T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM PROJELERİNDE KALİTEYİ DÜŞÜREN KODLARIN BULUNMASI

SEVİLAY TANIŞ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
YRD. DOÇ. DR. YUNUS EMRE SELÇUK

İSTANBUL, 2016

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM PROJELERİNDE KALİTEYİ DÜŞÜREN KODLARIN BULUNMASI VE İYİLEŞTİRME METODLARININ ÖNERİLMESİ

SEVİLAY TANIŞ tarafından hazırlanan tez çalışması 27.07.2016 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Yrd. Doç. Dr. Yunus Emre Selçuk

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Yrd. Doç. Dr. Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi _____

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Yrd. Doç. Dr. Tolga OVATMAN

İstanbul Teknik Üniversitesi _____

ÖNSÖZ

Bu çalışmada yazılım kalitesi yüksek olan uygulamalardan, kod kusurunu belirleyen alt ve üst limitler elde edilerek, yazılım projerilerinin kaynak kodları üzerinde kod kusurunun bulunması hedeflenmektedir. Yrd. Doç. Dr. Yunus Emre Selçuk'un dinamik alt ve üst limit oluşturarak kod kusurunun bulunması önerisi, bu tez çalışmasının ilham kaynağı olmuştur.

Çözüm için önceki çalışmalarda kullanılan yöntemlerden esinlenilmiştir.

Tüm yüksek lisans ve tez çalışmam boyunca destek olan değerli danışmanım Yrd. Doç. Dr. Yunus Emre Selçuk'a, tez yazım sürecinde desteklerini esirgemeyen aileme teşekkürü borç bilirim.

Temmuz, 2016

Sevilay TANIŞ

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ vi

ŞEKİL LİSTESİ vii

ÇİZELGE LİSTESİ viii

ÖZET ix

ABSTRACT x

BÖLÜM 1

GİRİŞ 1

1.1 Literatür Özeti 1
1.1.1 Kod Kusuru, Tekrar Tasarım (Refactoring) ve Antipattern Tanımı 2
1.2 Tezin Amacı 3
1.3 Hipotez 3
1.3.1 Çalışmada Kullanılan Ölçütler 4
1.3.2 Dosya Ayrıştırma Yöntemi 5
1.3.3 Alt Limit ve Üst Limitlerin Belirlenmesi: 5
1.3.4 Limitleri Kullanılarak Bulunması Hedeflenen Kod Kusurları 6

BÖLÜM 2

İLGİLİ ÇALIŞMALAR 10

2.1 İlgili Araçlar 10
2.1.1 Teknikler 12

BÖLÜM 3

ÖNERİLEN MODEL 17

3.1 Kullanılan Teknolojiler 17
3.1.1 Roslyn 17
3.1.2 DuplicateFinder 18

3.1.3 C# to Java Converter 18
3.1.4 Visual Studio Calculate Code Metrics 19
3.1.5 IPlasma 20
3.1.6 Essere 20
3.2 Geliştirme 21
3.2.1 Brain Method Model Detayı 27
3.2.2 Data Class Model Detayı 29
3.2.3 Duplicate Code Model Detayı 32
BÖLÜM 4
BULGULAR 35
4.1 Diğer Modeller ile Karşılaştırma 35
BÖLÜM 5
SONUÇ VE ÖNERİLER 42
KAYNAKLAR 44
ÖZGEÇMİŞ 46

KISALTMA LİSTESİ

AST	Abstract Syntax Tree
CBO	Coupling Between Object Classes
CYCLCO	McCabe's Cyclomatic Number
DECOR	DEtection & CORrection
DETEX	DETection Expert
LOC	Line of Code
NOAM	Number of Accesor Methods
NOM	Number of Method
NON	Numeber of Namespace
NOPA	Number of Public Attributes
ORT	Ortalama
PatOIS	Primitives, Operations, Iterator, and Accessors
SPM	Sapma Değeri
WMC	Weighted Method Count

ŞEKİL LİSTESİ

Sayfa

Şekil 1. 1 NReFactory Kütüphanesi ile C# Kodlarının Ayrıştırması 5
Şekil 1. 2 Alt ve Üst Limit Hesaplama 6
Şekil 1. 3 Brain Method Kod Bulma Stratejisi 7
Şekil 1. 4 Data Class Kod Bulma Stratejisi 8
Şekil 3. 1 C# to Java Converter 18
Şekil 3.2 CBO Hesaplama 19
Şekil 3. 3 Visual Studio Code Metrics Örneği 19
Şekil 3. 4 IPlasma Örnek Ekran Görüntüsü 20
Şekil 3. 5 Essere Örnek Ekran Görüntüsü 21
Şekil 3. 6 C# Dosya Ayrıştırma Yöntemi 22
Şekil 3. 7 AST Ağacından Metot Listesini Çağırma 22
Şekil 3. 8 Metotlara Ait Metrik Hesaplamaları 23
Şekil 3. 9 Metrik Sonuçlarını Saklama 23
Şekil 3. 10 Metrik Sonuçlarının Saklandığı Dosyanın İçeriği 23
Şekil 3. 11 C# Sınıfları İçin Metrik Hesaplama 24
Şekil 3. 12 Metrik Sonuçlarını Saklama 25
Şekil 3. 13 Metrik Sonuç Dosyası 25
Şekil 3. 14 Normal Havuzu Hesaplama 26
Şekil 3. 15 Brain Method Kod Kusuru İçeren Metotlar(YTÜ Code Smell Detector) 27
Şekil 3. 16 Brain Method Kod Kusuru İçeren Metotlar(IPlasma) 28
Şekil 3. 17 Brain Method Listesi Oluşturma 29
Şekil 3. 18 Data Classs Kod Kusuru İçeren Sınıflar(YTÜ Code Smell Detector) 30
Şekil 3. 19 Data Classs Kod Kusuru İçeren Sınıflar(IPlasma) 30
Şekil 3. 20 Data Classs Kod Kusuru İçeren Sınıflar(Essere) 31
Şekil 3. 21 Data Classs Kod Kusuru Listesi Oluşturma 32
Şekil 3. 22 Duplicate Code 33
Şekil 3. 23 Duplicate Code Bloğu 34

ÇİZELGE LİSTESİ

Sayfa

Çizelge 2. 1 Marinescuescu'nun kod kusuru bulma doğruluk oranları 13
Çizelge 2. 2 Munro'nun kod kusuru bulma doğruluk oranları 14
Çizelge 2. 3 Moha'nın kod kusuru bulma doğruluk oranları 15
Çizelge 2. 4 Çalışmaların karşılaştırılması 16
Çizelge 3. 1 Normal havuzu CBO değerleri 26
Çizelge 4. 1 Önceki çalışmalar ile karşılaştırılması 36
Çizelge 4. 2 DataClass kod kusuru içeren sınıflar 37
Çizelge 4. 3 YTU-CSD DataClass kod kusuru bulma doğruluk oranları 38
Çizelge 4. 4 IPlasma DataClass kod kusuru bulma doğruluk oranları 38
Çizelge 4. 5 Essere DataClass kod kusuru bulma doğruluk oranları 38
Çizelge 4. 6 BrainMethod kod kusuru içeren metotlar 39
Çizelge 4. 7 YTU-CSD BrainMethod kod kusuru bulma doğruluk oranları 40
Çizelge 4. 8 IPlasma BrainMethod kod kusuru bulma doğruluk oranları 40
Çizelge 4. 9 (YTÜ-CSD/IPlasma/Essere) Doğruluk oranlarının karşılaştırılması 41

ÖZET

YAZILIM PROJELERİNDE KALİTEYİ DÜŞÜREN KODLARIN BULUMASI ve İYİLEŞTİRME METODLARININ ÖNERİLMESİ

Sevilay TANIŞ

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

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

Günümüzde yazılım onarım maliyeti geliştirme maliyetinden çok daha yüksek olabilmektedir. Sınıfların karmaşıklığı arttıkça projeye sonradan dâhil olan yazılımcıların adaptasyonu zorlaşmaktadır bu da yazılım maliyetini oldukça yükseltmektedir. Bu yüzden kodun mümkün olduğunca anlaşılabilir ve kusursuz olması önem taşır. Üstelik günümüzde yazılım projeleri tek bir kişi tarafından değil, 3-5 veya daha fazla kişi tarafından aynı anda geliştirilebilmektedir. Kodun ilerde anlaşılmayacak derecede karmaşık olmasını engellemek için, geliştirme esnasında kirli kodların bulunup düzeltilmesi, yazılım projesini daha anlaşılır kılacaktır. Bu şekilde projenin geliştirme ve onarım maliyeti minimize edilecektir.

Yazılım projelerinde kirli kodları yakalayabilmek için eşik değerlerini belirlemek gerekmektir. Bu değerler dinamik olarak belirlenmelidir, çünkü limitler firmalar arasında farklılık gösterebilmektedir. Yani mutlak alt limit ya da mutlak üst limit diye sınır yoktur. Değinilen ihtiyacı karşılamak için çalışmamızda, kurumların kendi limitlerini oluşturmalarına olanak sağlanmıştır.

Anahtar Kelimeler: Kirli kodların ve antipatternlerin belirlenmesi

ABSTRACT

FINDING CODE THAT DECREASE QUALİTY OF SYSTEM AND PROPOSSING REFACTORING METHODS FOR SOFTWARE PROJECTS

Sevilay TANIŞ

Department of Computer Engineering

MSc. Thesis

Adviser: Assist. Prof. Dr. Yunus Emre Selçuk

Today, software maintenance is more expensive than development cost. While class complexity increases, adaptation of new programmers to software projects also increases, so the cost of software goes up. Therefore, it's important to produce fautless and understandable code. Moreover, software projects are not developed by only one person; even a small scale project needs 3 or more participants working on the code at the same time. To prevent complexty of code, fixing inappropriate codes (code smells) during development has significant value as this process makes software projects more understandable and leads to higher code quality. Consequently in this way, costs of software projects' maintanence will decrease.

Thresholds should be determined to detect code smells in projects. Because of every firm has its own upper and lower limits, they can generate these thresholds dynamically by using their trusted software source codes. In other words, there are no absolute limits. To solve this problem about the aforementioned issue, we provide an opportunity to firms so that they can create their own limits dynamically.

Keywords: Code smell and antipattern detection

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Günümüzde yazılım projelerinde kod kusuru bulmayla alakalı birçok çalışma mevcuttur. Literatürde var olan kod kusuru bulma teknikleri ve ilgili araçlarla alakalı gerçekleştirilen incelemeler aşağıdaki alt bölümlerde ayrıntılandırılmıştır.

Yazılım projesi içinde manuel olarak kirli kodları bulmak kolay iş olmamakla birlikte maliyeti artırmaktadır. Yazılım boyutu büyüdükçe kirli kod bulmak bir o kadar zorlaşmaktadır. Yardımcı bir araç kullanarak kirli kod bulma işinin üstesinden kolayca gelinip maliyet düşürülebilir [1].

Yazılımcılar daha önceden miras projelere dâhil olabilirler. Yazılım projesindeki kirli kodları düzeltmeleri, onlar için uygulamayı daha anlaşılır hale getirecektir, böylece kısa zamanda daha çok iş çıkararak yazılım geliştirme ve onarım maliyetini düşüreceklerdir.

Bu çalışmada özellikle birden fazla yazılımcının geliştirmekte olduğu projelerde kirliliğe sebebiyet veren kodların, geliştirilecek olan bir yazılım ile bulunması amaçlanmaktadır. Araştırmada yazlım kalitesini düşüren faktörler önceden tanımlanacak ve ölçülendirilen bu faktörlerin kaynak kodda aranması veya ölçülmesi yolu ile kirli kodların bulunabilmesi sağlanacaktır.

Tekrar tasarım(refactoring) yazılım projelerinde yapıyı değiştirmeden kaliteyi arttırma yaklaşımıdır. Yani verdiğiniz girdi ve aldığınız çıktı değişmez. Tekrar tasarım kirli kod problemlerine çözüm olarak sunulmuştur. Çalışmamızda kirli kodların bulunmasıyla birlikte, bunların hangi iyileştirme metotlarıyla çözülebileceğinin önerilmesi de

hedeflenmektedir. Geliştirilecek olan gereç bulunan kirli kodlara karşılık çözüm olarak iyileştirme metotları sunulacaktır.

Özetle bu araştırmada kod kusurlarını azaltmak suretiyle yazılım kalitesini artıracak, yazılım onarım maliyetinin düşürecek, yeni katılan yazılımcıların projeye adaptasyonunu kısa surede sağlayacak bir eklentinin geliştirilmesi amaçlanmaktadır.

1.1.1 Kod Kusuru, Tekrar Tasarım (Refactoring) ve Antipattern Tanımı

Kod kusurları zayıf kodlamayı veya tasarlamayı referans eder. Eğer kodda tekrar tasarlanmayı gerektiren yerler varsa orda kod kusurunun varlığından bahsedebiliriz [2]. Çeşitli yazarlar çeşitli kod kusuru tanımlarını literatüre kazandırmıştır. Örneğin Fowler sistemdeki derin problemlere sebebiyet veren ve kendini yüzeyde gösteren kod parçalarını kod kusuru olarak tanımlamıştır [3].

Yazılım projelerini daha basit, daha anlaşılır hale getirmek, var olan kod kusurlarını asgari düzeye indirmek için projelerin iç yapısınında yapılan ve dış davranışı etkilemeyen değişikliklere tekrar tasarım denir. Tekrar tasarım gerçekleştirmek için adımlar basittir. Örneğin bir sınıftan diğer sınıfa alan taşınabilir, satır sayısı uzun olan metotlarda kodun bir kısmı taşınarak yeni bir metot oluşturulabilir ya da birbirine hiyerarşi ile bağlı olan sınıflar arasında kodun bir kısmı üst sınıfa ya da alt sınıfa taşınabilir. Sıralanan bu değişimlerin birleşimi ile kod tasarımında iyileştirmeler yapılarak kod kalitesi artırılır ve ilerde oluşabilecek bulgular engellenmiş olur [3].

Karşıt kalıp (Antipattern) olgusu ilk olarak Koenig tarafından ortaya çıkarılmıştır [4]. Karşıt kalıp çözüm zannedilen bir tasarım kalıbı gibi görünebilir ancak öyle değildir. Karşıt kalıplar aslında tasarım kalıplarının karşıtları olarak yazılım geliştirme sürecinde tekrar eden bazı yanlışları ifade etmektedir [5]. Karşıt kalıp kavramlarını bilmek, yazılım geliştirme sürecinde karşılaşılabilecek ciddi problemleri önceden tahmin edebilmeyi ve tedbir almayı kolaylaştırır. Sık karşılaşılan karşıt kalıplar için spaghetti code, copy-paste programming, god object gibi örnekler verilebilir [6].

1.2 Tezin Amacı

Çalışma yönteminde ilk olarak kod kalitesini düşüren faktörlerin analizi yapılarak, bu faktörlerin kod içerisinde tespit edilebilmesini sağlayacak ölçütler tanımlanacaktır. Tanımlanan bu ölçütlerden faydalanarak kodların kalite durumu hakkında bilgi edilecektir. Kod kalitesini düşüren faktörlerden çalışmaya dâhil edilmesi planlanan kod kusurları şunlardır:

- Aynı kodun birden fazla kopyası
- Data Class
- Brain Method

Yazılım kalitesini düşüren faktörler ile iyileştirme metotları eşleştirilecektir. Böylece yazılımda bulunan kötü kodların çözümüne karşılık eşleştirilen iyileştirme yöntemi önerilecektir. Bulunan kirli kodlara karşılık olarak önerilebilecek metotlar aşağıdaki gibi olabilir:

- Sınıf çıkarma
- Metot çıkarma
- Metotları yeniden isimlendirme

Kirli kodlar tanımladıktan sonra, çözüm olarak iyileştirme metotlarıyla eşleştirilecektir. Bu eşleştirme sonucunda kullanıcıya önerilerde bulunulacaktır. Geliştirilecek olan yazılım C# dilinde Microsoft Visual Studio için eklenti veya bağımsız bir yazılım olarak tasarlanacaktır.

1.3 Hipotez

Bu bölümde kod kusurlarını bulmak için önerilen ölçütler ele alınmaktadır. Kullanılacak ölçütlerle birlikte hangi kusurun ne sıklıkla hangi seviyede ihlal edildiğinin bulunması amaçlanmıştır. Bu yüzden kod kusurlarını bulmak için kullanmak üzere eşik değerleri oluşturulması gerekmektir. Mükemmel eşik değeri diye bir şey yoktur, bundan dolayı çalışmamızda var olan C# projeleri üzerinden bilgi toplanarak eşik değerleri

oluşturuldu. Eşik değerlerinin normal aralıklarının belirlenmesi için kullanılan projelerin kaynak kodlarının oluşturduğu havuza “normal havuzu” adı verilmiştir.

Sistem için nesneye yönelik ve yazılım kalitesinin yüksek olduğu 8 proje üzerinde veri toplandı. Bu verilerle ortalama metot satır sayısı, ortalama if/else sayısı, ortalama yerel değişken sayısı, ortalama while döngü sayısı, ortalama for döngü sayısı, ortalama switch sayısı, ortalama try/catch sayısı, ortalama and ve or işleç sayısı, ortalama get/set sayısı, ortalama public değişken sayısı, en yüksek iç içe koşul sayısı bilgileri elde edildi.

1.3.1 Çalışmada Kullanılan Ölçütler

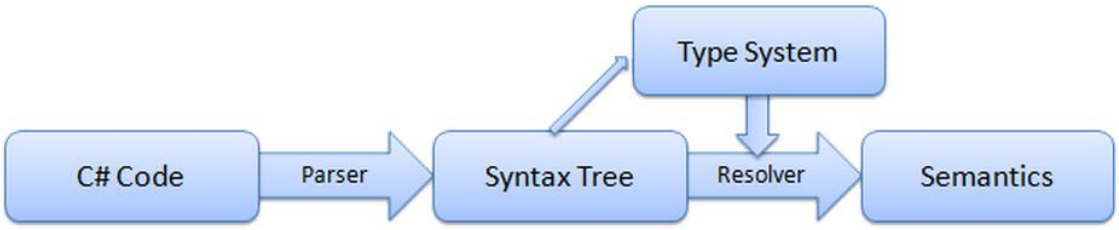
Bu bölümde çalışmada bulunması hedeflenen kod kusuruları için kullanmak üzere belirlenen yazılım ölçütlerinden bahsedilmektedir. Birden fazla ölçüt bir araya getirilerek, farklı kod kusurları ile eşleştirildi. İlerleyen bölümlerde ölçütlerin nasıl bir araya getirildiğini gösteren şekiller, Lanza ve Marinescu’nun çalışmasında önerilen yöntemle çizildi [8]. Bu yöntemde bireysel ölçütler kutularla, bağlantıları ise ve ile veya işleçlerinin mantık devrelerinde gösterildikleri biçimde çizilir. Kod kusuruları ve eşleştirilen ölçütlerle tanımları ise aşağıdaki gibi ayrıntılandırıldı:

- CYCLCO (McCabe’s Cyclomatic Number): Bir metotta bulunan liner bağımsız operasyon sayısını veren metriktir. Brain Method kod kusurunu bulmak için kullanıldı.
- LOC (Lines of Code): Bir metot ya da sınıftaki satır sayısını bulan metriktir. Kodda bulunan satırlar arası boşluklar bu sayıya dâhil değildir. Brain Method kod kusurunu bulmak için kullanıldı.
- NOAM (Number of Accessor Methods): Bir sınıfta bulunan erişim metotların (get/set)sayısını verir. Data Class kod kusurunu bulmak için kullanıldı.
- NOM (Number of Methods): Bir sınıfta bulunan metot sayısını veren metriktir. Brain Method kod kusurunu bulmak için kullanıldı.
- NOPA (Number of Public Attributes): Bir sınıfta bulunan public değişken sayısını verir. Data Class kod kusurunu bulmak için kullanıldı.

- WMC (Weighted Methods per Class): Bir sınıfta bulunan metotların karmaşıklık sayısını veren metriktir. Metot karmaşıklığını hesaplamak için CYCLCO metriğinden faydalanıldı. Data Class kod kusurunu bulmak için kullanıldı.
- MAXNESTING (Maximum Nesting Level): Metot içinde bulunan kontrol ifadelerinin derinliğini hesaplayan metriktir. Brain Method kod kusurunu bulmak için kullanıldı.
- NOAV (Number of Accessed Variables): Ölçülmek istenen metot içinde kullanılan değişken sayısını hesaplayan metriktir. Brain Method kod kusurunu bulmak için kullanıldı.
- CBO (Coupling Between Object Classes): Sınıflar arası bağlılığı ölçen metriktir. Data Class kod kusurunu bulmak için kullanıldı.

1.3.2 Dosya Ayrıştırma Yöntemi

Dosya ayrıştırma işlemi için ICSharpCode.NRefactory.CSharp olan C# kütüphanesi kullanıldı. NReFactory C# kodlarına ait olan sözdizimi ve anlamsal yapılarına erişimi sağlayan kütüphanedir. Bu kütüphane ile sadece C# kodları ele alanınabilmektedir. NRefactory ile C# kodlarına ait söz dizimi ve anlamsal verilere erişim aşamaları Şekil 1.1'de gösterilmektedir [7].



Şekil 1. 1 NReFactory Kütüphanesi ile C# Kodlarının Ayrıştırması

1.3.3 Alt Limit ve Üst Limitlerin Belirlenmesi:

Metrikleri kullanarak yazılım projelerini ölçümlendirip gerekli değerleri elde ettikten sonra bu verileri limitler dâhilinde değerlendirebilmek gerekmektedir. Metot satır sayısı kaç olmalıdır? Bir sınıfta kullanılması gereken ortalama If/else sayısı ne olmalıdır?

Bir sınıfta kullanılması gereken ortalama döngü sayısı kaç olmalıdır? Bu soruların cevaplarını bulmak için alt ve üst limitler belirlendi.

Alt ve üst limitleri belirlemek için gerekli olan iki değere ihtiyacımız bulunmaktadır. Bunların ilki ortalama değer(ORT), diğeri ise sapma değeri (SPM)'dir. Alt limit ve üst limit Şekil 1.2'deki gibi hesaplanmaktadır.

SPM=ORT/100
Alt Limit= ORT – SPM
Üst Limit= ORT+SPM

Şekil 1. 2 Alt ve Üst Limit Hesaplama

Sapma değeri ortalama değerin 100'e bölünmesiyle bulunmaktadır. Bölen değer arttıkça sistemin hassasiyeti de artmaktadır. İhtiyaçlar doğrultusunda bu değer artırılabilir ya da azaltılabilir. Alt limit elde edilen sapma değeri kadar eksik, üst limit ise sapma değeri kadar fazla olarak belirlendi. ORT değerinin bulunması Modelin Detayı başlıklı konuda detaylandırılmıştır.

1.3.4 Limitleri Kullanılarak Bulunması Hedeflenen Kod Kusurları

Elde edilen alt ve üst limit değerleri ile test sistemlerine ait Brain Method ve Data Class kod kusurlarının bulunması sağlandı.

Brain Method

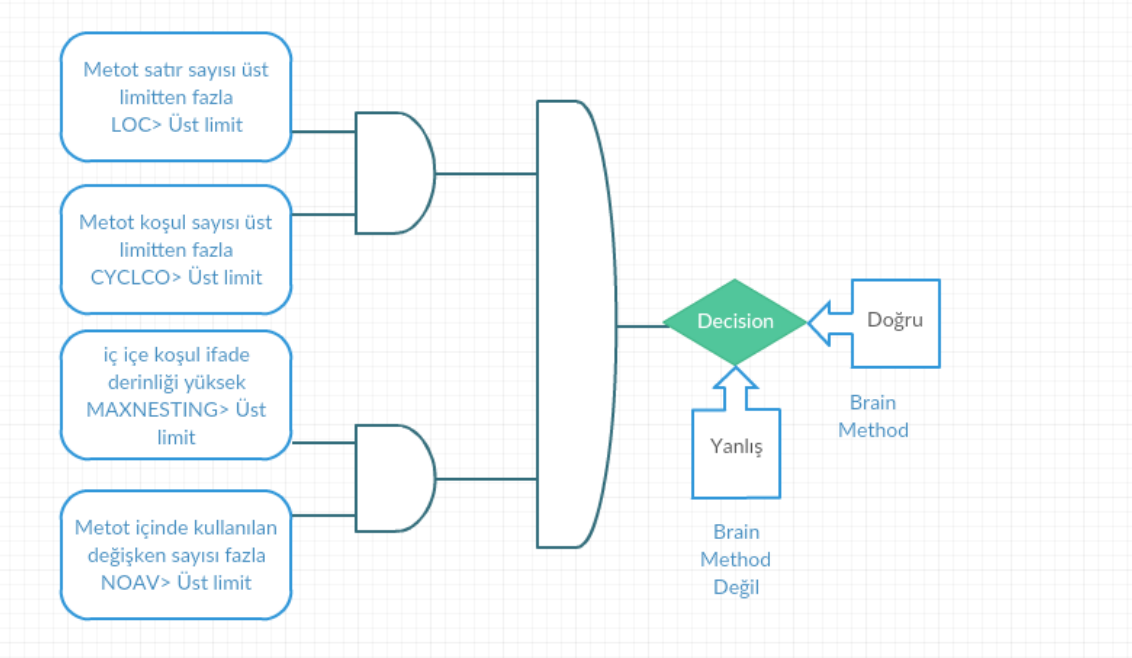
Normal olarak başlayan metotlar zamanla yeni fonksiyonlar eklenerek büyümekte, anlaşılması ve onarılması zor olmaktadır. Brain method kod kusuru bu özellikleri içerme eğiliminde bulunan metotlara denmektedir [8].

Brain Method aşağıdaki stratejiler kullanılarak bulunmaktadır:

- LOC ölçütü kullanarak uzun metotlar bulunur. Uzun metotlar birden fazla iş içerir ve bu da kod kusuruna sebebiyet verir.
- Ayrıştırılmış metot içinde kullanılan yerel değişken sayısına bakılır. Üst limiti aşan değişken sayısı, bu durum metodu Brain Method kod kusuruna daha da yakınlaştırır.

- Ayrıştırılmış metot içinde kullanılan if/else koşullarının sayısına bakılır. Eğer bu sayı üst limiti aşıyorsa, bu durum metodu Brain Method kod kusuruna daha da yakınlaştırır.

Brain Method kod kusuru bulma stratejisi Şekil 1.3'te ayrıntılı bir şekilde gösterilmiştir. Gösterim şekli Lanza ve Marinescu tarafından önerilen şekildedir [8]. Eğer sınanacak olan metot bloğunun satır sayısı üst limiti aşıyorsa, koşul sayısı üst limitten daha fazlaysa, iç içe koşul ifade derinliği üst limiti aşıyorsa ve metot içinde kullanılan değişken sayısı (parametre, metot içi değişkenler, nesneler) üst limiti aşıyorsa bu metot Brain Method kod kusurunu içeriyor demektir. Satır sayısını bulmak için LOC, metot içerisindeki koşul sayısını bulmak için CYCLCO, koşul derinliğini hesaplamak için MAXNESTING, değişken sayısını bulmak için ise NOAV ölçütlerinden faydalanıldı.



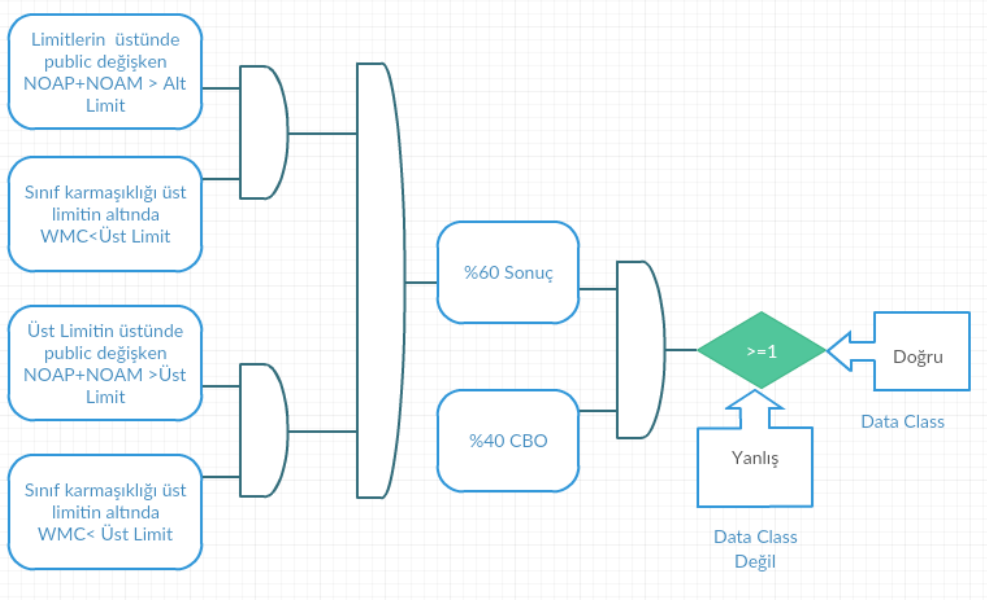
Şekil 1. 3 Brain Method Kod Bulma Stratejisi

Data Class

Data Class başka sınıflar tarafından çok kullanılan ancak fonksiyonu az olan sınıflara denir. Veri kapsüllenmesi az ve fonksiyonu zayıf olan bu sınıflar, nesneye yönelik programlama tasarım prensiplerine ters düşmektedir. Bu tarz sınıfların anlaşılabilirliği düşük, onarım maliyeti yüksek ve test edilmesi zordur [8]

Data Class aşağıdaki stratejiler kullanılarak bulunmaktadır:

- Sınıflar arası bağlılıklara bakılır. CBO metrik kullanılarak, analiz edilmesi istenen sınıfın toplam bağlı olduğu sınıf sayısı bulunur.
- Data class özelliği taşıyan sınıf çok büyük değildir ve fazla fonksiyon sunmaz ancak değişken ve değişken erişim metotları sağlamaktadır. NOPA(Number of Public Metric) ve NOAM(Number of Accessor Methods) metrikleri kullanılarak sınıfta değişken ve değişken erişim metot sayısı bulunur.



Şekil 1. 4 Data Class Kod Bulma Stratejisi

Data class kod kusuru bulma stratejisi Şekil 1.4 'te ayrıntılı olarak gösterilmektedir. Alt limiti aşan public değişken sayısı, sınıf karmaşıklığı üst limitin altında olan kodlar veya üst limiti aşan public değişken sayısı ve karmaşıklığı üst limitten düşük olan kodların toplam değerinin yüzde 60'ı ile proje içinde bulunan diğer sınıflara bağlılık sayısının yüzde 40'ı toplandığında bu sayı 1'i geçiyorsa, sınıf Data Class olarak adlandırılmaktadır.

Duplicate Code

Kaynak kod kopyalarının birden fazla yerde kullanılmasına duplicate code denir. Kopya kodlar yazılım projelerinde değişiklik ve onarım yapılmasını zorlaştırmaktadır, bunun sebebi birden fazla kod bloğunun aynı anda güncellenmesi gerektiğindendir.

Fowler "Birden fazla yerde kullanılan kod parçacıkları ile karşılaşıyorsanız eğer, bunları teke indirerek sistemin daha iyi olacağına emin olabilirsiniz" der [3]. En basit duplicate code problemi aynı sınıfta bulunan, farklı metotlardaki kopyalanmış kod parçacıklarıdır. Çözüm olarak metot çıkarma yapılır ve farklı metotlardan oluşturulan yeni metot çağrılır. Diğer basit duplicate code problemi ise kardeş sınıflar arasında kullanılan ortak yapılardır. Bunun çözümü için ise metot çıkarma ve bir üst sınıfa bu metotu çıkarmakla yapılır.

Eğer alakasız iki sınıf arasında duplicate code problemi varsa, yeni bir sınıf üretilir ve new[1] bileşeni ile ilgili metot çağrılır.

[1] New bileşeni, C# programlama dilince yeni bir örnek oluşturmaktadır. Genellikle sınıflardan yeni bir nesne oluşturmak için kullanılmaktadır.

BÖLÜM 2

İLGİLİ ÇALIŞMALAR

2.1 İlgili Araçlar

Köşker tarafından yapılan "Sınıf ve dosyaların yazılım ölçütleri kullanılarak tekrar tasarım durumlarının irdelenmesi" çalışmasında projelerin sürüm geçmişleriyle kod karmaşıklığı analiz edilerek tekrar tasarlanması gereken sınıflar belirlenmektedir. Tekrar tasarlanması gereken sınıflar için makine öğrenme temelli model kullanılmıştır. Maliyet/fayda analizi yapıldıktan sonra, bu çalışmanın manuel kod kontrolüne göre %78 ile %83 arasında onarım maliyetini azalttığı görülmülmüştür [9].

Astekin tarafından yapılan "Simülasyon Yazılımlarında Kod Klonları" çalışmasında klon tespit yönteminin alan analizinde kullanılabilirliği ve etkinliği araştırılmıştır. Yapılan analiz sonuçlarına göre kod klon tespiti kullanılarak yapılan alan analizi çalışmasında, yeniden kullanılabilir bileşen tanımlanmasını ve referans mimari tasarmını desteklediği görülmüştür [10].

Kapdan tarafından yapılan "Nesneye Dayalı Programlama Tabanlı Yazılımlarda Yazılım Metrikleri Kullanarak Yapısal Kod Klon Tespiti" çalışmasında metriklerden faydalanılarak klon tespiti önerilmiştir [11].

Kendi çalışmamızda kaynak kodların yapısal analizi tercih ederek daha doğru sonuçların üretilmesi amaçlanmaktadır. Kod kusurlarını belirleyen ve iyileştirme yapan PMD, Essere, CBSDetector gibi çeşitli kaynak gereçler mevcuttur.

PMD kaynak kod analizörüdür. Eclipse, JBuilder, BlueJ, NetBeans, dâhil olmak üzere çeşitli IDE'lerle tümleşil olarak çalışabilen PMD'nin bulabildiği başlıca kod kusurları aşağıdaki gibidir:

- Boş try/catch/finally/switch blokları
- Kullanılmayan yerel değişkenler, parametreler
- Gereksiz String kullanımı
- Gereksiz if ifadeleri
- While yerine kullanılan for döngüleri

PMD aracının çalışma adımları ise şu şekildedir:

- PMD'ye analiz edilmesi istenen dosya ismi ve kuralları girilir.
- JAVA_CC olarak üretilmiş kodu dosyaya yazar
- Dosyayı ayrıştırıcı ile AST(Abstract Syntax Tree) 'ye alır
- PMD AST kullanarak açıklamalar, değişkenler gibi kümeler oluşturur.
- Eğer herhangi bir kural akış analizine ihtiyaç duyulursa PMD AST'yi kullanarak kontrol akış ve veri akış düğümlerini oluşturur.
- Kural kümesinde bulunan herbir kural AST'deki düğümü bulur ve problemin var olup olmadığını kontrol eder.
- Sonuçlar son kullanıcıya XML ya da HTML olarak sunulur.

Essesre Kod kusurlarını bulma MARPLE projelerinin bir parçasıdır. Uygulama makine öğrenmesinden faydalanarak Java kaynak kodlarında bulunan kusurları bulur. Sadece Eclipse ortamına entegre edilebilen Essere'nin kod kusuru bulmada odaklandığı alanlar şu şekildedir [12]:

- Veri Sınıfları: Mantıksal hiçbir operatör içermeyen sınıflara veri sınıfı denir.
- Tembel Sınıf(Lazy Class): İçinde birkaç metod barındıran küçük sınıflara Lazy Class denir. Eğer toplam özellik ve metot sayısı oranı aynı pakkette bulunan özellik ve

metot toplamlamlarının ortalama sayısından 2/3 kadarı veya daha azı ise sınıf tembeldir.

- Ortadaki Adam (Middle Man): Middle man 2 veya daha fazla sınıf arasında aracı rolü üstlenir. Nesneler arası dolaylı iletişimi sağlar. Eğer bir sınıfta başka sınıftaki metodları çağıran metot sayısı diğer metotların sayısından büyükse sınıf ortadaki adamdır.
- Uzun Parametre Listesi: Okumayı ve çağırmayı zorlaştıracak derecede parametresi olan metodlar uzun parametre listeli metot olarak tanımlanır.
- Özellik Kıskançlığı(Feauture Envy): Özellik kıskançlığı olan sınıf başka sınıflardaki kaynakları kendi sınıfındaki bulunan kaynaklar yerine kullanır.
- Gereksiz Özellik Açılımı(Indecent Exposure): Eğer bir sınıf erişilmemesi gereken kaynaklarını dışarıya açıyorsa bu sınıfa indecent exposure denir.
- Büyük Sınıf: Eğer bir sınıf içinde çok fazla veri ve metod barındırıyorsa büyük sınıf olarak tanımlanır.

CBSDetector uygulaması Java tabanlı olup, herhangi bir IDE ile tümleşik çalışamamaktadır. CBSDetector, ayrıştırıcı (parser) sınıfları ile verilen kaynak kodu düğüm kümelerine ayırmaktadır. Bu sınıflandırmalar için herbir ayrıştırıcı metoduna kurallar yazılmıştır. Bu uygulama Java kodlarında aşağıdaki kod kusurları ile ilgilenir [13]:

- Veri Yığınları
- Switch ifadeleri
- Mesaj zincirleri
- Ortadaki Adam (Middle Man)

2.1.1 Teknikler

Travasos kirli kodları bulmak için manuel süreci tanıtmıştır. Kod kusuru bulma tekniği otomatize edilmediğinden, büyük sistemler için uygulanması kolay değildir [14].

Marinescu metrik tabanlı yaklaşımı ile kod kusurlarını IPLASMA aracını kullanarak bulmayı amaçlamaktadır. Ölçülebilir kural ifadeleri ile kodda kirliliği bulmayı sağlar. Tasarım problemlerini sınıf, metot ve alt sistem gibi farklı yazılım katmanlarında bulmayı amaçlamaktadır. Filtreleme ve birleştirme mekanizması kullanarak oluşturduğu metriklerle, büyük ölçekli projelerde bile başarı sağlanabilmiştir. Otomatik kod bulma özelliği kullanıldığında ortalama %67 doğruluk oranı tespit edilmiştir. Bu çalışmada bağımlılık (shotgun surgery), wide subsytem interface, özellik kıskançlığı (feauture envy), yanlış konumlandırılmış sınıf (misplaced class), uzun metot (god method), büyük sınıf (god class), büyük paket (god package), veri sınıfları (data class), anti kalıtım (refused bequest) gibi kod kusurları bulunmaktadır. Çizelge 2.1'de kod kusuru bulma tekniğinin sistem üzerindeki sonucu gösterilmektedir. Bu teknik 93 kloc , 18 package, 152 sınıf ve 1284 metod içeren sistem üzerinde uygulanmıştır. Bu çizelgede kod kusurlarına ait yanlış pozitif ve doğru pozitif sayılar bulunarak son alanda doğruluk oranları hesaplanmıştır [15].

Çizelge 2. 1 Marinescuescu'nun (2004) kod kusuru bulma doğruluk oranları

Kod Kusuru	Yanlış Pozitif	Doğru Pozitif	Doğruluk Oranı
FeatureEnvy	11	25	%63
God Method	1	3	%75
Shotgun Surgery	6	9	%60
Refused Bequest	4	18	%81
God Class	2	3	%60
God Package	1	1	%50
Misplaced Class	1	3	%75
Wide Subsys. Interface	1	4	%80
Data Class	1	2	%66

Munro kod kusurlarını sistematik olarak bulabilmek için bir taslak önerir. Bu taslağı bir tasarım desenine benzetmek mümkündür. Taslak kod kusurunun adı, özelliği ve bulma sezgileri olarak 3 bölümden oluşmaktadır. Munro, Marinescu'nun tekniğinde olduğu gibi ayrıca metriklerden faydalanmaktadır. Özellikle java kaynak kodunda otomatik kod kusuru bulma özelliği ile tasarım problemlerini bulmaya odaklanmıştır. Tembel sınıf (lazy class), geçici değişkenler ile alakalı kod kusurlarını bulmayı sağlar. Bu çalışmada amaç java kaynak kodlarındaki kod kusurlarını bularak, sonucu son kullanıcıya excel dökümanı halinde çıktı olarak verebilmektir. Manuel sistemlerle kıyaslanınca daha hızlı sonuç veren bu yöntemde ayrıca hata oranı da daha az olmaktadır [16].

Çizelge 2. 2 Munro'nun kod kusuru bulma doğruluk oranları

	Pozitif	Negatif
Doğru	7	5
Yanlış	1	0

Alikacem ve Sahraoui nesne yönelimli sistemlerde kod kusurları ve kod kalitesi için gerekli prensiplere uyulup uyulmadığını tespit etmeyi amaçlamaktadır. Bu yöntem mühendislerin beklentilerine göre kalıtımlara, nesneler arasındaki ilişkilere ve metriklere göre kuralları uygulayabilmektedir. Metrik üretme için birçok araç mevcut olduğu halde, bu araçlara yeni özellikler eklemek ya da geliştirmek kolay değildir. Bu çalışmada metrik yığınlarının esnek bir şekilde manipüle edilmesi amaçlanmaktadır. Varolan metrikler ile kullanıcının kendi istediği metrikler adapte edilerek yeni metrikler üretilebilinecektir. Alikacem ve Sahraoui çalışmalarında seçme, filtreleme, navigasyon ve operasyon mekanizmalarını birleştirerek PatOIS adında metrik tanımlama dili oluşturmuşlardır. PatOIS ile yeni metrikler varolanlar ile birleştirilerek yeni metrikler üretilebilmektedir. Elle hesaplamanın kolayca mümkün olmadığı, karmaşık uygulamalar için bu yöntem kullanılabilir [17].

Gue´he´neuc, Meur ve Moha yapısal programlama kullanmanın, spagetti kod oluşumuna sebebiyet verdiğini ileri sürmektedirler. Naouel Moha tarafından yapılan çalışmada tasarım kusurları ve bunlara bağlı oluşan kod kusurları ele alınmıştır. Örneğin

spagetti kodun varlığı aslında nesne tabanlı programlama yerine yapısal programlama kullanmaktan kaynaklanır. Bu çalışmanın 3 temel amacı bulunmaktadır. DECOR (DEtection & CORrection) ile kod kusurlarını tanımlamak ve bulmak için somutlaştırma yapılır. DETEX(DETection Expert) ise kod kusuru bulma tekniğidir. Son olarak DETEX tekrar çağrılır ve deneysel doğrulama yapılır. Çalışma antipatterns blob, functional decomposition, spaghetti code ve swiss army knife tasarım desenleri ile ilgilenilir ve buna bağlı olarak 15 kod kusuru üzerinde çalışılmıştır. Sonuçlara göre doğruluk en az Swiss Army Knife karşıt kalıbının belirlenmesinde olmaktadır. Blob karşıt kalıbının belirlenmesinde ise %90 ile en yüksek başarı oranı sağlanmıştır. Ortalamada ise %60,5 başarı sağlanmıştır [18].

Çizelge 2. 3 Moha'nın kod kusuru bulma doğruluk oranları

Kod Kusuru	Doğru – Pozitif	Bulunan Kusur Sayısı	Doğruluk	Tekrar Çağırma	Süre
Blob	39/513 (%7.6)	44/513 (%8.6)	%88.6	%100	2.45s
F.D	15/513 (%3.0)	29/513 (%5.6)	%51.7	%100	0.16s
S.C	46/513 (%9.0)	76/5113 (%15)	%60.5	%100	0.22s
S.A.K	23/513 (%4.5)	56/513 (%11)	%41.1	%100	0.05s
Ortalama	N/A	N/A	%60.5	%100	0.72s

Çizelge 2. 4 Çalışmaların karşılaştırılması

	Ortalama Doğruluk Oranı	Büyük Uygulamalarda Kullanıma Uygunluk	Bulduğu Kod Kusur Sayısı
Travasos	%69	Değil	5
Marinescu	%67	Uygun	9
Munro	%93	Uygun	2
Alikacem ve Sahraoui	N/A	Uygun	N/A
Gue´he´neuc, Meur ve Moha	%60.5	Uygun	4

BÖLÜM 3

ÖNERİLEN MODEL

Önerilen modelde kod ölçütlerinden faydalanılarak Brain Method ve Data Class kod kusurlarının bulunması hedeflenmektedir. Ölçümlerden faydalanılarak kod kusurlarının daha doğru sonuçlar ile bulunması amaçlanmaktadır.

Önerilen modeli gösteren bir Windows uygulaması geliştirilmiştir. Bu uygulama ile kod kusurları son kullanıcı tarafından görüntülenebilmektedir. Önerilen modelde eşik değerlerinin normal aralıklarının belirlenmesi için kullanılan projelerin kaynak kodlarının oluşturduğu havuza "normal havuzu" adı verildiği belirtilmişti. Uygulama, kullanıcının kendi normal havuzunu oluşturmasına da imkân vermektedir.

3.1 Kullanılan Teknolojiler

Kod kusuru bulmak için C# Roslyn, DuplicateFinder, C# to Java Converter, Visual Studio Calculate Code Metrics, Iplasma ve Essere teknolojilerinden faydalanılmıştır. Alt başlıklar ile aşağıda bu teknolojiler detaylandırılmıştır.

3.1.1 Roslyn

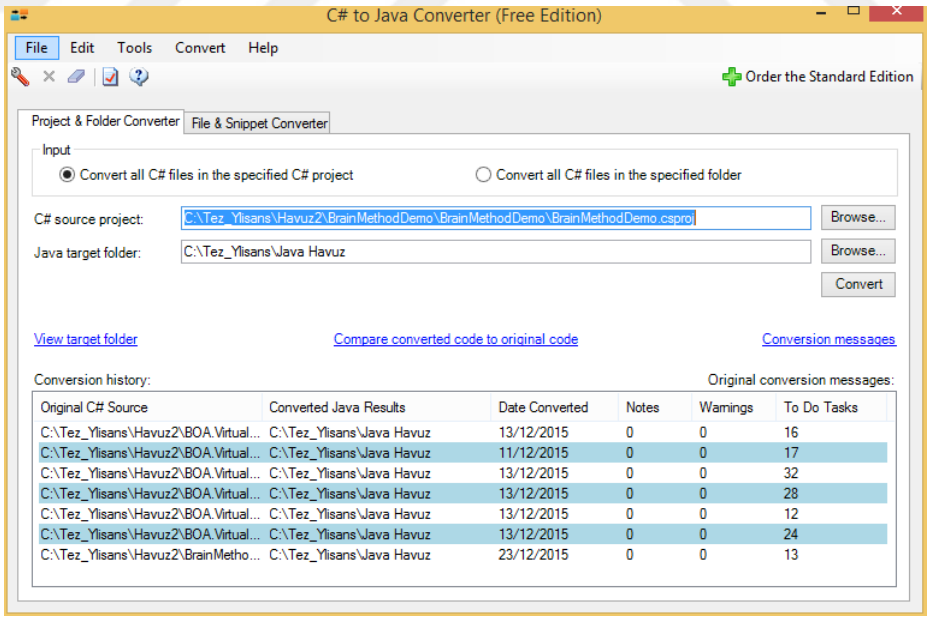
Roslyn YTU Code Smell Detector projesinin geliştirme sürecinde, proje aşamalarını basitleştirmek için kullanılmıştır. Roslyn proje sınıflarını AST (Abstract Syntax Tree) ağacına atarak, ayrıştırılmasını sağlamaktadır. AST içinde ayrıştırılmış olan sınıflar analiz edilme aşamasını kolaylaştırmaktadır. Bu kütüphane hem alt limit ve üst limit elde etme, hem de kod kusuru bulma aşamalarında kullanıldı [19].

3.1.2 DuplicateFinder

DuplicateFinder[1], açık kaynaklı kopya kodları bulan koddur. YTU Kod Smell Detector duplicate kod kusuru bulma aşamasını basitleştirmek için kullanılmıştır. DuplicateFinder doküman tabanlı kaynakları kontrol ederek aynı olan satırları bulmaktadır.

3.1.3 C# to Java Converter

Doğruluk sınaması için YTÜ Code Smell Detector uygulaması ile IPlasma ve Essere uygulamaları karşılaştırıldı. Bu iki aracı kıyaslayabilmek için C# to JavaConverter[2] uygulamasına ihtiyaç duyuldu. Çünkü YTÜ Code Smell Detector C# kaynak kodlarını analiz edebiliyorken, Iplasma ve Essere araçları java kodlarını analiz edebilmektedir, böylece aynı projeler üzerinde testleri yapabilmek için, diller arası çevirici olan bu araca ihtiyaç duyuldu. Test için ayrılan C# kaynak kodlu projeler, programlama dili çevirici araç ile java kaynak koduna dönüştürüldü. Böylece java kodlarını analiz eden uygulamalarla YTÜ Code Smell Detector uygulaması eşit koşullarda mukayese edilebildi. Şekil 3.1'de C# to Java Converter uygulamasının ara yüzü gösterilmektedir.



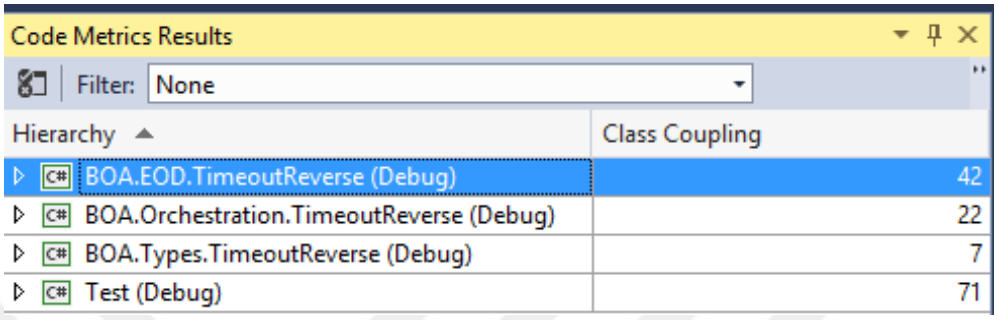
Şekil 3. 1 C# to Java Converter

[1] https://duplicatefinder.codeplex.com/

[2] http://www.tangiblesoftwaresolutions.com/Product_Details/CSharp_to_Java_Converter_Details.html

3.1.4 Visual Studio Calculate Code Metrics

Normal havuzuna CBO değeri düşük projeleri dahil etmek için Visual Studio eklentisi olan Calculate Code Metrics eklenrisi kullanıldı. Bu eklenti kaliteli projeleri seçme aşamasında kolaylık sağlamıştır. Şekil 3.2'de bu eklentiye ait ekran görüntüsü bulunmaktadır.



Şekil 3. 2 Visual Studio Code Metrics Örneği

Code Metrics uygulamasından elde edilen Class Coupling değerleri ve toplam sınıf sayısından faydalanarak CBO değerleri hesaplandı. Şekil 3.3'te CBO hesaplanma formülü gösterilmektedir. NumberOfLinks Alanına Class Coupling olarak gelen toplam sayı, NumberOfClasses alanına ise Projeye ait olan toplam sınıf sayısı girilerek, o uygulama için CBO değeri elde edilmiştir[1]. Bu uygulama alt ve üst limit oluşturmak toplanan projelerde, kalitenin düşük olmadığını göstermek için kullanıldı. Norm havuzu oluşturmak için CBO değeri düşük olan projeler tercih edildi.

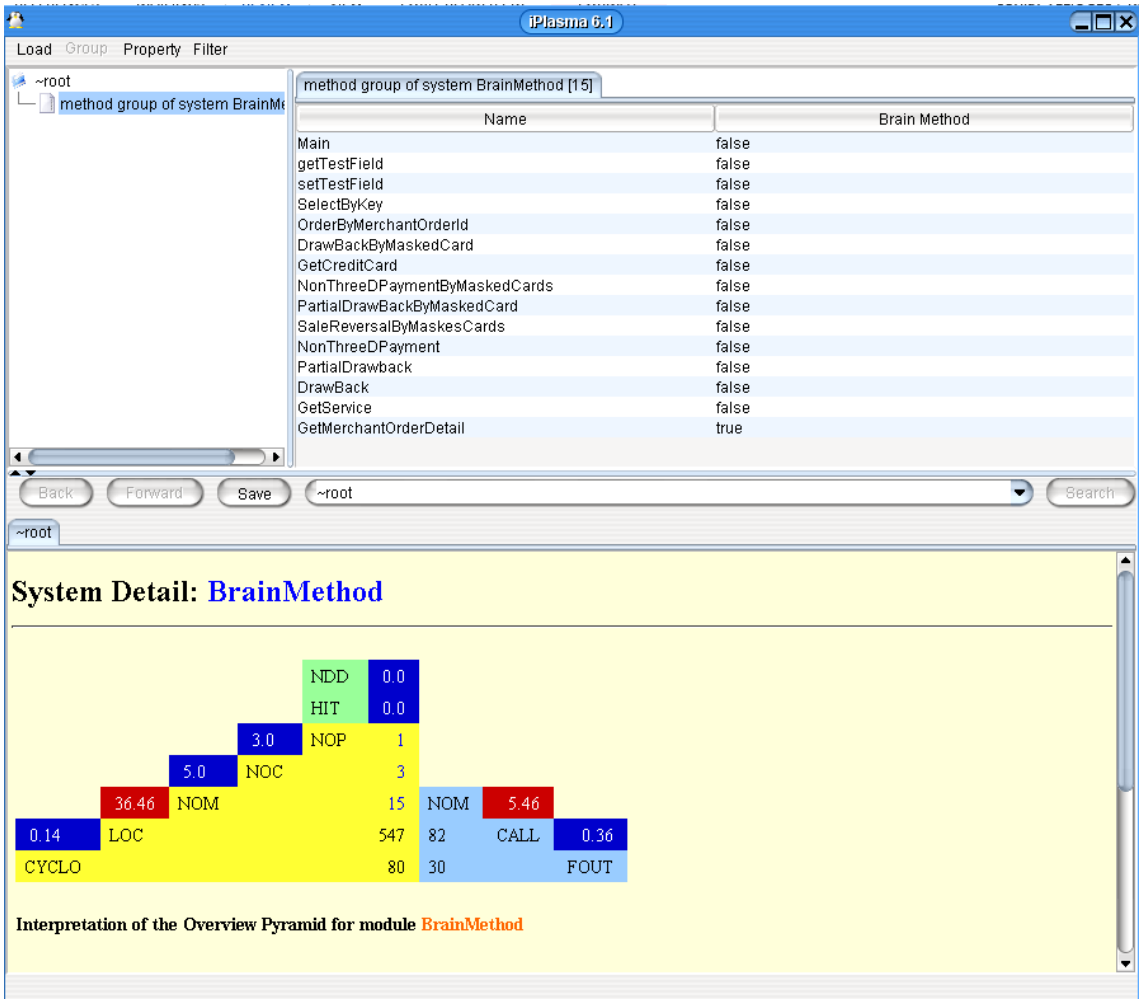
$$CBO = \frac{NumberOfLinks}{NumberOfClasses}$$

Şekil 3.3 CBO Hesaplama

[1]http://support.objecteering.com/objecteering6.1/help/us/metrics/metrics_in_detail/coupling_between_object_classes.htm

3.1.5 IPlasma

Iplasma uygulaması, YTÜ Code Smell Detector uygulaması ile kıyaslanmak için kullanıldı [20]. Iplasma java kodlarına ait kod kusurlarını bulurken YTÜ Code Smell Detector C# kodlarına ait kusurları bulabilmektedir. Aynı projeler üzerinde test yapabilmeyi sağlamak için, projenin C# ve C# to Java Converter uygulaması kullanılarak Java versiyonu oluşturuldu. Bu projeler ile elde edilen sonuçlar kıyaslandı. Şekil 3.4'te IPlasma uygulamasına ait Brain Method kod kusurunun BOA.VirtualPos.WCFService projesi içinde arandığı ekran görüntüsü gösterilmektedir.

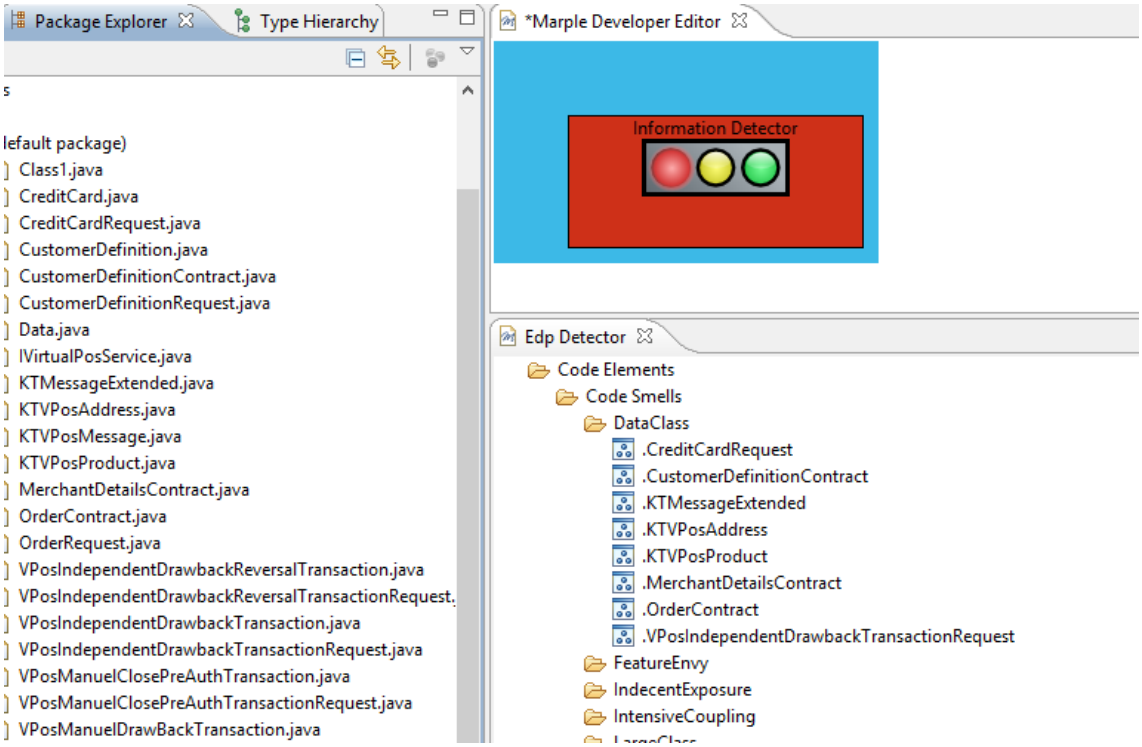


Şekil 3. 4 IPlasma Örnek Ekran Görüntüsü

3.1.6 Essere

Essere eclipse eklentisi, YTÜ Code Smell Detector uygulaması ile kıyaslanmak için kullanıldı. Essere java kodlarına ait kod kusurlarını bulurken YTÜ Code Smell Detector

C# kodlarına ait kusurları bulabilmektedir [21]. Aynı projeler üzerinde test yapabilmeyi sağlamak için, projenin C# ve C# to Java Converter uygulaması kullanılarak Java versiyonu oluşturuldu. Bu projeler ile elde edilen sonuçlar kıyaslandı. Şekil 3.5'te Essere eklentisine ait Data Class kod kusurunun BOA.VirtualPos.WCFService projesi içinde arandığı ekran görüntüsü gösterilmektedir.



Şekil 3. 5 Essere Örnek Ekran Görüntüsü

3.2 Geliştirme

Modelde eğitim verisi ve sınama verisi olmak üzere iki tür veri sınıfı bulunmaktadır. İlk aşamada eğitim verisine ait proje grubu alınarak kod kusuruları için alt ve üst sınırlar oluşturulacaktır. Elde edilen bu sınırlar daha sonra sınama verileri için kullanılmak üzere dosya sisteminde saklanacaktır. Böylece kod kusurularının bulunması otomatize edilecektir.

Şekil 3.6'te yer alan kod bloğunda C# dosyalarının ayrıştırılarak anlamsal ve söz dizimi verilerine erişim aşamaları gösterilmektedir. Dosya ayrıştırma yöntemi Hipotez başlığı altında ayrıca detaylandırılmıştı.

```
public static List<Method> GetMethodMetric(string filename) {
     List<Method> MethodList = new List<Method>();
     var sb = new StringBuilder();
     if (Path.GetExtension(filename) == ".cs") {
          string sourceCode = File.ReadAllText(filename);
          CSharpParser parser = new CSharpParser();
          SyntaxTree syntaxTree = parser.Parse(sourceCode, filename);
          if (parser.HasErrors) {
               foreach (var error in parser.ErrorsAndWarnings) {
                  sb.AppendLine(string.Format("ErrorType: {0} \nRegion:{1}
                  \nMessage:{2}", error.ErrorType, error.Region, error.Message));
               }
          }
          else  {
               var methods = syntaxTree.Descendants.OfType<MethodDeclaration>();
               foreach (var method in methods) {
                    var body = method.Body;
                    var response = GetMethodDetail(body.ToString());
                    var rr = method.GetChildrenByRole(Roles.Variable);
                    Method mth = new Method();
                    mth.MethodName = method.Name;
                    mth.ClassName = filename;
                    mth.NumberOFStatment = response.NumberOfIfStatment;
                    mth.NumberOfVariable = response.NumberOfVariable;
                    mth.MaxNestingLevel = response.MaxNestingLevel;
                    mth.LOCOfMethod = method.EndLocation.Line -
                    method.StartLocation.Line;
                    mth.NumebrOfParameter = method.Parameters.ToList().Count();
                    mth.NumberOfForStatment = response.NumberOfForStatment;
                    mth.NumberOfWhileStatment = response.NumberOfWhileStatment;
                    mth.NumberOfSwitchStatment = response.NumberOfSwitchStatment;
                    mth.NumberOfIfStatment = response.NumberOfIfStatment;
                    MethodList.Add(mth);
               }
          }
      }
      return MethodList;
}
```

Şekil 3. 6 C# Dosya Ayrıştırma Yöntemi

Şekil 3.7'de son kullanıcıdan alınan kaynak kod NReFactory kütüphanesi içinde bulunan SyntaxTree nesnesinin içine atılarak, o kod bloğu için AST ağacı oluşturuldu. SyntaxtTree nesnesi farklı tiplere göre filtrelenerek, ağaca ait olan o özellikteki yapraklar getirildi. Kütüphane için de bu yapraklar, descendant yani torun olarak adlandırılmaktadır. Örneğin Şekil 3.7'de tip olarak MethodDeclaration seçilerek, ağaç içinde bulunan metot tanımlarının getirilmesi sağlandı. Elde edilen bu metotların listesi daha sonra kullanılmak üzere methods değişkenine atandı.

```
          CSharpParser parser = new CSharpParser();
          SyntaxTree syntaxTree = parser.Parse(sourceCode, filename);
          var methods = syntaxTree.Descendants.OfType<MethodDeclaration>();
```

Şekil 3. 7 AST Ağacından Metot Listesini Çağırma

Elde edilen bu metot listesi içinde her bir metot için tek tek metot adı, sınıf adı, içinde bulunan If/Else koşullarının sayısı, o metoda ait toplam değişken sayısı ve toplam satır sayısı gibi bilgileri içermektedir. Bu değerlerin hesaplanması Şekil 3.8'de gösterilmektedir.

```
var methods = syntaxTree.Descendants.OfType<MethodDeclaration>();
foreach (var method in methods) {
        var body = method.Body;
        var response = GetMethodDetail(body.ToString());
        var rr = method.GetChildrenByRole(Roles.Variable);
        Method mth = new Method();
        mth.MethodName = method.Name;
        mth.ClassName = filename;
        mth.NumberOFStatment = response.NumberOfIfStatment;
        mth.NumberOfVariable = response.NumberOfVariable;
        mth.MaxNestingLevel = response.MaxNestingLevel;
        mth.LOCOfMethod = method.EndLocation.Line-
        method.StartLocation.Line;
        mth.NumebrOfParameter = method.Parameters.ToList().Count();
        mth.NumberOfForStatment = response.NumberOfForStatment;
        mth.NumberOfWhileStatment = response.NumberOfWhileStatment;
        mth.NumberOfSwitchStatment = response.NumberOfSwitchStatment;
        mth.NumberOfIfStatment = response.NumberOfIfStatment;
        MethodList.Add(mth);
}
```

Şekil 3. 8 Metotlara Ait Metrik Hesaplamaları

Metotlara ait olan toplam satır sayısı, ortalama satır sayısı, toplam metot sayısı, ortalama metot sayısı ve toplam if/else sayısı, ortalama if/else sayısı bilgileri Şekil 3.9'da gösterilen kod bloğu ile "C:\Tez_Ylisans\Veri_Havuzu\MethodData.txt" dosyasında daha sonra kullanılmak üzere saklanmaktadır. Şekil 3.9'daki kod bloğun MethodData.txt dosyasına verdiği çıktı, Şekil 3.10'da gösterilmektedir.

```
private static void SaveMethodData(List<String> methodList)  {
   string path = @"C:\Tez_Ylisans\Veri_Havuzu\MethodData.txt";
   File.Create(path).Close();
   if (File.Exists(path)) {
      System.IO.File.Delete(@"C:\Tez_Ylisans\Veri_Havuzu\MethodData.txt");
      System.IO.File.WriteAllLines(@"C:\Tez_Ylisans\Veri_Havuzu\MethodData.txt"
      , methodList);
   }
}
```

Şekil 3. 9 Metrik Sonuçlarını Saklama

```
totalLine > 340860
totalMethod > 13037
averageLine > 27
```

Şekil 3. 10 Metrik Sonuçlarının Saklandığı Dosyanın İçeriği

Metotla ilgili detay bilgiler elde edildikten sonra sınıfla ilgili detay bilgileri elde etme ve saklama adımı başlamaktadır. AST ağacına atılmış kaynak kod CyclomaticCodeComplexity sınıfına SyntaxTree sınıfı tipinde bir parametre olarak gönderilerek o sınıfa ait If/Else sayısı, While sayısı, For sayısı, değişken sayısı, erişim metot sayısı(get/set), try/cath sayısı, and ve or sayısı hesaplanmaktadır. İlgili kod Şekil 3.11'da gösterilmiştir.

```
Class cls = new Class();
var ifStatements = syntaxTree.Descendants.OfType<IfElseStatement>().ToList();
var whileStatements = syntaxTree.Descendants.OfType<WhileStatement>();
var forStatements = syntaxTree.Descendants.OfType<ForStatement>();
var foreachStatements = syntaxTree.Descendants.OfType<ForeachStatement>();
var switchStatments = syntaxTree.Descendants.OfType<SwitchStatement>();
var operators = syntaxTree.Descendants.OfType<Statement>();
var variableStatment =
syntaxTree.Descendants.OfType<VariableDeclarationStatement>();
var objectStatement =
syntaxTree.Descendants.OfType<VariableDeclarationStatement>().Where(x =>
x.Type.GetType().Name != "PrimitiveType");
var fieldStatement =
syntaxTree.Descendants.OfType<FieldDeclaration>().ToList();
var accesors = syntaxTree.Descendants.OfType<Accessor>().ToList();
var tryCatchStatement =
syntaxTree.Descendants.OfType<TryCatchStatement>().ToList();
var parameterList = syntaxTree.Descendants.OfType<ParameterDeclaration>();
Int64 counter = 0;
foreach (var i in operators) {
            var k = i.Descendants.ToList();
            foreach (var item in k) {
                  Role role = item.Role;
                  String value = role.ToString();
                  if (value == "||" || value == "&&") {
                        counter = counter + 1;
                  }
            }
}
counter = counter / 2;
cls.NumberOfVariable = variableStatment.Count();
cls.NumberOfPublicVariable = fieldStatement.Where(x => x.Modifiers ==
Modifiers.Public).Count();
cls.NumberOfPublicObjectVariable = fieldStatement.Where(x => x.Modifiers ==
Modifiers.Public && x.ReturnType.GetType().Name != "PrimitiveType").Count();
cls.NumberOfAccesors = accesors.Count();
cls.NumberOfIfStatment = ifStatements.Count();
cls.NumberOfWhileStatment = whileStatements.Count();
cls.NumberOfForStatment = forStatements.Count();
cls.NumberOfSwitchStatment = switchStatments.Count();
cls.NumberOfAndOrStatment = counter;
cls.NumberOfTryCatchStatement = tryCatchStatement.Count();
cls.ProjectPath = syntaxTree.FileName;
cls.NumberOfObjectVariable = objectStatement.Count();
cls.NumberOfForeachStatment = foreachStatements.Count();
return cls;
```

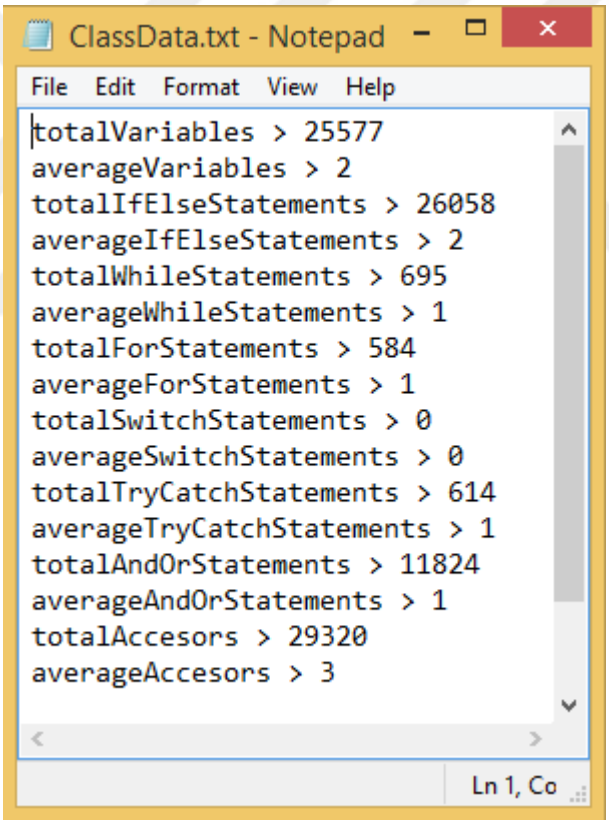
Şekil 3. 11 C# Sınıfları İçin Metrik Hesaplama

Elde edilen bu sınıf özellikleri daha sonra sınama verisinde kullanılmak üzere "C:\Tez_Ylisans\Veri_Havuzu\ClassData.txt" dosyasında saklanmaktadır. Şekil 3.12'de sınıflar için hesaplanan bu değerlerin dosyaya yazıldığı kod bloğunu göstermektedir.

```
private static void SaveClassData(List<String> classList) {
    string path = @"C:\Tez_Ylisans\Veri_Havuzu\ClassData.txt";
    File.Create(path).Close();
    if (File.Exists(path)) {
       System.IO.File.Delete(@"C:\Tez_Ylisans\Veri_Havuzu\ClassData.txt");
       System.IO.File.WriteAllLines(@"C:\Tez_Ylisans\Veri_Havuzu\ClassData.txt"
       , classList);
    }
}
```

Şekil 3. 12 Metrik Sonuçlarını Saklama

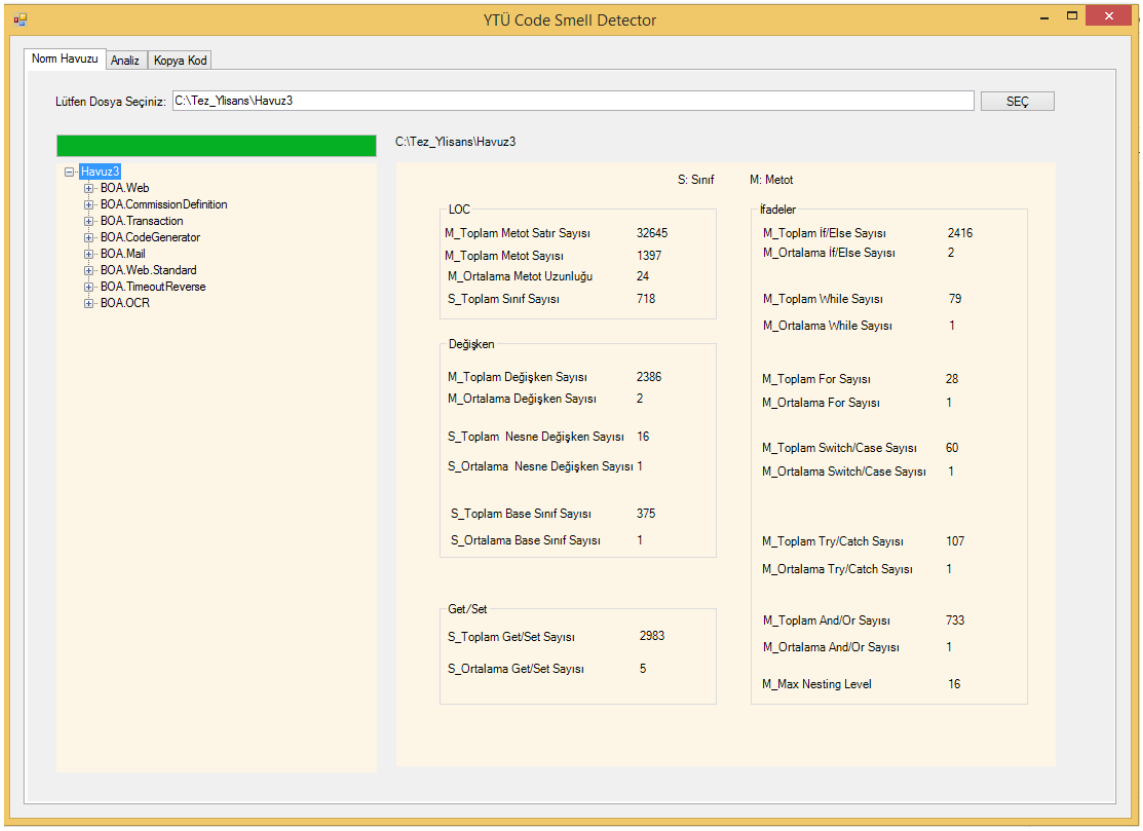
Şekil 3.12'deki kod bloğun çıktısı Şekil 3.13'deki gibidir.



Şekil 3. 13 Metrik Sonuç Dosyası

Geliştirmiş olduğumuz araçta eğitim verisi olarak CBO değeri düşük olan 8 proje ele alınmıştır. Şekil 3.14'de gösterildiği gibi, sistem ölçümlendirildiğinde 32645 kod satırı ve 1397 metot sayısı, 2386 değişken sayısı, 2416 if/else koşul sayısı, 79 while döngü sayısı,

28 for döngü sayısı, 60 switch/case sayısı, 107 try/catch sayısı, 733 and/or sayısı, 2983 erişim metodu(get/set) sayısı, 16 Max Nesting sayısı bilgileri elde edilmiştir. Bu değerler YTÜ Code Smell Detector uygulamasının Normal Havuzu ile dinamik olarak hesaplanmıştır.



Şekil 3. 14 Normal Havuzu Hesaplama

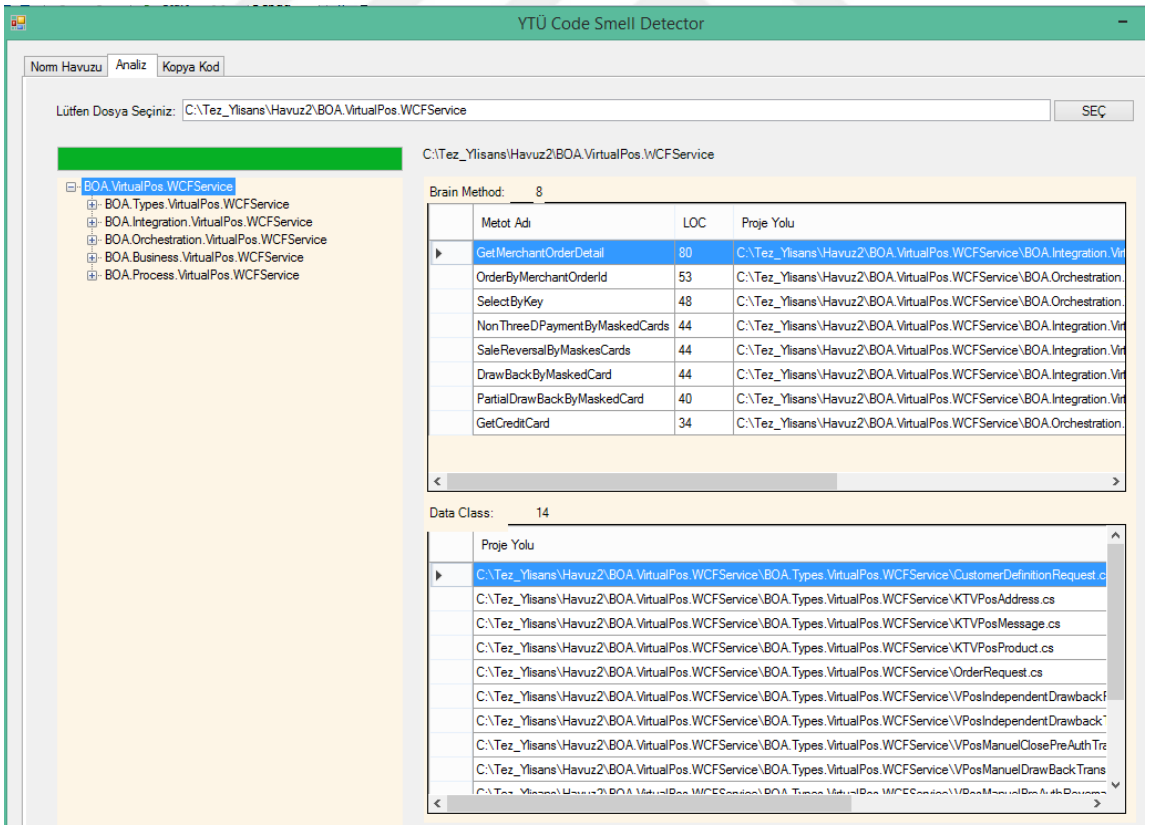
Çizelge 3. 1 Normal havuzu CBO değerleri

Proje Adı	Toplam Sınıf Sayısı	Ortalama CBO
BOA.Web	153	4
BOA.CommisionDefinition	175	4
BOA.Transaction	127	3
BOA.CodeGenerator	78	4
BOA.Mail	60	2
BOA.Web.Standard	58	2
BOA.TimeoutReverse	35	4
BOA.OCR	32	4

Normal havuzunu oluşturan projelere ait CBO bilgileri çizelge 3.1'de gösterilmektedir. Normal havuzu için CBO değeri düşük olan projelerin kod kalitesi daha yüksek olduğundan seçilmiştir. Çizelge 3.1'de CBO sayısı 2 ve 4 arasında olan 8 projenin bilgileri gösterilmiştir.

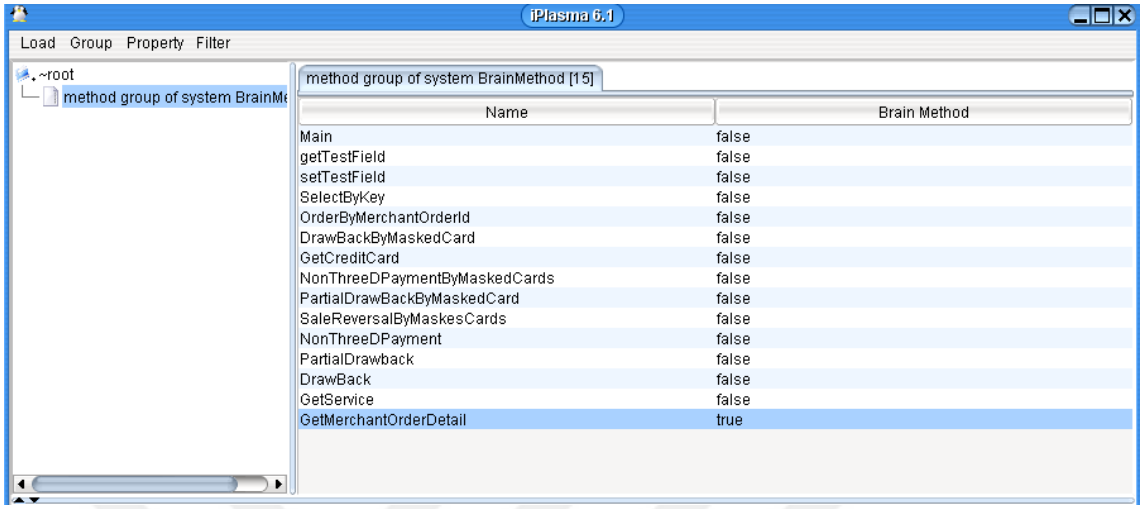
3.2.1 Brain Method Model Detayı

Eğitim verilerinin ölçümlendirilerek edilen değerlerden faydalanılarak Brain Method kod kusurunun, geliştirilen araç ile bulunması hedeflenmektedir. Aşağıdaki şekilde gösterildiği gibi Analiz sayfasında BOA.VirtualPos.WCFService projesine ait Brain Method kod kusuru içeren metotların listesi ilk tabloda gösterilmektedir. Şekil 3.15'e göre bu proje içinde şu 8 metot brain method kod kusuru içermektedir: GetMerchantOrderDetail, OrderByMerchantOrderId, SelectByKey, NonThreeD-PaymentByMaskedCards, SaleReversalByMaskesCard, DrawBackByMaskedCard, PartialDrawBackByMaskedCard, GetCreditCard.



Şekil 3. 15 Brain Method Kod Kusuru İçeren Metotlar(YTÜ Code Smell Detector)

Şekil 3.16'de BOA.VirtualPos.WCFService projesinin java versiyonu ile analiz edilen iPlasma çıktısı görüntülenmektedir. Buna göre BOA.VirtualPos.WCFService projesinde sadece GetMerchantOrderDetail adlı metot brain method kod kusurunu içermektedir.



Şekil 3. 16 Brain Method Kod Kusuru İçeren Metotlar(IPlasma)

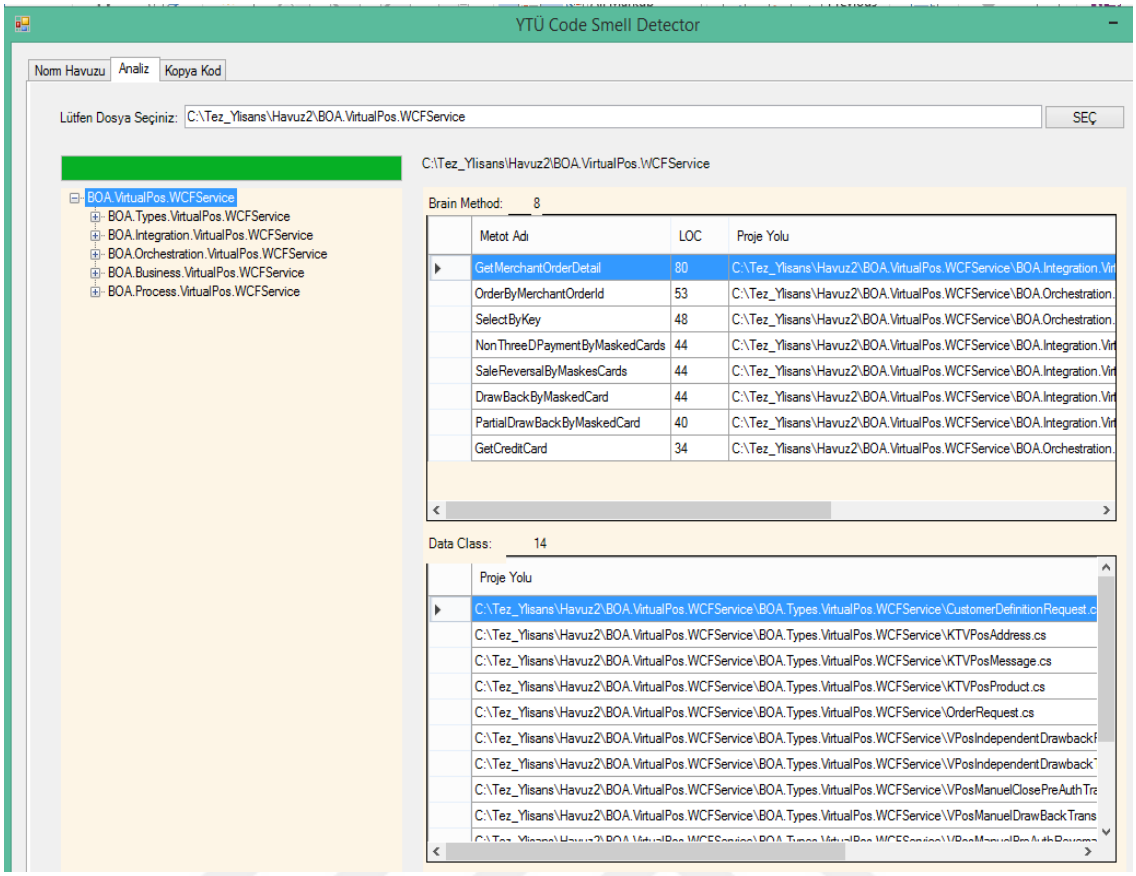
Brain method analiz aşamasında, eğitim verisinde elde ettiğimiz ölçüm sonuçlarını kullanarak metotları Brain Method filtresinden geçirerek, kod kusuru içerenlerin son kullancıya gösterilmesi sağlanmaktadır. Aşağıdaki kod bloğunda gösterildiği gibi LOC, variable ve statements değişkenlerine, eğitim verisinden elde edilen değerler atanır. Böylece Şekil 3.17'de ayrıntılandırıldığı gibi metot listesinde bulunan her bir metot için LOC metriğinin üst ve alt sınırlarının arasında kalmayan metotlar için +1 puan, CYCLCO metriğinin üst limitini aşanlar için +1 puan, MaxNestingLevel metriğinin üst limitini aşanlar için +1 puan, NOAV metriğinin üst limitini aşanlar için +1 puan verilerek, puanı 4 olanların Brain Method havuzuna düşürülmesi sağlanmıştır.

```
foreach (var item in methodResult) {
    var cyclco = CalculateCYCLCONorm(item);
    #region defectBrainMethod
    if (item.LOCOfMethod < loc.Item1 || item.LOCOfMethod > loc.Item2) {
        defectBrainMethodScore = defectBrainMethodScore + 1;
    }
    if (cyclco > cyclcoNorm.Item2) {
        defectBrainMethodScore = defectBrainMethodScore + 1;
    }
    if (item.MaxNestingLevel > maxNesting.Item2) {
        defectBrainMethodScore = defectBrainMethodScore + 1;
    }
    if (item.NumberOfVariable > noav.Item2) {
        defectBrainMethodScore = defectBrainMethodScore + 1;
    }
    if (defectBrainMethodScore == 4) {
        DataRow row = table.NewRow();
        row["Proje Yolu"] = item.ClassName;
        row["LOC"] = item.LOCOfMethod;
        row["Metot Adı"] = item.MethodName;
        row["Değişken Sayısı"] = item.NumberOfVariable;
        row["Koşul Sayısı"] = item.NumberOFStatment;
        table.Rows.Add(row);
        numberOfBrainMethod++;
    }
    defectBrainMethodScore = 0;
    #endregion
}
```

Şekil 3. 17 Brain Method Listesi Oluşturma

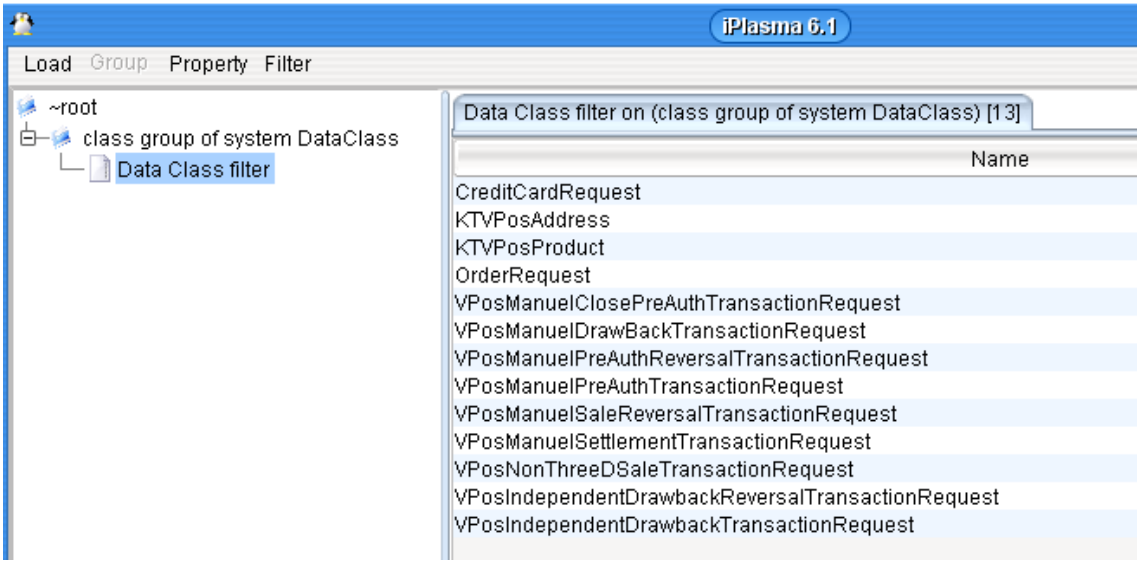
3.2.2 Data Class Model Detayı

Eğitim verisinden elde edilen değerler doğrultusunda limitler belirlendikten sonra, geliştirilen araç ile Data Class kod kusuru içeren sınıfların bulunması hedeflenmektedir. BOA.VirtualPos.WCFService sınama verisine ait Data Class kod kusuru içeren sınıflar Şekil 3.18'de gösterilmektedir. Uygulamanın sağ alt köşesinde gösterilen listede bu projeye ait Data Class kod kusurunu içeren sınıflar listelenmektedir. Bu proje için toplamda 14 Data Class kod kusuru içeren sınıf bulunmaktadır.



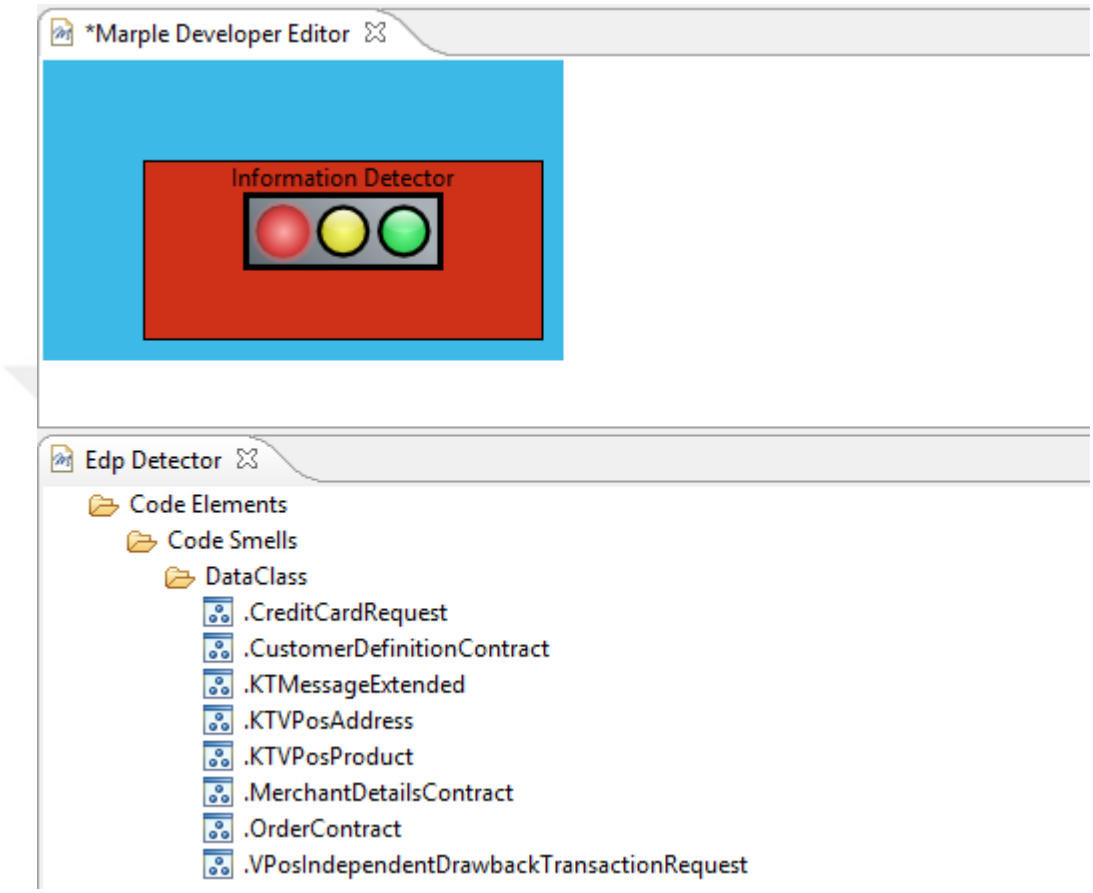
Şekil 3. 18 Data Classs Kod Kusuru İçeren Sınıflar(YTÜ Code Smell Detector)

Şekil 3.19'da BOA.VirtualPos.WCFService projesi için IPlasma aracında Data Class kod kusuru hesaplanmıştır. IPlasma aracına göre bu projede toplam 13 adet Data Class kod kusuru içeren sınıf bulunmaktadır. Şekil 3.19'da Data Class kriterine göre sınıflar filtrelenmiştir.



Şekil 3. 19 Data Classs Kod Kusuru İçeren Sınıflar(IPlasma)

Şekil 3.20'de BOA.VirtualPos.WCFService projesi için Essere eklentisinde Data Class kod kusuru hesaplanmıştır. Essere eklentisine göre bu projede toplam 8 adet Data Class kod kusuru içeren sınıf bulunmaktadır. Edp Detector alanında bulunan, DataClass klasörü altındaki sınıflar, Data Class kod kusurunu içermektedirler.



Şekil 3. 20 Data Classs Kod Kusuru İçeren Sınıflar(Essere)

Şekil 3.21'de ayrıntılandırıldığı gibi NOAP+NOAM metriğinden dönen sonuç alt limitin üstünde kalıyorsa ve WMC metriğinden dönen sonuç alt limitin altında kalıyorsa veya NOAP+NOAM metriğinden dönen sonuç üst limiti aşıyorsa ve WMC metriğinden dönen sonuç üst limitin altında kalıyorsa bu değerin %60'ı alınır ve CBO değerinin %40'ı ile toplanır. Eğer bu değerlerin toplamı 5'i aşıyorsa bu sınıf Data Class kod kusurunu içeriyor demektir.

```
foreach (var item in ClassResult) {
    #region defectDataClass
    var condition = item.NumberOfAndOrStatment + item.NumberOfForStatment +
    item.NumberOfIfStatment + item.NumberOfWhileStatment;
    var switchCase = item.NumberOfSwitchStatment;
    var tryCatch = item.NumberOfTryCatchStatement;
    var wmcNorm = 1 + condition + switchCase + tryCatch;
    if (item.NumberOfPublicVariable + item.NumberOfAccesors > noap.Item1
            && wmcNorm < wmc.Item1) {
        defectDataClassScore = 100;
    }
    if (item.NumberOfPublicVariable + item.NumberOfAccesors > noap.Item2
            && wmcNorm < wmc.Item2) {
        defectDataClassScore = 100;
    }

    #region cbo
    Int64 averageCbo = 0;
    var aveargePublicObject = item.NumberOfPublicObjectVariable;
    var averageBaseClass = item.NumberOfBaseType;
    var publicData = aveargePublicObject + averageBaseClass;
    if (publicData > cbo.Item2) {
        averageCbo = publicData / cbo.Item2;
        averageCbo = averageCbo * 100;
    }
    #endregion

    defectDataClassScore = (defectDataClassScore * 60 / 100)
        + (averageCbo * 40 / 100);
    if (defectDataClassScore > 5) {
        DataRow row = table.NewRow();
        row["Proje Yolu"] = item.ProjectPath;
        row["Değişken Sayısı"] = item.NumberOfPublicVariable +
            item.NumberOfAccesors;
        row["Koşul Sayısı"] = wmcNorm;
        table.Rows.Add(row);
        numberOfDataClass++;
    }
}
```

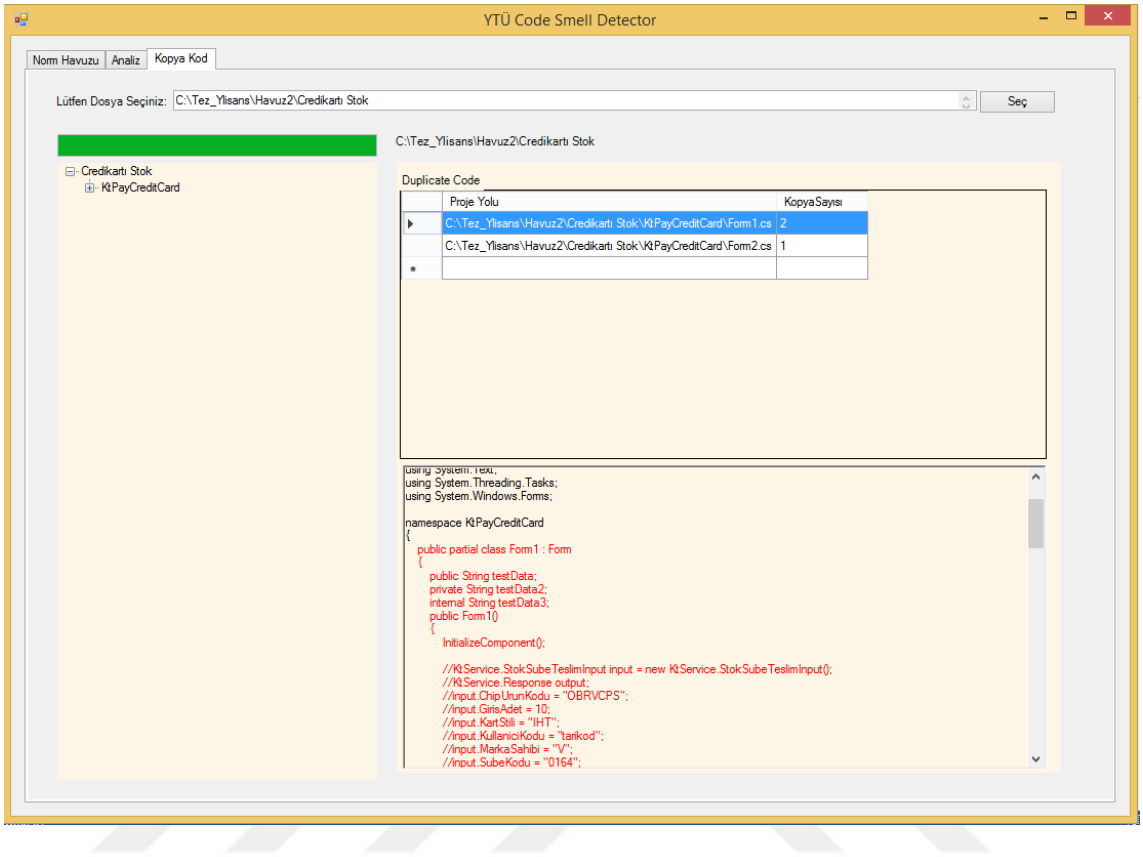
Şekil 3. 21 Data Classs Kod Kusuru Listesi Oluşturma

3.2.3 Duplicate Code Model Detayı

Geliştirilen araçta DuplicateFinder adlı açık kaynak kodlu uygulamadan faydalanılarak, sınama verisi içinde bulunan kopyalar bulundu. DuplicateFinder dosyalar arası karşılaştırma yaparak kesişimi bulunan kodları bulur. Satır satır karşılaştırma yapılır ve benzerliği 5 satır veya daha fazla olan kod parçacıkları kopya kod olarak adlandırılır.

Şekil 3.22'de YTÜ Code Smell Detector uygulaması kullanılarak KTPayCreditCard uygulamasında duplicate kod kusuru bulunan sınıflar gösterilmiştir. Sağ üst alanda bu projeye ait kod kusuru bulunan sınıflar listelenirken, sağ alt alanda ise kopyası bulunan

satırlar gösterilmektedir. Kırmızı renk ile gösterilen satırlar kopya kod anlamına gelmektedir.



Şekil 3. 22 Duplicate Code

Şekil 3.23'de duplicate code kod kusuru aranırken, istisnai durumlar gösterilmiştir. Using ile başlayan satırlar, referans edilen kütüphaneleri gösterdiğinden duplicate kod kusuru aranırken bu satırlar atlanmıştır.

Namespace ile başlayan satırlar, sınıfların gruplanarak ayrıştırılması için kullanıldığından, bu satırlar duplicate kod kusuru aranırken atlanmıştır.

ServiceReferences, Proporties, obj, bin klasörleri altında bulunan sınıflar proje taslakları ile geldiğinden kod kusuru ararken bu klasör içinde bulunan kodlar, arama işlemine dahil edilmemiştir.

.cs dosyalı uzantılı tüm dosyalar birbiri ile kıyaslanarak, minimum 5 satır sayısı kadar benzerliği olan sınıflar duplicate code kod kusuru içeren sınıf listesine dâhil olmuştur.

```
var selectedFiles=subDirs.ToList().Where(d=>!d.Contains("Service References"));
selectedFiles = selectedFiles.Where(d => !d.Contains("Properties"));
selectedFiles = selectedFiles.Where(d => !d.Contains("obj"));
selectedFiles = selectedFiles.Where(d => !d.Contains("bin"));
foreach (var currentDir in selectedFiles) {
     FileDuplicateFinderTester tester = new   FileDuplicateFinderTester();
     FileDuplicateFinder process = tester.Process;
     process.AddIgnoredLinePrefix("using");
     process.AddIgnoredLinePrefix("namespace");
     process.ReadFile(sourceDirectory, "*.cs", true);
     var sourceFile = System.IO.Path.GetFileName(currentDir);
     int processedCount = process.FindDuplicates(sourceFile);
     List<DuplicateInfo> infoList = new List<DuplicateInfo>();
     if (tester.DuplicateCount > 0) {
         List<DuplicateInfo> resultList = Calculate(process, currentDir);
         duplicateInfoList.Add(currentDir, resultList);
     }
 }
```

Şekil 3. 23 Duplicate Code Bloğu

BÖLÜM 4

BULGULAR

Kod kusuru tespitleri ortak alanda çalışma yapmış olan uygulamalarla(YTU CSD, Iplasma, Essere) mukayese edilerek sonuç çıktısı oluşturulmuştur. Uzman görüşlerin çoğunluk oylaması sonucuna göre çıktıların doğruluk oranları hesaplanmıştır.

4.1 Diğer Modeller ile Karşılaştırma

YTÜ Code Smell Detector uygulaması ile kod kusurlarını bulmak için limitlerin dinamik bir şekilde oluşturulması sağlanarak, kod kusurularının bulunmasının otomatize edilmesi hedeflenmiştir.

Çizelge 4.1'de YTÜ Code Smell Detector uygulaması ile önceki çalışmalar karşılaştırılmıştır. Çizelge 4.2'de ise üzerinde Data Class analizi yapılan 14 sınıfın adı gösterilmektedir. Bu sınıfların Data Class içerip içermedikleri alanında uzman 3 yazılımcının çoğunluk oylaması sonucunda alınan karara göre belirlenmiştir. Elde edilen değerler Çizelge 4.2'de Data Class kolonu altında gösterilmektedir. Bu değerlere karşılık, YTU-CSD, Iplasma ve Essre uygulamalarının bu sınıflar için verdikleri kod kusuru değerlerinin çıktıları, ayrı sütunlar altında detaylandırılmıştır.

Çizelge 4. 1 Önceki çalışmalar ile karşılaştırması

Özellikler	YTÜ-CSD	IPlasma	Travasos	Munro	Alikacem ve Sahraoui	Gue´he´neuc, Meur ve Moha	Essere
Dinamik alt ve üst limit	Var	Yok	Yok	Yok	Yok	Yok	Yok
Bulduğu Kod Kusur Sayısı	3	9	5	2	N/A	4	10
C# desteği	Var	Yok	Yok	Yok	Yok	Yok	Yok
Büyük Uygulamalarda Kullanıma:	Uygun	Uygun	Uygun değil	Uygun	Uygun	Uygun	Uygun

Çizelge 4. 2 DataClass kod kusuru içeren sınıflar

Sınıf Adı	Data Class	YTU-CSD	IPlasma	Essere
CustomerDefinitionRequest	Evet	Evet	Hayır	Hayır
KTVPosAddress	Evet	Evet	Evet	Hayır
KTVPosMessage	Evet	Evet	Hayır	Evet
KTVPosProduct	Evet	Evet	Evet	Evet
OrderRequest	Evet	Evet	Evet	Hayır
CreditCardRequest	Hayır	Hayır	Evet	Evet
CustomerDefinitionContract	Evet	Hayır	Hayır	Evet
OrderContract	Evet	Hayır	Hayır	Evet
VPosIndependentDrawbackReversalTransactionRequest	Evet	Evet	Evet	Hayır
VPosIndependentDrawbackTransactionRequest	Evet	Evet	Evet	Evet
VPosManuelClosePreAuthTransactionRequest	Evet	Evet	Evet	Hayır
VPosManuelSettlementTransactionRequest	Evet	Evet	Evet	Hayır
VPosManuelSaleReversalTransactionRequest	Evet	Evet	Evet	Hayır
VPosNonThreeDSaleTransactionRequest	Evet	Evet	Evet	Hayır
VPosManuelDrawBackTransactionRequest	Evet	Evet	Evet	Hayır
VPosManuelPreAuthReversalTransactionRequest	Evet	Evet	Evet	Hayır
VPosManuelPreAuthTransactionRequest	Evet	Evet	Evet	Hayır
MerchantDetailsContract	Evet	Hayır	Hayır	Evet

Çizelge 4.3'de elde edilen sonuçlara göre, YTU Code Smell Detector aracı Data Class kod kusuru bulmak için toplamda 18 sınıf üzerinde test edildiğinde, 15 doğru, 3 yanlış sonucu elde edilmiştir.

Çizelge 4. 3 YTU-CSD DataClass kod kusuru bulma doğruluk oranları

	Pozitif	Negatif
Doğru	14	1
Yanlış	0	3

Çizelge 4.4'de elde edilen sonuçlara göre, IPlasma aracı Data Class kod kusuru bulmak için toplamda 18 sınıf üzerinde test edildiğinde, 12 doğru, 6 yanlış sonucu elde edilmiştir.

Çizelge 4. 4 IPlasma DataClass kod kusuru bulma doğruluk oranları

	Pozitif	Negatif
Doğru	12	0
Yanlış	1	5

Çizelge 4.5'de elde edilen sonuçlara göre, Essere aracı Data Class kod kusuru bulmak için toplamda 18 sınıf üzerinde test edildiğinde, 6 doğru, 12 yanlış sonucu elde edilmiştir.

Çizelge 4. 5 Essere DataClass kod kusuru bulma doğruluk oranları

	Pozitif	Negatif
Doğru	6	0
Yanlış	1	11

Çizelge 4.6'da üzerinde Brain Method analizi yapılan 11 metot adı gösterilmektedir. Bu metotların Brain Method içerip içermedikleri alanında uzman 3 yazılımcının çoğunluk oylaması sonucunda alınan karara göre belirlenmiştir. Elde edilen değerler Çizelge 4.6'da Brain Method kolonu altında gösterilmektedir. Bu değerlere karşılık, YTU-CSD ve Iplasma uygulamalarının bu metotlar için verdikleri çıktılar, ayrı sütunlar altında detaylandırılmıştır.

Çizelge 4. 6 BrainMethod kod kusuru içeren metotlar

Metot Adı	Brain Method	YTU-CSD	IPlasma
GetMerchantOrderDetail	Evet	Evet	Evet
OrderByMerchantOrderId	Evet	Evet	Hayır
SelectByKey	Evet	Evet	Hayır
NonThreeDPaymentByMaskedCards	Hayır	Evet	Hayır
SaleReversalByMaskesCard	Hayır	Evet	Hayır
DrawBackByMaskedCard	Hayır	Evet	Hayır
PartialDrawBackByMaskedCard	Hayır	Evet	Hayır
GetCreditCard	Evet	Evet	Hayır
NonThreeDPayment	Hayır	Hayır	Hayır
PartialDrawback	Hayır	Hayır	Hayır
DrawBack	Hayır	Hayır	Hayır

Çizelge 4.7'de elde edilen sonuçlara göre, YTU Code Smell Detector aracı Brain Method kod kusuru bulmak için toplamda 11 metot üzerinde test edildiğinde, 7 doğru, 4 yanlış sonucu elde edilmiştir.

Çizelge 4. 7 YTU-CSD BrainMethod kod kusuru bulma doğruluk oranları

	Pozitif	Negatif
Doğru	4	3
Yanlış	4	0

Çizelge 4.8'de elde edilen sonuçlara göre, IPlasma aracı Brain Method kod kusuru bulmak için toplamda 11 metot üzerinde test edilerek, 7 doğru, 4 yanlış sonucu elde edilmiştir.

Çizelge 4. 8 IPlasma BrainMethod kod kusuru bulma doğruluk oranları

	Pozitif	Negatif
Doğru	1	7
Yanlış	0	3

YTÜ Code Smell Detector ile Iplasma uygulaması ve Essere eklentisi ile ortak olan fonksiyonlar karşılaştırılmıştır. Aynı veriler üzerinde test edildiklerinde Data Class kod kusuru için Essere eklentisi 18 sınıf içinden 6'sını kusurlu olarak sınıflandırırken, YTÜ-CSD 18 sınıf içinden 15'ini ve IPlasma aracı ise 18 sınıf içinden 11'ini kusurlu olarak sınıflandırmışlardır. Brain Method kod kusuru Essere eklentisi tarafından desteklenmediğinden, YTÜ-CSD ve Iplasma araçları üzerinde denenmiştir. Sonuçlara göre YTÜ CSD aracı 11 metot içinden 7 Brain Method kod kusuru bulurken, IPlasma aracı 11 metot içinden 8 Brain Method kod kusuru bulmuştur. Çizelge 4.9'da karşılaştırma sonuçları gösterilmektedir.

Çizelge 4. 9 (YTÜ-CSD/IPlasma/Essere) Doğruluk oranlarının karşılaştırılması

Kod Kusuru Adı	YTÜ-CSD Sınıf/Metod Sayısı	IPlasma Sınıf/Metod Sayısı	Essere Sınıf/Metod Sayısı
Data Class Kod Kusuru	15/18 = %83,3	12/18 = %66,7	6/18 = %33,3
Brain Method Kod Kusuru	7/11 = %63,6	8/11 = %72,7	Desteklemiyor

BÖLÜM 5

SONUÇ VE ÖNERİLER

YTÜ Code Smell Detector uygulaması ile kod kusurlarını bulmak için limitlerin dinamik bir şekilde oluşturulması sağlanarak, kod kusurularının bulunmasının otomatize edilmesi hedeflenmiştir. Yapılan gerçeklemenin avantajları aşağıdaki gibidir:

- Kod kusurları için alt ve üst limitlerin belirlenmesi otomatize edilmiştir. Böylece firmalar kendi alt ve üst limitlerini, kalitesine güvendikleri uygulamalar ile oluşturabileceklerdir.
- Yazılım metriklerine ait alt ve üst limitler txt dosyalarında saklanmaktadır, bu sayede analiz sırasında limitler istenildiği zaman kullanılabilecektir.
- Kopya kod kusur bulunması hedeflenen C# uygulamasında, duplicate code kusuru bulunan sınıf listesi ve hangi satırlarda kopya kod bulunduğu gösterilmektedir.
- Data Class kod kusuru hesaplanırken, diğer çalışmalardan farklı olarak sınıfların CBO değerleri de hesaplama kriterine dahil edilmiştir.

Sonuç olarak Data Class, Brain Method ve Duplicate Code kusurlarını otomatik olarak tespit edebilen bir gereç elde edilmiştir. Sürecin otomatik işlemesine rağmen Bölüm 4'te incelendiği üzere yeterli doğruluk oranlarına erişilebilmiştir.

YTÜ Code Smell Detecor, yeni özellikler eklenerek ve var olan özellikler geliştirilerek yazılımcılar için daha kullanışlı bir şekilde sunulabilir. Aşağıda geliştirilebilecek ya da eklenebilecek özellikler listelenmiştir:

- Yeni kod kusurlarını belirleyecek şekilde genişletilebilir.
- Eklenti olarak sunulabilir.

- Var olan kod kusurlarına karşılık gelen iyileştirme metotları önerilebilir.
- Yazılım projelerine ait geçmiş bilgiler tutularak, kod kalitesini düşüren veya arttıran takım çalışanları için grafik sunulabilir.

KAYNAKLAR

[1] O'Keeffe, M., ve O´Cinneide, M., (2008). "Search-based refactoring for software maintenance", Journal of Systems and Software, 81:502-516.

[2] Laplante, Philip A., (2007). What every engineer should know about software engineering, CRC Press, 2007, Florida.

[3] Fowler, M., (2011). Refactoring Improving the Design of Existing Code, 26, Longman, Boston.

[4] AntiPattern, http://martinfowler.com/bliki/AntiPattern.html, 25 Haziran 2016.

[5] Gamma, E., vd., (1994). Design Patterns – Elements of Reusable Object-Oriented Software, Longman, Boston.

[6] Rising, L., (1998). The patterns handbook: techniques, strategies, and applications, Cambridge, New York.

[7] Using NRefactory for Analyzing Csharp Code, http://www.codeproject.com/Articles/408663/Using-NRefactory-for-analyzing-Csharp-code, 18 Ekim 2015.

[8] Lanza, M. ve Marinescu ,R., (2010). Object-Oriented Metrics In Practice, 1, Springer, Berlin.

[9] Köşker, Y., (2009). Prediction of Code Refactoring Using Class and File Level Software Metrics, Yüksek Lisans Tezi, Boğaziçi Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

[10] Astekin, M., (2012). Simülasyon Yazılımlarında Kod Klonları, Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

[11] Kapdan, M., (2014). Nesneye Dayalı Programlama Tabanlı Yazılımlarda Yazılım Metrikleri Kullanarak Yapısal Kod Klon Tespiti, Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul.

[12] Software Evolotion and Reverse Engineering, http://essere.disco.unimib.it/reverse/CodeSmellDetector.html, 21 Haziran 2016.

[13] Open Source Software, Code Bad Smell Detector, https://sourceforge.net/projects/cbsdetector/, 14 Ocak 2014.

[14] Travassos, G., Shull, F., Fredericks, M., ve Basili, V.R., (1999). "Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality", Proc. 14th Conf. Object-Oriented Programming, Systems, Languages, and Applications, 01 October 1999, New York.

[15] Marinescu, R., (2004). "Detection Strategies: Metrics-Based Rules for Detecting Design Flaws," Proc. 20th Int'l Conf. Software Maintenance, 11 September 2004, Romania.

[16] Munro, M.J., (2005). "Product Metrics for Automatic Identification of "Bad Smell" Design Problems in Java Source-Code", Proc. 11th Int'l Software Metrics Symposium, F. Lanubile and C. Seaman, eds., 19 September 2005, University of Strathclyde Glasgow, UK.

[17] Alikacem, E.H. ve Sahraoui, H., (2006). "Generic Metric Extraction Framework," Proc. 16th Int'l Workshop Software Measurement and Metrik Kongress, 21 September 2009, Postdam.

[18] Moha, N., Gueheneuc, Y., Duchien, L., ve Le Meur, A., (2010). "DECOR: A Method for the Specificationand Detection of Code and Design Smells", Triskell Team, University of Rennes 1, Rennes, France.

[18] .NET Compiler Platform ("Roslyn"), https://roslyn.codeplex.com/, 23 Nisan 2016.

[19] Iplasma, http://loose.upt.ro/iplasma/, 1 Nisan 2016.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı :Sevilay TANIŞ

Doğum Tarihi ve Yeri :13.02.1988 Mardin

Yabancı Dili :İngilizce

E-posta :sevilaytanis@gmail.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Yüksek Lisans	Bilgisayar Mühendisliği	Yıldız Teknik Üniversitesi	2016
Lisans	Bilgisayar Mühendisliği	Fatih Üniversitesi	2011
Lise	Fen	Milli Piyango Lisesi	2005

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2012	Kuveytturk Katılım Bankası	Yazılım Mühendisi
2011	Android Pazarı	Yazılım Mühendisi