T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM ÖLÇÜTLERİ ÜZERİNDEN BEYİN METOT VE BEYİN SINIF KOD KUSURLARININ TESPİTİ

DENİZ NACAR

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI BİLGİSAYAR MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

DANIŞMAN
DR. ÖĞRETİM ÜYESİ YUNUS EMRE SELÇUK

İSTANBUL, 2019

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM ÖLÇÜTLERİ ÜZERİNDEN BEYİN METOT VE BEYİN SINIF KOD KUSURLARININ TESPİTİ

DENİZ NACAR tarafından hazırlanan tez çalışması 11/07/2019 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Dr. Öğretim Üyesi Yunus Emre Selçuk

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Dr. Öğretim Üyesi Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi _____

Prof.Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Prof.Dr. Selim AKYOKUŞ

Medipol Üniversitesi _____

ÖNSÖZ

Bu çalışmada yazılım kalitesi yüksek olan uygulamalardan kod kusurunu belirleyen alt ve üst limitler otomatik olarak elde edilerek ya da yapılan çalışmalar sonucu kod kusurunu belirleyen alt ve üst limitler elde edilerek, yazılım projelerinin kaynak kodları üzerinde beyin sınıf kod kusurunun bulunması hedeflenmektedir. Dr. Yunus Emre Selçuk'un manuel olarak ya da dinamik alt ve üst limit oluşturarak kod kusurlarının 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 Dr. Yunus Emre Selçuk'a, tez yazım sürecinde desteklerini esirgemeyen aileme teşekkürü borç bilirim.

Haziran, 2019

Deniz NACAR

İÇİNDEKİLER

Sayfa

KISALTMA LİSTESİ

AST	Abstract Syntax Tree
CBO	Coupling Between Object Classes
CYCLCO	Cyclomatic Complexity
LOC	Line of Code
MAXNESTING	Maximum Nesting Level
NOAM	Number of Accessor Methods
NOAV	Number of Accessed Variables
NOM	Number of Method
NON	Number of Namespace
NOPA	Number of Public Attributes
ORT	Ortalama
SPM	Standart Sapma Değeri
VS	Visual Studio
WMC	Weighted Method Per Class

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

ÖZET

YAZILIM ÖLÇÜTLERİ ÜZERİNDEN BEYİN METOT VE BEYİN SINIF KOD KUSURLARININ TESPİTİ

Deniz NACAR

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Dr. Yunus Emre SELÇUK

Günümüzde yazılım maliyetleri gittikçe artmakta ve şirketler için önemli bir sorun haline gelmektedir. Yazılımların boyutlarının büyümesi ve sınıfların karmaşıklığının artması bakım masraflarının ve geliştirme zamanının artmasına sebep olmuştur. Yazılım projeleri birden fazla kişinin çalıştığı ekipler tarafından geliştirildiği için geliştirme aşamasında doğru tasarım ve kodlama yapılması projenin daha anlaşılabilir ve daha kolay bakım yapılabilir olmasını sağlar. Dolayısıyla doğru tasarım ve kodlama anlaşılabilirliği arttırır, karmaşıklığı ve bakım maliyetlerini azaltır.

Bu makale derleme zamanında yazılım ölçütleri aracılığıyla Beyin Metot ve Beyin Sınıf kod kusurlarının nasıl tespit edildiğini göstermeyi amaçlamaktadır. Yazılım projelerinde kusurlu kodları yakalayabilmek için eşik değerlerin belirlenmesi gerekmektir. Bu değerler başka çalışmalar sonucunda elde edilen değerler olabilir ya da eşik değerler dinamik olarak belirlenen değerler olabilir. Yani mutlak alt limit ya da mutlak üst limit diye sınır yoktur. Değinilen ihtiyacı karşılamak için çalışmamızda, manuel olarak kendi limitlerini sisteme girebilmesine ve kurumların kendi limitlerini oluşturmalarına olanak sağlanmıştır.

Anahtar Kelimeler: Beyin Metot Tespiti, Beyin Sınıf Tespiti, Kod Kusurları, Kod Hataları

ABSTRACT

DETECTION OF BRAIN METHOD AND BRAIN CLASS CODE SMELLS THROUGH SOFTWARE METRICS

Deniz NACAR

Department of Computer Engineering

MSc. Thesis

Adviser: Dr. Yunus Emre Selçuk

Nowadays, software costs are increasing and becoming an important problem for companies. Growth of software size has also led to an increase in maintenance costs and development time. As the software projects are developed by teams of more than one person, the right design and coding in the development phase makes the project more understandable and easier to maintain. Thereby correct design and coding also increase understandability and reduce complexity and maintenance costs. Hence, it is necessary to detect and refactor poorly designed or improved classes in the software projects.

This article aims to show how to detect the Brain Method and Brain Class code smell with the help of some compile-time software metrics. 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 way thresholds can be detemine manually on other works. 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 or manually.

Keywords: Brain Method Detection, Brain Class Detection, Code Smells, Code Defects

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

Kod kusuru yazılımlar için büyük sorunlara yol açtığından üzerinde çok fazla çalışma yapılan ve çözüm üretilen bir konudur. Literatürde var olan kod kusuru bulma teknikleri ve kod kusuru bulmak için geliştirilen araçlarla ilgili yapılan çalışmalar aşağıdaki bölümlerde detaylandırılmıştır.

Yazılım projesi içinde yazılım geliştiricilerin kod kusurlarını manuel olarak tespit etmeleri ciddi maliyet oluşturmakta ve proje boyutu arttıkça zorlaşmaktadır. Bu nedenlerden dolayı kod kusurlarını daha hızlı tespit ederek tespit maliyetini düşürmek için kod kusuru tespit araçları kullanılır. Daha önce geliştirilmiş projelerin bakımlarında görev alan yazılımcılar işlerin yeni yazılımcılara devredileceğini düşünerek tasarım ve kodlama yapmalıdır. Bu şekilde anlaşılabilir projeler yaratılarak projelerin bakım ve geliştirme maliyetleri azaltılabilir.

Bu çalışmada özellikle birden fazla yazılımcının olduğu yazılım ekiplerinin geliştirmekte olduğu projelerde kod kusurunun oluşmasına sebebiyet veren kodların, geliştirilecek olan araç ile bulunması amaçlanmaktadır. Çalışmada yazılım kalitesini düşüren, kod kusuruna neden olan faktörler önceden belirlenecek ve ölçülendirilen bu faktörlerin kaynak kodda aranması veya ölçülmesi yolu ile kod kusurlarının bulunabilmesi sağlanacaktır. Geliştirilecek BB Code Smell Detector (BB-CSD) isimli araç ile Beyin Metot (Brain Method) ve Beyin Sınıf (Brain Class) tespiti yapılarak kod kusurlarının azaltılması amaçlanmaktır.

Özetle bu çalışmanın amacı proje yaşam süresince gittikçe daha büyük sorunlara yol açabilecek Beyin Metot ve Beyin Sınıf kod kusurlarını tespit eden bir araç geliştirmektir. Bu araç ile geliştirilecek yazılımların bakım ve geliştirme maliyetleri azaltılacak, yazılımcıların projelere daha hızlı adapte olması sağlanacaktır.

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

Bazı yazarlar kötü tasarım ve kodlamanın azaltılması için çeşitli kod kusuru tanımlamaları ile literatüre katkıda bulunmuşlardır. Fowler [1] kod kusurunu zayıf tasarım ve kodlamanın tekrar eden izleri olarak tanımlar. Eğer kod içinde tekrar tasarlanmayı ya da geliştirilmeyi gerektiren yerler varsa orada kod kusurunun varlığından bahsedilebilir. Fowler [1] ayrıca kod kusurlarının yazılım projelerinde gün geçtikçe büyüyerek daha önemli sorunlara yol açacak göstergeler olduklarını belirtmiştir. Mens ve Tourwe´ [2], çalışmalarında yeniden düzenleme (ing. Refactoring) ve yeniden yapılandırma (ing. Restructuring) kavramlarına açıklık getirmiş; çalışan bir kod yapısı için gerekli olan yeniden düzenleme yöntemlerini uygulamış ve tasarımı daha anlaşılır, kolay müdahale edilebilir, nesneye dayalı tasarım ilkelerine uygun hale getirmiştir.

Kod kusuru tespiti yapılan kaynak kodlarda yer alan kusurlar; geliştirilen uygulamanın doğru çalışmasını engellemezler, bu nedenle beyaz kutu (white box) ve siyah kutu (black box) testleri yapılarak tespit edilemezler. Kod kusurları fonksiyonel hatalar değillerdir fakat kodun kalitesi üzerinde çok etkilidirler; çünkü bu kod kusurları kodun ne zaman hangi bölümünün yeniden düzenlenmesi gerektiğinin cevabını verirler. Yeniden düzenleme işlemi sistemin davranışını değiştirmeden kodu daha okunabilir, daha anlaşılabilir yapan ve kodlama hatasını en aza indiren bir değiştirme tekniğidir. Yeniden düzenleme işlemi çeşitleri: ortak olarak yapılan işlemler yeni metotlar içinde toplanır ve bu metotlar ilgili metotlardan çağrılır, sınıfların ortak metotları base sınıf içine taşınır, metotlar tek sorumluluk prensibine (Single Responsibility Principle) göre tek görev üstlenecek şekilde yeni metot çıkarımı yapılır. Sıralanan bu değişimlerin birleşimi ile kod tasarımında iyileştirmeler yapılarak kod kalitesi artırılır ve ilerde oluşabilecek bulgular engellenmiş olur [1].

1.2 Tezin Amacı

Çalışmada kod kalitesini düşüren faktörlerin analizi yapılarak, bu faktörlerin kod içerisinde tespit edilebilmesini sağlayacak ölçütler tanımlanacaktır. Tanımlanan bu ölçütlerden faydalanarak kod kalitesi üzerinde incelemeler yapılacaktır. Kod kalitesini ölçmek için çalışmaya dâhil edilmesi planlanan kod kusurları şunlardır:

- Beyin Metot
- Beyin Sınıf

Yazılım kalitesini düşüren bu kod kusurlarının giderilmesi için iyileştirme metotları sunulacaktır. Bu şekilde tespit edilmiş olan kod kusurlarının giderilmesi için bu iyileştirme metotları önerilecektir. Bulunan kod kusurlarının giderilmesi için önerilecek metotlar şunlardır:

- Metot Çıkarma
- Sınıf Çıkarma

Kod kusurları tanımlandıktan sonra bu sorunların giderilmesi için iyileştirme metotları ile eşleştirilecektir. Geliştirilen yazılım, belirlenen ölçütleri kullanarak Beyin Metot ve Beyin Sınıf kod kusurlarının tespitini yapmak için C# dilinde bir Windows Form uygulaması olarak tasarlanacaktır.

1.3 Hipotez

Bu bölümde Beyin Metot ve Beyin Sınıf kod kusurlarını bulmak için önerilen ölçütler ele alınmaktadır. Beyin Metot ve Beyin Sınıf kod kusurlarını bulmakta kullanılacak eşik değerler uygulamada iki türlü oluşturulmaktadır. Geliştirilen uygulama bu eşik değerlerin dinamik olarak oluşturulmasını ya da analizler sonucunda elde edilen eşik değerlerin manuel olarak uygulamaya girişini sağlamaktadır. Eşik değerlerin optimal değeri diye bir şey yoktur, bundan dolayı dinamik olarak oluşturulan eşik değerler yapmış olduğumuz çalışmada kalitesine güvenilen C# projeleri üzerinden ölçütlerin değerleri toplanarak oluşturulmuştur. Bu eşik değerlere ek olarak kullanıcıların başka çalışmalarda hesaplanmış eşik değerleri sisteme manuel olarak tanımlamalarına olanak

sağlanmıştır. Eşik değerlerin hesaplanabilmesi için kullanılan projelerin kaynak kodlarının oluşturduğu havuza “Eşik Değer Havuzu” adı verilmiştir.

Dinamik olarak oluşturulan eşik değerlerin bulunmasında nesneye yönelik programlama prensiplerinin kullanıldığı ve yazılım kalitesinin yüksek olduğu 11 proje eğitim veri seti olarak alınarak ölçütler elde edildi. Bu verilerle sınıf sayısı, metot sayısı, sınıflardaki satır sayısı, metotlardaki satır sayısı, WMC (Sınıfın ağırlıklı metot sayısı), CBO (Nesneler arası bağımlılık), en yüksek iç içe koşul sayısı, if/else sayısı, yerel değişken sayısı, while döngü sayısı, for döngü sayısı, foreach döngü sayısı, switch sayısı, switch içinde sayısı, try/catch sayısı, and ve or işlem sayısı, get/set sayısı, public değişken sayısı bilgileri elde edilmiştir.

1.3.1 Çalışmada Kullanılan Ölçütler

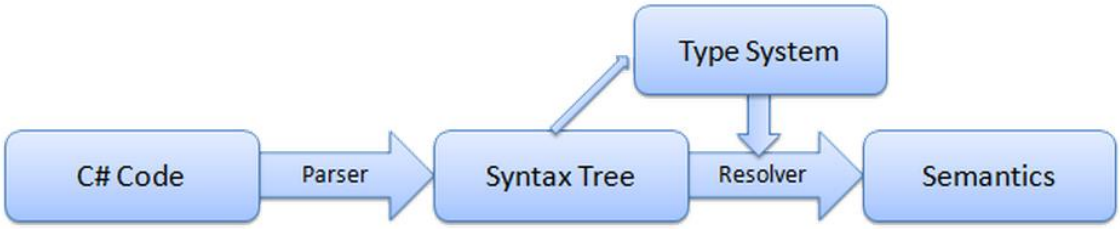
Bu bölümde çalışmada tespit edilmesi hedeflenen Beyin Metot ve Beyin Sınıf kod kusurlarının belirlenmesinde kullanılmak üzere hesaplanan yazılım ölçütlerinden bahsedilmektedir. Beyin Sınıf kod kusurunun bulunması için önce Beyin Metot kod kusuru bulunmalıdır. Birden fazla ölçüt bir araya getirilerek, Beyin Metot ve Beyin Sınıf kod kusurları tespit edildi. Sonraki bölümlerde ölçütlerin nasıl kullanıldığını gösteren şekiller, Lanza ve Marinescu’nun [3] çalışmasında önerilen mantıksal kapılar ile çizildi. Bu mantıksal kapılarda ölçütler kutularla, bağlantıları ise “ve” ile “veya” işleçlerinin mantık devrelerinde gösterildikleri biçimde çizildi. Kod kusurları ve bu kusurları tespit etmede kullanılan ölçütler aşağıdaki gibi ayrıntılandırıldı:

- LOC (Lines of Code): Bir metot ya da sınıftaki satır sayısını bulan ölçüttür. Kodda bulunan satırlar arası boşluklar ve yorum satırları bu sayıya dâhil değildir. Beyin Metot ve Beyin Sınıf kod kusurlarını bulmak için kullanıldı.
- CYCLCO (Cyclomatic Complexity): Bir metotta bulunan operasyon sayısını veren ölçüttür. Bu ölçüt bağımsız işlemlerin sayısını hesaplar. If/else sayısı, while sayısı, do while sayısı, for sayısı, foreach sayısı, switch sayısı ve case sayısı toplamlarından oluşur. CYCLO, Beyin Metod ve Beyin Sınıf kod kusurlarının tespitinde kullanılır ayrıca WMC (Weighted Method Per Class) ölçütünü hesaplamak için kullanılır.
- NOAM (Number of Accessor Methods): Bir sınıftaki get/set erişim metotlarının sayısını hesaplar.

- NOM (Number of Methods): Bir sınıftaki metotların sayısını hesaplar. Beyin Sınıf kod kusurunu bulmak için kullanıldı.
- CBO (Coupling Between Object Classes - Nesneler arası bağımlılık): Bir sınıfın başka sınıflara ne kadar bağlı olduğunu hesaplayan ölçüttür. Beyin Metot ve Beyin Sınıf kod kusurlarının tespitinde kullanılır.
- MAXNESTING (Maximum Nesting Level): Bir metot içindeki işlemlerin maksimum derinliğini hesaplayan ölçüttür. Bu ölçüt Beyin Metot ve Beyin Sınıf kod kusurlarını bulmak için kullanılır.
- NOAV (Number of Accessed Variables): Bir metottaki değişken sayısını hesaplayan ölçüttür. Bu ölçüt Beyin Metot kod kusurunu bulmak için kullanılır.
- WMC (Weighted Method Per Class): Bir sınıftaki metotların karmaşıklığını veren ölçüttür, sınıftaki metotların CYCLO ölçütü değerlerinin toplamından oluşur. Beyin Sınıf kod kusuru tespitinde kullanılır.

1.3.2 Dosya Ayrıştırma Yöntemi

Dosya ayrıştırma işlemi için bir C# kütüphanesi olan ICSharpCode.NRefactory.CSharp kütüphanesi kullanıldı. NReFactory [4] C# kodlarının sözdizimi ve anlamsal yapılarına erişimi sağlayan bir kütüphanedir. NRefactory ile C# kodlarına ait söz dizimi ve anlamsal verilere erişim aşamaları Şekil 1.1'de gösterilmektedir.



Şekil 1.1 NReFactory kütüphanesi ile C# kodlarının ayrıştırması

1.3.3 Alt Limit ve Üst Limitlerin Belirlenmesi:

Ölçütleri kullanarak yazılım projelerinin eşik değerlerini elde ettikten sonra projeleri bu verilerin limitleri dâhilinde değerlendirmek gerekmektedir. Geliştirilmiş kodlar bu ölçütler üzerinden kod kusuru analizi yapmaktadır. Sınıf satır sayısı ne olmalıdır? Metot satır sayısı kaç olmalıdır? Bir sınıf ortalama karmaşıklık değeri ne olmalıdır? Bir sınıfta kullanılması gereken ortalama If/else sayısı ne olmalıdır? Bir sınıfın diğer sınıflarla bağlantı sayısı ortalama kaç olmalıdır? Bu soruların cevaplarını bulmak için alt ve üst limitler belirlenmiştir.

Analizde kullanılacak ölçütler önceki çalışmalar baz alınarak manuel hesaplanabilir ya da kalitesine güvenilen projelerin eğitim veri seti olarak kullanılmasıyla dinamik olarak hesaplanabilmektedir. Analizde kullanılacak eşik değerlerin alt ve üst limitlerinin belirlenmesi için iki değer hesaplanmaktadır. Bunların ilki ortalama değer (ORT), diğeri ise sapma değeri (SPM)'dir. ORT ± SPM formülüyle eşik değerlerin alt ve üst limitleri belirlenmiştir. Sapmanın değerini hesaplamak için ortalama değerin %10'u alınmıştır. Alt limit ve üst limit Şekil 1.2'deki gibi hesaplanmaktadır.

$$SPM = ORT/10$$

$$\text{Alt Limit} = ORT - SPM$$

$$\text{Üst Limit} = ORT + SPM$$

Şekil 1.2 Alt ve üst limit hesaplama

Bölen değer arttıkça eşik değerlerin aralığı daralacaktır ve hassasiyet artacaktır. İhtiyaçlara göre SPM hesaplamasında bu şekilde değişiklikler yapılabilir, standart sapmada kullanılan oran arttırılabilir ya da azaltılabilir. Alt limit elde edilen sapma değeri kadar eksik, üst limit ise sapma değeri kadar fazla olarak belirlenmiştir. ORT değerinin hesaplanması Modelin Detayı başlığı altında detaylandırılmıştır.

1.3.4 Eşik Değerler Kullanılarak Bulunması Hedeflenen Kod Kusurları

Elde edilen alt ve üst limit değerleri ile test veri seti olarak kullanılan projelerde önce Beyin Metot sonrasında da Beyin Sınıf kod kusurlarının bulunması sağlandı. Beyin Sınıf

tespiti yapabilmek için Beyin Metotların bilinmesi gerektiğinden önce Beyin Metotlar belirlendi.

Beyin Metot

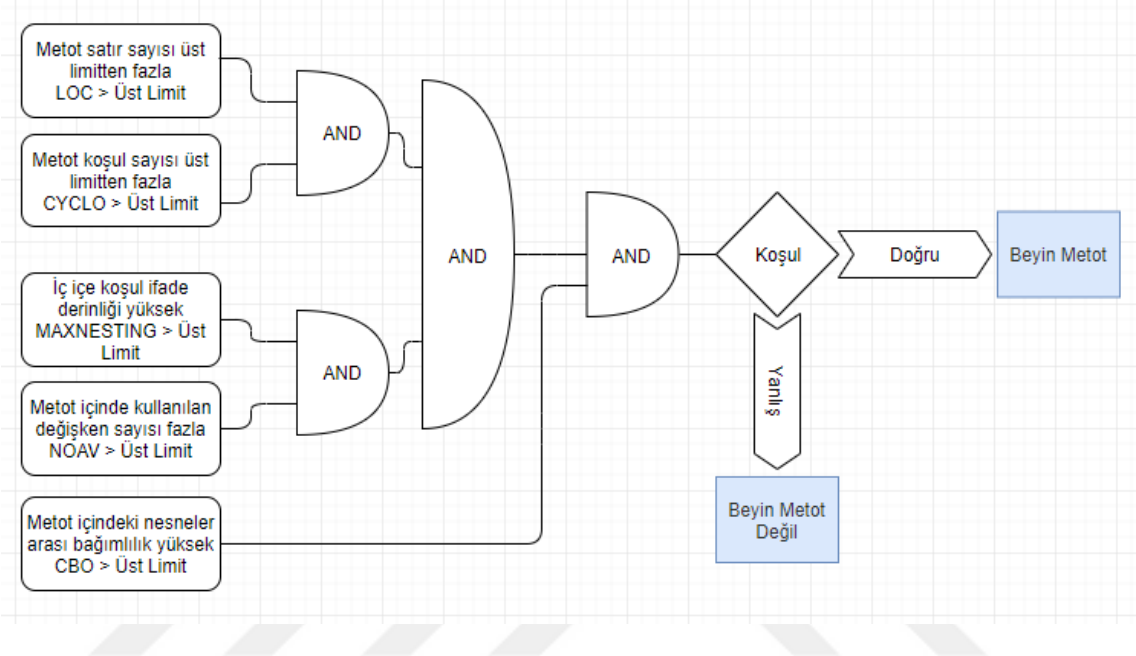
Normal boyutlarda yazılmış olan metotlar zamanla yeni işlevler eklenerek büyümekte, anlaşılması ve onarılması zor bir hale gelmektedir. Bu özellikleri içeren, yapılacak işlerin bir metot içinde toplanıp merkezileştirildiği metotlara beyin metot denmektedir. Bu metotların tespiti sonrasında metot çıkarma yöntemi ile kod kusurları giderilebilir.

Beyin metodu tespit etmek için tespit skoru hesaplanmıştır. Bu tespit skorunu takip etmek adına tespit skoru değişkeni tanımlanır. Beyin metot analizi için aşağıdaki koşullardan her biri sağlandıkça bu değişken 1 arttırılır. Bu değişkenin değeri 5 olduğunda ilgili metoda beyin metottur diyebiliriz.

- Metottaki satır sayısı eşik değerlerde belirtilen satır sayısının üst limitinden fazla ise skor değişkeni 1 arttırılır. Bu koşulda LOC ölçütü kullanılarak uzun metotlar bulunur. Uzun metotlar birden fazla iş içerir ve bu da kod kusuruna sebebiyet verir.
- Metottaki CYCLO değeri eşik değerlerde belirtilen CYCLO değerinin üst limitinden fazla ise skor değişkeni 1 arttırılır. Bu koşulda CYCLO ölçütü kullanılarak metotların karmaşıklık değerleri bulunur. Karmaşık metotlar iş yükünün fazla olduğu metotlardır bu da kod kusuruna sebebiyet verir.
- Metottaki işlemlerin maksimum derinliği eşik değerlerde belirtilen maksimum derinlik değerinin üst limitinden fazla ise skor değişkeni 1 arttırılır. Bu koşulda Maxnesting level ölçütü kullanılarak işlem derinliği fazla olan metotlar bulunur. İşlem derinliği fazla olan metotlar iş karmaşıklığının olduğu metotlardır bu da kod kusuruna sebebiyet verir.
- Metottaki yerel değişkenlerin sayısın eşik değerlerde belirtilen yerel değişken sayısının üst limitinden fazla ise skor değişkeni 1 arttırılır. Bu koşulda NOAV ölçütü kullanılarak metotların içindeki değişken sayısı bulunur.
- Metot içinde başka nesnelere bağımlılık eşik değerlerde belirtilen nesneler arası bağımlılık değerinin üst limitinden fazla ise skor değişkeni 1 arttırılır. Bu şekilde diğer

nesnelerle yoğun ilişkide olan metotlar tespit edilir. Başka nesnelerle yoğun ilişkide olan metotlar fazla işin yapıldığı metotlardır bu da kod kusuruna sebebiyet verir.

Analiz sonucunda tespit skoru 5 olursa analiz edilen metot "Beyin Metot" olarak sınıflandırılır. Beyin Metot kod kusuru tespit yöntemi Şekil 1.3'te ayrıntılı bir şekilde gösterilmiştir.



Şekil 1.3 Beyin metot kod kusuru tespit yöntemi

Önerilen yöntem Lanza ve Marinescu [3] tarafından önerilen şekilde mantık kapıları ile çizilmiştir. Eğer analiz edilen metodun satır sayısı eşik değerlerin üst limitinden fazlaysa, metot karmaşıklık sayısı eşik değerlerin üst limitinden fazlaysa, iç içe koşul ifadesi derinliği eşik değerlerin üst limitinden fazlaysa, metot içinde kullanılan değişken sayısı eşik değerlerin üst limitinden fazlaysa ve nesneler arası bağımlılık eşik değerlerin üst limitinden fazlaysa bu metot Beyin Metot kod kusurunu içeriyor demektir.

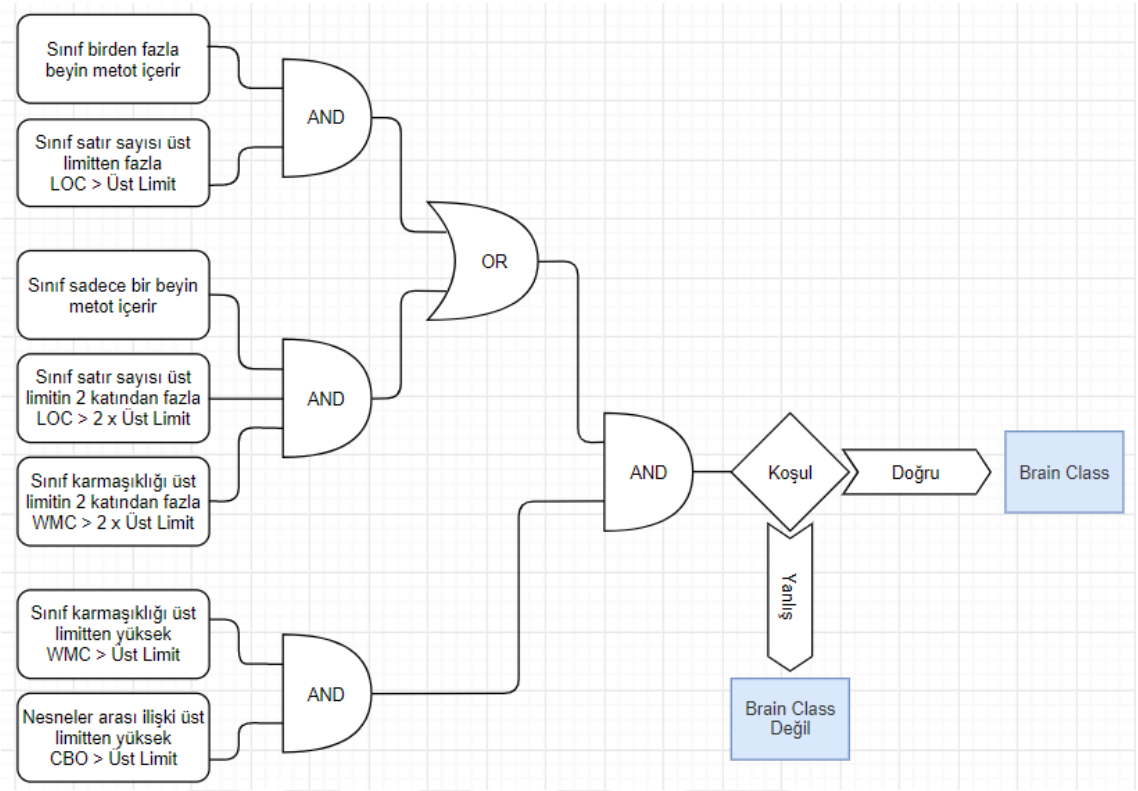
Beyin Sınıf

Sistemde yapılacak işlerin bazı sınıfların üzerinde merkezileştirilmesi problemidir. Sistemdeki birçok sorumluluk ve görev bu sınıfa yüklenmiştir. Tek sorumluluk prensibine aykırı tasarlanmış olan bu sınıflar, nesneye yönelik programlama tasarım prensiplerine ters düşmektedir. Bu sınıfların anlaşılabilirliği düşük, bakım maliyeti yüksek ve test edilmesi zordur [3]. Bu sınıfların tespiti sonrasında sınıf çıkarma yöntemi ile kod kusurları

giderilebilir. Beyin sınıf tespit etmek için tespit skoru hesaplanmıştır. Bu tespit skorunu takip etmek adına tespit skoru değişkeni tanımlanır. Tespit skorunu arttırmak için belirlenen eşik değerlerin alt ve üst limitleriyle ölçütler kontrol edilir ve kontrol edilen her ölçüt sonrasında koşul sağlanıyorsa bu skor 1 arttırılır. Eğer bu tespit skoru değeri 2 olursa ilgili sınıf Beyin Sınıf olarak sınıflandırılabilir. İlgili sınıfın Beyin Sınıf olup olmadığını tespit etmek için sırayla aşağıdaki adımlar uygulanır.

- Bir sınıf birden fazla beyin metodu içeriyorsa ve aynı zamanda sınıfın satır sayısı (LOC) belirlenen eşik değerlerin üst limitinden fazlaysa tespit skoru 1 arttırılır. Bu koşulda LOC ölçütü kullanılarak uzun sınıflar bulunur. İşlerin merkezileştirildiği birden fazla beyin metoda sahip uzun sınıflar karmaşık sınıflardır ve bu da kod kusuruna sebebiyet verir.
- İlk koşul sağlanmıyorsa diğer durumda sınıfın içinde bir beyin metot olup satır sayısı (LOC) üst limitin 2 katından fazla ve karmaşıklık (WMC) üst limitin 2 katından fazlaysa tespit skoru 1 arttırılır. Bu koşulda WMC ölçütü kullanılarak sınıflardaki CYCLO karmaşıklık değerlerinin toplamları bulunur. Satır sayısının ve karmaşıklığın fazla olduğu sınıflar iş yükünün fazla olduğu sınıflardır bu da kod kusuruna sebebiyet verir.
- Son olarak sınıfın karmaşıklığı (WMC) eşik değerlerin üst limitinden fazlaysa ve nesneler arası ilişki (CBO) üst limitten yüksekse tespit skoru 1 arttırılır. Bu koşulda WMC ölçütü kullanılarak sınıflardaki CYCLO karmaşıklık değerlerinin toplamları bulunur buna ek olarak CBO ölçütü ile sınıflar arası bağımlılık hesaplanır. Karmaşıklığın ve sınıflar arası bağımlılığın fazla olduğu sınıflar beyin sınıf olma ihtimali yüksek olan sınıflardır.

Analiz sonucunda gerekli koşullar sağlanıp tespit skoru 2 olursa analiz edilen sınıf “Beyin Sınıf” olarak sınıflandırılır. Beyin sınıf kod kusuru tespit yöntemi Şekil 1.4’te ayrıntılı bir şekilde gösterilmiştir.



Şekil 1.4 Beyin sınıf kod kusuru tespit yöntemi

Analiz edilen sınıfın birden fazla beyin metodu içerdiği durumda ve aynı zamanda sınıfın satır sayısı (LOC) belirlenen eşik değerlerin üst limitinden fazla olduğu durumda ya da sınıfın içinde bir beyin metot olup satır sayısı (LOC) ve karmaşıklık (WMC) üst limitin 2 katından fazla olduğunda tespit skoru 1 arttırılır. Buna ek olarak sınıfın karmaşıklığı (WMC) ve nesneler arası ilişki (CBO) eşik değerlerin üst limitinden fazlaysa tespit skoru 1 arttırılır ve bu sınıf beyin sınıf olarak sınıflandırılmış olur.

BÖLÜM 2

İLGİLİ ÇALIŞMALAR

2.1 İlgili Araçlar

Kod kusurları kötü tasarım ve kötü kodlamanın belirtileridir. Bazı yazarlar kötü tasarım ve kodlamanın azaltılması için çeşitli kod kusuru tanımlamaları ile literatüre katkıda bulunmuşlardır. Bunlara örnek verecek olursak Fowler [1], kod kusurlarının yazılım projelerinde gün geçtikçe büyüyerek daha önemli sorunlara yol açacak göstergeler olduğunu belirtir. Mens ve Tourwe´ [2], çalışmalarında yeniden düzenleme (ing. Refactoring) ve yeniden yapılandırma (ing. Restructuring) kavramlarına açıklık getirmiştir. Çalışan bir kod yapısı için gerekli olan yeniden düzenleme yöntemlerini uygulamış ve tasarımı daha anlaşılır, kolay müdahale edilebilir, nesneye dayalı tasarım ilkelerine uygun hale getirmiştir.

Kapdan [6] yaptığı "Nesneye Dayalı Programlama Tabanlı Yazılımlarda Yazılım Ölçütleri Kullanarak Yapısal Klon Kod Tespiti" çalışmasında yazılım ölçütleri kullanarak kopya kodların tespitini önermiştir.

Kod kusurlarını bulabilen çok sayıda araç vardır. Bazı araçlar tümleşik geliştirme ortamına (Integrated Development Environment – IDE) eklenti olarak entegredirler, bazıları da ayrı uygulamalar olarak çalışmaktadırlar. Bu araçların bir kısmı tespit edilen kod kusurlarının giderilmesi için yeniden düzenleme tavsiyelerinde de bulunurlar. Bu tespit araçlarından inCode, JDeodorant ve iPlasma araçlarıyla ilgili bilgiler aşağıda yer almaktadır.

inCode [7] kod kusuru algılaması için bir Eclipse eklentisidir, Eclipse eklentisi olması açısından başarılı bir araçtır. Bu aracın temel özelliği, yazılımcıları kodlama yaptıkları esnada takip edip uyarmaktır. Eclipse IDE'sinin arka planında sürekli çalışır ve programlama sırasında, programcı eğer kötü bir kod yazarsa, bu kod kusurlarını "eclipse show error" olarak gösterir ve kırmızı renkli bloklar şeklindeki uyarıları kodla birlikte gösterir. inCode 4 kod kusuru algılar, bunlar "Feature Envy", "God Class", "Duplicate Code" ve "Data Class" kod kusurlarıdır. Bu kod kusurların tespitleri nesne yönelimli ölçüt ölçümlerine dayanır. Bu aracın incelediği kod kusurlarının detayları aşağıda yer almaktadır:

- Feature Envy: Bir metodun, başka bir sınıfın verilerine ve metotlarına kendi sınıfındaki veri ve metotlardan daha çok ihtiyaç duyması problemidir.
- God Class: Sistemde yapılacak işlerin bazı sınıfların üzerinde merkezileştirilmesi problemidir. Sistemdeki birçok sorumluluk ve görev bu sınıfa yüklenmiştir [8]. Düşük bir iç sınıf uyumuna ve yabancı sınıfların verilerine yoğun erişime sahiptir [9].
- Duplicated Code: Benzer kodların farklı sınıflarda ya da metotlarda bulunması problemidir. Aynı kod farklı yerlerde bulunuyorsa Extract Method yöntemi ile ortak bir metoda eklenir ve diğer yerlerden bu metot çağırılır.
- Data Class: Bu sınıflar kendi sınıfları üzerinde işlem yapmayan, sadece sistemdeki verileri tutan ve diğer sınıfların bu verilere erişmesini sağlayan veri modeli sınıflarıdır.

JDeodorant [10], Java dilinde yazılmış kaynak kod üzerinden "Feature Envy", "Type Checking", "Long Method", "God Class" ve "Duplicated Code" olmak üzere 5 çeşit kod kusuru tanımlayan bir Eclipse eklentisidir. Bu araç, kaynak kodda kod kusurunu bulmak için Eclipse Java Geliştirme Aracı'nın ASTParser API'sini ve yeniden düzenlemeyi (refaktöringi) uygulamak için ASTRewrite API'larını kullanır. JDeodorant geliştirme ortamına entegrasyonu olması açısından başarılı bir araçtır. Bu aracın incelediği kod kusurlarının detayları aşağıda yer almaktadır:

- Feature Envy: Bir metodun, başka bir sınıfın verilerine ve metotlarına kendi sınıfındaki veri ve metotlardan daha çok ihtiyaç duyması problemidir.

- Type Checking: Tanımlanan veri tiplerinin uygunluğunu kontrol eder. Statik ve dinamik olarak ikiye ayrılır, statik tip kontrolü hatası kod yazımı esnasında ortaya çıkar, dinamik tip kontrolü hatası ise çalışma zamanında ortaya çıkar.
- Long Method: Bir metot çok fazla sayıda kod satırı içermesi problemidir. Bu tür metotlar yazılımcının metot içinde yapılan işi anlamasını zorlaştırır.
- God Class: Sistemde yapılacak işlerin bazı sınıfların üzerinde merkezileştirilmesi problemidir. Sistemdeki birçok sorumluluk ve görev bu sınıfa yüklenmiştir [8]. Düşük bir iç sınıf uyumuna ve yabancı sınıfların verilerine yoğun erişime sahiptir [9]
- Duplicated Code: Benzer kodların farklı sınıflarda ya da metotlarda bulunması problemidir. Aynı kod farklı yerlerde bulunuyorsa Extract Method yöntemi ile ortak bir metoda eklenir ve diğer yerlerde bu metot çağırılır.

iPlasma, Java ve C++ dillerinde yazılmış kaynak kod üzerinden “Brain Class”, “Brain Method”, “Data Class”, “Dispersed Coupling”, “Feature Envy”, “God Class”, “Intensive Coupling”, “Shotgun Surgery”, “Refused Parent Bequest”, “Tradition Breaker” olmak üzere 10 çeşit kod kusuru tanımlayan bir araçtır [11]. Herhangi bir geliştirme platformu ile entegrasyonu yoktur ve bu nedenle kod geliştirilirken eş zamanlı olarak kod kusurlarının tespiti mümkün değildir. Bu aracın incelediği kod kusurlarının detayları aşağıda yer almaktadır:

- Brain Class: Sistemdeki yapılacak işlerin bazı sınıfların üzerinde merkezileştirilmesi problemidir. Sistemdeki birçok sorumluluk ve görev bu sınıfa yüklenmiştir, uygulamadaki çoğu işlem bu sınıflar aracılığıyla yapılmaktadır. God Class’a benzerdir fakat beyin sınıfları yabancı sınıflardan çok fazla veri kullanmaz ve tanrı sınıfa göre biraz daha uyumludur [9].
- Brain Method: Bir süre boyunca yeni işlevsellikler eklenerek aşırı büyütülmüş metotlara beyin metod denir. Bu daha fazla karmaşıklığa ve bakımın artmasına neden olur.
- Data Class: Bu sınıflar kendi sınıfları üzerinde işlem yapmayan, sadece sistemdeki verileri tutan ve diğer sınıfların bu verilere erişmesini sağlayan veri modeli sınıflarıdır.

- Dispersed Coupling: Bu, sistemdeki diğer birçok işleme aşırı bağlı bir işlemdir ve ek olarak bu sağlayıcı metotlar birçok sınıf arasında dağılmışlardır [3].
- Feature Envy: Bir metodun, başka bir sınıfın verilerine ve metotlarına kendi sınıfındaki veri ve metotlardan daha çok ihtiyaç duyması problemidir.
- God Class: Sistemde yapılacak işlerin bazı sınıfların üzerinde merkezileştirilmesi problemidir. Sistemdeki birçok sorumluluk ve görev bu sınıfa yüklenmiştir [8]. Düşük bir iç sınıf uyumuna ve yabancı sınıfların verilerine yoğun erişime sahiptir [9].
- Intensive Coupling: Aşırı bağlantı durumunda oluşan hatadır. Sistemdeki diğer birçok işleme bağlı bir metodun sağlayıcı işlemlerinin yalnızca bir veya birkaç sınıfa dağıldığı durumdur.
- Shotgun Surgery: Bir değişiklik yapılması gerektiğinde kodda birçok sınıfta küçük değişiklik yapılmasının gerekmesi problemidir. Böyle bir durumda önemli bir değişikliğin atlanma ihtimali yüksektir. Bu sınıflarda karmaşıklık ve bağımlılık metriğinin yüksek olması beklenir.
- Refused Parent Bequest: Bir alt sınıf, ebeveynlerinden miras alınan metotların ve özelliklerin yalnızca bir kısmını kullanıyorsa, hiyerarşinin dengesi bozulmuştur. Gereksiz metotlar kullanılmayabilir veya yeniden tanımlanabilir ve bunlar hata çıkarabilir.
- Tradition Breaker: Bu sorun sınıfın, miras alınan sınıf tarafından sağlanan özellik ve metotlarla ilgisi olmayan geniş bir hizmet kümesi sunması problemidir. Miras alınan sınıfın metotları ve özelliklerini özelleştirmeyip tamamen yeni metot ve özellik eklenmesi durumunda bu sınıfın tanımında sorun olduğu görülür [3].

2.1.1 Teknikler

Marinescu [12] ölçüt tabanlı çözümleri ile kod kusurlarını tespit etmek için iPlasma aracını geliştirmiştir. iPlasma aracı, ölçülebilir kural ifadeleri ile kodlarda yer alan kod kusurlarını bulmayı sağlar. Marinescu, iPlasma'da yapılacak kod kusuru tespitlerinde kullandığı eşik değerleri 45 Java projesinin ve 37 C++ projesinin istatistiksel sonuçlarından toplamıştır [3]. Filtreleme ve birleştirme mekanizması kullanarak

oluşturduğu ölçütlerle, 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 Marinescu'nun kod kusuru tespiti doğruluk oranları (2004)

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

Malhotra ve Pritam [13] nesneye dayalı bir sistemdeki mevcut kod kusurlarının bir sınıf için tespit edilme eğilimini deneysel olarak onaylamak amacıyla bu çalışmayı gerçekleştirdiler. Quartz olarak adlandırılan açık kaynak zamanlayıcıdan 79 sınıf alarak veri seti olarak kullandılar. Eğilimi düşük ve yüksek olarak sınıflandırdılar. Ayrıca Java sınıflarındaki 13 farklı kod kusurunu tespit etmek için eşik değerleri kullanarak bir araç geliştirmişlerdir.

Munro [14] kod kusurlarını sistematik olarak bulabilmek için bir şablon önermiştir. Şablon; kod kusurunun adı, özelliği ve bulma sezgileri olarak 3 bölümden oluşmaktadır. Munro, Marinescu'nun da yönteminde olduğu gibi yazılım ölçütlerinden faydalanmaktadır. Java kaynak kodunda otomatik kod kusuru bulma özelliği ile tasarım problemlerini bulmaya odaklanmıştır. Lazy class, geçici değişkenler ile alakalı kod

kusurlarını bulmayı sağlar. Bu çalışmada amaç Java kaynak kodlarındaki kod kusurlarını bularak, sonucu kullanıcıya excel dökümanı halinde çıktı olarak verebilmektir. Manuel sistemlere göre oldukça hızlı sonuç veren bu yöntemde ayrıca hata oranı da manuel sistemlere göre daha az olmaktadır.

Çizelge 2.2 Munro'nun kod kusuru tespiti doğruluk oranları (2005)

	Pozitif	**Negatif**
Doğru	7	5
Yanlış	1	0

Çizelge 2.3 Çalışmaların karşılaştırılması

	Ortalama Doğruluk Oranı	Büyük Uygulamalarda Kullanıma Uygunluk	Bulduğu Kod Kusur Sayısı
Marinescu	%67	Uygun	9
Malhotra ve Pritam	%81	Uygun	13
Munro	%93	Uygun	2

BÖLÜM 3

ÖNERİLEN MODEL

Önerilen yöntemde yazılım ölçütleri kullanılarak Beyin Metot ve Beyin Sınıf kod kusurlarının tespiti hedeflenmektedir. Beyin sınıf kod kusuru bulunurken öncesinde beyin metot kod kusurunun tespiti yapılmıştır. Ölçütlerden faydalanılarak kod kusurlarının daha doğru sonuçlar ile bulunması amaçlanmaktadır.

C# ile önerilen yöntemi gösteren bir Windows Form uygulaması geliştirilmiştir. Bu uygulama ile kod kusurları son kullanıcı tarafından görüntülenebilmektedir. Uygulama, kullanıcının analiz aşamasında kullanılacak kod ölçütlerini kalitesine güvendiği projelerden dinamik olarak oluşturup analizleri bu ölçütler üzerinden yapabilmesine imkan vermektedir. Uygulama ayrıca kullanıcının daha önce yapılan analizler sonucunda elde edilen ölçütleri uygulamaya girebilmesini ve analizlerini bu ölçütler üzerinden yapabilmesine de imkan vermektedir. Önerilen yöntemde ölçütlerin eşik değerlerinin normal aralıklarının belirlenmesi için kullanılan projelerin kaynak kodlarının oluşturduğu havuza “eşik değer havuzu” adı verilmiştir.

3.1 Kullanılan Gereçler

Kod kusuru tespiti için C# Nrefactory, Roslyn, Visual Studio Calculate Code Metrics, C# to Java Converter, IPlasma ve Codecity gereçlerinden faydalanılmıştır. Bu bölümdeki alt başlıklar kullanılan gereçler hakkında bilgi vermektedir.

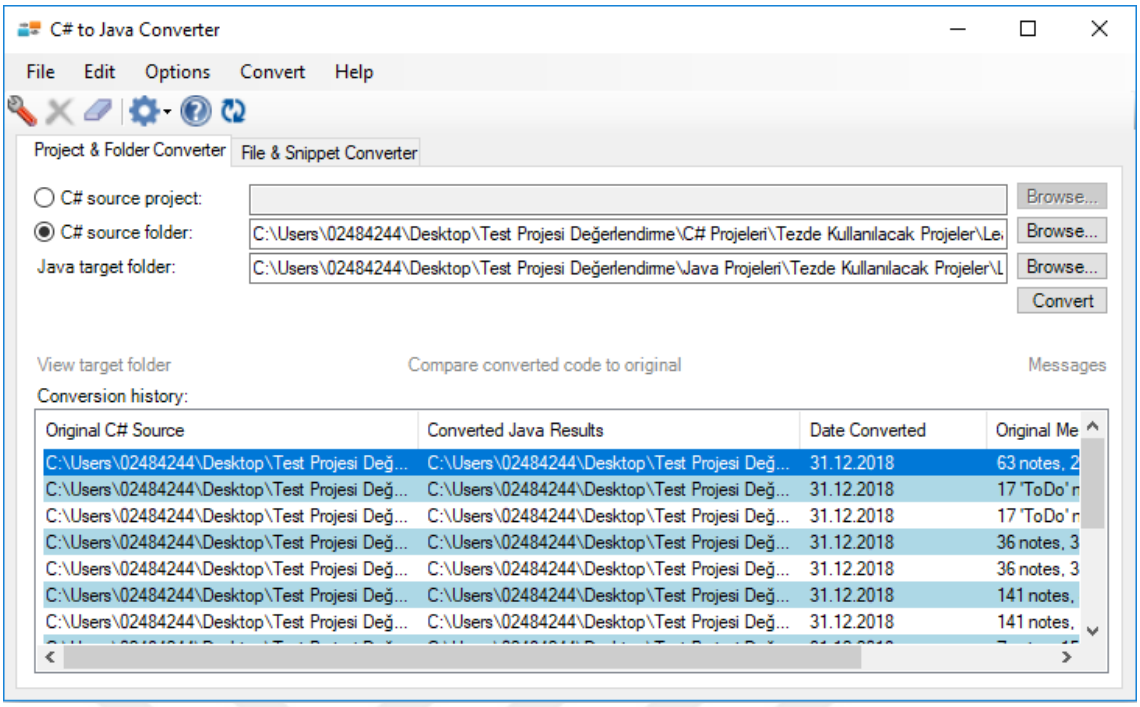
3.1.1 Roslyn

Roslyn [15], BB Code Smell Detector projesinin geliştirme sürecinde, projelerin analiz edilmesi için nesneler oluşturulmasında kullanılmıştır. Roslyn proje sınıflarını AST (Abstract Syntax Tree) ağacına atarak, kodların ayrıştırılmasını ve ölçümlendirilmesini sağlamaktadır. AST'leri oluşturulmuş olan sınıflar üzerinden analiz aşamasında kullanılacak ölçütler elde edilmektedir. Bu kütüphane hem alt limit ve üst limit elde etme, hem de kod kusuru bulma aşamalarında kullanılmaktadır.

3.1.2 C# to Java Converter

Analiz sonuçlarının doğruluk oranlarını değerlendirmek için BB Code Smell Detector uygulaması ile IPlasma ve CodeCity uygulamaları karşılaştırıldı. BB Code Smell Detector C# kaynak kodlarını analiz edebiliyorken, IPlasma ve CodeCity araçları Java kodlarını analiz edebilmektedir. Bu analiz araçlarının sonuçlarını karşılaştırabilmek için aynı projeler üzerinde analiz yapılması gerektiğinden, test veri seti olarak kullanılan projelerin kaynak kodlarını C#'tan Java'ya çeviren C# to JavaConverter[1] [16] aracına ihtiyaç duyuldu. Test için ayrılan C# kaynak kodlu projeler, programlama dili çevirici araç ile Java kaynak koduna dönüştürüldü. C#'ta olup Java'da olmayan using kullanımı ve linq kod blokları çevirici araç tarafından çevrilemediğinden, dönüştürme işlemi sonrasında kodlarda manuel düzeltmeler yapıldı. Böylece Java kodlarını analiz eden IPlasma ve CodeCity araçlarıyla BB Code Smell Detector uygulamasının değerlendirme sonuçları eşit koşullarda karşılaştırıldı. Şekil 3.1'de C# to Java Converter uygulamasının ara yüzü gösterilmektedir.

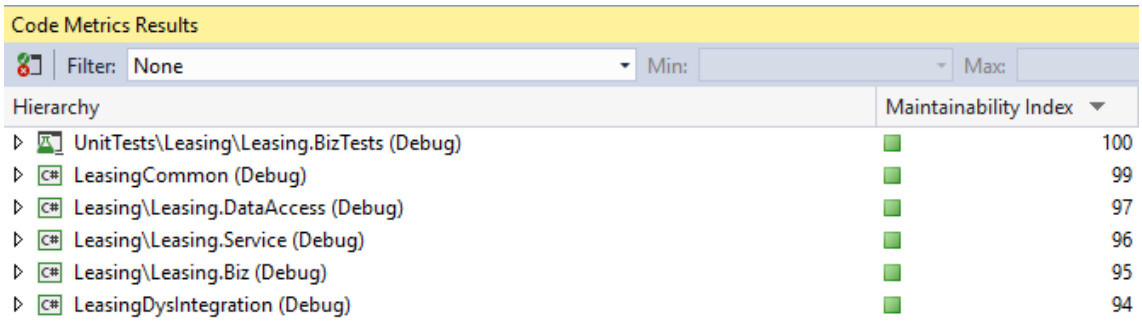
[1] http://www.tangiblesoftwaresolutions.com/Product_Details/CSharp_to_Java_Converter_Details.html



Şekil 3.1 C# to Java Converter

3.1.3 Visual Studio Code Metrics

Eşik değer havuzuna Maintainability Index (Bakım İndeksi) değeri yüksek projeleri dahil etmek için Visual Studio eklentisi olan Code Metrics eklentisi kullanıldı. Bu eklenti bakım indekslerine göre kaliteli projelerin belirlenmesi aşamasında kullanılmıştır. Şekil 3.2'de bu eklentiye ait ekran görüntüsü bulunmaktadır.



Şekil 3.2 VS Code Metrics bakım indeksi örneği

Code Metrics uygulamasından elde edilen bakım indeksi değerleri 3.1 numaralı denklemdeki formül kullanılarak hesaplanmaktadır. Bu uygulama alt ve üst limit oluşturmak için toplanan projelerin kalitesinin yüksek olduğunu göstermek için

kullanılmaktadır. Eşik değer havuzu oluşturmak için bakım indeksi değeri yüksek olan projeler tercih edilmiştir.

$$\boldsymbol{Maintainability\ Index} = Max(0, (171 - 5.2 * \ln(Halstead\ Volume) - 0.23 * Cyclomatic\ Complexity - 16.2 * \ln(Lines\ Of\ Code)) * \frac{100}{171})$$	(3.1)

Bu formüle göre maksimum kontrolü olmadığında (−∞, 171] arasında bir sonuç çıkmaktadır. Eksi değerlerin etkisinin çok az olduğunu görüldüğünden yani eksi değer çok çok kötü anlamına geldiğinden 0'dan küçük değerlerin tümü 0 olarak kabul edilmiştir ve [0, 171] aralığının [0, 100] arasına normalize edilmesi için hesaplanan değer 171'e bölünmüştür. Bu ölçüte göre 0'a yaklaştıkça bakım yapılabilirlik azalır, 100'e yaklaştıkça bakım yapılabilirlik artar. Formülde kullanılan değişkenler aşağıda açıklanmaktadır.

- Halstead Volume (HV): Bir algoritmanın uygulanmasının boyutunu açıklar. Değerin hesaplanması, gerçekleştirilen işlem sayısına ve algoritmada işlenen işlemlere dayanmaktadır.

  µ1 = tekil operatörlerin sayısı

  µ2 = tekil operandların sayısı

  µ = µ1 + µ2

  V = N x Log2 µ (Halstead Volume)

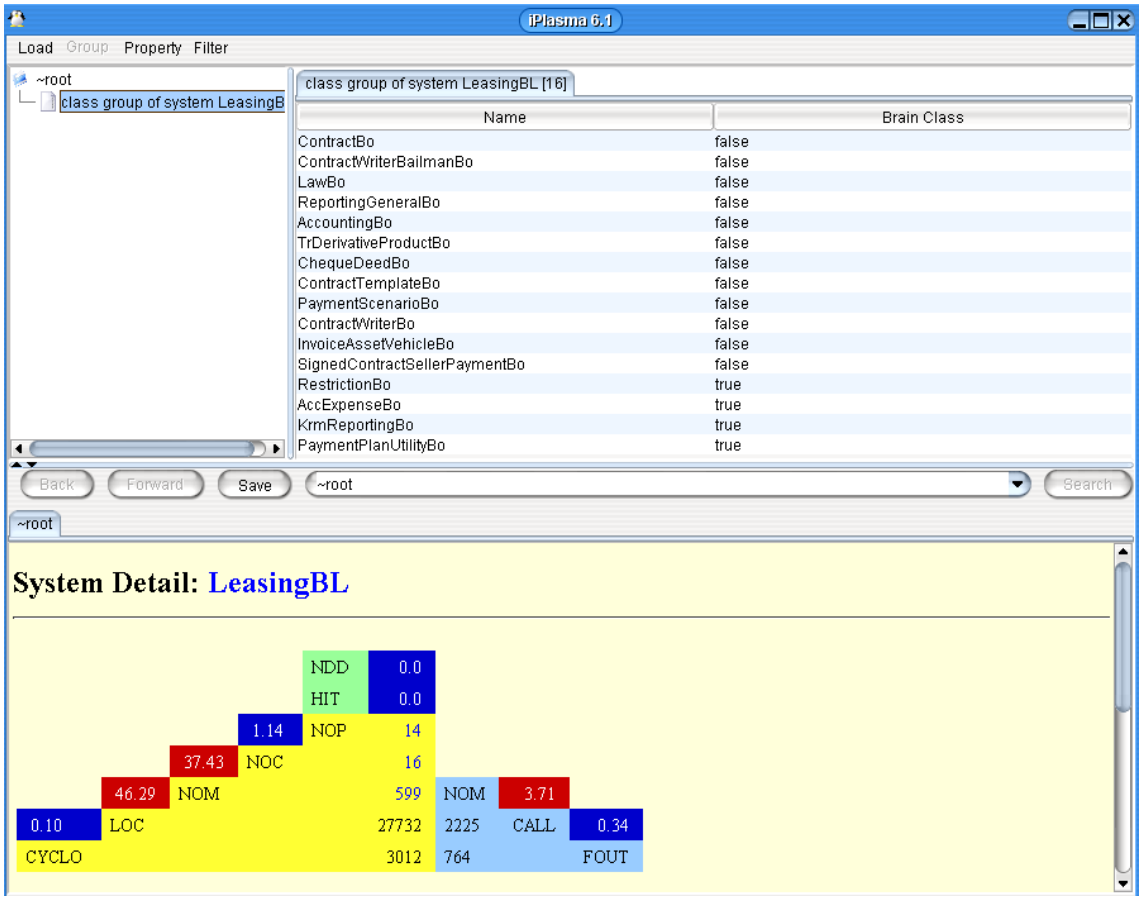
- Cyclomatic Complexity (CYCLCO): Bu ölçüt bağımsız işlemlerin sayısını hesaplar. If/else sayısı, while sayısı, do while sayısı, for sayısı, foreach sayısı, switch sayısı ve case sayısı toplamlarından oluşur.

- Lines Of Code (LOC): Kaynak kodun satır sayısıdır.

3.1.4 IPlasma

Iplasma (http://loose.upt.ro/iplasma/) uygulaması, Beyin Metot ve Beyin Sınıf analiz sonuçlarını BB Code Smell Detector uygulamasının analiz sonuçları ile kıyaslamak için kullanıldı. IPlasma Java kodlarında kod kusurlarını bulurken BB Code Smell Detector C# kodlarında kod kusurları bulabilmektedir. Aynı projeler üzerinde analiz yapabilmeyi

sağlamak için, test veri seti olarak kullanılan C# projelerinin C# to Java Converter aracı kullanılarak Java versiyonu oluşturuldu. Bu projeler IPlasma aracı ile analiz edilip elde edilen sonuçlar BB Code Smell Detector aracının sonuçlarıyla karşılaştırıldı. Şekil 3.3'te IPlasma uygulamasına ait Beyin Sınıf kod kusurunun FineLease.LeasingBL projesi içinde arandığı ekran görüntüsü gösterilmektedir.

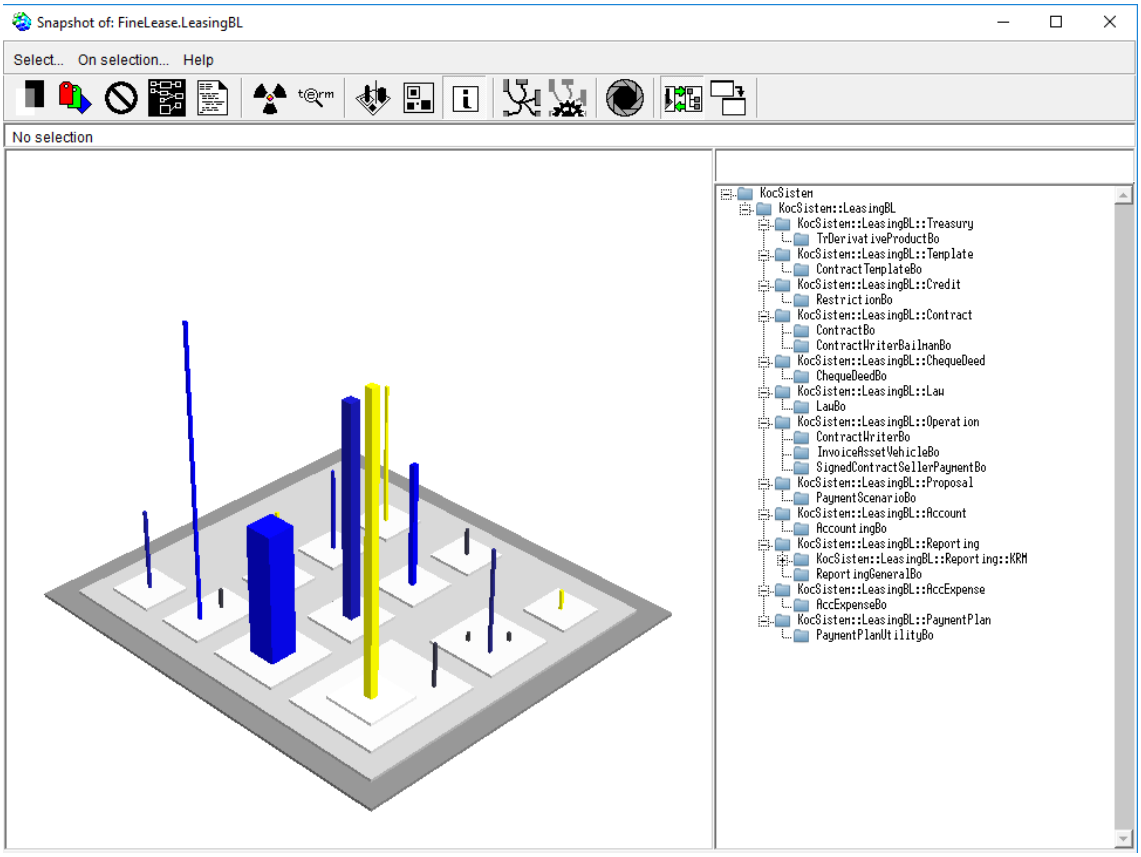


Şekil 3.3 FineLease.LeasingBL IPlasma örnek ekran görüntüsü

3.1.5 CodeCity

CodeCity (https://wettel.github.io/codecity-download.html) Beyin Metot ve Beyin Sınıf analiz sonuçlarını BB Code Smell Detector uygulaması ile kıyaslamak için kullanıldı. CodeCity Java kodlarında kod kusurlarını bulurken BB Code Smell Detector C# kodlarında kod kusurları bulabilmektedir. Aynı projeler üzerinde analiz yapabilmeyi sağlamak için, test veri seti olarak kullanılan C# projelerinin C# to Java Converter aracı kullanılarak Java versiyonu oluşturuldu. CodeCity aracı mse uzantılı dosyaları analiz edebildiğinden IPlasma aracı ile test edilecek projelerin Java kodları mse uzantılı dosyalara

dönüştürüldü. Sonrasında analiz edilecek projeler CodeCity aracı ile analiz edilip elde edilen sonuçlar BB Code Smell Detector aracının sonuçlarıyla karşılaştırıldı. Şekil 3.4'te CodeCity uygulamasına ait Beyin Sınıf kod kusurunun FineLease.LeasingBL projesi içinde arandığı ekran görüntüsü gösterilmektedir.



Şekil 3.4 FineLease.LeasingBL CodeCity örnek ekran görüntüsü

3.2 Geliştirme

Yöntemde eğitim verisi ve test verisi olmak üzere iki tür veri sınıfı bulunmaktadır. İlk aşamada eğitim verisine ait projeler alınarak kod kusurları için ölçütlerin eşik değerlerinin alt ve üst limitleri oluşturulacaktır. Eğitim veri seti üzerinden eşik değerler belirlenmek istenmezse kullanıcılar kendi eşik değerlerini sisteme girebileceklerdir. Elde edilen bu limitler daha sonra test veri seti projelerinin değerlendirilmesinde kullanılmak üzere veritabanında saklanacaktır. Böylece kod kusurlarının bulunması otomatize edilecektir.

Şekil A-1.1'deki kod bloğunda C# kodlarının anlamsal söz dizimi (AST) verilerine erişiminin nasıl yapıldığı gösterilmektedir. Dosya ayrıştırma yöntemi Hipotez başlığı altında ayrıca detaylandırılmıştı.

Şekil 3.5'te kaynak kodlar NReFactory kütüphanesi içinde yer alan SyntaxTree nesnesinin içine atılarak o kod bloğu için AST (Abstract Syntax Tree) nesnesi oluşturuldu. SyntaxtTree nesnesi istenen farklı ölçütlere göre filtrelenerek, ağaca ait olan o özellikteki yapraklar getirildi. Kütüphanede bu yapraklar, descendant yani torun olarak adlandırılmaktadır, bu torun yapraklar değerlendirmede kullanılacak ölçütlerin bilgilerini içermektedir. Örneğin Şekil 3.5'te tip olarak MethodDeclaration seçilerek, ağaç içinde bulunan metot tanımlarının getirilmesi sağlandı. Elde edilen bu metot listesi sonra kullanılmak üzere methodList değişkenine atandı.

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

Şekil 3.5 AST ağacı üzerinden metot listesi oluşturma

Bu metot listesi içinde her metot için metot adı, sınıf adı, içindeki If/Else koşullarının sayısı, o metottaki toplam değişken sayısı, döngü sayısı, nesne sayısı ve satır sayısı gibi bilgileri içermektedir. Bu değerlerin hesaplanması Şekil A-2.1'de gösterilmektedir.

```
public static string[] GetDirectoryFiles(string path, string fileExtension)
{
 var parser = new CSharpParser();
 SyntaxTree syntaxTree;

 var classPaths = Directory.GetFiles(path, "*.cs",
SearchOption.AllDirectories);


 foreach (var classPath in classPaths)
 {
  syntaxTree = parser.Parse(fileBody, classPath);

  var classList = syntaxTree.Descendants.OfType<TypeDeclaration>().Where(k =>
k.ClassType == ClassType.Class)
 }
}
```

Şekil 3.6 AST ağacı üzerinden sınıf listesi oluşturma

Şekil 3.6'da AST içinden sınıf tipindeki nesnelerden bir sınıf listesi oluşturuldu. Elde edilen bu sınıf listesi içinde her bir sınıf için toplam satır sayısı, ortalama satır sayısı,

toplam metot sayısı, ortalama metot sayısı ve toplam if/else sayısı, ortalama if/else sayısı, Cyclomatic Complexity gibi bilgileri içermektedir. Bu değerlerin hesaplanması Şekil A-3.1'de gösterilmektedir.

Cyclomatic Complexity hesaplanırken AST üzerinden hesaplanmış ölçütler CalculateCyclomaticComplexity metoduna MethodInfo sınıfı tipinde bir parametre olarak gönderilerek o sınıfa ait If/Else sayısı, While sayısı, Do while, For sayısı, Foreach sayısı, Switch İçinde Case sayısı üzerinden hesaplanmaktadır. Bu işlemleri yapan kod Şekil 3.7'de gösterilmiştir.

```
private int CalculateCyclomaticComplexity(MethodInfo methodInfo)
{
 return 1 + methodInfo.NumberOfIfElseStatement +
            methodInfo.NumberOfWhileStatement +
            methodInfo.NumberOfDoWhileStatement +
            methodInfo.NumberOfForStatement +
            methodInfo.NumberOfForeachStatement +
            methodInfo.NumberOfCaseInSwitchStatement;

}
```

Şekil 3.7 Cyclomatic complexity hesaplama

Sınıf ve metotlara ait olan toplam satır sayısı, ortalama satır sayısı, toplam metot sayısı, ortalama metot sayısı ve toplam if/else sayısı, ortalama if/else sayısı, Cyclomatic Complexity bilgileri Şekil A-4.1'de gösterilen kod bloğu ile Mssql veritabanında daha sonra kullanılmak üzere saklanmıştır.

Şekil A-4.1'deki kod bloğuyla veritabanında saklanan sonuçlar Şekil 3.8'de gösterilmektedir.

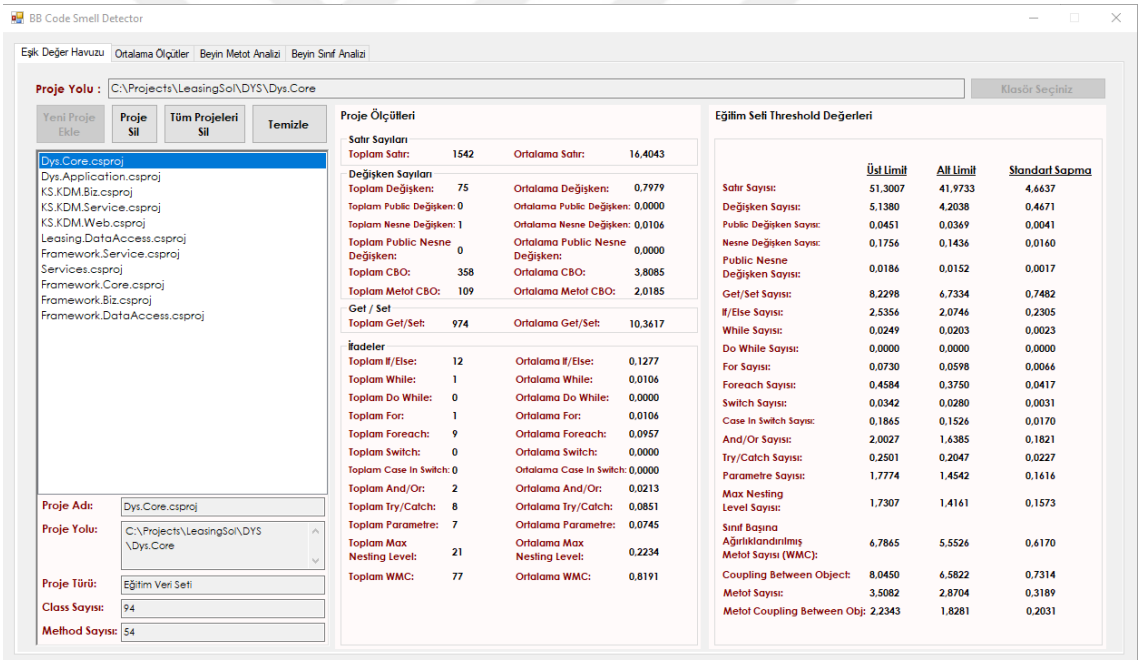
Sınıf sayısı: 708	Değişken sayısı: 3307
Metot sayısı: 2258	if/else: 1632
Sınıflardaki kod satırı: 33019	Foreach: 295
Metotlardaki kod satırı: 21737	For: 47
WMC (Weighted Method Per Class): 4368	While: 16

Şekil 3.8 Ölçütlerin sonuçlarının saklandığı veritabanı içeriği

CBO (Coupling Between Object): 5178	Try/Catch sayısı: 161
get/set sayısı: 5297	Switch: 22
Maxnesting: 1114	Switch içinde case: 120

Şekil 3.8 Ölçütlerin sonuçlarının saklandığı veritabanı içeriği (devamı)

Geliştirmiş olduğumuz araçta eğitim verisi olarak Maintainability Index değeri yüksek olan 11 proje ele alınmıştır. Şekil 3.8'de gösterildiği gibi, sistem ölçümlendirildiğinde 708 sınıf sayısı, 2258 metot sayısı, 33019 sınıflardaki kod satırı, 21737 metotlardaki kod satırı, 4368 WMC değeri, 5178 CBO değeri, 5297 get/set sayısı, 1114 maxnesting, 3307 değişken sayısı, 1632 if/else, 295 foreach, 47 for, 16 while, 161 try/catch sayısı, 22 switch, 120 switch içinde case sayısı bilgileri elde edilmiştir. Bu değerler BB Code Smell Detector uygulamasının eşik değer havuzu ile dinamik olarak hesaplanmıştır.



Şekil 3.9 Eğitim veri seti uygulama görüntüsü

Çizelge 3.1 Eşik değer havuzu maintainability index değerleri

Proje Adı	**Toplam Sınıf Sayısı**	**Toplam Metot Sayısı**	**Maintainability Index**
Dys.Core.csproj	94	54	91
Dys.Application	78	212	86
KS.KDM.Biz	12	20	97

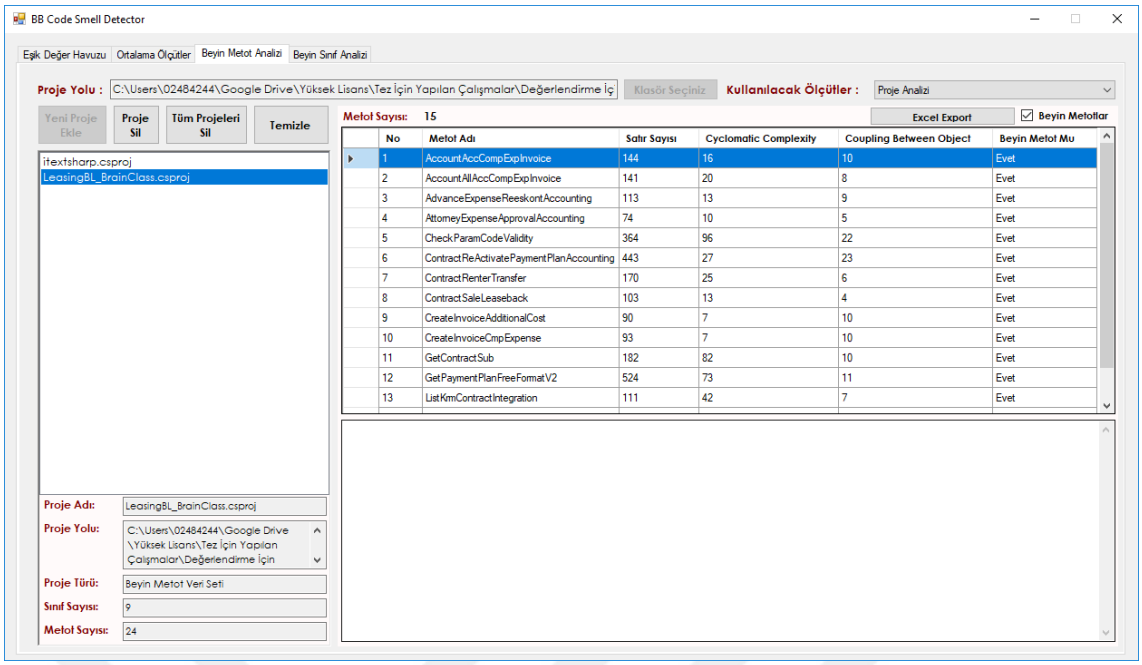
Çizelge 3.1 Eşik değer havuzu maintainability index değerleri (devamı)

KS.KDM.Service	86	331	91
KS.KDM.Web	130	294	86
Leasing.DataAccess	87	420	98
Framework.Service	22	114	94
Services	36	121	94
Framework.Core	92	295	92
Framework.Biz	56	238	86
Framework.DataAccess	15	159	84

Eşik değer havuzunu oluşturan projelere ait Maintainability Index bilgileri çizelge 3.1'de gösterilmektedir. Eşik değer havuzu için Maintainability Index değeri yüksek olan projeler seçilmiştir, Maintainability Index bu projelerin kod kalitesinin daha yüksek olduğunu göstermektedir. Çizelge 3.1'de Maintainability Index değeri 80 ve 100 arasında olan 11 projenin bilgileri gösterilmiştir.

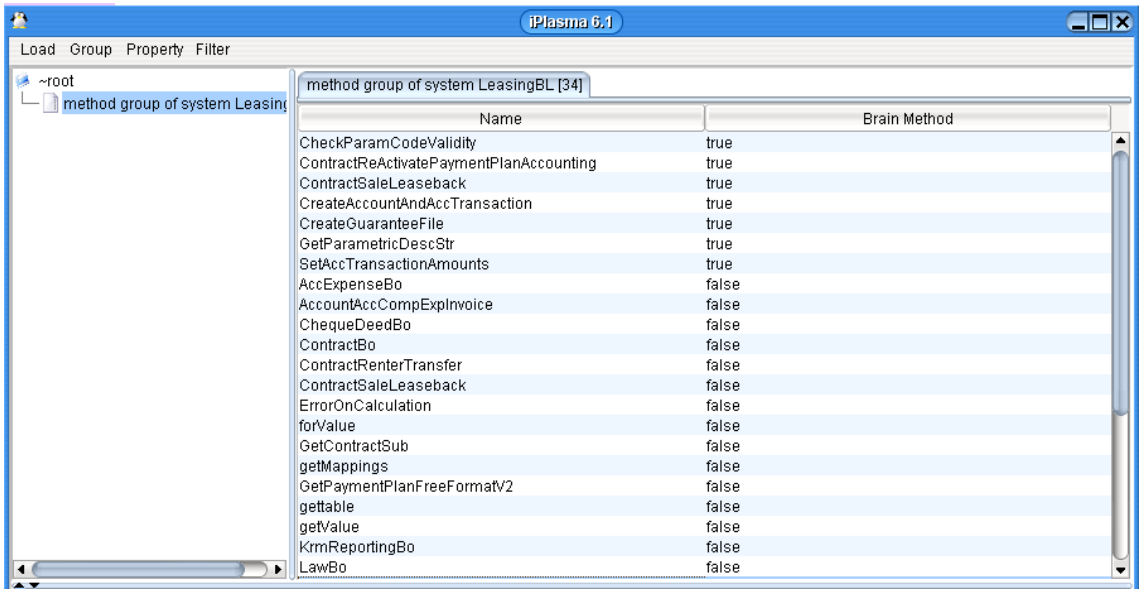
3.2.1 Beyin Metot Model Detayı

Geliştirilen araç ile eğitim verileri ölçütlendirilerek elde edilen eşik değerler kullanılarak analiz edilecek kodların Beyin Metot kod kusurunu içerip içermediğinin bulunması hedeflenmektedir. Aşağıdaki şekillerde analiz sayfasında FineLease.LeasingBL ve iTextSharp projelerine ait Beyin Metot kod kusuru içeren metotların listesi gösterilmektedir. Şekil 3.10'da FineLease.LeasingBL projesinin analizi sonucunda BB Code Smell Detector'un çıktısı görüntülenmektedir. BB Code Smell Detector'e göre FineLease.LeasingBL projesinde AdvanceExpenseReeskontAccounting, AccountAllAccCompExpInvoice, ContractRenterTransfer, ContractSaleLeaseback, CreateInvoiceAdditionalCost, CreateGuaranteeFile, CreateAccountAndAccTransaction, CreateCreditBalancedPayment, GetParametricDescStr, GetPaymentPlanFreeFormatV2, CreateInvoiceCmpExpense, IterateRevisionPaymentPlan, InsertDiscDecCalcAmounts, AttorneyExpenseApprovalAccounting, ListKrmContractIntegration, GetContractSub, PrepaidDecomposedExpenseAccounting, ContractReActivatePaymentPlanAccounting, CheckParamCodeValidity, PrepaidExpenseAccounting, AccountAccCompExpInvoice, IteratePaymentPlan isimli 15 metot Beyin Metot kod kusuru içermektedir.



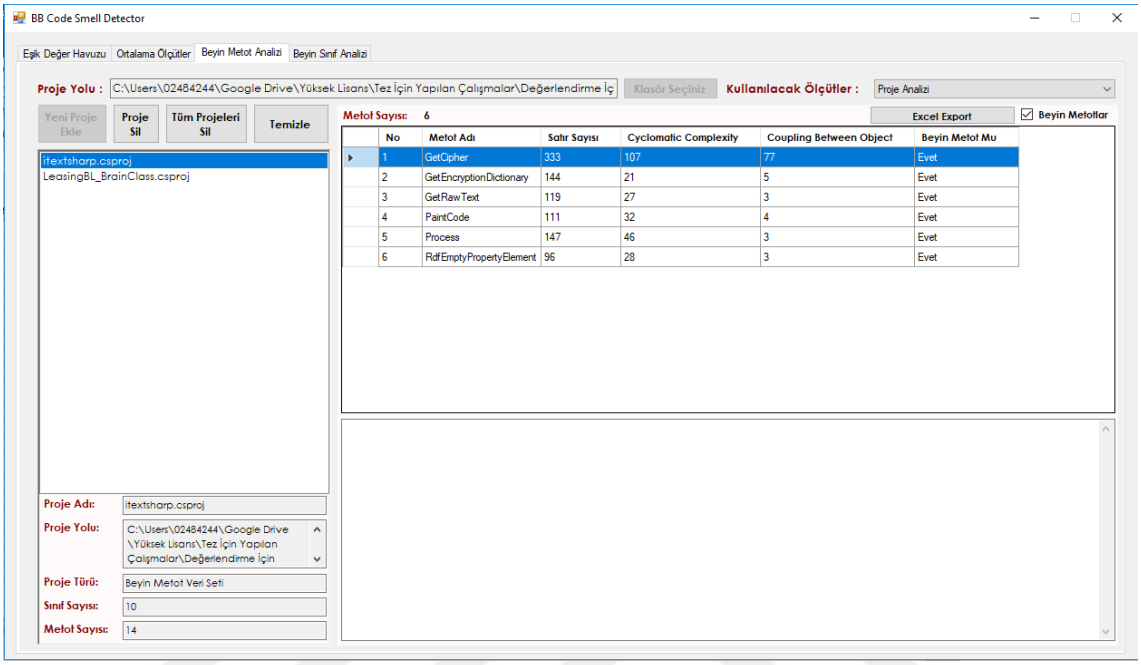
Şekil 3.10 FineLease.LeasingBL - Beyin metot kod kusuru içeren metotlar (BB-CSD)

Şekil 3.11'de FineLease.LeasingBL projesinin Java versiyonu ile analiz edilen iPlasma çıktısı görüntülenmektedir. iPlasmaya göre FineLease.LeasingBL projesinde CreateGuaranteeFile, ContractSaleLeaseback, CreateAccountAndAccTransaction, CheckParamCodeValidity, SetAccTransactionAmounts, GetParametricDescStr, ContractReActivatePaymentPlanAccounting isimli 7 metot Beyin Metot kod kusurunu içermektedir.



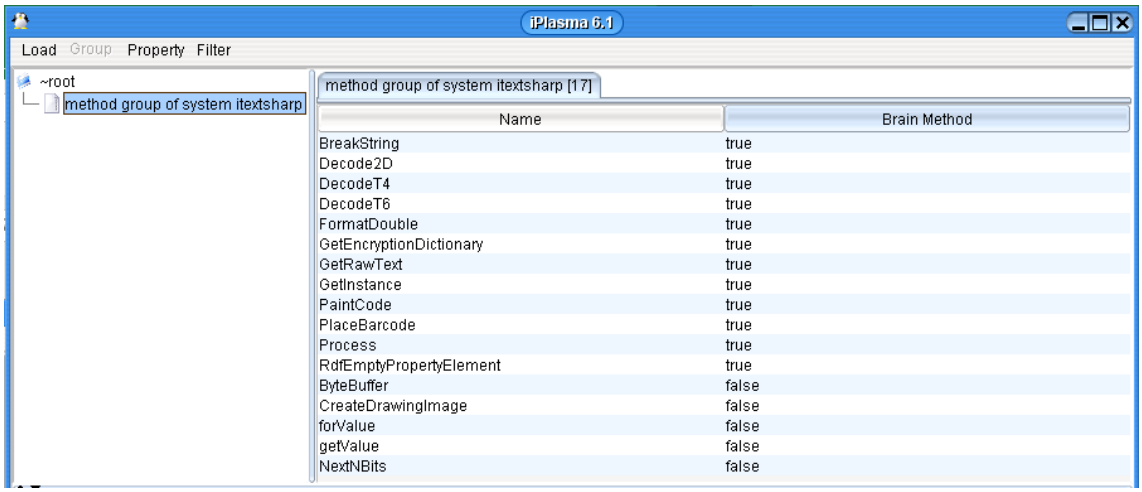
Şekil 3.11 FineLease.LeasingBL - Beyin metot kod kusuru içeren metotlar (IPlasma)

Şekil 3.12’de iTextSharp projesinin analizi sonucunda BB Code Smell Detector’un bulduğu sonuçlar görüntülenmektedir. BB Code Smell Detector’e göre iTextSharp projesinde GetEncryptionDictionary, GetCipher, GetRawText, PaintCode, Process, RdfEmptyPropertyElement isimli 6 metot Beyin Metot kod kusuru içermektedir.



Şekil 3.12 iTextSharp - Beyin metot kod kusuru içeren metotlar (BB-CSD)

Şekil 3.13’te iTextSharp projesinin Java versiyonu ile analiz edilen iPlasma çıktısı görüntülenmektedir. iPlasmaya göre iTextSharp projesinde BreakString, Decode2D, DecodeT4, DecodeT6, FormatDouble, GetEncryptionDictionary, GetInstance, GetRawText, PaintCode, PlaceBarcode, Process, RdfEmptyPropertyElement isimli 12 metot Beyin Metot kod kusurunu içermektedir.

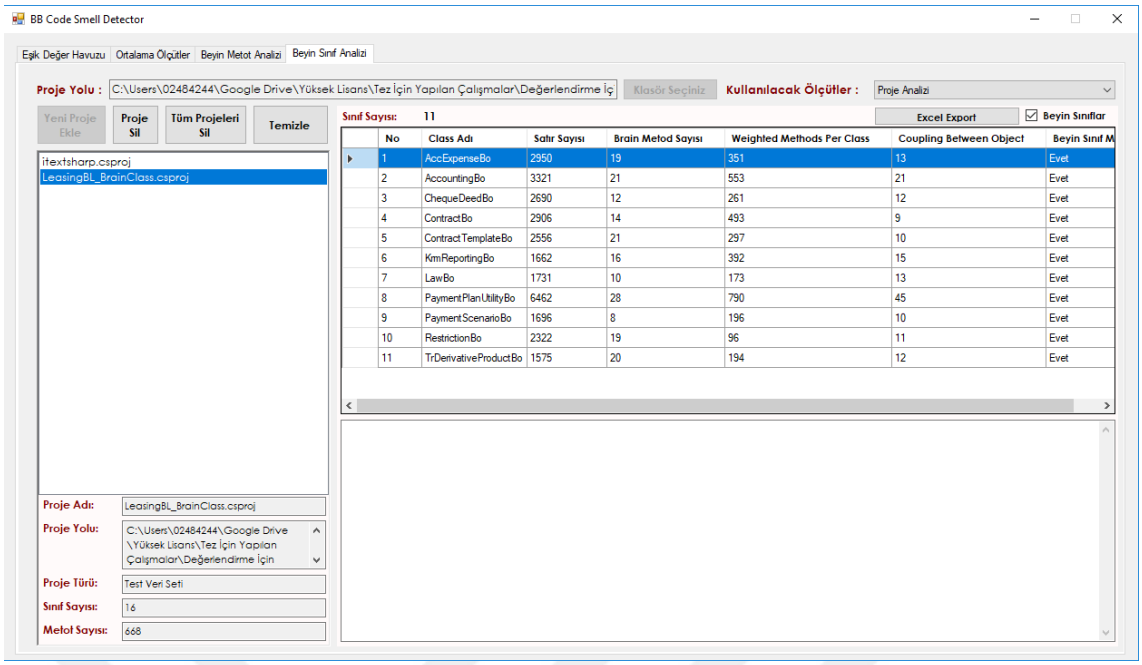


Şekil 3.13 iTextSharp - Beyin metot kod kusuru içeren metotlar (IPlasma)

Beyin Metot analizinde eğitim veri setinde elde edilen ölçütler kullanılarak, metotları Beyin Metot filtresinden geçirip, Beyin Metot kod kusuru içeren metotların son kullanıcıya gösterilmesi sağlanmaktadır. Şekil A-5.1'de ayrıntılandırıldığı gibi metot listesinde bulunan her bir metot için tespit skoruna; satır sayısı belirlenen ölçütlerin üst limitini aşanlar için +1, CYCLCO ölçütünün üst limitini aşanlar için +1, MaxNestingLevel metriğinin üst limitini aşanlar için +1, NOAV metriğinin üst limitini aşanlar için +1, CBO metriğinin üst limitini aşanlar için +1 eklenerek, toplam tespit skoru 5 olanların Beyin Metot olarak sınıflandırılması sağlanmıştır. Anlatılan yöntemin şeması 1. bölümde Şekil 1.3'te verilmiştir.

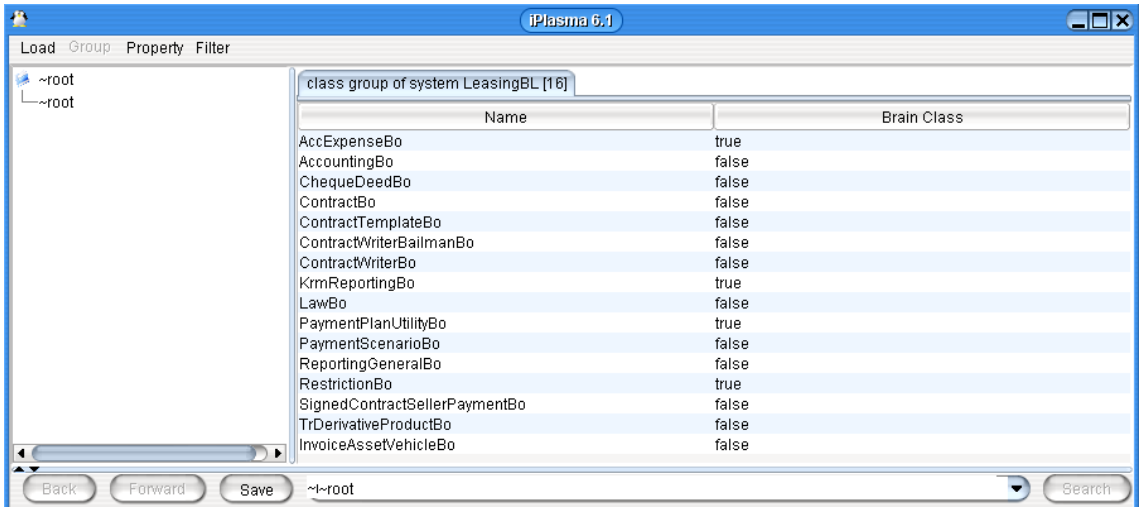
3.2.2 Beyin Sınıf Model Detayı

Geliştirilen araç ile eğitim verileri ölçütlendirilerek elde edilen eşik değerler kullanılarak analiz edilecek kodların Beyin Sınıf kod kusurunu içerip içermediğinin bulunması hedeflenmektedir. Aşağıdaki şekillerde analiz sayfasında FineLease.LeasingBL ve iTextSharp projelerine ait Beyin Sınıf kod kusuru içeren sınıfların listesi gösterilmektedir. Şekil 3.14'te FineLease.LeasingBL projesinin analizi sonucunda BB Code Smell Detector'un çıktısı görüntülenmektedir. BB Code Smell Detector'e göre FineLease.LeasingBL projesinde AccExpenseBo, AccountingBo, ChequeDeedBo, ContractBo, ContractTemplateBo, KrmReportingBo, LawBo, PaymentPlanUtilityBo, PaymentScenarioBo, RestrictionBo, TrDerivativeProductBo isimli 11 sınıf Beyin Sınıf kod kusuru içermektedir.



Şekil 3.14 FineLease.LeasingBL - Beyin sınıf kod kusuru içeren sınıflar (BB-CSD)

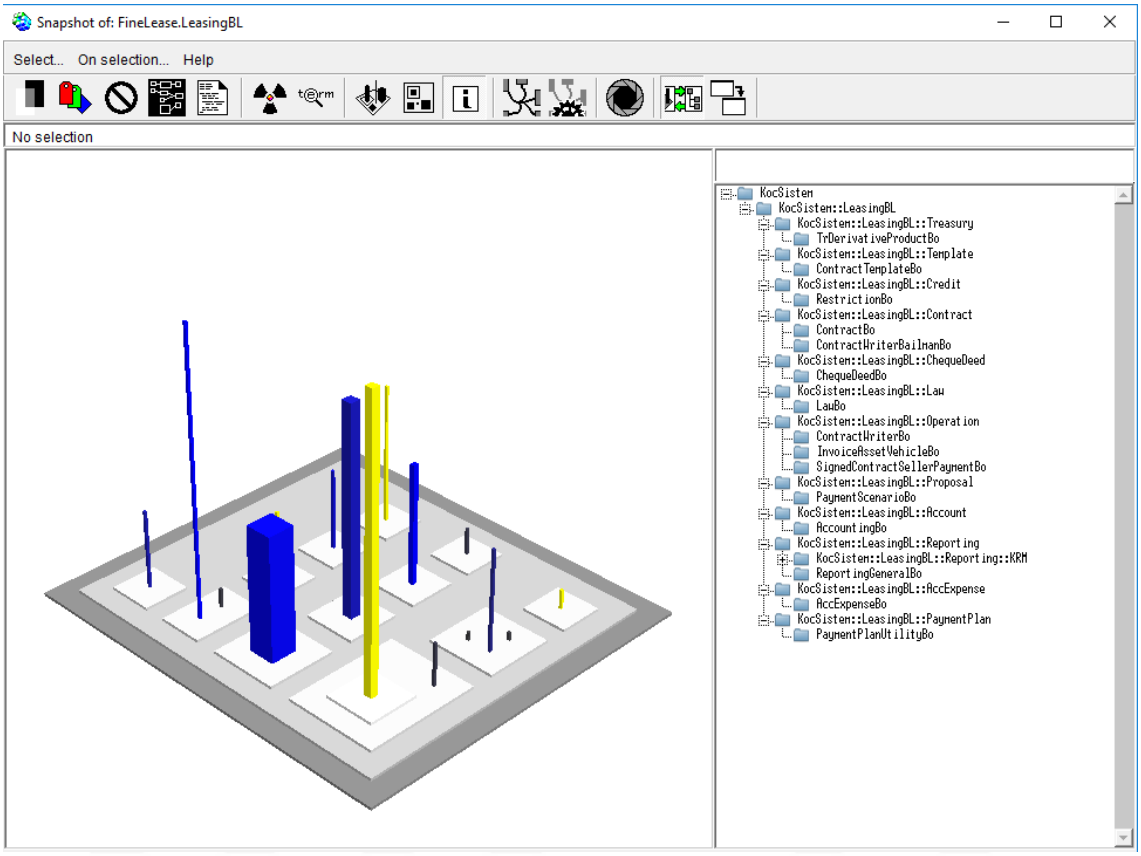
Şekil 3.15'de FineLease.LeasingBL projesinin Java versiyonu ile analiz edilen iPlasma çıktısı görüntülenmektedir. iPlasma'ya göre FineLease.LeasingBL projesinde AccExpenseBo, KrmReportingBo, PaymentPlanUtilityBo, RestrictionBo isimli 4 sınıf Beyin Sınıf kod kusurunu içermektedir.



Şekil 3.15 FineLease.LeasingBL - Beyin sınıf kod kusuru içeren sınıflar (IPlasma)

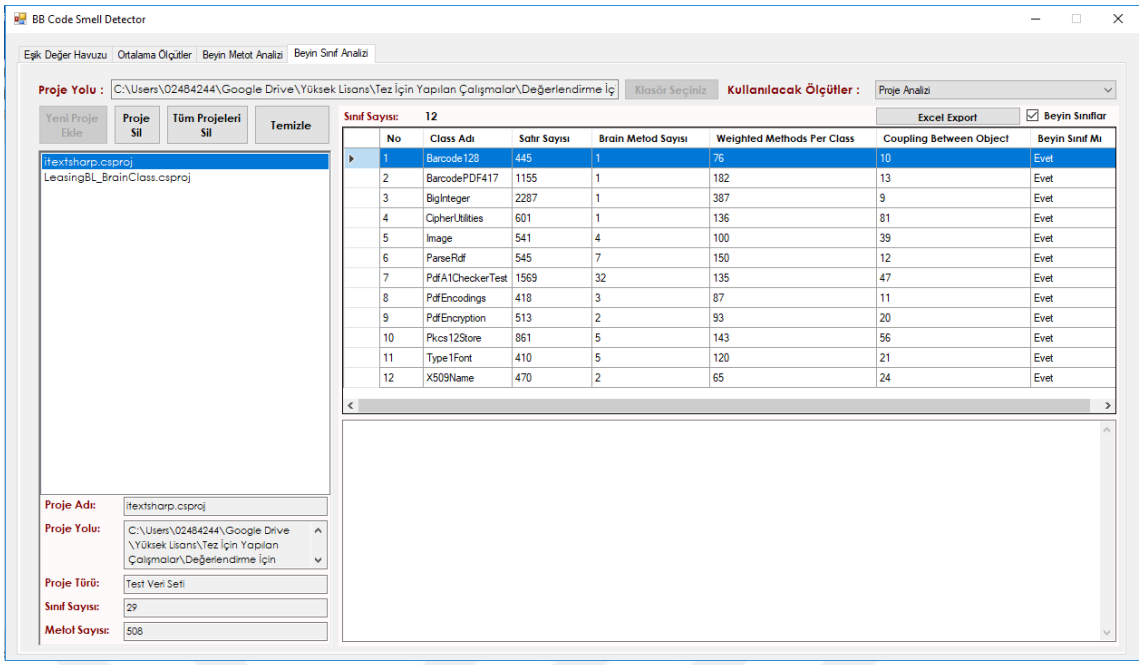
Şekil 3.16'da FineLease.LeasingBL projesinin Java versiyonu ile analiz edilen CodeCity çıktısı görüntülenmektedir. Bu görüntüde sarı ile gösterilen sınıflar Beyin Sınıf olarak sınıflandırılmışlardır. CodeCity'e göre FineLease.LeasingBL projesinde AccExpenseBo,

KrmReportingBo, LawBo, PaymentPlanUtilityBo isimli 4 sınıf beyin sınıf kod kusurunu içermektedir.



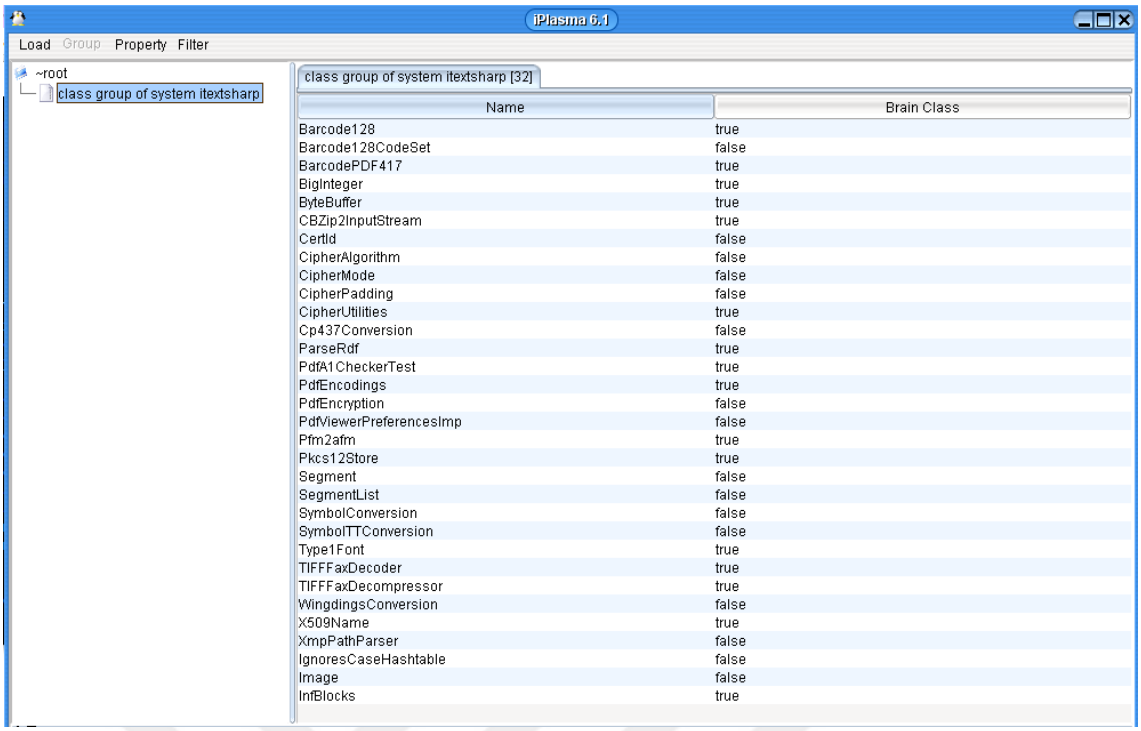
Şekil 3.16 FineLease.LeasingBL - Beyin sınıf kod kusuru içeren sınıflar (CodeCity)

Şekil 3.17'de iTextSharp projesinin analizi sonucunda BB Code Smell Detector'un çıktısı görüntülenmektedir. BB Code Smell Detector'e göre iTextSharp projesinde Barcode128, BarcodePDF417, BigInteger, CipherUtilities, Image, ParseRdf, PdfA1CheckerTest, PdfEncodings, PdfEncryption, Pkcs12Store, Type1Font, X509Name isimli 12 sınıf Beyin Sınıf kod kusuru içermektedir.



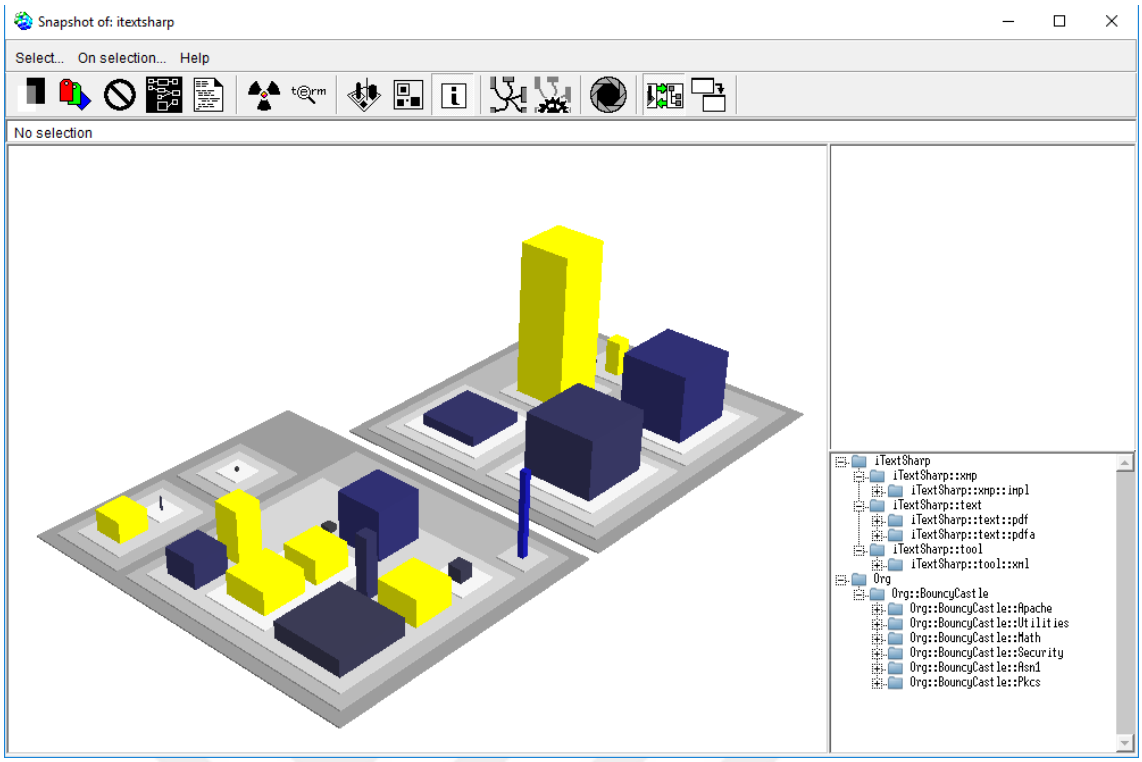
Şekil 3.17 iTextSharp - Beyin sınıf kod kusuru içeren sınıflar (BB-CSD)

Şekil 3.18'de iTextSharp projesinin Java versiyonu ile analiz edilen iPlasma çıktısı görüntülenmektedir. iPlasma'ya göre iTextSharp projesinde Barcode128, BarcodePDF417, BigInteger, ByteBuffer, CBZip2InputStream, CipherUtilities, InfBlocks, ParseRdf, PdfA1CheckerTest, PdfEncodings, Pfm2afm, Pkcs12Store, TIFFFaxDecoder, TIFFFaxDecompressor, Type1Font, X509Name isimli 16 sınıf Beyin Sınıf kod kusurunu içermektedir.



Şekil 3.18 iTextSharp - Beyin sınıf kod kusuru içeren sınıflar (IPlasma)

Şekil 3.19'da iTextSharp projesinin Java versiyonu ile analiz edilen CodeCity çıktısı görüntülenmektedir. Bu görüntüde sarı ile gösterilen sınıflar Beyin Sınıf olarak sınıflandırılmışlardır. CodeCity'e göre iTextSharp projesinde BarcodePDF417, BigInteger, ParseRdf, Pkcs12Store, TIFFFaxDecoder, TIFFFaxDecompressor, Type1Font isimli 7 sınıf Beyin Sınıf kod kusurunu içermektedir.



Şekil 3.19 iTextSharp - Beyin sınıf kod kusuru içeren sınıflar (CodeCity)

Beyin Sınıf analiz aşamasında, eğitim veri setinde elde ettiğimiz ölçüm sonuçlarını kullanarak sınıfları Beyin Sınıf filtresinden geçirip, Beyin Sınıf kod kusuru içeren sınıfların son kullanıcıya gösterilmesi sağlanmaktadır. Şekil A-6.1'de ayrıntılandırıldığı gibi her bir sınıf için tespit skoruna; sınıf içinde birden fazla beyin metot varsa ve sınıfın satır sayısı eşik değerlerdeki satır sayısının üst limitinden yüksek olanlar için +1 ya da sınıf içinde sadece bir beyin metot varsa ve sınıfın satır sayısı eşik değerlerdeki satır sayısının üst limitinin 2 katından yüksekse ve WeightedMethodsPerClass değeri eşik değerlerdeki WMC değerinin üst limitinin 2 katından yüksek olanlar için +1, Beyin Metot sayısına bakılmaksızın WeightedMethodsPerClass (WMC) değeri eşik değerlerdeki WeightedMethodsPerClass (WMC)'un üst limitinden yüksekse ve CouplingBetweenObject (CBO) değeri eşik değerlerdeki CouplingBetweenObject (CBO)'in üst limitinden yüksekse tespit skoruna +1 eklenerek, toplam tespit skoru 2 olanların Beyin Metot olarak sınıflandırılması sağlanmıştır. Anlatılan yöntemin şeması 1. bölümde Şekil 1.4'te verilmiştir.

BÖLÜM 4

BULGULAR

Kod kusuru tespit sonuçlarını aynı kod kusuru türlerinde çalışma yapmış olan uygulamalar (BB CSD, IPlasma, CodeCity) karşılaştırılarak sonuç çıktısı oluşturulmuştur. Analiz aşamasında geliştirilen araç ile diğer araçların sonuçlarının doğruluğunun karşılaştırılabilmesi adına açık kaynak kodlu iTextSharp projesi de analiz edilmiştir. Uzmanların görüşlerinin ortalaması alınarak sonuçlara karar verilmiş ve doğruluk oranları hesaplanmıştır.

4.1 Diğer Modeller ile Karşılaştırma

BB Code Smell Detector uygulaması ile kod kusurlarını bulmak için eşik değerlerin dinamik bir şekilde oluşturulması sağlanarak, kod kusurlarının bulunmasının otomatize edilmesi hedeflenmiştir. Buna ek olarak istenen eşik değerlerle analiz yapılabilmesi için kullanıcıların eşik değerleri sisteme manuel olarak girebilmesine imkan tanınmıştır.

Çizelge 4.1'de BB Code Smell Detector uygulaması ile önceki çalışmalar karşılaştırılmıştır. Çizelge 4.2'de ise FineLease.LeasingBL projesinde Beyin Metot analizi yapılan 24, çizelge 4.3'de ise iTextSharp projesinde Beyin Metot analizi yapılan 14 metot gösterilmektedir. Bu metotların Beyin Metot kod kusuru içerip içermedikleri alanında uzman 3 yazılımcının çoğunluk oylaması sonucunda alınan karara göre belirlenmiştir. Elde edilen değerler Çizelge 4.2 ve Çizelge 4.3'de Beyin Metot kolonu altında gösterilmektedir. Bu değerlere karşılık BB-CSD ve IPlasma uygulamalarının bu metotlar için verdikleri kod kusuru değerlerinin çıktıları, ayrı sütunlar altında detaylandırılmıştır. Verilen sonuçlarda hatalı tespit edilenler kalın ve italik yazılmıştır.

Çizelge 4.1 Önceki çalışmalar ile karşılaştırma

Özellikler	BB – CSD	IPlasma	Malhotra ve Pritam	Munro	JDeodorant	InCode	CodeCity
Dinamik alt ve üst limit	Var	Yok	Yok	Yok	Yok	Yok	Yok
Bulduğu Kod Kusuru Sayısı	2	10	13	2	4	2	8
C# desteği	Var	Yok	Yok	Yok	Yok	Yok	Yok
Otomatik Refaktoring	Yok	Yok	Yok	Yok	Var	NA	Yok
Koda Entegrasyon	Yok	Yok	Yok	Yok	Var	Var	Yok

Çizelge 4.2 FineLease.LeasingBL projesi beyin metot kod kusuru içeren metotlar

Metot Adı	**BB-CSD**	**IPlasma**	**Beyin Metot**
AccountAccCompExpInvoice	Evet	***Hayır***	Evet
AccountAllAccCompExpInvoice	Evet	***Hayır***	Evet
AdvanceExpenseReeskontAccounting	Evet	***Hayır***	Evet
AttorneyExpenseApprovalAccounting	***Evet***	Hayır	Hayır
CheckParamCodeValidity	Evet	Evet	Evet
ContractReActivatePaymentPlanAccounting	Evet	Evet	Evet
ContractRenterTransfer	Evet	***Hayır***	Evet
ContractSaleLeaseback	Evet	Evet	Evet

Çizelge 4.2 FineLease.LeasingBL projesi beyin metot kod kusuru içeren metotlar (devamı)

CreateAccountAndAccTransaction	***Hayır***	Evet	Evet
CreateCreditBalancedPayment	Hayır	Hayır	Hayır
CreateGuaranteeFile	***Hayır***	Evet	Evet
CreateInvoiceAdditionalCost	Evet	***Hayır***	Evet
CreateInvoiceCmpExpense	Evet	***Hayır***	Evet
GetContractSub	***Evet***	Hayır	Hayır
GetParametricDescStr	Hayır	***Evet***	Hayır
GetPaymentPlanFreeFormatV2	Evet	***Hayır***	Evet
InsertDiscDecCalcAmounts	Hayır	Hayır	Hayır
IteratePaymentPlan	***Hayır***	***Hayır***	Evet
IterateRevisionPaymentPlan	***Hayır***	***Hayır***	Evet
ListKrmContractIntegration	***Evet***	Hayır	Hayır
PrepaidDecomposedExpenseAccounting	Evet	***Hayır***	Evet
PrepaidExpenseAccounting	Evet	***Hayır***	Evet
SaveContributionAmounts	Hayır	Hayır	Hayır
SetAccTransactionAmounts	***Hayır***	Evet	Evet

Çizelge 4.3 iTextSharp projesi beyin metot kod kusuru içeren metotlar

Metot Adı	BB-CSD	IPlasma	Beyin Metot
BreakString	Hayır	***Evet***	Hayır
Decode2D	***Hayır***	Evet	Evet
DecodeT4	***Hayır***	Evet	Evet
DecodeT6	***Hayır***	Evet	Evet
FormatDouble	Hayır	***Evet***	Hayır
GetCipher	Evet	***Hayır***	Evet
GetEncryptionDictionary	Evet	Evet	Evet
GetInstance	***Hayır***	Evet	Evet
GetRawText	Evet	Evet	Evet
NextNBits	Hayır	Hayır	Hayır
PaintCode	Evet	Evet	Evet
PlaceBarcode	Hayır	***Evet***	Hayır
Process	Evet	Evet	Evet
RdfEmptyPropertyElement	Evet	Evet	Evet

Çizelge 4.4'te BB Code Smell Detector ve iPlasma araçlarının FineLease.LeasingBL ve iTextSharp projelerindeki kod kusuru tespit sonuçları gösterilmiştir. Bu sonuçlara göre

BB Code Smell Detector aracı FineLease.LeasingBL projesinde Beyin Metot kod kusuru bulmak için 24 metot üzerinde analiz edildiğinde 16 doğru 8 yanlış, iTextSharp projesinde ise 14 metot üzerinde 10 doğru 4 yanlış tespitte bulunmuştur. iPlasma aracı ise FineLease.LeasingBL projesinde Beyin Metot kod kusuru bulmak için 24 metot üzerinde analiz edildiğinde 12 doğru 12 yanlış, iTextSharp projesinde ise 14 metot üzerinde 10 doğru 4 yanlış tespitte bulunmuştur.

Çizelge 4.4 Beyin metot kod kusuru bulma doğruluk (accuracy) sonuçları

Araç	**Proje Adı**		**Pozitif**	**Negatif**
BB-CSD	FineLease.LeasingBL	Doğru	12	4
		Yanlış	3	5
	iTextSharp	Doğru	6	4
		Yanlış	0	4
iPlasma	FineLease.LeasingBL	Doğru	6	6
		Yanlış	1	11
	iTextSharp	Doğru	9	1
		Yanlış	3	1

Çizelge 4.5'de FineLease.LeasingBL projesinde Beyin Sınıf analizi yapılan 16, Çizelge 4.6'da ise iTextSharp projesinde Beyin Sınıf analizi yapılan 29 sınıfın adı gösterilmektedir. Bu sınıfların Beyin Sınıf kod kusuru içerip içermedikleri alanında uzman 3 yazılımcının çoğunluk oylaması sonucunda alınan karara göre belirlenmiştir. Elde edilen değerler Çizelge 4.5 ve Çizelge 4.6'da Beyin Sınıf kolonu altında gösterilmektedir. Bu değerlere karşılık BB-CSD, IPlasma ve CodeCity uygulamalarının bu sınıflar için verdikleri kod kusuru değerlerinin çıktıları, ayrı sütunlar altında detaylandırılmıştır.

Çizelge 4.5 FineLease.LeasingBL projesi beyin sınıf kod kusuru içeren sınıflar

Sınıf Adı	**BB-CSD**	**iPlasma**	**Codecity**	**Beyin Sınıf**
AccExpenseBo	Evet	Evet	Evet	Evet
AccountingBo	Evet	***Hayır***	***Hayır***	Evet
ChequeDeedBo	***Evet***	Hayır	Hayır	Hayır
ContractBo	Evet	***Hayır***	***Hayır***	Evet
ContractTemplateBo	***Evet***	Hayır	Hayır	Hayır
ContractWriterBailmanBo	Hayır	Hayır	Hayır	Hayır
ContractWriterBo	Hayır	Hayır	Hayır	Hayır
InvoiceAssetVehicleBo	Hayır	Hayır	Hayır	Hayır
KrmReportingBo	Evet	Evet	Evet	Evet
LawBo	Evet	***Hayır***	Evet	Evet
PaymentPlanUtilityBo	Evet	Evet	Evet	Evet
PaymentScenarioBo	Evet	***Hayır***	***Hayır***	Evet
ReportingGeneralBo	Hayır	Hayır	Hayır	Hayır
RestrictionBo	Evet	Evet	***Hayır***	Evet
SignedContractSellerPaymentBo	Hayır	Hayır	Hayır	Hayır
TrDerivativeProductBo	***Evet***	Hayır	Hayır	Hayır

Çizelge 4.6 iTextSharp projesi beyin sınıf kod kusuru içeren sınıflar

Sınıf Adı	BB-CSD	iPlasma	Codecity	Beyin Sınıf
Barcode128	Evet	Evet	***Hayır***	Evet
BarcodePDF417	Evet	Evet	Evet	Evet
BigInteger	Evet	Evet	Evet	Evet
ByteBuffer	Hayır	***Evet***	Hayır	Hayır
CBZip2InputStream	***Hayır***	Evet	***Hayır***	Evet
CertId	Hayır	Hayır	Hayır	Hayır
CipherUtilities	***Evet***	***Evet***	Hayır	Hayır
Cp437Conversion	Hayır	Hayır	Hayır	Hayır
IgnoresCaseHashtable	Hayır	Hayır	Hayır	Hayır
Image	Evet	***Hayır***	***Hayır***	Evet
InfBlocks	***Hayır***	Evet	***Hayır***	Evet
ParseRdf	Evet	Evet	Evet	Evet
PathPosition	Hayır	Hayır	Hayır	Hayır
PdfA1CheckerTest	Evet	Evet	***Hayır***	Evet
PdfEncodings	Evet	Evet	***Hayır***	Evet
PdfEncryption	Evet	***Hayır***	***Hayır***	Evet
PdfViewerPreferencesImp	Hayır	Hayır	Hayır	Hayır
Pfm2afm	Hayır	***Evet***	Hayır	Hayır

Çizelge 4.6 iTextSharp projesi beyin sınıf kod kusuru içeren sınıflar (devamı)

Pkcs12Store	Evet	Evet	Evet	Evet
Segment	Hayır	Hayır	Hayır	Hayır
SegmentList	Hayır	Hayır	Hayır	Hayır
SymbolConversion	Hayır	Hayır	Hayır	Hayır
SymbolTTConversion	Hayır	Hayır	Hayır	Hayır
TIFFFaxDecoder	***Hayır***	Evet	Evet	Evet
TIFFFaxDecompressor	***Hayır***	Evet	Evet	Evet
Type1Font	Evet	Evet	Evet	Evet
WingdingsConversion	Hayır	Hayır	Hayır	Hayır
X509Name	***Evet***	***Evet***	Hayır	Hayır
XmpPathParser	Hayır	Hayır	Hayır	Hayır

Çizelge 4.7'de BB Code Smell Detector, iPlasma ve CodeCity araçlarının FineLease.LeasingBL ve iTextSharp projelerindeki Beyin Sınıf kod kusuru bulma sonuçları gösterilmiştir. Bu sonuçlara göre BB Code Smell Detector aracı FineLease.LeasingBL projesinde Beyin Sınıf kod kusuru bulmak için 16 sınıf üzerinde analiz edildiğinde 13 doğru 3 yanlış, iTextSharp projesinde ise 29 sınıf üzerinde 23 doğru 6 yanlış tespitte bulunmuştur. iPlasma aracı FineLease.LeasingBL projesinde Beyin Sınıf kod kusuru bulmak için 16 sınıf üzerinde analiz edildiğinde 12 doğru 4 yanlış, iTextSharp projesinde ise 29 metot üzerinde 23 doğru 6 yanlış tespitte bulunmuştur. CodeCity aracı ise FineLease.LeasingBL projesinde Beyin Sınıf kod kusuru bulmak için 16 sınıf üzerinde analiz edildiğinde 12 doğru 4 yanlış, iTextSharp projesinde ise 29 metot üzerinde 22 doğru 7 yanlış tespitte bulunmuştur.

Çizelge 4.7 Beyin metot kod kusuru bulma doğruluk (accuracy) sonuçları

Araç	Proje Adı		Pozitif	Negatif
BB-CSD	FineLease.LeasingBL	Doğru	8	5
		Yanlış	3	0
	iTextSharp	Doğru	10	13
		Yanlış	2	4
iPlasma	FineLease.LeasingBL	Doğru	4	8
		Yanlış	0	4
	iTextSharp	Doğru	12	11
		Yanlış	4	2
CodeCity	FineLease.LeasingBL	Doğru	4	8
		Yanlış	0	4
	iTextSharp	Doğru	7	15
		Yanlış	0	7

BB Code Smell Detector, Iplasma ve CodeCity uygulamaları karşılaştırılmıştır. Beyin Metot kod kusuru CodeCity aracı tarafından desteklenmediğinden, BB-CSD ve Iplasma araçları üzerinde denenmiştir. Aynı veriler üzerinde test edildiklerinde BB-CSD aracı FineLease.LeasingBL projesinde 24 metot içinden 15'ini, iTextSharp projesinde ise 14 metot içinden 6 tanesini kusurlu olarak sınıflandırırken, iPlasma aracı FineLease.LeasingBL projesinde 24 metot içinden 7'sini, iTextSharp projesinde ise içinden 14 metot içinden 12 tanesini kusurlu olarak sınıflandırmışlardır. Beyin Sınıf kod kusuru BB-CSD, iPlasma ve CodeCity üzerinde denenmiştir. Sonuçlara göre BB-CSD aracı FineLease.LeasingBL projesinde 16 sınıf içinden 13'ünü, iTextSharp projesinde ise 29 sınıftan 23'ünü kusurlu olarak sınıflandırırken, iPlasma aracı FineLease.LeasingBL projesinde 16 sınıf içinden 12'sini, iTextSharp projesinde ise 29 sınıftan 23'sini kusurlu olarak sınıflandırırken, CodeCity aracı FineLease.LeasingBL projesinde 16 sınıf içinden 12'sini, iTextSharp projesinde ise 29 sınıftan 22'sini kusurlu olarak sınıflandırmıştır.

Çizelge 4.8'de araçların doğruluk (accuracy) oranlarının karşılaştırma sonuçları, Çizelge 4.9'da araçların hassasiyet (precision) oranlarının karşılaştırma sonuçları, Çizelge 4.10'da araçların duyarlılık (sensitivity) oranlarının karşılaştırma sonuçları, Çizelge 4.11'de araçların özgüllük (specificity) oranlarının karşılaştırma sonuçları, Çizelge 4.12'de ise araçların f1 skor (f-one score) oranlarının karşılaştırma sonuçları gösterilmektedir. Çizelgelerde altı çizili ve kalın olarak belirtilen değerler en iyi sonuçları göstermektedir.

Çizelge 4.8 (BB-CSD/IPlasma/CodeCity) doğruluk (accuracy) oranlarının karşılaştırılması

Kod Kusuru	**Proje Adı**	**BB–CSD**	**iPlasma**	**CodeCity**
Beyin Metot	Leasing Projesi	**66.67%** (16/24)	50.00% (12/24)	Desteklenmiyor
	iTextSharp	**71.43%** (10/14)	71.43% (10/14)	Desteklenmiyor
Beyin Sınıf	Leasing Projesi	**81.25%** (13/16)	75.00% (12/16)	75.00% (12/16)
	iTextSharp	**79.31%** (23/29)	**79.31%** (23/29)	75.86% (22/29)

Çizelge 4.9 (BB-CSD/IPlasma/CodeCity) hassasiyet (precision) oranlarının karşılaştırılması

Kod Kusuru	**Proje Adı**	**BB–CSD**	**iPlasma**	**CodeCity**
Beyin Metot	Leasing Projesi	80.00% (12/15)	**85.71%** (6/7)	Desteklenmiyor
	iTextSharp	**100.00%** (6/6)	75.00% (9/12)	Desteklenmiyor
Beyin Sınıf	Leasing Projesi	72.73% (8/11)	**100.00%** (4/4)	**100.00%** (4/4)
	iTextSharp	83.33% (10/12)	75.00% (12/16)	**100.00%** (7/7)

Çizelge 4.10 (BB-CSD/IPlasma/CodeCity) duyarlılık (sensitivity) oranlarının karşılaştırılması

Kod Kusuru	Proje Adı	BB–CSD	iPlasma	CodeCity
Beyin Metot	Leasing Projesi	**70.59%** (12/17)	35.29% (6/17)	Desteklenmiyor
	iTextSharp	60.00% (6/10)	**90.00%** (9/10)	Desteklenmiyor
Beyin Sınıf	Leasing Projesi	**100.00%** (8/8)	50.00% (4/8)	50.00% (4/8)
	iTextSharp	71.43% (10/14)	**85.71%** (12/14)	50.00% (7/14)

Çizelge 4.11 (BB-CSD/IPlasma/CodeCity) özgüllük (specificity) oranlarının karşılaştırılması

Kod Kusuru	Proje Adı	BB–CSD	iPlasma	CodeCity
Beyin Metot	Leasing Projesi	57.14% (4/7)	**85.71%** (6/7)	Desteklenmiyor
	iTextSharp	**100.00%** (4/4)	25.00% (1/4)	Desteklenmiyor
Beyin Sınıf	Leasing Projesi	62.50% (5/8)	**100.00%** (8/8)	**100.00%** (8/8)
	iTextSharp	86.67% (13/15)	73.33% (11/15)	**100.00%** (15/15)

Çizelge 4.12 (BB-CSD/IPlasma/CodeCity) f1-skor (f-one score) oranlarının karşılaştırılması

Kod Kusuru	**Proje Adı**	**BB–CSD**	**iPlasma**	**CodeCity**
Beyin Metot	Leasing Projesi	**75.00%**	50.00%	Desteklenmiyor
	iTextSharp	75.00%	**81.82%**	Desteklenmiyor
Beyin Sınıf	Leasing Projesi	**84.21%**	66.67%	66.67%
	iTextSharp	76.92%	**80.00%**	66.67%

BÖLÜM 5

SONUÇ VE ÖNERİLER

BB Code Smell Detector uygulaması ile kod kusurlarını bulmak için limitlerin dinamik bir şekilde oluşturulması ya da manuel olarak girilmesi sağlanarak, kod kusurlarının bulunmasının otomatize edilmesi hedeflenmiştir. Yapılan gerçeklemenin avantajları aşağıdaki gibidir:

- Kod kusurları için alt ve üst limitlerin belirlenmesi otomatize edilebilmektedir. Böylece firmalar kendi alt ve üst limitlerini, kalitesine güvendikleri uygulamalar ile oluşturabileceklerdir.
- Kod kusurları için alt ve üst limitleri kullanıcılar kendileri sisteme girebilmektedirler. Böylece istedikleri değerler ile analizlerini yapabileceklerdir.
- Beyin metot ve beyin sınıf kod kusuru hesaplanırken, diğer çalışmalardan farklı olarak sınıflar arası ilişki (CBO) değerleri hesaplama kriterine dahil edilmiştir.

Sonuç olarak beyin metot ve beyin sınıf kod kusurlarını otomatik olarak tespit edebilen bir araç elde edilmiştir. Bölüm 4'te incelendiği üzere geliştirilen aracın tespit sonuçları diğer araçlarla karşılaştırıldığında, sonuçlar diğer araçlarla ya aynı ya da diğer araçlardan daha iyidir.

BB Code Smell Detector, yeni özellikler eklenerek ve var olan özellikler geliştirilerek yazılım geliştiriciler için daha kullanışlı bir şekilde sunulabilir. Aşağıda geliştirilebilecek ya da eklenebilecek özellikler listelenmiştir:

- Yeni kod kusurlarını belirleyecek şekilde genişletilebilir.

- Eklenti olarak sunulabilir.
- Uygulamaya entegre çalışarak yapılan hatalar anlık olarak yazılım geliştiriciye sunulabilir.

KAYNAKLAR

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke and D. Roberts, “Refactoring: Improving the Design of Existing Code”, Addison-Wesley, (1999).

[2] Mens, T. & Tourwe´, T., " A Survey of Software Refactoring" , IEEE Transactions on Software Engineering, 30(2), 126-139 (2004).

[3] M. Lanza and R. Marinescu, Object-Oriented Metrics in Practice, Springer, 2006.

[4] http://www.codeproject.com/Articles/408663/Using-NRefactory-for-analyzingCsharp-code, 21 Şubat 2019.

[5] A. Koenig, “Patterns and antipatterns,” Journal of Object-Oriented Programming, 8(1), pp. 46-48, 1995.

[6] Melih Altıntas & Ebru Akçapınar Sezer “Kaynak Kodlardaki Kötü Kokuların Otomatik Tespiti için Eclipse Eklenti Önerisi”, Hacettepe Üniversitesi, UGES - Aselsan, Ankara, (2018).

[7] Almas Hamid, Muhammad Ilyas, Muhammad Hummayun and Asad Nawaz, “A Comparative Study on Code Smell Detection Tools”, International Journal of Advanced Science and Technology, (2013), pp.25-32

[8] Sevilay Velioğlu, Yunus Emre Selçuk “An Automated Code Smell and Anti-Pattern Detection Approach”, Computer Engineering, Yıldız Technical University (2016)

[9] Steffen M. Olbrich, Daniela S. Cruzes, Dag I.K. Sjøberg, “Are all Code Smells Harmful? A Study of God Classes and Brain Classes in the Evolution of three Open Source Systems”, 26th IEEE International Conference on Software Maintenance in Timişoara, Romania 978-1-4244-8628-1/10/$26.00©2010 IEEE

[10] M. Fokaefs, N. Tsantalis and A. Chatzigeorgiou, “JDeodorant: Identification and Removal of Feature Envy Bad Smells”, IEEE International Conference on Software Maintenance, 2007 October, pp. 519-520.

[11] Francesca Arcelli Fontana, Elia Mariani, Andrea Morniroli, Raul Sormani, Alberto Tonello, “An experience report on using code smells detection tools”, University of Milano Bicocca, 2011

[12] Marinescu R.,(2004), “Detection Strategies: Metrics-Based Rules for Detecting Design Flaws,” Proc. 20th Int’l Conf. Software Maintenance, (2004), Romania.

[13] Malhotra, R., Pritam, N.," Assessment of Code Smells for Predicting Class Change Proneness", Software Quality Professional, 15(1), 33-40 (2012).

[14] Munro M.J. (2005), “Product Metrics for Automatic Identification of“Bad Smell” Design Problems in Java Source-Code,” Proc. 11th Int’l Software Metrics Symp., F. Lanubile and C. Seaman, eds., Sept. 2005, University of Strathclyde Glasgow, UK.

[15] https://roslyn.codeplex.com/ 18 Ocak 2019

[16] http://www.tangiblesoftwaresolutions.com/Product_Details/CSharp_to_Java_Converter_Details.html, 11 Şubat 2019.

[17] Richard Wettel and Michele Lanza, “CodeCity”, University of Lugano, Switzerland

EK-A

GELİŞTİRME

Yöntemde kod kokusu tespiti için projelerin ölçütleri üzerinden işlem yapılmaktadır. Bu nedenle analiz edilecek kodların anlamsal söz dizimi çıkartılır, bu anlamsal söz dizimi üzerinden ölçütler elde edilerek analiz yapılır.

A-1 Ölçüt Ayrıştırma Metodu

Şekil A-1.1'deki kod bloğunda C# kodlarının anlamsal söz dizimi (AST) verilerine erişiminin nasıl yapıldığı gösterilmektedir. Dosya ayrıştırma yöntemi Hipotez başlığı altında ayrıca detaylandırılmıştı.

```
public List<MethodInfo> GetMethodsWithMetrics(SyntaxTree classSyntaxTree, int
projectType)
{
    var methodList = new List<MethodInfo>();
    var methods = classSyntaxTree.Descendants.OfType<MethodDeclaration>();

    foreach (var method in methods)
    {
      var body = method.Body;
      var response = GetMethodDetail(body);

      var methodInfo = new MethodInfo
      {
        ClassID = 0,
        MethodName = method.Name,
        NumberOfIfElseStatement = response.NumberOfIfElseStatement,
```

```
        NumberOfWhileStatement = response.NumberOfWhileStatement,
        NumberOfDoWhileStatement = response.NumberOfDoWhileStatement,
        NumberOfForStatement = response.NumberOfForStatement,
        NumberOfForeachStatement = response.NumberOfForeachStatement,
        NumberOfSwitchStatement = response.NumberOfSwitchStatement,
        NumberOfCaseInSwitchStatement = response.NumberOfCaseInSwitchStatement,
        NumberOfVariableStatement = response.NumberOfVariableStatement,
        NumberOfAndOrStatement = response.NumberOfAndOrStatement,
        NumberOfTryCatchStatement = response.NumberOfTryCatchStatement,
        NumberOfParameter = response.NumberOfParameter,
        MaxNestingLevel = response.MaxNestingLevel,
        CyclomaticComplexity = response.CyclomaticComplexity,
        CBO = response.CBO,
        LineOfCode = method.Descendants.OfType<NewLineNode>().Where(k =>
k.NextSibling.Role != Roles.NewLine || k.PrevSibling.Role ==
Roles.Comment).Count() - method.Descendants.OfType<Comment>().Where(k =>
k.CommentType == CommentType.SingleLine || k.CommentType ==
CommentType.Documentation).Count() -
method.Descendants.OfType<Comment>().Where(k => k.CommentType !=
CommentType.SingleLine && k.CommentType != CommentType.Documentation).Sum(i =>
i.ToString().CountLinesInString()) - 1,
        MethodBody = method.Body.ToString(),
        MethodStartLine = method.StartLocation.Line,
        MethodEndLine = method.EndLocation.Line
};


        methodList.Add(methodInfo);
      }
      return methodList;
}
```

Şekil A-1 C# ölçüt ayrıştırma metodu

A-2 Metot Ölçüt Hesaplamaları

Şekil A-2.1'deki kod bloğunda C# kodlarında metotların ölçütlerine erişiminin nasıl yapıldığı gösterilmektedir.

```
private MethodInfo GetMethodDetail(BlockStatement methodBody)
{
  var methodInfo = new MethodInfo
  {
    NumberOfIfElseStatement =
methodBody.Descendants.OfType<IfElseStatement>().Count(),
    NumberOfWhileStatement =
methodBody.Descendants.OfType<WhileStatement>().Count(),
    NumberOfDoWhileStatement =
methodBody.Descendants.OfType<DoWhileStatement>().Count(),
    NumberOfForStatement =
methodBody.Descendants.OfType<ForStatement>().Count(),
    NumberOfForeachStatement =
methodBody.Descendants.OfType<ForeachStatement>().Count(),
    NumberOfSwitchStatement =
methodBody.Descendants.OfType<SwitchStatement>().Count(),
    NumberOfCaseInSwitchStatement =
methodBody.Descendants.OfType<SwitchStatement>().SelectMany(i =>
i.Descendants.Where(k => k.Role.ToString() == "case")).Distinct().Count(),
    NumberOfVariableStatement =
methodBody.Descendants.OfType<VariableDeclarationStatement>().Count(),
    NumberOfTryCatchStatement =
methodBody.Descendants.OfType<TryCatchStatement>().Count(),
    NumberOfParameter =
methodBody.Descendants.OfType<ParameterDeclaration>().Count(),
    CBO = methodBody.Descendants.OfType<SimpleType>().Select(k =>
k.ToString()).Distinct().Count()
  };

  var operators = methodBody.Descendants.OfType<Statement>();

  var andOrCounter = 0;
  foreach (var i in operators)
  {
   var k = i.Descendants.ToList();

    foreach (var item in k)
    {
     var value = item.Role.ToString();
```

```
     if (value == "||" || value == "&&")
      andOrCounter = andOrCounter + 1;
    }
  }

   andOrCounter = andOrCounter / 2;
   methodInfo.NumberOfAndOrStatement = (int)andOrCounter;

   var statementList = methodBody.Descendants.Where(i => controlList.Any(p => p
== i.GetType().Name)).ToList();

   maxNestingLevel = 0;

   foreach (var item in statementList)
   {
    int counter = 1;

    GetMaxNestingLevel(statementList.Where(i => i != item).ToList(),
item.Parent.Parent, ref counter);

    if (maxNestingLevel < counter)
    {
      maxNestingLevel = counter;
    }
   }

   methodInfo.MaxNestingLevel = maxNestingLevel;
   methodInfo.CyclomaticComplexity = CalculateCyclomaticComplexity(methodInfo);

   return methodInfo;
}
```

Şekil A-2 Metotların ölçütlerinin hesaplanmaları

A-3 Sınıf Ölçüt Hesaplamaları

Şekil A-3.1'deki kod bloğunda C# kodlarında sınıfların ölçütlerine erişiminin nasıl yapıldığı gösterilmektedir.

```
public List<ClassInfo> GetClassesWithMetrics(string projectPath, int
projectType)
{
 var classInfoList = new List<ClassInfo>();
 var classInfo = default(ClassInfo);
 var methodBo = new MethodBo();
 var fileBody = default(string);

 var classPaths = Utility.GetDirectoryFiles(projectPath, FileExtension.Class);

 var parser = new CSharpParser();
 SyntaxTree syntaxTree;
 SyntaxTree classSyntaxTree;
 var sb = new StringBuilder();


 foreach (var classPath in classPaths.Where(k => controlList.Count(i =>
Path.GetFileName(k).ToLowerInvariant().EndsWith(i) ||
Path.GetFileName(k).ToLowerInvariant().StartsWith(i)) == 0))
 {
  if (Path.GetExtension(classPath) == ".cs")
  {
   try
   {
    fileBody = GetFileBodyFormatted(classPath);
    syntaxTree = parser.Parse(fileBody, classPath);

    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));
     }
    }

   foreach (var item in syntaxTree.Descendants.OfType<TypeDeclaration>().Where(k
=> k.ClassType == ClassType.Class))
   {
    if (item.Members.Count > 0)
    {
     classInfo = new ClassInfo
     {
      ClassName = item.Name,
      ClassBody = item.ToString()
     };

    if (!string.IsNullOrEmpty(classInfo.ClassBody))
    {
     classSyntaxTree = parser.Parse(classInfo.ClassBody);

     classInfo.ClassPath = Path.GetDirectoryName(classPath);
```

```
     classInfo.NumberOfObjectVariableStatement =
classSyntaxTree.Descendants.OfType<FieldDeclaration>().Where(x =>
x.ReturnType.GetType().Name != "PrimitiveType").Count();


     classInfo.NumberOfPublicVariableStatement =
classSyntaxTree.Descendants.OfType<FieldDeclaration>().Where(x => x.Modifiers ==
Modifiers.Public).Count();

     classInfo.NumberOfPublicObjectVariableStatement =
classSyntaxTree.Descendants.OfType<FieldDeclaration>().Where(x => x.Modifiers ==
Modifiers.Public && x.ReturnType.GetType().Name != "PrimitiveType").Count();

     classInfo.NumberOfAccesorsStatement =
classSyntaxTree.Descendants.OfType<Accessor>().Count();

     classInfo.CouplingBetweenObjectList =
classSyntaxTree.Descendants.OfType<SimpleType>().Select(k =>
k.ToString()).Distinct().ToList();

     classInfo.Methods = methodBo.GetMethodsWithMetrics(classSyntaxTree,
projectType);

     classInfo.LineOfCode = item.Descendants.OfType<NewLineNode>().Where(k =>
k.NextSibling.Role != Roles.NewLine || k.PrevSibling.Role ==
Roles.Comment).Count() - item.Descendants.OfType<Comment>().Where(k =>
k.CommentType == CommentType.SingleLine || k.CommentType ==
CommentType.Documentation).Count() - item.Descendants.OfType<Comment>().Where(k
=> k.CommentType != CommentType.SingleLine && k.CommentType !=
CommentType.Documentation).Sum(i => i.ToString().CountLinesInString()) - 2 *
classInfo.MethodCount - 1;

     classInfoList = MergeClassInfoWithExistsClass(classInfoList, classInfo);
    }
  }
 }
}
catch (Exception error)
  {
   sb.AppendLine(string.Format("StackTrace: {0} \nMessage:{1}",
error.StackTrace, error.Message));
  }
 }
}

   return classInfoList;
}
```

Şekil A-3 Sınıfların ölçütlerinin hesaplanmaları

A-4 Hesaplanan Ölçütlerin Veritabanı İşlemleri

Şekil A-4.1'deki kod bloğunda hesaplanan ölçütlerin MSSQL veritabanına kayıt işlemlerinin nasıl yapıldığı gösterilmektedir.

```
public void BulkInsertProjectsAndRelatedObjects(List<ProjectInfo> projects)
{
  foreach (var item in projects)
  {
   item.ProjectName = item.ProjectName ?? string.Empty;
   item.ProjectPath = item.ProjectPath ?? string.Empty;

   foreach (var clas in item.Classes)
   {
    clas.ClassBody = clas.ClassBody ?? string.Empty;
    clas.ClassName = clas.ClassName ?? string.Empty;
    clas.ClassPath = clas.ClassPath ?? string.Empty;
   }
  }

  using (var conn = new SqlConnection(connectionString))
  {
   conn.Open();
   SqlTransaction trans = conn.BeginTransaction();

   var projectId = 0;
   var classId = 0;

   foreach (var project in projects)
   {
    projectId = conn.Query<int>(insertProjectSql, project, trans).ToList()[0];

    project.Classes.ForEach(Class =>
    {
     Class.ProjectID = projectId;
     classId = conn.Query<int>(insertClassSql, Class, trans).ToList()[0];

     if (Class.Methods != null)
     {
      conn.Execute(insertMethodSql, Class.Methods.Select(method => {
method.ClassID = classId; return method; }), trans);
     }
    });
  }
  trans.Commit();  conn.Close(); } }
```

Şekil A-4 Hesaplanan ölçütlerin veritabanına kayıt işlemleri

A-5 Beyin Metot Tespiti

Şekil A-5.1'deki kod bloğunda önerilen yöntemin nasıl gerçekleştirildiği gösterilmektedir.

```
public bool IsBrainMethod(MethodInfo methodInfo, MetricThresholdInfo
metricThresholdInfo)
{
  var detectionScore = 0;

  if (methodInfo.LineOfCode > metricThresholdInfo.MethodLineOfCode.Max)
  {
    detectionScore++;
  }

  if (methodInfo.CyclomaticComplexity >
metricThresholdInfo.WeightedMethodsPerClass.Max)
  {
    detectionScore++;
  }

  if (methodInfo.MaxNestingLevel > metricThresholdInfo.MaxNestingLevel.Max)
  {
    detectionScore++;
  }

  if (methodInfo.NumberOfVariableStatement >
metricThresholdInfo.VariableStatement.Max)
  {
    detectionScore++;
  }

  if (methodInfo.CBO > metricThresholdInfo.MethodCBO.Max)
  {
    detectionScore++;
  }

    return detectionScore == 5;
}
```

Şekil A-5 Beyin metot kod kusuru belirleme

A-6 Beyin Sınıf Tespiti

Şekil A-6.1'deki kod bloğunda önerilen yöntemin nasıl gerçekleştirildiği gösterilmektedir.

```
public bool IsBrainClass(ClassInfo classInfo, MetricThresholdInfo
metricThreshold)
{
  var detectionScore = 0;

  //Bir class içinde birden fazla beyin metot varsa ve classın satır sayısı eşik
değerlerdeki satır sayısının üst limitinden yüksekse
  if (classInfo.Methods.Count(i => i.IsBrainMethod) > 1 &&
  classInfo.LineOfCode >= metricThreshold.LineOfCode.Max)
  {
    detectionScore++;
  }
  /*
   * Bir class içinde sadece bir beyin metot varsa ve classın satır sayısı eşik
değerlerdeki satır sayısının üst limitinin 2 katından yüksekse ve
   * WeightedMethodsPerClass değeri eşik değerlerdeki WMC değerinin üst
limitinin 2 katından yüksekse
  */
  else if (classInfo.Methods.Count(i => i.IsBrainMethod) == 1 &&
  classInfo.LineOfCode >= (2 * metricThreshold.LineOfCode.Max) &&
  classInfo.WeightedMethodsPerClass >= (2 *
metricThreshold.WeightedMethodsPerClass.Max))
  {
    detectionScore++;
  }

  /*
   * Marinescu: Class is very complex and non-cohesive demiş, biz de non-
cohesive dediği için tersi durum olan coupling'i yüksekseyi kontrol ettik.

   * Beyin metot sayısına bakılmaksızın;
   * WeightedMethodsPerClass (WMC) eşik değerlerdeki WeightedMethodsPerClass
(WMC)'un üst limitinden yüksekse ve
   * CouplingBetweenObject (CBO) değeri eşik değerlerdeki CouplingBetweenObject
(CBO)'in üst limitinden yüksekse
  */

  if (classInfo.WeightedMethodsPerClass >
metricThreshold.WeightedMethodsPerClass.Max && classInfo.CBO >
metricThreshold.CBO.Max)
  {
    detectionScore++;
  }

  return detectionScore == 2;
}
```

Şekil A-6 Beyin sınıf kod kusuru belirleme

EK-B

EĞİTİM VERİ SETİ PROJE METRİKLERİ

Yöntemde yazılım ölçütleri üzerinden hesaplamalar yapılmaktadır. Eğitim veri seti olarak kullanılan projelerden ölçütler elde edilmektedir.

B-1 Dys.Application Projesi Ölçütleri

Dys.Application projesinden elde edilen ölçütler Şekil B-1.1'de gösterilmektedir.

Sınıf sayısı: 78	Değişken sayısı: 323
Metot sayısı: 212	if/else: 151
Sınıflardaki kod satırı: 2535	Foreach: 41
Metotlardaki kod satırı: 1791	For: 0
WMC (Weighted Method Per Class): 450	While: 0
CBO (Coupling Between Object): 452	Try/Catch sayısı: 14
get/set sayısı: 320	Switch: 6
Maxnesting: 121	Switch içinde case: 46

Şekil B-1 Dys.Application projesi ölçütleri

B-2 Dys.Core Projesi Ölçütleri

Dys.Core projesinden elde edilen ölçütler Şekil B-2.1'de gösterilmektedir.

Sınıf sayısı: 94	Değişken sayısı: 75
Metot sayısı: 54	if/else: 12
Sınıflardaki kod satırı: 1542	Foreach: 9
Metotlardaki kod satırı: 389	For: 1
WMC (Weighted Method Per Class): 77	While: 1
CBO (Coupling Between Object): 358	Try/Catch sayısı: 8
get/set sayısı: 974	Switch: 0
Maxnesting: 21	Switch içinde case: 0

Şekil B-2 Dys.Core projesi ölçütleri

B-3 Framework.Biz Projesi Ölçütleri

Framework.Biz projesinden elde edilen ölçütler Şekil B-3.1'de gösterilmektedir.

Sınıf sayısı: 56	Değişken sayısı: 350
Metot sayısı: 238	if/else: 222
Sınıflardaki kod satırı: 3392	Foreach: 40
Metotlardaki kod satırı: 2418	For: 4
WMC (Weighted Method Per Class): 538	While: 6
CBO (Coupling Between Object): 526	Try/Catch sayısı: 35
get/set sayısı: 207	Switch: 4
Maxnesting: 154	Switch içinde case: 28

Şekil B-3 Framework.Biz projesi ölçütleri

B-4 Framework.Core Projesi Ölçütleri

Framework.Core projesinden elde edilen ölçütler Şekil B-4.1'de gösterilmektedir.

Sınıf sayısı: 92	Değişken sayısı: 347
Metot sayısı: 295	if/else: 243
Sınıflardaki kod satırı: 3824	Foreach: 51
Metotlardaki kod satırı: 2339	For: 13
WMC (Weighted Method Per Class): 624	While: 4
CBO (Coupling Between Object): 579	Try/Catch sayısı: 19
get/set sayısı: 284	Switch: 4
Maxnesting: 171	Switch içinde case: 18

Şekil B-4 Framework.Core projesi ölçütleri

B-5 Framework.DataAccess Projesi Ölçütleri

Framework.DataAccess projesinden elde edilen ölçütler Şekil B-5.1'de gösterilmektedir.

Sınıf sayısı: 15	Değişken sayısı: 156
Metot sayısı: 159	if/else: 166
Sınıflardaki kod satırı: 1535	Foreach: 22
Metotlardaki kod satırı: 1218	For: 10
WMC (Weighted Method Per Class): 367	While: 0
CBO (Coupling Between Object): 289	Try/Catch sayısı: 6
get/set sayısı: 101	Switch: 2
Maxnesting: 123	Switch içinde case: 10

Şekil B-5 Framework.DataAccess projesi ölçütleri

B-6 Framework.Service Projesi Ölçütleri

Framework.Service projesinden elde edilen ölçütler Şekil B-6.1'de gösterilmektedir.

Sınıf sayısı: 22	Değişken sayısı: 107
Metot sayısı: 114	if/else: 29
Sınıflardaki kod satırı: 637	Foreach: 13
Metotlardaki kod satırı: 575	For: 0
WMC (Weighted Method Per Class): 156	While: 0
CBO (Coupling Between Object): 165	Try/Catch sayısı: 1
get/set sayısı: 57	Switch: 0
Maxnesting: 32	Switch içinde case: 0

Şekil B-6 Framework.Service projesi ölçütleri

B-7 KS.KDM.Biz Projesi Ölçütleri

KS.KDM.Biz projesinden elde edilen ölçütler Şekil B-7.1'de gösterilmektedir.

Sınıf sayısı: 12	Değişken sayısı: 58
Metot sayısı: 20	if/else: 34
Sınıflardaki kod satırı: 544	Foreach: 9
Metotlardaki kod satırı: 356	For: 3
WMC (Weighted Method Per Class): 67	While: 1
CBO (Coupling Between Object): 104	Try/Catch sayısı: 1
get/set sayısı: 48	Switch: 0
Maxnesting: 26	Switch içinde case: 0

Şekil B-7 KS.KDM.Biz projesi ölçütleri

B-8 KS.KDM.Service Projesi Ölçütleri

KS.KDM.Service projesinden elde edilen ölçütler Şekil B-8.1'de gösterilmektedir.

Sınıf sayısı: 86	Değişken sayısı: 807
Metot sayısı: 331	if/else: 422
Sınıflardaki kod satırı: 7025	Foreach: 61
Metotlardaki kod satırı: 4924	For: 8
WMC (Weighted Method Per Class): 830	While: 2
CBO (Coupling Between Object): 805	Try/Catch sayısı: 29
get/set sayısı: 1189	Switch: 3
Maxnesting: 245	Switch içinde case: 6

Şekil B-8 KS.KDM.Service projesi ölçütleri

B-9 KS.KDM.Web Projesi Ölçütleri

KS.KDM.Web projesinden elde edilen ölçütler Şekil B-9.1'de gösterilmektedir.

Sınıf sayısı: 130	Değişken sayısı: 446
Metot sayısı: 294	if/else: 208
Sınıflardaki kod satırı: 5659	Foreach: 29
Metotlardaki kod satırı: 3207	For: 0
WMC (Weighted Method Per Class): 532	While: 1
CBO (Coupling Between Object): 823	Try/Catch sayısı: 9
get/set sayısı: 2030	Switch: 0
Maxnesting: 138	Switch içinde case: 0

Şekil B-9 KS.KDM.Web projesi ölçütleri

B-10 Leasing.DataAccess Projesi Ölçütleri

Leasing.DataAccess projesinden elde edilen ölçütler Şekil B-10.1'de gösterilmektedir.

Sınıf sayısı: 87	Değişken sayısı: 306
Metot sayısı: 420	if/else: 14
Sınıflardaki kod satırı: 3459	Foreach: 1
Metotlardaki kod satırı: 2557	For: 1
WMC (Weighted Method Per Class): 436	While: 0
CBO (Coupling Between Object): 809	Try/Catch sayısı: 1
get/set sayısı: 3	Switch: 0
Maxnesting: 8	Switch içinde case: 0

Şekil B-10 Leasing.DataAccess projesi ölçütleri

B-11 Services Projesi Ölçütleri

Services projesinden elde edilen ölçütler Şekil B-11.1'de gösterilmektedir.

Sınıf sayısı: 36	Değişken sayısı: 332
Metot sayısı: 121	if/else: 131
Sınıflardaki kod satırı: 2867	Foreach: 19
Metotlardaki kod satırı: 1963	For: 7
WMC (Weighted Method Per Class): 291	While: 1
CBO (Coupling Between Object): 268	Try/Catch sayısı: 38
get/set sayısı: 84	Switch: 3
Maxnesting: 75	Switch içinde case: 12

Şekil B-11 Services projesi ölçütleri

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Deniz NACAR

Doğum Tarihi ve Yeri : 06.08.1990, İstanbul

Yabancı Dili : İngilizce

E-posta : nacardeniz@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Yüksek Lisans	Mühendislik ve Teknoloji Yönetimi	Çukurova Üniversitesi	2015
Lisans	Bilgisayar Mühendisliği	Çukurova Üniversitesi	2013

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2016	KoçSistem Bilgi ve İletişim Hizmetleri A.Ş.	Yazılım Geliştirme Danışmanı
2015	Asseco See Teknoloji A.Ş.	Yazılım Mühendisi
2013	2 Adam Yazılım	Yazılım Mühendisi

ÖDÜLLER

1.	Yüksek Onur (1)	Çukurova Üniversitesi	Bilgisayar Mühendisliği (2013)
2.	Onur (1)	Çukurova Üniversitesi	Bilgisayar Mühendisliği (2011)