaları için otomatik birim seviyesinde test senaryoları oluşturmak için kullanılır.	Birim testleri için kapsama hedeflerini hesaplama için yeni ve tamamlayıcı bir yöntem uygular.
Higo ve çalışma arkadaşları [14]	Otomatik test oluşturma tekniklerini kullanarak bir veri kümesi oluşturma aracı geliştirilmiş.	Yaklaşık 36 milyon satırlık bir kaynak kodundan fonksiyonel olarak eşdeğer Java metotları veri kümesi oluşturmuşlardır.
Lukasczyk ve çalışma arkadaşları [15]	Dinamik programlama dili olan Python için Pynguin adında otomatik bir birim test oluşturma yazılımı geliştirilmiş.	Yüksek kod kapsamına sahip regresyon testleri üreten genişletilebilir bir Python test oluşturma çerçevesi geliştirdiler.

girilebilir ve bytecode'unu değiştirebilir veya karışıklık çıkarabilir.

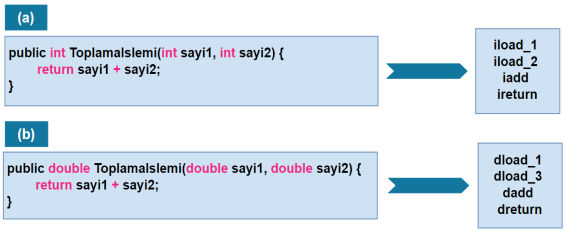
Javassist: Java programları üzerinde dinamik olarak bytecode üretme, değiştirme ve analiz etme işlemlerini sağlayan bir kütüphanedir. Bu kütüphane, bytecode düzeyinde programları manipüle etmek için kullanılır. Java sınıflarını oluşturmak, düzenlemek veya analiz etmek için kullanıcıya geniş olanaklar sunar.

Java Bytecode: C ve C++ derleyicileri assembler ile temsil edildiği gibi, java programları da bytecode ile temsil edilir. Bir java derleyicisinin üreteceği bytecode aslında programın kendisidir. Aynı zamanda bytecode, Java'nın taşınabilirlik ve güvenlik gibi sorunlarına bir çözüm olmak zorundadır. Java derleyicisinin çıktısı yürütülebilir olmadığı için bytecode'a ihtiyaç duyulur. Bytecode'lar JVM tarafından yorumlanır. Bu sayede bytecode'lar çalışma zamanında iyi bir şekilde optimize edilir. Bytecode sayesinde bir java programı birçok farklı ortamda çalıştırılabilir. Şekil 1'de bir bytecode'un çalışma akışı verilmiştir.



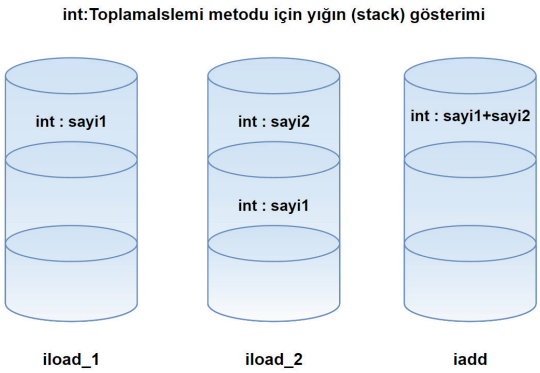
Şekil-1: Java Bytecode'un çalışma akışı

Bir metodun bytecode akışı, JVM için bir dizi talimatı içerir. Her talimat, bir byte opcode'undan ve bu opcode'u takip eden sıfır veya daha fazla operand'dan oluşur. Operand'lar, bir komutun veya talimatın işleme sokulacak olan veri veya hedef değerleridir. Bu değerler, komutun işlevini gerçekleştirmek için kullanılır. Bu bağlamda, bir opcode (bytecode talimatı) belirli bir işlemi temsil ederken, operand'lar bu işlemi tamamlamak için gerekli olan değerleri veya hedefleri içerir. Opcode, gerçekleştirilecek işlemi gösterir. JVM işlem yapmadan önce daha fazla bilgi gerekiyorsa, bu bilgi opcode'u takip eden bir veya daha fazla operanda kodlanır. Her opcode türü bir mnemonic'e sahiptir. Mnemonic, bir bilgisayar programında veya işletim sistemlerinde, spesifik bir işlemi veya komutu temsil etmek için kullanılan sembolik veya hatırlatıcı bir kısaltma yada semboldür. Özellikle bellek adresleri, komutlar veya işlemcilerin komut setleri gibi teknik veya karmaşık konuları temsil etmek için kullanılırlar. Bu bağlamda, bir opcode'un işlevini veya



Şekil-2: Java ile yazılmış parametre alarak toplama işlemini gerçekleştiren bir metodun farklı veri türündeki operand durumları verilmiştir [16]. (a)'da int, (b)'de ise double veri türlerine örnek verilmiştir.

Şekil 2'de java ile yazılmış basit bir metodun almış olduğu farklı veri türündeki parametrelerle kullanımı ve bu kodlara karşılık gelen opcode'lar verilmiştir. Şekil 3'de ise Şekil 2.(a)'da verilen metot yapısının operand yığın gösterimi basit bir şekilde incelenmiştir.



Şekil-3: int veri türü dönüşümlü ToplamaIslemi metodunun yığın üzerinde gösterimi

Bytecode ile ilgili kodların olduğu talimat seti karmaşık olacak şekilde tasarlanmıştır. Tüm talimatlar, tablo oluşturmak için iki kod dışında bayt sınırlarına hizalanmıştır. Opcode'ların toplam sayısı azdır, bu nedenle bytecode'lar

yalnızca bir byte yer kaplar. Böylece, JVM tarafından çalıştırılmadan önce sınıf dosyalarının boyutunu en aza indirmeye yardımcı olur. Ayrıca, JVM uygulamasının boyutunu küçük tutmaya da yardımcı olur. JVM'deki tüm hesaplamalar yığın üzerinde gerçekleştirilir. Bu nedenle, bytecode talimatları öncelikle yığın üzerinde çalışır [17].

Java bytecode, .class uzantılı dosya biçimindeki makine kodudur. Java'da bytecode kullanımı, JVM için komut setidir ve derleyiciye benzer şekilde çalışır. Bytecode'un detaylı incelenmesi, belirli opcode'ların olduğunu ortaya koyar. Bazı opcode'ların önünde şekil 2'de de görüldüğü a veya i gibi harfler bulunur. Örneğin, aload_0 ve iload_2. Bu ön ekler, opcode'un hangi türle çalıştığını temsil eder. a ön eki, opcode'un bir nesne referansını değiştirdiğini ifade eder. i ön eki, opcode'un bir tam sayıyı işlediğini belirtir. Diğer opcode'lar; byte için b, char için c ve double için d olarak kullanılır. Bu ön ekler, işlenen veri türü hakkında bilgi sağlar.

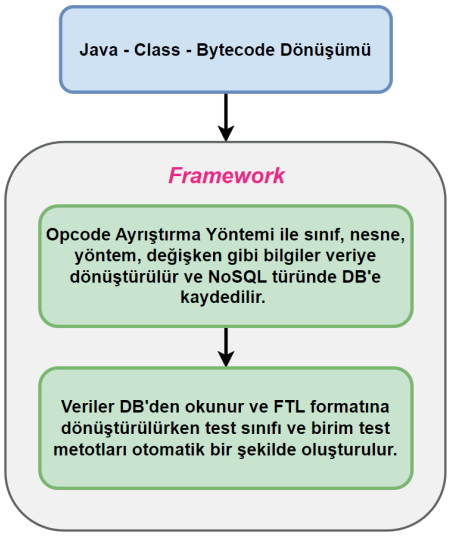
JVM tarafından bytecode'un yürütülmesi için yığın tabanlı bir makine kullanılır. Her bir iş parçacığının bir JVM yığını vardır; bu yığın, verilerini bir çerçevede depolar ve bir çerçeve yığınına dönüştürür. Bir yöntem çağrıldığında her seferinde bir çerçeve yığını oluşturulur ve operand yığını, yerel değişkenlerin bir seti ve mevcut sınıfın çalışma zamanı gibi veriler içerir. Yerel değişkenler seti (dizi), aynı zamanda yerel değişkenlerin değerlerini tutmak için kullanılan yerel değişkenler tablosu olarak da bilinir. Parametreler, dizindeki 0›dan başlayarak saklanır. Yapı bir yapıcı ya da dinamik yöntem içinse, referans 0 pozisyonunda saklanır. Ardından, 1. pozisyon ilk formal parametreyi ve 2. pozisyon ikinci formal parametreyi alır. Bir statik yöntem için, ilk formal yöntem parametresi 0 pozisyonunda saklanır, ikincisi ise 1. pozisyonda. Yerel değişkenler dizisinin boyutu derleme zamanında belirlenir ve yerel değişkenlerin sayısı ve boyutu bir formal prosedür parametine bağlıdır. Operand yığını, değerlerin yerini değiştirmek için LIFO (Son Giren İlk Çıkar) yöntemini kullanır. Belirli opcode talimatları, operand yığınına değerler taşır; diğerleri operatörleri yığından çeker, manipüle eder ve sonucu oluşturur. Operand yığını aynı zamanda yöntemlerin tümünde dönüş değerlerini almak için de kullanılır.

3.3. Opcode Ayrıştırma Yöntemi

Geliştirlen opcode ayrıştırma yöntemi için öncelikle bir Java sınıfı sisteme tanıtılır. Tanıtılan Java sınıfı, bir java agent yardımıyla bytecode dosya formatına dönüştürülür. Bu dönüşüm gerçekleştirilirken; sınıf adı, değişkenler, nesneler, metotlar ve bunların girdi-çıktı parametreleri gibi birçok opcode, java string fonksiyonları kullanılarak satır satır ve kelime kelime ayrıştırılır. Örneğin, invokevirtual, invokestatic, ifge, iflt, ifeq, iinc gibi opcode'lar, bytecode dosyasında java string fonksiyonları kullanılarak kontrol edilir ve karşılık gelen parametreler belirlenir, değişkenlere aktarılır. Aynı zamanda, sınıfta tanımlanan nesne türlerini gösteren opcode'ların verdiği parametreler kontrol edilerek Mock-Stub yöntemlerinin kullanımı için de ayrım yapılabilir. Değişkenlere aktarılan tüm parametreler anlık olarak JSON formatında listelenir ve NoSql veritabanı yapısı kullanılarak veri halinde kaydedilir. Tüm dönüşüm işlemleri tamamlandıktan sonra kaydedilen veriler, FTL şablon motorunun yardımıyla anında birim test formatına otomatik olarak dönüştürülür. FreeMarker Java Template Engine-FTL (Java Template Engine), Apache tarafından üretilen bir şablon motorudur. Şablonlar ve değişen verilere dayalı olarak HTML web sayfaları, e-postalar, yapılandırma dosyaları ve kaynak kodları gibi metin çıktıları üretebilen bir Java kütüphanesidir. Genellikle java gibi genel amaçlı bir programlama diline verileri hazırlamak için kullanılır. Ardından FTL şablonları kullanılarak hazırlanan verileri görüntüler. FTL şablonlarında verilerin "nasıl" sunulacağı ve şablon dışında "hangi" verilerin sunulacağına odaklanır [18]. Böylece, esnek bir framework olarak üretilen opcode ayrıştırma yöntemi sayesinde istenen opcode türü için gerekli kod değişiklikleri yapılabilir. Bu tamamen açık kaynak kodlu olması nedeniyle projeye katkı sağlamaktadır.

Bu çalışmada, çalışma zamanında veri toplayarak otomatik birim testlerin oluşturulmasına imkan sağlayan bir opcode ayrıştırma yöntemi geliştirilmiştir. Söz konusu araç,

Java programlama dili kullanılarak Netbeans ortamında konsol ve masaüstü uygulaması olarak geliştirilmiştir. Bu çalışma, Java programlama dili üzerine kurulu olduğundan, Java tabanlı kodları test etmek için JUnit framework'ü kullanıldı ve birim testlerin oluşturulması bu bağlamda yapıldı. Geliştirilen yöntemin akış şeması Şekil 4'de verilmiştir.



Şekil-4: Çalışma Zamanı Verilerini Toplayarak Otomatik Birim Test Oluşturmada Kullanmak için Opcode Ayrıştırma Yönteminin Akış Şeması

Şekil 5'te opcode ayrıştırma yönteminden bir kesit verilmiştir. Burada getstatic ve ifge gibi ilgili opcode'ların dosya içerisinden nasıl okunacağı hakkında bir algoritma kullanılmıştır. Tamamen string fonksiyonlarından yararlanılarak her bir satırdaki kodlar taranmış ve yakalanan ifadeler ilgili MongoDB'ye gönderilmesi için aracı değişkenlere aktarılmıştır. Ayrıca ifge gibi koşul belirten yapıların kullandığı < ve > gibi operatörlerde kontrol edilmiştir.

3.3.1. Opcode Ayrıştırma Yönteminin Tasarımı

Java - Class - Bytecode **Dönüşümleri:**

Geliştirilen ayrıştırma yöntemi kullanıcılar tarafından kolaylık olması amacıyla bir arayüz olarak NetBeans ortamında tasarlanmıştır. Öncelikle bir Java sınıfı bu sisteme yüklenir. Bu Java dosyası ara yüzde bulunan bir buton yardımıyla .class dosyasına dönüştürülür ve geliştirilen yazılımın neredeyse her bölümünde .class dosyası kullanılır. Sınıf dosyasını derlemek için JavaCompiler kütüphanesi kullanılmıştır. Burada birden fazla Java sınıfı ile ilişkili ve bağlantılı olan bir sınıf sisteme tanıtılırsa, dosyanın bulunduğu konumdaki ilgili diğer sınıflarda sistem tarafından otomatik olarak algılanmaktadır. Geliştirilen sistem okuduğu tüm dosyaları Iterable koleksiyon listesinde saklar ve daha sonra bu liste opcode dönüşüm işlemlerinde kullanılır.

• Java dosyasından Sınıf dosyasına dönüşümün ardından, bytecode dönüşüm işlemleri gerçekleştirilir. Bu işlem için javassist kütüphanesindeki ClassPool, CtClass, CtMethod ve InstructionPrinter alt sınıflar kullanılmıştır. Bu alt sınıflardan bahsedecek olursak;

• ClassPool: Yüklenen sınıfları ve bunların metodlarını, değişkenlerini ve diğer özelliklerini yöneten bir veri havuzu olarak işlev görür. Sınıfların ve bunların bileşenlerinin depolanması, erişilmesi ve değiştirilmesi için kullanılır. Genellikle sınıf dosyalarını temsil eder ve bu sınıflar üzerinde yapılan işlemleri sağlar.

```
// _" + linenumber + ""
if (line_.contains(": getstatic #")) {
    objectClassName2 = line_.substring(line_.indexOf("Field") + 6, line_.indexOf("("));
    document.append("$set", new BasicDBObject().append("ObjectInClass_Field", convertToByteCodeField(objectClassName2)));
    searchQuery = new BasicDBObject().append("Execution Id", "'" + executionId + "'");
    table.update(searchQuery, document, true, false);
    document.append("$set", new BasicDBObject().append("ImportField", StringUtils.substringBeforeLast(objectClassName2, ".")));
    searchQuery = new BasicDBObject().append("Execution Id", "'" + executionId + "'");
    table.update(searchQuery, document, true, false);
}

if (line_.contains(": ifge")) {   //value>=0
    ifMap.add(new bcList("ifge", Integer.parseInt(line_.substring(line_.indexOf("ifge") + 5, line_.length()))));
    ifSetVariable = ifGetVariable("<", linenumber);
    writeToDBtoIf("ifge" + linenumber, ifSetVariable);
}
```

Şekil-5: Opcode ayrıştıma yönteminden bir örnek

• CtClass: Javassist kütüphanesinde temsil edilen bir sınıfın yapısını ve özelliklerini içerir. Bu sınıf, yüklü sınıfları temsil eden bir veri yapısıdır. Sınıfların yapısını değiştirmek, özelliklerini incelemek veya düzenlemek için kullanılır.

• CtMethod: Javassist kütüphanesindeki bir sınıfın veya arabirimin bir yöntemini temsil eder. Bu sınıf, yöntemlerin özelliklerini içerir, örneğin yöntem adı, parametreleri, dönüş türü vb. yöntemleri oluşturmak, değiştirmek veya incelemek için kullanılır.

InstructionPrinter: Javassist kütüphanesinde kullanılan bir alt sınıf olup, bytecode veya işlem talimatlarını, metotların içeriğini yazdırmak ve görüntülemek için kullanılır. Bu sınıf, bytecode'larını okunabilir bir formatta görüntülemek için kullanılır ve genellikle bytecode'unu veya metot içeriğini denetlemek ve anlamak için kullanıcıya yardımcı olur.

Özetle, yapılan çalışmada, okunan java sınıf dosyasındaki sınıf ve metot bilgileri classPool adlı sınıf havuzunda tutulur. Daha sonra, classPool'dan InstructionPrinter ile bilgi alınır ve sonuçlar bir çıktı olarak görüntülenir.

Şekil 6›de java sınıfının bir bytecode'a dönüştürülmüş haline ait örnek bir çıktı verilmiştir. Sınıftaki her metot MethodName satırları arasında ayrılır. Her metot 0 numaralı satır ile başlar ve geri kalan diğer metotlardan ayrımı otomatik olarak yapılmış olur. Bu metot, dreturn anahtar kelimesiyle bir değer döndürür ve aynı zamanda lcd2_w anahtar kelimesiyle parametreleri aldığını gösterir.

```
MethodName : InterestRate
0: dload_0
1: ldc2_w #2 = int 100.0
4: ddiv
5: dload_2
6: dmul
7: dreturn
MethodName : InterestRateEx
0: dload_2
1: dconst_1
2: dcmpg
3: ifge 8
6: dconst_1
7: dstore_2
8: dload_0
9: ldc2_w #2 = int 100.0
12: ddiv
13: dload_2
14: dmul
15: dreturn
```

Şekil-6: Bir Bytecode örneği

Bu bytecode'lar, Javassist kütüphanesi tarafından okunur ve ilgili sınıfın dosya adı ve yolu gibi parametrelerde ClassPool, CtClass ve CtMethod gibi yapılar içinde saklanır. Bytecode'a dönüştürülen tüm kod satırları, Netbeans arayüzünün konsol ekranında görüntülenebilir; ayrıca metin belgeleri olarak da çıktı alınabilir. Bu ayrıştırma yöntemi için geliştirilen başka bir sınıf ise FindObjectsInClasses'dir. Bu sınıfın yardımıyla, her bir bytecode, bytecode dosyasının çıktısından satır satır okunur ve bu kodlar NoSQL formatında MongoDB veritabanına depolanır. Bunun için, ConnectionDB olarak adlandırılan bir sınıf framework içerisinde oluşturulmuştur. Bu sınıfta, com.mongodb kütüphanesinin yöntemlerini kullanarak bir koleksiyon oluşturma ve koleksiyondaki verilere erişme gibi özellikler bir araya getirilmiştir. Bu sınıf aynı zamanda tüm JSON formatındaki verilerin bir metin belgesinde arşivlenmesini sağlar.

Burada önemli olan nokta, bilgilerin geri alınması ve veritabanına kaydedilmesinde java string fonksiyonlarından yardım alınmasıdır. Bu, çalışma için geliştirilen opcode ayrıştırma yöntemi yardımcı olmaktadır. Sınıfta kullanılan değişkenler, varsa metot isimleri, aldıkları parametreler ve gönderdikleri sonuçlar, oluşturulan tüm nesneler, kullanılan koşul ve döngü bloklarına ilişkin bilgiler, sırasıyla string fonksiyonlarıyla alınır. Alınan bilgiler, Şekil

7 ve Şekil 8'de görüldüğü gibi MongoDB'de saklanır. Bunun yanı sıra, invokevirtual, getfield, invokestatic, getstatic, ifge, ifle, iflt, ifgt, ifeq, ifne, iinc, if_icmp ve goto gibi en sık kullanılan opcode'larla ilgili bilgiler de kaydedilir. Aynı zamanda, bu bilgiler işlemin yapıldığı gün ve saatine göre sistem dosyaları altında metin belgesi (.txt) biçiminde arşivlenir. Böylece MongoDB tarafından iki koleksiyon kullanılır. Bunlardan ilki, Şekil 7'da görülen byteCoding adlı koleksiyondur.


```
/* 1 */
{
    "_id" : ObjectId("622e1bb20fb2df14e4b6d6ce"),
    "Execution Id" : "'25'",
    "MethodName" : "'InterestRate'",
    "ClassName" : "'CalculateCredit'",
    "Returned" : "'728.28'"
}

/* 2 */
{
    "_id" : ObjectId("622e1bb20fb2df14e4b6d6cf"),
    "Execution Id" : "'26'",
    "MethodName" : "'InterestRateEx'",
    "ifge12Line" : "< 1.0",
    "ClassName" : "'CalculateCredit'",
    "Returned" : "'1936.69'"
}

/* 3 */
{
    "_id" : ObjectId("622e1bb20fb2df14e4b6d6d0"),
    "Execution Id" : "'27'",
    "MethodName" : "'InterestRateFor'",
    "Loop" : "InterestRateFor",
    "ClassName" : "'CalculateCredit'",
    "Returned" : "'475800'"
}

/* 4 */
{
    "_id" : ObjectId("622e1bb20fb2df14e4b6d6d1"),
    "Execution Id" : "'28'",
    "MethodName" : "'InterestRateIF'",
    "ifle41Line" : "> 7.0",
    "ifge45Line" : "< 9.0",
    "iflt52Line" : ">= 3.0",
    "ifgt56Line" : "<= 6.0",
    "ifle63Line" : "> 1.6",
    "ifgt67Line" : "<= 2.95",
    "ClassName" : "'CalculateCredit'",
    "Returned" : "'4.56'"
}
```


Şekil-7: *byteCoding* adlı koleksiyonun yapısı

NoSQL yapısı altında, veri anahtar-değer (key-value) yapısıyla listelenir. Anahtar alanlarının adlandırılması bazı alanlarda yapılacak işlemlere ilişkin olurken, diğerlerinde bytecode terimlerini hatırlatır. Örneğin, MethodReturnType adlı anahtar alanı, methodun gönderdiği sonucu depolar; ifge45Line adlı anahtar alanı ise if bloğunun hangi işleme karşılık geldiğinin değerini saklar. MongoDB'de kullanılan ikinci koleksiyon adı ise kayitlar'dır. Şekil 7'de görüldüğü gibi, sınıflar ve metodlar hakkında tüm bilgiler kaydedilmiştir. Öyle ki, mock uygulanabilecek aday işlemlere ait nesnelerin sınıf ve nesne adları, bu liste içinde mocking anahtar kelimesi altında yer alır.

Veri Aktarımı - Şablon Dönüştürme:

MongoDB'de arşivlenen tüm veriler, veri okuma veya yazma gibi işlemler için otomatik birim test yazılımı ile iletişim kurabilmelidir. Bu nedenle, veritabanı ile yazılım arasında iletişimi sağlamak için bir POJO sınıfı yazılmıştır. Her türlü basit seviyedeki test senaryolarına cevap verebilecek şekilde kurucu ve getter-setter yöntemleri, değişkenleriyle birlikte tanımlanmıştır. POJO sınıfı, geliştirilen ayrıştırma yöntemi içerisinde veri taşıyıcı olarak işlev görür.

Şekil 7 ve Şekil 8'de de görüldüğü üzere veriler, veritabanında iki tabloda yani koleksiyonda depolanmaktadır. Bu koleksiyonlar, birleştirilmiş ve id'ler kullanılarak birbirleriyle ilişkilendirilmiştir. Bu ilişkilendirilmiş koleksiyonlardan veri okunabilmesi için iki farklı sınıf yazılmıştır. Bunlardan ilki, framework için geliştirilmiş GetItFromMongoDB sınıfıdır. Çalışma zamanında okunan koşul ve döngülerin kodları byteCoding koleksiyonunda saklanmıştır. Bu koleksiyonda okunan veriler, birim test koduna dönüştürülmek üzere FTL formatına yönlendirilir ve bu işlem POJO sınıfının yardımıyla gerçekleştirilir. ByteCoding koleksiyonunda 'if' ile başlayan anahtar kelimeler için değişkenler; sınıf adı, metot adı ve dönüş değerleri ile birlikte POJO sınıfına iletilir. 'if' ile başlayan tüm değerler bir metot içerisinde kullanılan şart yapılarının tüm olasılıklarını karşılayacak şekilde opcode ayrıştırma yönteminde alt algoritması geliştirilmiştir.

MongoDB veritabanından veri çekmek amacıyla ReadDataFromDB sınıfı oluşturulmuştur. Bu sınıfta, birleştirme işlemleri için iki koleksiyona bağlanmıştır. İlgili tüm değişkenler hem veritabanından okunur hem de birleştirme işlemleri için güncellenmesi gereken alanlar olup olmadığı kontrol edilir. Tüm değişkenler, çerçevenin ana sınıfına yönlendirilir. Veritabanından alınan tüm veriler ise, taşıyıcı sınıf POJO ile çekilir. Veritabanında depolanan bu verilerle birlikte, birim testleri istenen dosya biçiminde FTL kullanılarak otomatik olarak hazırlanır ve bir çıktıya dönüştürülür. Bu çıktıların içeriği, çalışma zamanında toplanan

verilere dayalı olarak oluşturulan yeni bir sınıfa ait kodlardır.

4. DENEYSEL SONUÇLAR

Opcode Ayrıştırma Yönteminin Uygulanması:

Uygulamanın backend kodları tamamlandıktan sonra, frontend kodları ile arayüzü hazırlandı. Böylece, uygulama sadece bir konsol ortamı olarak değil, aynı zamanda bir masaüstü platformu olarak da geliştirildi. Bunun için UnitTestGeneratorGUI adlı bir sınıf tasarlandı. Bu tasarım için hem İngilizce hem de Türkçe dillerinde arayüzler oluşturuldu. Şekil 9'de verilen ekran görüntüsü tasarlanan bu sınıfa aittir. Sınıf, framework'ün ilgili bölümlerinin çalışması için gerekli hiyerarşiyi kullanır; tüm komut düğmeleri, listeler ve metin kutuları bu hiyerarşideki sınıflara ait işlevleri gerçekleştirir. Deneyler, Windows 10 işletim sistemine sahip, i7 2.50 GHz CPU ve 16.0 GB belleğe sahip bir bilgisayarda JDK 11 çalıştırılarak gerçekleştirildi.

Çalışma kapsamında çeşitli java sınıfları ile ilgili test sınıfları gerçekleştirildi. Geliştirilen yöntem sadece değişkenler, nesneler ve metotlar üzerinde çalışmıyor. Aynı zamanda, koşullu yapılar ve döngüler gibi ek özellikleri de kapsamaktadır. Şekil 9›da da görüldüğü gibi, arayüz iki ana bölümden oluşmaktadır. İlk olarak, test dosyasının hazırlanacağı java dosyasını sisteme yüklemek için dosya seç düğmesi tıklanır. Dosya yüklendikten sonra, ikinci kısımda sekmeler yer almaktadır. Seçilen Java sınıfının içeriği, bir döküm olarak Java Kod Dosyası penceresine gelir. Sınıf kodlarındaki her bir kod bytecode'a dönüştürülecek olsa da, yorum satırlarındaki ifadeler ya da kod cümleleri bytecode'a dönüştürülmez. Bu özellik tüm derleyicilerde bulunmaktadır. Ayrıca şekildeki örnekte, bu sınıfın hem koşul hem de döngü yapıları içerdiği görülmektedir. Bilgi al düğmesine tıklandığında, sınıfın adı ve ait olduğu metodların adları ekranın sağ tarafında listelenmektedir. Bunun için arka panda çalışan GetInfoAboutJavaAndByteCodeFile adlı bir sınıf tasarlandı. Burada öncelikle, dosya yoluyla okunan java dosyası, JavaCompiler kütüphanesi kullanılarak bir sınıf dosyasına dönüştürülmektedir. Bu dönüşümle birlikte ilgili sınıfın bytecode'u da elde edilmektedir. Bu dönüşümler tamamlandıktan sonra ilgili sınıfın adı ve sahip olduğu metotların adları bir dizi değişkeni ile toplanmakta ve bu bilgi ekranın sağ tarafındaki listede gösterilmektedir. Bytecode dosyasını oluştur düğmesine tıklandığında, Opcode ayrıştıma yöntemine ait alt fonksiyon ve özellikler çalışır ve Şekil 10'da gösterilen Bytecode Dosyası sekmesi aktif hale gelir.

Javassist kütüphanesinin yardımıyla, ilgili örnek sınıfın .class dosyası oluşturuldu ve bytecode'a dönüştürüldü. Bytecode'ları, javassist kütüphanesi tarafından belirlenen standarta uygun olarak, metot isimlerine göre ayrılmıştır. TestFile Oluştur düğmesine tıklanıldığında tüm bilgiler MongoDB'ye aktarılır ve FTL dönüşüm işlemleri gerçekleştirilir.

Şekil 11'de üçüncü sekme ile FTL şablonuna ait çıktıların bu ekrana aktarılan test sınıfı ve birim test metotları gösterilmiştir.

Şekil 12'de dördüncü sekme ile MongoDB


```
/* 1 */
{
    "_id" : ObjectId("622e16570fb2df4654f26bbb"),
    "Execution Id" : "'1:1'",
    "Kaynak" : "'public static double org.brutusin.instrumentation.logging.CalculateCredit3.CalculateInterest(double)'",
    "Baslangic Suresi" : "'Sun Mar 13 19:05:43 EET 2022'",
    "Degiskenler" : "'[484.0]'",
    "ClassName" : "'CalculateCredit3'",
    "MethodName" : "'CalculateInterest'",
    "MethodReturnType" : "'double'",
    "MethodParameters" : "'[double arg0]'",
    "ObjectInClass_Field" : "intRate",
    "ObjectClassName_Method" : "InterestRate",
    "Mocking" : "InterestRate.monthlyRate",
    "ImportMethod" : "org.brutusin.instrumentation.logging.InterestRate",
    "ImportField" : "org.brutusin.instrumentation.logging.CalculateCredit3",
    "Toplam Suresi" : "'277 ms'",
    "Returned" : "'5.808'"
}
```


Şekil-8: *kayitlar* isimli koleksiyonun yapısı

- JSon Görünüm dosya formatlarına ulaşılabilmektedir. İki sekme bulunmaktadır. MongoDB yazılımından çekilmiş iki NoSQL formatında koleksiyonlar bulunmaktadır. İlk sekmede; framework tarafından toplanan veriler kaydedildiğinde, ilgili java sınıfının adı ve metod isimleri, aldığı parametreler, nesnelerle ilişkilendirilen değerler, bunlara sahip oldukları sınıf adlarına ait isimler, metodlardan dönen değerler gibi bilgiler içeren kayıtlar koleksiyonu listesidir. Ayrıca, bu listede bir taklit (mocking) olup olmadığı gibi bilgiler de yer alır. İkinci sekmede ise; byteCoding koleksiyonunun verileri listelenmektedir. Java sınıfındaki her metodun döngü ve koşul yapılarının ayrı ayrı analizi yapıldı ve bunlarla ilgili değerlerin özet bilgisi ayrı ayrı toplandı ve kaydedildi.

Literatürde yaygın olarak kullanılan diğer otomatik birim test oluşturma araçlarıyla bu geliştirilen otomatik birim test oluşturma yazılımının ve bu yazılım ile elde edilen çalışma zamanı verilerinin toplanmasıyla gerçekleştirilen işlemlerin karşılaştırması, tartışma, sonuç ve önerilier bölümünde ele alınmıştır.

Bu geliştirilen çalışmanın uygulaması https://github.com/SevdanurGENC/Nano-Automatic-Unit-Test-Generator adresinde yayınlanmıştır.

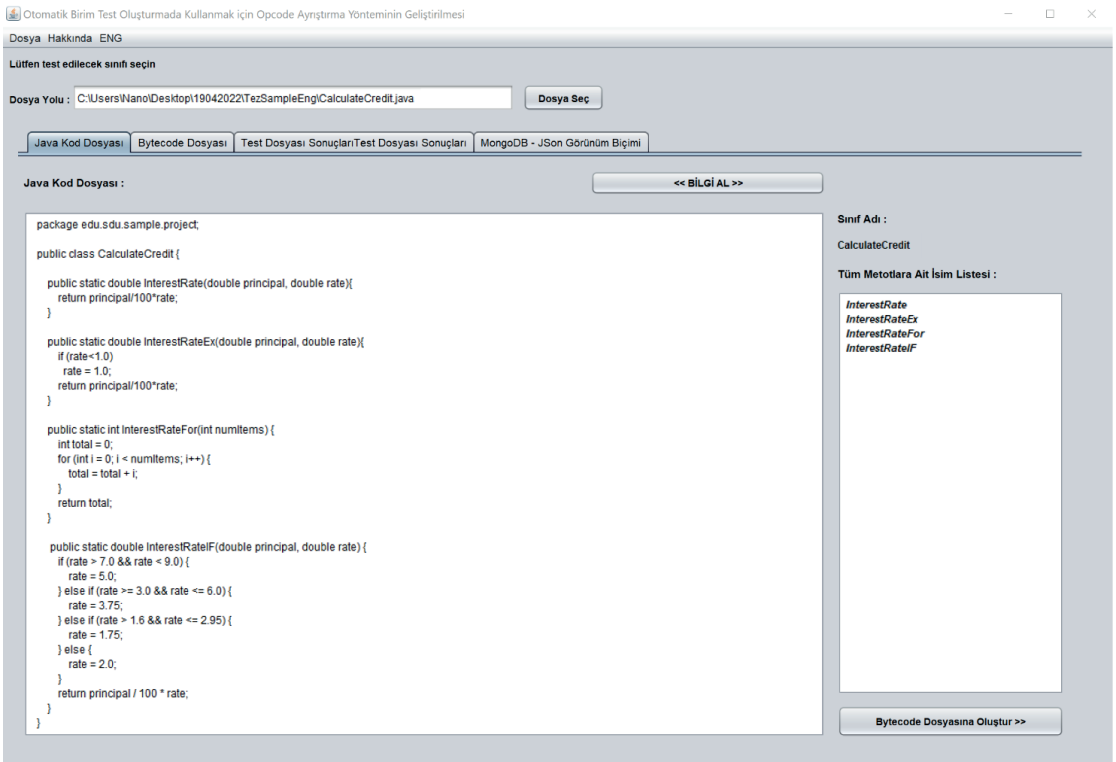
5. TARTIŞMA, SONUÇ VE ÖNERİLER

Bu çalışmada, bir Java Agent yardımıyla çalışma zamanında veri toplayan, bu verileri NoSQL veritabanında depolayan ve bu verileri JTL şablon motorunu kullanarak birim testine dönüştüren bir uygulama geliştirildi. Yapılan çalışmalar ve geliştirilen araç, literatürdeki önceki çalışmalara kıyasla çeşitli açılardan farklı bir yapı kullanmaktadır.

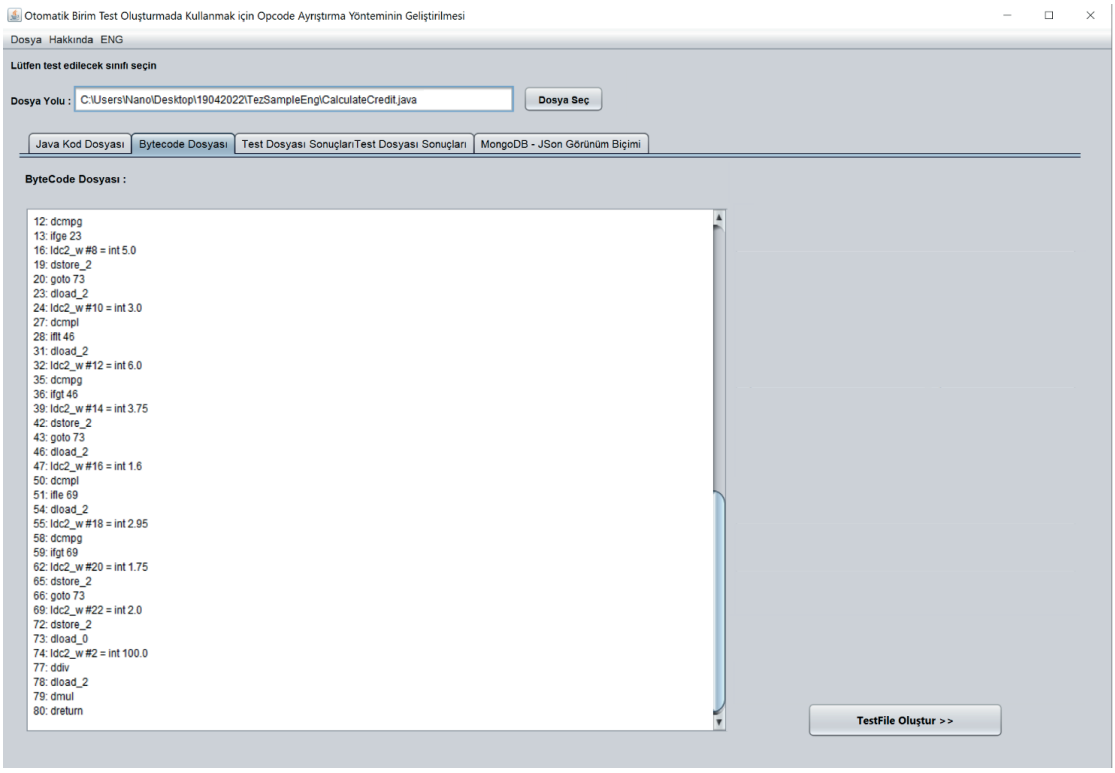
• Opcode ayrıştırma yönteminin üstünlük ve eksikliklerinden bahsedecek olursak;

• Bu çalışmada, önceki literatürdeki bazı çalışmalar test görevlerini otomatize etmek için genetik algoritmalar gibi meta-sezgisel optimizasyon arama tekniklerini kullanmıştır [19]. Ancak bu çalışmada herhangi bir optimizasyon arama veya genetik algoritma yöntemi kullanılmamıştır. Doğrudan, bytecode'ları oluşturulduğu bir java sınıfının opcode'ları, java string metotları kullanılarak ayrıştırılmıştır.

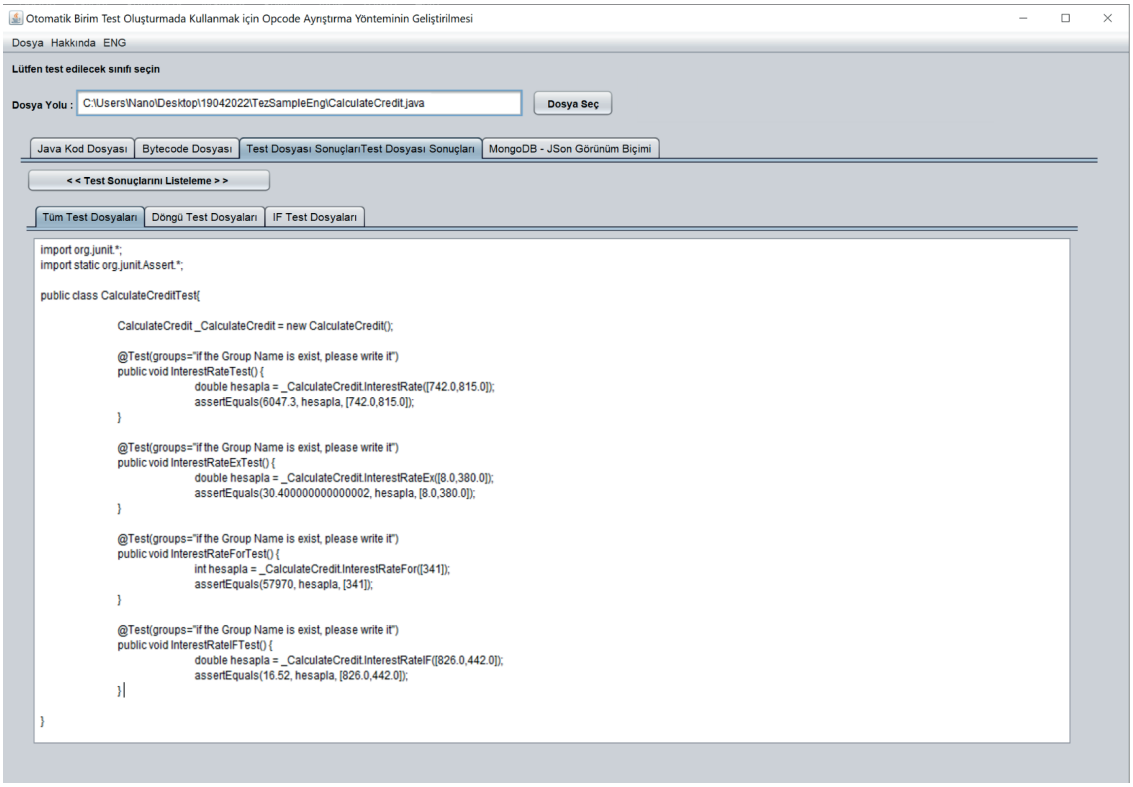
• Bytecode API tarafından sunulan sınırlı ve



Şekil-9: Otomatik birim testi oluşturma Framework'ünün ekran görüntüsü



Şekil-10. Java sınıfının bytecode dönüşümü



Şekil-11: MongoDB'de yer alan *kayitlar* koleksiyonunun JSon formatı

hazır fonksiyonların yanı sıra geliştirilen bu opcode ayrıştırma yöntemiyle, istenen opcode'a karşılık gelen giriş-çıkış parametrelerine bytecode'a dönüştürülen çıktıda erişilebilir. Aynı zamanda, bu geliştirilen yöntem açık kaynak kodlu olduğundan, kullanıcılar her bir fonksiyon içinde kendi ihtiyaçlarına göre istedikleri özellikleri ekleyebilir veya kaldırabilirler.

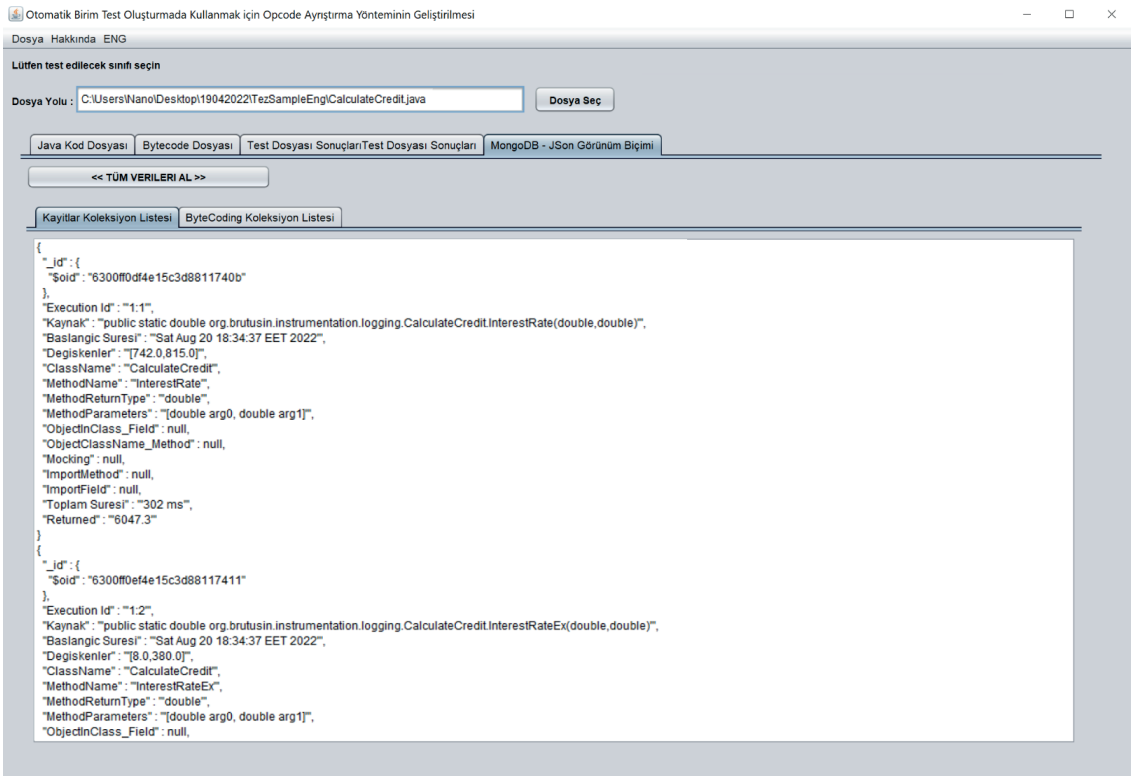
•Bir sınıfta üretilen bir nesneye ait diğer sınıflar aynı yerde olduğu sürece, temel sınıf ile birlikte otomatik olarak işlenebilir. Özellikle mock-stub uygulamaları için düşünüldüğünde, kullanıcıdan ekstra bir eylem gerektirmez ve geliştirilen çerçeve bunu kendisi algılar.

• Bu çalışmada, literatürde Venkatesan ve diğerlerinin çalışmasında olduğu gibi, opcode ayrıştırma yönteminden elde edilen tüm veri türlerini depolamak için esnek bir veri depolama sistemi gerekiyordu [20]. Opcode ayrıştırma yöntemi sonrasında çıktı parametreleri JSON formatında kaydedildiği için, bu veriler herhangi bir NoSQL tabanlı veritabanı yönetim sisteminde kolayca incelenebilmektedir.

• Şimdilik geliştirilen uygulama, belirli bir java sınıfı üzerinde detaylı işlemler yapabilir. Sonraki çalışmalarda geliştirilen bu yöntemle, birbirine bağlı birçok sınıfa ait bir projenin konumundan sonra projenin tüm özelliklerine göre otomatik birim testler oluşturmayı planlanmaktadır.

Test verisi olarak geliştirilen sistemde, int ve double gibi sayısal veri türlerine ait rasgele veriler atanmaktadır. Gelecek çalışmada, tüm veri tiplerini kapsayan rasgele veriler üzerinden otomatik veri seçimi veya hatta içe aktarılabilen örnek bir veri seti planlanmaktadır.

Bu çalışma kapsamında geliştirilen aracın, ulusal yazılım testi alanında da önemli bir yer edineceği düşünülmektedir. Geliştirilen opcode ayrıştırma yöntemi, yazılım testçileri tarafından gerçekleştirilecek birim testlerde etkin bir şekilde kullanılması amaçlanmaktadır. Mevcut formuyla en temel test senaryolarına yanıt verebilen bu araç, bytecode tabanlı bir yapıya sahip olduğundan, çok daha gelişmiş senaryolarda nasıl davranması gerektiği konusunda geliştirilebilecek bir yapıya sahiptir. Bunun en büyük nedenlerinden biri Java'nın açık kaynak kodlu bir sistem olmasıdır. Gelecekte



Şekil-12: MongoDB'de yer alan *kayitlar* koleksiyonunun JSon formatı

ihtiyaç duyulabilecek diğer ilgili bytecode'ları da tercüme edecek olan bu çerçeve için farklı modüller de geliştirilebilir.

Yerli bir ürün olarak hedeflenen otomatik birim test oluşturma yazılımının geliştirilmesiyle, opcode dönüşüm işlemleri gerçekleştirildikten sonra basit bir şekilde otomatik birim testi oluşturulmaktadır.

Bu çalışma şu anda, sınıf içinde oluşturulan nesne bağlantılı diğer sınıfları hariç tutarak, tek bir Java dosyası için çalışmaktadır. Gelecekteki çalışmalarda, önceden tanımlanmış test senaryolarının kullanıcı tarafından seçilmesi ve opcode ayrıştırma yöntemi ile toplanan tüm bilgilerin bu senaryolara direkt otomatik olarak uygulanacağı bir uygulama geliştirilmesi planlanmaktadır. Ayrıca bu ikinci aşamada, framework bütünlüğü olarak birden fazla Java dosyası ekledikten sonra, belirtilen test senaryolarına göre olası otomatik birim testlerinin üretilmesi de hedeflenmektedir. Ek olarak, SF110 sınıf örnekleri gibi en son gelişmelerin takip edildiği örneklerle karşılaştırma yapılacaktır [21]. Kod satırları ve sistemdeki kullanılan bellek süreleri hakkında kıyaslamalar yapılarak analizler hakkında bilgi verilmesi hedeflenmektedir.

Kaynakça

Damar M, Özdağoğlu G, Özdağoğlu A, Eylül Üniversitesi Rektörlüğü D. Küresel Ölçekte Yazılım Kalitesi ve Standartları: Sektörel ve Bilimsel Perspektiften Literatürdeki Eğilimler. DergiparkOrgTr 2018;6:325–48. https://doi.org/10.17093/alphanumeric.404102.

Felice S. JUnit Vs TestNG: Differences Between JUnit and TestNG | BrowserStack n.d. https://www.browserstack.com/guide/junit-vs-testng (accessed January 29, 2023).

Fewster M, Graham D. Software test automation 1999.

Csallner C, Smaragdakis Y. JCrasher: an automatic robustness tester for Java. Softw Pract Exp 2004;34:1025–50. https://doi.org/10.1002/SPE.602.

Pacheco C, Ernst MD. Randoop: Feedback-directed random testing for Java. Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA 2007:815–6. https://doi.org/10.1145/1297846.1297902.

Pacheco C, Lahiri SK, Ernst MD, Ball T. Feedback-directed random test generation. Proceedings - International Conference on Software Engineering 2007:75–84. https://doi.org/10.1109/ICSE.2007.37.

Simons AJH. JWalk: A tool for lazy, systematic testing of java classes by design introspection and user interaction. Automated Software Engineering 2007;14:369–418. https://doi.org/10.1007/S10515-007-0015-3/METRICS.

Sen K. Cute: A concolic unit testing engine for c and java 2007. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=Sen%2C+K.+%282007%29.+Cute+%3AA+concolic+unit+testing+engine+for+c+and+java.+Web%2C+%5Bcited+at+p.39%2C+44%5D.+Retrieved+from++http%3A%2F%2Fosl.cs.uiuc.edu%2F%CB%9Cksen%2Fcute%2F&btnG= (accessed January 29, 2023).

Charreteur F, International AG-2010 I 21st, 2010 undefined. Constraint-based test input generation for java bytecode. IeeexploreIeeeOrg n.d.

Fraser G, symposium AA-P of the 19th AS, 2011 undefined. Evosuite: automatic test suite generation for object-oriented software. DlAcmOrg 2011:416–9. https://doi.org/10.1145/2025113.2025179.

Sakti A, … GP-IT on, 2014 undefined. Instance generator and problem representation to improve object oriented code coverage. IeeexploreIeeeOrg n.d.

Tanno H, Zhang X, … TH-2015 I 37th I, 2015 undefined. TesMa and CATG: automated test generation tools for models of enterprise applications. IeeexploreIeeeOrg n.d.

Tzoref-Brill R, Sinha S, … AN-… IC on, 2022 undefined. TackleTest: A Tool for Amplifying Test Generation via Type-Based Combinatorial Coverage. IeeexploreIeeeOrg n.d.

Higo Y, Matsumoto S, … SK-P of the 19th, 2022 undefined. Constructing dataset of functionally equivalent Java methods using automated test generation techniques. DlAcmOrg 2022:2022. https://doi.org/10.1145/3524842.3528015.

Lukasczyk S, International GF-P of the A 44th, 2022 undefined. Pynguin: Automated unit test generation for python. DlAcmOrg 2022. https://doi.org/10.1145/3510454.

Wan B, Dong S, Zhou J, Qian Y. SJBCD: A Java Code Clone Detection Method Based on Bytecode Using Siamese Neural Network. Applied Sciences 2023, Vol 13, Page 9580 2023;13:9580. https://doi.org/10.3390/APP13179580.

Venners B. Bytecode basics : A first look at the bytecodes of the Java virtual machine. 1996. https://www.

infoworld.com/article/2077233/bytecode-basics.html (accessed January 29, 2023).

FreeMarker Java Template Engine n.d. https://freemarker.apache.org/ (accessed January 29, 2023).

on PM-2011 IFIC, 2011 undefined. Search-based software testing: Past, present and future. IeeexploreIeeeOrg n.d.

Venkatesan P, Rozario RG, Fiaidhi J. Junit framework for unit testing. pdf 2020.

Fraser G, and AA-AT on SE, 2014 undefined. A large-scale evaluation of automated unit test generation using evosuite. DlAcmOrg n.d. https://doi.org/10.1145/0000000.0000000.