BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIMLARIN BAKIM KOLAYLIĞI ÖLÇÜMÜ İÇİN YAZILIM ÖLÇÜTLERİ ÖNERİSİ

ALPER KIRAL

YÜKSEK LİSANS TEZİ
2019

YAZILIMLARIN BAKIM KOLAYLIĞI ÖLÇÜMÜ İÇİN YAZILIM ÖLÇÜTLERİ ÖNERİSİ

SOFTWARE METRICS PROPOSAL TO MEASURE MAINTAINABILITY

ALPER KIRAL

Başkent Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin
BİLGİSAYAR Mühendisliği Anabilim Dalı İçin Öngördüğü
YÜKSEK LİSANS TEZİ
olarak hazırlanmıştır.

2019

“Yazılımların Bakım Kolaylığı Ölçümü İçin Yazılım Ölçütleri Önerisi” başlıklı bu çalışma, jürimiz tarafından, 31/01/2019 tarihinde, **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI 'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan : Dr. Öğr. Üyesi Emre SÜMER

Üye (Danışman) : Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

Üye : Doç. Dr. İhsan Tolga MEDENİ

ONAY

..../…./2019

Prof. Dr. Faruk ELALDI

Fen Bilimleri Enstitüsü Müdürü

BAŞKENT ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS TEZ ÇALIŞMASI ORİJİNALLİK RAPORU

Tarih: 08 / 02 / 2019

Öğrencinin Adı, Soyadı : Alper KIRAL
Öğrencinin Numarası : 21710541
Anabilim Dalı : Bilgisayar Mühendisliği
Programı : Tezli Yüksek Lisans
Danışmanın Unvanı/Adı, Soyadı : Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ
Tez Başlığı : Yazılımların Bakım Kolaylığı Ölçümü için Yazılım Ölçütleri Önerisi

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 73 sayfalık kısmına ilişkin, 08/02/2019 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 4'tür.

Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası:…….

Onay
08 / 02/ 2019
Öğrenci Danışmanı Unvan, Ad, Soyad,

Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

TEŞEKKÜR

Başta çok kıymetli sevgili aileme…

Yüksek Lisans süreci boyunca yaptığım çalışmalarda değerli bilgilerini benimle paylaşan, yol gösterici olan ve çok kıymetli fikirler veren danışman hocam;

Dr. Tülin ERÇELEBİ AYYILDIZ'a, çalışmam boyunca benden yardımlarını esirgemeyen Mehmet Emin GÜLLÜOĞLU'na sonsuz teşekkürlerimi sunarım.

ÖZ

YAZILIMLARIN BAKIM KOLAYLIĞI ÖLÇÜMÜ İÇİN YAZILIM ÖLÇÜTLERİ ÖNERİSİ

Alper KIRAL
Başkent Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Yazılım projelerinde göz ardı edilen zaman kısıtlılığı, insan faktörü gibi birçok önemli nokta ileriye dönük büyük problemler ortaya çıkarmaktadır. Yazılım kalitesi ölçülerek Bakım-onarım, fonksiyonellik, güvenilirlik gibi önemli noktalarda meydana gelebilecek bu problemlerin önüne geçilebilir. Bu çalışmada ISO 9126 kalite standardı kapsamında yazılımın bakım kolaylığı özelliğinin ölçümünde kullanılan ölçütler incelenmiştir. Çalışmanın gerçekleştirilebilmesi için uzay ve havacılık alanından 40 adet nesne yönelimli açık kaynak kodlu, JAVA programlama dilinde kodlanan yazılım projesi seçilip kod karmaşıklık analizi yapılmıştır; Chidamber and Kemerer (CK), Lorenz and Kidd (LK) ve McCabe's Complexity Suite ölçüt kümelerine ait ölçüt değerleri Understand statik kod analiz aracıyla belirlenmiştir. Bu ölçüt değerlerinin literatür çalışmasında elde edilen eşik değerleri geçip geçmediği karşılaştırılmıştır. 40 adet nesne yönelimli açık kaynak kodlu JAVA yazılım projesi için eşik değerleri geçen ölçüt frekansları hesaplanmıştır. Tespit edilen frekans değerleri arasındaki uyum Waikato Environment for Knowledge Analysis (WEKA) makine öğrenme ile araştırılmıştır. Sonuçlar değerlendirildiğinde, yazılımın bakım kolaylığı özelliği ölçümünde kullanılan Weighted Methods per Class (WMC), Coupling Between Objects (CBO), Response for Class (RFC) gibi CK ölçütlerine ek olarak Number of Children (NOC), Number of Inherited Methods (NIM) ve Ratio of Comment per Code (C/C) ölçütlerinin de anlamlı ölçüm sonuçları verdiği gözlemlenmiştir.

ANAHTAR SÖZCÜKLER: Yazılım kalite ölçütleri, yazılım kalite ölçümü, bakım kolaylığı, CK, LK, McCabe, ISO 9126, Understand, WEKA.

Danışman: Dr. Tülin ERÇELEBİ AYYILDIZ, Başkent Üniversitesi, Bilgisayar Mühendisliği Bölümü.

ABSTRACT

In software projects, many points that are overlooked such as time constraints and human factors are causing great problems in the future. By measuring the quality of software projects, problems that may arise in important parameters such as maintenance-repair, functionality and reliability can be eliminated. In this study, metrics that can be used for measuring maintainability quality attribute within the scope of ISO 9126 Quality Standard are examined. In order to perform the study, 40 open source object-oriented software belonging to space and aviation domain was selected and code complexity analysis was performed. Values of metric sets such as Chidamber and Kemerer (CK), Lorenz and Kidd (LK) and McCabe's complex Suite were determined by the Understand Code Analysis tool. It was determined whether the obtained values exceeded the threshold values indicated in the literature. Frequencies of metrics passing threshold values were determined for 40 open source object-oriented software projects, and the consistency among the metrics was evaluated using WEKA Machine Learning Software. When the results were evaluated, it was observed that in addition to CK metrics such as WMC, CBO, and RFC, which measure the maintainability quality attribute, NOC, NIM, and the Ratio of Comment/Code metrics have been observed to yield significant measurement results.

KEYWORDS: Software quality metrics, software quality measurement, maintainability, CK, LK, McCabe, ISO 9126, Understand, WEKA.

Supervisor: Dr. Tülin ERÇELEBİ AYYILDIZ, Başkent University, Department of Computer Engineering

İÇİNDEKİLER LİSTESİ

Sayfa

ŞEKİLLER LİSTESİ

Sayfa

ÇİZELGELER LİSTESİ

Sayfa

SİMGELER VE KISALTMALAR LİSTESİ

AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
C/C	Comment per Code: Kod başına Yorum Satırı
CBO	Coupling Between Objects
CC	Cyclomatic Complexity
CF	Coupling Factor
CK	Chidamber and Kemerer
DIT	Depth of Inheritance Tree
EC	Essential Complexity
KLOC	Kilo Line of Code: Bin Adet Kod Satırı
LCC	Loose Class Coupling
LCOM	Lack of Cohesion of Methods
LK	Lorenz and Kidd
LOC	Line of Code: Kod Satırı
MHF	Method Hiding Factor
MIF	Method Inheritance Factor
MLOC	Million Lines of Code: Milyon Adet Kod Satırı
NIM	Number of Instance Methods
NIV	Number of Instance Variables
NOC	Number of Children: Alt Sınıf Sayısı
OO	Object-Oriented: Nesne-Yönelimli
PF	Polymorphism Factor
RFC	Response for a Class
WEKA	Waikato Environment for Knowledge Analysis
WMC	Weighted Methods per Class
TCC	Tight Class Cohesion

1 GİRİŞ

Teknolojinin hızlı ilerleyişi ile birlikte yüksek işlem gücüne sahip bilgisayar donanımları düşük maliyetlerle üretilebilmektedir. Bu durum beraberinde daha kapsamlı yazılım ihtiyaçları, daha büyük yazılım projeleri ve bu projelerin yürütülmesi görevini sağlayan daha kalabalık yazılım ekiplerini getirmektedir. Yazılım projelerindeki tüm bu büyüme ile bakım-onarım masrafları, proje maliyeti, yazılım geliştirme zamanı gibi çok önemli faktörler de artmaktadır. Bu ve benzeri faktörlerde meydana gelebilecek sorunlar projenin erken safhalarında doğru şekilde öngörülemezse, sonradan düzeltilmesi çok maliyetli olacak ve projenin tamamı tehlikeye girebilecektir. Bu gibi artan maliyetler yüzünden iptal edilen, kullanılamayan, yazılım kalitesi düşüklüğü sebebiyle müşteriler tarafından reddedilen ya da yüksek bakım-onarım maliyetleri gerektiren yazılım projelerinin ekonomiye olan zararları ise bir hayli fazladır. Örnek olarak, 2002 yılı verilerine göre başarısız yazılım projelerinin Amerikan ekonomisine yıllık zararı yaklaşık 59 Milyar $ civarındadır [1]. Ayrıca Tricentis firmasının 2017 yılı araştırmasına göre de başarısız yazılımların küresel ekonomiye verdiği zarar 1.7 Trilyon $ dolaylarındadır [2].

Kaliteli bir yazılımın başlıca gereksinimleri için; kod satırı başına hata oranı ve operasyon süresi boyunca meydana gelen hata oranının en az seviyede olması gerekliliğinden bahsedilmektedir [3]. ISO 9126 kalite standardında bakım kolaylığı özelliği için özetle, yazılımın ürününün güncellenebilir olması denilmektedir. Bu güncellemeler; yazılımdaki hata düzeltmeleri, yazılımın iyileştirilmesi ve değişen-gelişen ihtiyaçlara göre yazılımın uyarlanmasıdır [4]. Yazılımın temelleriyle alakalı olan bu değişiklikler, bakım kolaylığı özelliğini çok önemli bir noktaya taşımaktadır. Eğer bir yazılım uygulaması bakım kolaylığı kaynaklı değişiklikleri iyi şekilde destekleyemiyorsa, bakım-onarım çalışmaları hayli masraflı hale gelmektedir. Bu nedenle erken safhalarda yazılımın; tasarım, kodlama, test, onarım ve gereksinim analizi safhalarında bakım kolaylığı özelliğine büyük önem verilmelidir [5], [6]. Bu şekilde yazılım proje başarımında büyük rol sahibi olan bakım kolaylığı özelliği doğru yönetilebilir ve sonuç olarak başarısız proje sayısının azalması beklenmektedir.

ISO 9126 kalite standardına göre bakım kolaylığı özelliğinin ölçülebilmesi için alt karakteristik özellikler belirtilmiştir. Bunlar analiz edilebilirlik, değişebilirlik, kararlılık, test edilebilirlik ve riayettir [4]. Literatürde, belirtilen alt karakteristikler, yazılım kalite ölçütleri yardımıyla ölçülerek genel bir bakım kolaylığı değerlendirilmesi yapılmaktadır. Seçilen ölçüt kümeleri, kullanılan yöntemler ve yapılan yaklaşımlar yazılımın bakım kolaylığı ölçümü için farklı bakış açıları ortaya koymaktadır. Yapılan çalışmalarda ilerleme kaydedilmesi bakım kolaylığı ölçümünün başarımının yükselmesi anlamına gelmektedir.

Genel olarak nesne-yönelimli (object-oriented (OO)) yazılımlarda bakım kolaylığı özelliği için önemli olduğu söylenen ölçütler; NIM, Depth of Inheritance Tree (DIT), NOC, Cyclomatic Complexity (CC), WMC, Line of Code (LOC), Lack of Cohesion of Methods (LCOM), CBO, RFC olarak görülmektedir [7]. Literatürde incelenen çalışmalarda bu ve benzeri ölçüt kümeleri kullanılarak ölçümler yapılmakta ve sonuçları değerlendirilmektedir. Farklı yaklaşımlarla yeni ve tutarlı çalışmalar yapılıp yeni ölçüt kümeleri önerileri, bakım kolaylığı ölçümünde bilimsel başarım sağlanmasına olanak tanımaktadır.

Büyük yazılım projelerinde tek tek kod analizi yapmak zor olacağından, bu analizleri kısa sürelerde gerçekleştiren araçlar ve eklentiler bulunmaktadır. İlgili yazılım kalite ölçütleri için değerleri analiz sonucu olarak veren bu araçlar, yazılım sektöründe sıklıkla kullanılmaktadırlar. Çalışma kapsamında, açık kaynak kodlu 40 adet OO JAVA yazılım projesi, kod analiz araçlarıyla incelenmekte ve makine öğrenmesi yazılımı yardımıyla yazılımın bakım kolaylığı özelliği için yeni bir ölçüt kümesi önerimi yapılmaktadır.

Çalışmada araştırma soruları şu şekilde belirlenmiştir:

- Araştırma Sorusu 1: Yazılım kalite ölçütlerinin en önemlileri nelerdir?
- Araştırma Sorusu 2: Yazılım bakım kolaylığı kalite özelliği için yeni ve daha etkili bir ölçüt kümesi önerilebilir mi?

Yapılan çalışmanın organizasyonu aşağıda paragraflar halinde verilmektedir.

Giriş bölümü; yazılım projelerinde karşılaşılan sorunları ve bu sorunlar erken safhalarda fark edilip önlem alınmazsa meydana gelebilecek zararları konu

almaktadır. Yazılımın bakım kolaylığı özelliğinin bu aşamada önemi vurgulanıp, gerekli ölçümlerin yapılış şekilleri özet bilgi olarak verilmektedir.

Literatür çalışması bölümü; bakım kolaylığı özelliği için daha önce yapılan çalışma örnekleri incelenmektedir. Ayrıca ISO 9126 kalite özelliği ve OO ölçütleri incelenmektedir. Bulgular bu çalışma kapsamında yapılan araştırmayla kıyaslanmaktadır.

Yürütülen çalışma bölümü; İlgili çalışmanın detaylandırıldığı bölümdür. Yazılım kalite ölçütleri yapısal ve OO ölçütleri olarak tarif edilmektedir. CK, LK ve McCabe's Complexity Suite ölçüt kümelerinden bahsedilmektedir. Kıral ve Erçelebi Ayyıldız, [33]'ün CK ölçütleri önem sırası hakkında yaptıkları bilimsel çalışma açıklanmaktadır. Ayrıca ISO 9126 kalite standardı incelenip neden bakım kolaylığı çalışması yapıldığı açıklanmaktadır. Çalışma kapsamında kullanılan uzay ve havacılık sektörüne ait veri seti incelemesi yapılmaktadır. Kıral ve Erçelebi Ayyıldız, [34]'ün bakım kolaylığı için ortaya koyduğu bilimsel çalışmanın yöntemleri anlatılmakta ve çalışma sonuçları değerlendirilmektedir.

Tartışma bölümü; Çalışma kapsamında bulunan sonuçlar değerlendirilmektedir.

Sonuç bölümü; Değerlendirilen çalışma sonuçları literatürle kıyaslanıp, genel bir yargıya varılmaktadır.

2 LİTERATÜR ÇALIŞMASI

Literatür çalışması;

- Yazılım ölçütleri ve önemleri hakkında yapılan çalışmalar,
- ISO 9126 kalite standardı hakkında yapılan çalışmalar,
- Yazılımın bakım kolaylığı karakteristiği hakkında yapılan çalışmalar

olarak üç alt başlıkta incelenmektedir.

2.1 Yazılım Ölçütleri ve Önemleri

Literatürde çeşitli sayıda yazılım ölçütleri ve ölçüt kümeleri bulunmaktadır. En çok kullanılanlardan biri CK [8] ölçüt kümesidir ve bu kümeye ait ölçütler; CBO, RFC, WMC, LCOM, DIT ve NOC'dur. Bahsi geçen altı ölçüt, yazılımın kaynak kodunun bağımlılık, büyüklük, karmaşıklık, uyumluluk gibi çeşitli özelliklerini değerlendirmede kullanılmaktadırlar.

CK haricinde farklı ölçüt kümeleri de bulunmaktadır. Örnek olarak, QMOOD ölçüt kümesi 11 adet tasarım ölçütü içermektedir ve OO yazılımların kapsülleme, yapı ve çok şekillilik gibi özelliklerini değerlendirmektedir [9]. 1974 yılında ise yazılım karmaşıklığını ölçmek amacıyla Thomas McCabe tarafından kendi adını verdiği McCabe's Complexity Suite öne sürülmüştür [10]. İleride benzeri ölçüt kümelerinden daha detaylı bahsedilecektir.

2.1.1 Yazılım kalitesine kötü kodlama etkisinin araştırması

2013 yılında yapılan çalışmada Fontana et al., [11], OO JAVA yazılımlar için en sık rastlanılan kötü kodlama örneklerini ve yazılım projesi alanlarına bağlı olarak sıklıkla kötü kodlama yapılıp yapılmadığını araştırmaktadır. Ek olarak, kötü kodlama ihtimali ile ölçüt değerleri arasında bir bağ olup olmadığını da çalışma dahilinde incelemektedir.

İlgili çalışmada yazılım sistemlerinin ve yazılım ölçütlerinin hangi alana özgü seçileceğinin önemi vurgulanmaktadır. Bu nedenle çalışmanın tutarlılığını korumak amacıyla JAVA dilinde yazılan açık kaynak kodlu yazılımlar seçileceğinden bahsedilmektedir. Çalışma için Tempero et al., [12], tarafından geliştirilen Qualitas Corpus adlı açık kaynak kodlu, 106 adet sistem içeren JAVA projesi seçilmiştir.

Sistemler, yine tutarlılık açısından, boyutlarına göre kategorize edilmiştir (Çizelge 2.1). İlgili aralık gruplandırmasına göre 106 projeden 68 tanesi araştırma için seçilmiştir.

Çizelge 2.1 LOC aralığına göre gruplandırma

LOC Aralığı	**Sistem Boyutu**
0-4999	Küçük
5000-14.999	Küçük-Orta
15.000-39.999	Orta
40.000-99.999	Orta-Geniş
100.000-499.999	Geniş
≥500.000	Çok Geniş

Çalışma analizlerini yapabilmek için OO yazılım kalitesi ölçütlerinden, boyut, karmaşıklık, bağımlılık, veri soyutlama, uyumluluk ve kalıtım kategorilerine ait olanlar seçilmiştir. Her bir kategoriden en az bir ölçüt bulunmaktadır.

Kötü kod analizi için Çizelge 2.2'deki ölçütler ve araçlar kullanılmaktadır ve Çizelge 2.3'deki kötü kod sonuçları belirlenmiştir.

Çizelge 2.2 68 sistemde çalışan ölçütler

Ölçüt	Öğe	Araç
Boyut		
LOC	İşlem	iPlasma [13]
Number of Methods (NOM)	Sınıf	
Number of Packages (NOP)	Sistem	
NOC	Sistem	
Number of equal Lines of Code (EDUPLINES)	Metot	
Average Line of Code per Method (ALCM)	Sistem	CodePro Analytix [14]
Karmaşıklık		
McCabe's Cyclomatic Number (CYCLO)	İşlem	iPlasma [13]
WMC	Sınıf	
Average Method Weight (AMW)	Sınıf	
Program Volume (PV)	Sistem	CodePro Analytix [14]
Bağımlılık		
Abstractness (Abstr)	Sistem	CodePro Analytix [14]
Instability (I)	Sistem	
Distance from Main Sequence (DMS)	Sistem	
Changing Classes (CC)	Metot	iPlasma [13]
Number of Called Classes (FANOUT)	Sistem	
Veri Soyutlama		
Access to Foreign Data (ATFD)	Sınıf, Metot	iPlasma [13]
Locality of Attribute (LAA)	Metot	
Uyumluluk		
Tight Class Cohesion (TCC)	Sınıf	iPlasma [13]
LCOM	Sınıf	Eclipse Metrics [15]
Kalıtım		
Average Depth of Inheritance Hierarchy (ADIH)	Sistem	CodePro Analytix [14]

Çizelge 2.3 Kötü kod sonuçları

Kötü kod türü	Öğe
Kullanılan Araç: iPlasma [13]	
God Class	Sınıf
Refused Parent Bequest	Sınıf
Refused Parent Bequest 2	Sınıf
Intensive Coupling	Metot
Extensive Coupling	Metot
Shotgun Surgery	Metot
Feature Envy	Metot
Data Class	Sınıf
Brain Method	Metot
Brain Class	Sınıf
Tradition Breaker	Sınıf
Schizophrenic Class	Sınıf
Kullanılan Araç: NiCad [16]	
Duplicate Code	İşlem
Kullanılan Araç: Excel Makroları	
Speculative Generality	Sınıf
Long Method	Metot
Long Parameter List	Metot

Bu çalışmanın geri kalan kısımlarında 68 sistem 4 kategoriye ayrılmaktadır. (Uygulama Yazılımları, İstemci Sunucu Yazılımları, Veri Görselleştirme Yazılımları, Yazılım Geliştirme). Çizelge 2.3'de bulunan kötü kod sonuçları ilgili kategorilere dağıtılıp hangi alanda daha çok kötü kod örneği olduğu sorusuna cevap bulunmaktadır.

Literatür taraması kapsamında bu çalışma yardımıyla ölçütler hakkında fikir edinilmiştir. CK ve McCabe Complexity Suite ölçüt kümeleri hakkında bilgiler edinilmiştir.

2.1.2 Erken safha nesne-yönelimli yazılım tasarım ölçütlerinin kod büyüklüğüne olan etkilerinin araştırması

Çalışmanın amacı erken safha OO yazılım tasarım ölçütlerinin kod satırı sayısına etkilerini gözlemlemektir [19]. Bu amaçla literatür taraması yapılıp MOOD [17],

QMOOD [9], Martin [18] ve CK [8] gibi ölçüt kümelerinden OO yazılım ölçütleri seti toplanmıştır (toplam 42 adet ölçüt tanımlanmıştır).

Bu ölçütler için analiz yapılmadan önce; birbirini tekrar eden, aralarında tutarsızlıklar bulunan ve birbirinin muadili olan ölçütler literatür araştırması sonucu çıkarılmıştır. Geriye 7 adet erken safha OO yazılım tasarım ölçütü kalmış olup bu ölçütler Çizelge 2.4'te gösterilmektedir.

Çizelge 2.4 Nesne-yönelimli erken safha tasarım ölçütleri

Ölçüt Kümesi	**Ölçüt**	**Nesne-Yönelimli Yapı**
CK	DIT	Sınıf
CK	NOC	Sınıf & Tasarım
CK	LCOM	Metot tasarımı
Martin	Afferent Couplings (CA)	Paket bağımlılığı
Martin	Efferent Couplings (CE)	Paket bağımlılığı
QMOOD	Number of Methods (NOM)	Metot karmaşıklığı
Module Wide Metrics	Number of Fields (NF)	Kod, modül bazında

Veri seti olarak 252 paketlik açık kaynak kodlu, JAVA diliyle geliştirilen, Eclipse yazılım geliştirme ortamı eklentisi, yine Eclipse yazılım geliştirme ortamına ait kod analizi aracı kullanılarak analiz edilmiştir. Sonuçlar düzenlenip bir veri seti haline getirilmiştir. Toplam ve ortalama değerler, değişkenler normal dağılım sergilemediği için (Kolmogorov-Smirnov testi ile normal dağılım kontrolü yapılmış), Spearman korelasyon testi ile analiz edilmiştir. Sonuç olarak Million Lines of Code (MLOC), NOC, NOM ve NOF ölçütleri arasında yüksek ilişki tespit edilmiştir

(Çizelge 2.5). Korelasyon testi sonucunu garantiye almak için regresyon çalışması da yapılmıştır ve sonuç olarak tutarsızlık tespit edilmemiştir.

Çalışma sonucunda; MLOC, NOC, NOM ve NOF ölçütleri arasında yüksek ilişki bulunduğu söylenirken, DIT ölçütünün LOC değerine yani kod satırı sayısına negatif etkisi olduğu belirlenmiştir.

Çizelge 2.5 Spearman korelasyon matrisi

	MLOC	NOC	NOM	NOF	DIT	CA	CE	LCOM
MLOC	1.000	.914	.955	.937	.883	.571	.848	.530
NOC		1.000	.941	.910	.976	.581	.918	.545
NOM			1.000	.950	.913	.555	.875	.534
NOF				1.000	.872	.524	.826	.528
DIT					1.000	.537	.901	.506
CA						1.000	.679	.396
CE							1.000	.496
LCOM								1.000

Literatür taraması kapsamında kod satırı sayısı ile CK ölçüt kümesine ait NOC ölçütü arasındaki yüksek ilişki ve DIT ölçütü ile negatif ilişki öğrenilmiştir. Ayrıca yine ölçüt kümeleri ve ölçütler hakkında fikir edinilmiştir.

2.1.3 Nesneye dayalı yazılım ölçütleri ve yazılım kalitesi araştırması

Erdemir vd., [20]'nin çalışmasında yazılım ölçütleri ve yazılım kalitesi üzerine bir literatür taraması yapılmış olup, ölçütlerin tanımlaması ve kaliteyle bağlantıları araştırılmaktadır. Ek olarak kaliteyi artırma üzerine prensipler ve yöntemler araştırılarak bahsedilmektedir.

Literatür çalışması kapsamında yazılım kalitesinin can alıcılığı üzerine Amerikan Ordusu'nun yazılım projeleri hakkında yaptığı bir araştırmanın sonuçları paylaşılmıştır [20]. Araştırmaya göre yazılım projelerinin;

- %47'si kullanılmamakta,
- %29'u müşteri tarafından kabul edilmemekte,
- %19'u başladıktan sonra iptal edilmekte ya da büyük ölçüde değiştirilerek yeniden başlatılmakta,
- %3'ü bazı değişikliklerden sonra kullanılabilmekte,
- Geri kalan projelerin sadece %2'sinin teslim alındığı şekliyle kullanılabilmekte olduğu görülmüştür.

Yazılım kalitesinin, müşteri gözünden ve üretici gözünden kalite olarak ele alınması gerektiği söylenmektedir. Müşteriler hatasız, yeterli performansa sahip ve isteklerini karşılayan ürün isterlerken, yazılım üreticileri ise geliştirme ve bakım-onarım maliyetleri düşük, parçaları daha sonra başka yazılımlarda da kullanılabilecek bir ürün ortaya çıkarmayı istemektedirler. Bu kapsamda ortaya çıkan ISO 9126 kalite standardı çalışma kapsamında incelenmektedir ve alt başlıklarından bahsedilmektedir [21] (3. Bölüm; "Yürütülen Çalışma" kısmında daha detaylı bahsedilecektir).

- İşlevsellik
- Güvenilirlik
- Kullanılabilirlik
- Verimlilik
- Bakım kolaylığı
- Taşınabilirlik

Üretim süreçlerinin sürekli gelişmesi, daha kaliteli ürün ortaya çıkarma hedefini doğurmaktadır. Bu nedenle SPICE, ISO, CMMI gibi uluslararası standartların, yazılım şirketleri tarafından süreç çalışmalarında sıklıkla kullanıldığından bahsedilmektedir. Yazılım projesi süreci esnasında toplanan ölçümler, yazılımın değerlendirilebilmesi ve kontrolü için önemli veriler olmaktadır. Bu nedenle ölçütlere ve ölçme işlemini yürütmeye ihtiyaç vardır.

2.1.3.1 Yazılımın yapısal özellikleri

Yazılımın dış özellikleri müşterinin isteklerini tarif ederken, iç özellikler ise yazılımın yapısal durumunu belirtmektedir. Yazılımın önemli iç özelliklerinin; bağımlılık, uyumluluk ve karmaşıklık olduğu söylenmektedir. Bu noktada kaliteli bir yazılımın uyumluluğunun yüksek, bağımlılığın ve karmaşıklığının ise mümkün olduğunca düşük olması beklenmektedir [22].

Bağımlılık, genel olarak bir sınıfın kendi işleri için başka sınıfları kullanma miktarı ve/veya farklı sınıflara dahi içerdiği bilgiler şeklinde izah edilmektedir. Bir yazılımın kaliteli olabilmesi için bağımlılığın düşük olması istenmektedir. Bağımlılık arttıkça;

- Bir sınıfta yapılan bir değişiklik, bağlı olduğu sınıfları da etkiler.

- Sınıfları birbirinden ayrı olarak okumak zorlaşır.
- Sınıfların tekrar kullanılması zorlaşır.

Uyumluluk için bir sınıftaki metot ve niteliklerin birbiriyle alakalı olması durumu denilmektedir. Bir sınıfın metot değişkenlerindeki ve nitelik tiplerindeki güçlü ahenk uyumun fazla olduğunu göstermektedir. Sınıflar birbiri ile ilgili olmayan işler yapıyor ve birbiri ile alakalı olmayan nitelik değişkenlerine sahipse o sınıfın uyumluluğu düşüktür denilmektedir. Düşük uyumluluğun yol açtığı sorunlar;

- Anlaşılabilirlik güçleşir.
- Bakım yapmak güçleşir.
- Tekrar kullanılabilirlik güçleşir.
- Sınıfların değişikliklere olan hassasiyeti artar.

Karmaşıklığın sınıflar arası ilişkileri ve yapıları anlama zorluğu olduğu belirtilmektedir. Eğer bir nesne çok fazla özelliğe sahipse, o nesne karmaşıktır denilmektedir.

2.1.3.2 Yazılım ölçüt kümeleri

Yazılım ölçütleri statik ve dinamik olarak iki grupta toplanabilir. Statik kod ölçütleri yazılım çalıştırılmadan önce ölçüm yaparken kullanılmaktadırlar. Dinamik ölçütler ise yazılım çalışması esnasında veri ölçümünde kullanılmaktadırlar. Bu çalışmada literatürde yaygın kullanılan 3 adet OO yazılım ölçütü kümesi incelenmektedir.

CK ölçüt kümesi, 6 adet tanımlı ölçütten oluşmaktadır. Bunlar (detaylı bilgilendirme 3. Bölüm'de yapılacaktır) WMC, DIT, NOC, CBO, RFC ve LCOM ölçütleridir. Çalışma kapsamında Chidamber and Kemerer, [8]'in 1994 yılında, ölçüt kümelerini tanıtırken sunduğu bildiriden faydalanarak ilgili ölçütlerin tarifleri yapılmıştır.

MOOD ölçüt kümesi [17; 23], OO yaklaşımın temellerine dayanmaktadır. Bunlar kapsülleme (Method Hiding Factor (MHF) ve Attribute Hiding Factor (AHF)), kalıtım (Method Inheritance Factor (MIF) ve Attribute Inheritance Factor (AIF)), çok

şekillilik (Polymorphism Factor (PF)), mesaj aktarımı yani bağımlılık faktörü (Coupling Factor (CF)) olarak verilmektedir.

- MHF; Sınıflarda çağrılabilir metotların tüm metotlara oranıdır (Kalıtımla gelen metotlar dahil değil). Sınıf için görünürlük ölçülür.

- MIF; Sistemde tanımlı tüm sınıflardaki kalıtım ile gelen metotların, bütün metotlara oranıdır (kalıtım ile gelenler dahil).

- AHF; Sınıflardaki erişilebilir niteliklerin tüm niteliklere oranıdır (Kalıtımla gelenler hariç). Sınıf için görünürlük ölçülür.

- AIF; Sistemde tanımlı tüm sınıflardaki kalıtım ile gelen niteliklerin, bütün niteliklere oranıdır (kalıtım ile gelenler dahil).

- PF; Bir sınıfın sistemde tanımlı çok şekilli durumlarının, maksimum olası çok şekillilik sayısına oranıdır.

- CF; Sınıflar arasında var olan bağımlılık sayısının, oluşabilecek maksimum bağımlılık sayısına oranıdır (Kalıtımla oluşan bağımlılıklar hari).

QMOOD ölçüt kümesi [8], toplam kalite endeksi hesaplaması için 4 seviyeden oluşan hiyerarşik bir model içerisinde tanımlanmıştır.

4. seviyede sınıf, metot gibi OO yazılım bileşenleri bulunmaktadır.

3. seviyede QMOOD ölçütleri yer almaktadır.

2. seviyede karmaşıklık, uyumluluk gibi yazılım özellikleri bulunmaktadır ve QMOOD ölçütleri ile hesaplanmaktadırlar.

1. seviyede ise bir alt seviyedeki yazılım özellikleri kullanılarak hesaplanan anlaşılabilirlik, esneklik, tekrar kullanılabilirlik gibi kalite nitelikleri bulunmaktadır.

Kalite niteliklerinden faydalanılarak da toplam kalite endeksi elde edilmektedir. 11 QMOOD ölçütü aşağıdaki gibidir [9; 20];

Data Access Metric – DAM; Bir sınıfa ait private ve protected niteliklerin tüm niteliklere oranıdır. Kapsüllenmeyi belirtir.

Design Size in Classes – DSC; Tasarım aşamasındaki toplam sınıf sayısıdır.

Direct Class Coupling – DCC; Bir sınıfın değişken olduğu ve değişken olarak tanımlandığı sınıfların toplamıdır. Bağımlılığı belirtir.

Number of Polymorphic Methods – NOP; Metotlardan çok biçimli olanların sayısıdır.

Avarage Number of Ancestors – ANA; Kalıtım ağacı derinliğinin ortalamasıdır. Soyutlama durumunu belirtir.

Cohesion Among Methods – CAM; Metotlar arasındaki uyumluluk göstergesidir.

Class Interface Size – CIS; Bir sınıfa ait metotların public olanlarının sayısıdır. Sınıflar arası iletişimi gösterir.

Measure of Functional Abstraction – MFA; Yazılımdaki toplam türeyen metot sayısının, bütün metotların sayısına oranlanması ile bulunur. Kalıtımı gösterir.

Measure of Aggregation – MOA; Sınıf tanımlamalarının, toplam tanımlı veri tiplerine oranıdır.

Number of Methods – NOM; Sınıfa ait toplam metot sayısıdır. Karmaşıklığı belirtir.

Number of Hierarchies – NOH; Sınıf hiyerarşilerinin sayısını ölçen ölçüttür.

2.1.3.3 Yazılım kalite çalışmalarında kullanılabilecek yardımcı araçlar

Yazılım kod analizini elle yapmak oldukça zordur. Bu nedenle bu işi hızlı ve otomatik yapabilen araçlara ihtiyaç duyulmaktadır. Çalışma kapsamında yazılım kalitesinin artırılmasında kullanılabilecek, ölçümleri hızlı bir şekilde otomatik yapabilecek ticari ve açık kaynak kodlu araçlardan örnekler verilmektedir. Bu gibi araçlar kalite analizi çalışmalarını oldukça kolaylaştırabilmektedirler.

- FindBugs; statik kod analiz aracıdır. JAVA kodu üzerinde analiz yaparak, yazılım hatalarını ve tasarım kusurlarını kısa sürede bulabilmektedir [60].

- CheckStyle; yazılımın yapısından çok formatı ile ilgilenen açık kaynak kodlu yazılım analizi aracıdır. JAVA kodları üzerinde format analizi yaparak kod yazım standartlarına uyumlu çalışılmasını desteklemektedir [61].

- Metrics; JAVA projelerinde yaygın birçok yazılım ölçütünü değerlendirerek raporlamaktadır [63].

- PMD; başka bir statik kod analizi aracıdır. JAVA üzerinde analiz yaparak olası kusurları, gereksiz kod parçalarını, kullanılmayan ve gereksiz modülleri raporlamaktadır [64].

- Coverlipse; yazılım kodu, gereksinimler ve test senaryolarının arasındaki örtüşmesi inceleyen araçtır. Bu 3 temel arasındaki uyuma bakarak aradaki olası boşlukları raporlamaktadır [65].

- SDMetrics; UML tasarım dokümanları üzerinde sayısal analiz yapabilen bir araçtır. Programın tasarım aşamasında, tasarım analizi yaparak bağımlılık ve karmaşıklık gibi önemli unsurlarda uygunsuzlukları erkenden raporlayabilmektedir. Erken fark edilen bu unsurlar, ilerdeki bakım-onarım maliyeti gibi problemleri azaltabilmektedir [62].

Çalışma kapsamında yazılım kalite ölçütleri, ölçümü ve ölçmesi hakkında geniş bir inceleme kaleme alınmaktadır. Yazılım kalitesi ve ölçütlerinin temel kavramları kısaca açıklanmaktadır. Ölçüt tanımlamalarının ve mevcut araçların şu anki durumu değerlendirilerek, gelecekte varılması gereken yerleri tartışılmaktadır. Endüstri için bilgilendirici amaç benimseyen bir çalışma olduğu vurgusu yapılmaktadır.

Araştırma dahilinde, yazılım kalitesi ve yazılım kalite ölçütleri hakkında geniş bilgi edinimi sağlanmıştır.

2.1.4 Nesne-yönelimli tasarım ölçütleri ve kalite özellikleriyle ilişkisi

Calp ve Arıcı, [24]'ün yaptığı çalışmada, CK ölçüt kümesinin kısa tanımı ve kalite özelliklerinden bahsedilmektedir ve bu ölçütlerin yorumlanması ile ilgili bilgiler yer almaktadır. Seçilen bir yazılım projesinin 4 farklı sürümünün ölçüt değerleri ölçülmekte ve elde edilen sonuçlar kalite özellikleriyle ilişkilendirilmektedir.

Geleneksel tasarım ölçütleri doğrudan OO yazılımlarda kullanılamayabilirler. Bu nedenle OO yazılım ölçütleri araştırmaları sonucu CK, MOOD, QMOOD gibi OO ölçüt kümeleri ortaya çıkmıştır. Çalışma kapsamında CK ölçüt kümesinden yararlanılmaktadır.

Yazılım kalitesinin önemi sebebiyle birçok kalite modeli geliştirilmiştir. Bu modeller, ürün kalitesini veya kod kalitesini değerlendirmektedirler. Literatürdeki bazı kalite modelleri;

- McCall [25],
- Boehm [25],
- ISO 9126 [21; 25],
- Dromey [25],
- Bansiya [25],
- The Software Assurance Technology Center (SATC) tarafından geliştirilen kalite modelidir [26].

Çalışmada kod kalitesi değerlendirmesinde SATC kalite modelinden yararlanılmaktadır. SATC, kodlama ve tasarım evresi için; verimlilik, karmaşıklık, anlaşılırlık, yeniden kullanılabilirlik ve test edilebilirlik/dayanıklılık olmak üzere 5 kalite özelliği önermektedir.

- Verimlilik; Yazılımın zaman ve kaynak kullanımı gibi konularda yeterli performansa sahip olma derecesidir. OO tasarım özelliklerini kullanarak gerekli işlevsellik ve davranışı sağlamadaki tasarım yeterliliğidir [27].

- Karmaşıklık; OO yapıların, yazılım karmaşıklığına sebep verip vermediğini gösterir [27]. Yazılımın anlaşılırlık düzeyini olumsuz etkilemektedir.

- Anlaşılırlık; Modelin anlaşılması için gerekli olan çaba miktarıdır. Modelin tekrar kullanımının kolaylığını sağlamaktadır.

- Yeniden kullanılabilirlik; Bir modülün diğer bölümlerde tekrar kullanılabilmesidir. Tasarımın yüksek yeniden kullanılabilirlik desteklemesi olumlu bir özelliktir.

- Test edilebilirlik / dayanıklılık; Yazılımın değişiklik ya da düzeltme gerekliliklerine uyum becerisidir. OO yapıların test kolaylığını ve değişiklikleri destekleyip desteklemediği incelenmektedir [27].

OO yazılımlar belli karakteristiklere sahiptir. Çoğunlukla kullanılan yazılım kalite özellikleri kapsülleme, kalıtım, bağımlılık ve uyumdur. Bu kalite özellikleri ve CK ölçütlerinin SATC tarafından önerilen tasarım özellikleri ilişkisi Çizelge 2.6'da gösterilmektedir.

Çizelge 2.6 Tasarım karakteristikleri ve ölçütler [24]

OO Tasarım Karakteristikleri	**Ölçütler**
Kapsülleme	RFC, WMC
Kalıtım	DIT, NOC
Bağımlılık	CBO
Uyum	LCOM, (COM)

SATC'a göre WMC, RFC, CBO, DIT ve NOC ölçüt değerleri düşük, Cohesion of Methods (COM) değerinin yüksek olması istenilen durumdur. OO tasarım karakteristikleri, ölçüt değerlerinin arzu edilen hedefleri ve SATC kalite özellikleri birleştirilecek olursa Çizelge 2.7'deki gibi bir ilişki tablosu elde edilebilmektedir.

Çizelge 2.7 CK ölçüt kümesi – SATC kalite özellikleri ilişkisi [24]

OO Tasarım Karakteristikleri	CK ÖLÇÜT KÜMESİ					
	WMC	DIT	NOC	CBO	RFC	LCOM
Verimlilik		✓	✓	✓		✓
Karmaşıklık	✓	✓			✓	✓
Anlaşılırlık	✓	✓			✓	
Yeniden Kullanılabilirlik	✓	✓	✓	✓		✓
Test edilebilirlik		✓	✓		✓	
Dayanıklılık	✓				✓	

Calp ve Arıcı, [24]'ün yaptığı çalışmada incelenen yazılımın 4 farklı sürümü için; arzu edildiği üzere RFC, WMC, DIT, NOC, CBO değerleri her yeni sürümde düşürülerek yazılım iyileştirilmiş olmakta, COM değeri ise 2. sürümde kötüye gitse de son sürüme kadar iyileştirmeler yapılıp başlangıç seviyesine dönüşü sağlanmış bulunmaktadır.

İlgili çalışmada yazılım kalite ölçütleri ve kalite modelleri arasındaki bağlantının kurulma yolları ve yöntemleri hakkında fikirler edinilmiştir.

2.1.5 Tasarım aşaması kalite ölçütleri ve kalite özellikleri ile ilgili eşleştirme çalışması

Arvanitou et al., [28]'in yaptığı çalışma kapsamında, yazılım kalitesini raporlamak için yazılımın ait olduğu alana uygun kalite özellikleri seçmek ve bu özellikleri ölçmede kullanılacak ölçütleri doğru belirlemek gerektiği ele alınmaktadır. Bu sebeple, 154 adet bilimsel yayın toparlanmış ve kategorize edilmiştir.

Bilimsel yayınların;

- En çok araştırması yapılan kalite özelliklerine göre,
- Yazılımın ait olduğu alanlar baz alınarak en çok araştırması yapılan kalite özelliklerine göre,
- En çok araştırması yapılan kalite ölçütlerine göre

gruplama tabloları yapılıp, ilgili elemanlar için frekans değerleri bulunmuştur. Bu tez kapsamında yürütülen çalışmalarda da faydalanıldığı için en çok araştırması yapılan kalite ölçütleri tablosu Çizelge 2.8'de gösterilmektedir.

Çizelge 2.8 En çok araştırılan kalite ölçütleri frekansları [28]

Kalite Ölçütleri	**Kalite Özelliği**	**Frekans Değerleri**
LCOM -1	Uyum	23
LOC	Boyut	23
Halstead n1	Karmaşıklık	20
Halstead n2	Karmaşıklık	20
CC	Karmaşıklık	17
DIT	Kalıtım	17
Tight Class Cohesion	Uyum	16
WMC	Karmaşıklık	15
RFC	Bağımlılık	15
NOC	Kalıtım	14
CBO	Bağımlılık	13
Number of Methods	Boyut	12
Loose Class Cohesion	Uyum	12
Message Passing Coupling	Bağımlılık	10
Cohesion	Uyum	10
LCOM -2	Uyum	10
LCOM -5	Uyum	10
Data Abstraction Coupling	Bağımlılık	9
LCOM -3	Uyum	9
LCOM -4	Uyum	8

İlgili çalışma, şimdiye kadar yapılmış bilimsel araştırmaları kullanarak geniş bir bakış açısı oluşturmayı hedeflemektedir. Çalışma neticesinde en sık araştırılan ve üzerinde çalışılan kalite özelliği bakım kolaylığı (maintainability) olarak görülmektedir (Şekil 2.1). Ayrıca kalite özelliklerinin de CK ölçütleri ile yüksek oranda örtüştüğü Çizelge 2.8'de görülmektedir.

Bu çalışmanın ortaya koyduğu bakış açısıyla; bakım kolaylığı kalite özelliğinin ölçümünün iyileştirilmesi, CK ölçütlerinin kendi aralarında önem sırasının çıkarılması gibi bilimsel araştırmalar yapma fikri ve kararı alınmıştır, 3. Bölüm kapsamında bu çalışmalar tarif edilecektir.

Çalışma Alanı	Kalite Özellikleri	Geliştirme Aşamaları						
		Proje Yönetimi	Gereksinimler	Mimari	Tasarım	Uygulama	Test	Bakım
Genel	Bakım-onarım	X	X	X	X	X		X
	Kararlılık	X	X	X	X	X		X
	Test edilebilirlik	X		X	X	X	X	
	Anlaşılabilirlik	X	X	X	X			
	Değişkenlik	X	X	X	X	X		X
	Değişime yatkınlık				X	X		
	Değiştirilebilirlik		X		X	X		
	Fonksiyonellik	X	X	X		X		
	Kullanılabilirlik	X	X	X	X	X		
	Modülerite			X		X		
	Tekrar kullanılabilirlik	X		X	X	X		
	Çözümlenebilirlik	X		X	X			
	Verimlilik	X		X	X			
	Uyum yeteneği	X		X				
	Bütünlük		X					
Web Uygulamaları	Fonksiyonellik	X	X	X		X		
	Bakım-onarım	X	X	X	X	X		X
	Tekrar kullanılabilirlik	X		X	X	X		
Gömülü Sistemler	Doğruluk		X					
	Bakım-onarım	X	X	X	X	X		X
	İzlenebilirlik	X	X					
	Bütünlük		X					
	Tutarlılık		X		X			
	Uçuculuk		X					
	Anlaşılabilirlik	X	X	X	X			
	Tekrar kullanılabilirlik	X		X	X	X		
	Test edilebilirlik	X		X	X	X	X	
	Dokümantasyon		X			X		
	Bellek gereksinimleri		X					
	Güvenilirlik					X		
	Uygunluk	X		X	X			
Bilgi Sistemleri	Fonksiyonellik	X	X	X		X		
	Verimlilik	X		X	X			
	Kurtarılabilirlik	X		X				
	Bakım-onarım	X	X	X	X	X		X
	İzlenebilirlik	X	X					
	Anlaşılabilirlik	X	X	X	X			
Dağıtık Sistemler	Bakım-onarım	X	X	X	X	X		X
	Kavranabilirlik				X	X		
	Yerellik			X	X			
	Değiştirilebilirlik		X		X	X		
Veritabanı Uygulamaları	Bakım-onarım	X	X	X	X	X		X
	Çözümlenebilirlik	X		X	X			
	Anlaşılabilirlik	X	X	X	X			
	Kullanılabilirlik	X	X	X	X	X		
	Kesinlik	X		X	X			
	Uygunluk	X		X	X			
	Sağlamlık				X			
	Tutarlılık		X		X			
	Eminlik				X			
	Bağlılık				X			
	Geçerlilik				X			

Şekil 2.1 Uygulama alanına göre geliştirme aşamasında kullanılan kalite özellikleri [28]

2.2 Yazılım Kalite Özellikleri ve Önemi

Yazılım sistemlerinin giderek artan karmaşıklığı, tasarım aşamasında daha tutarlı yaklaşımlar izlenmesi gerekliliğini doğurmaktadır. Bu yaklaşımlar, yazılımın yapısal ve davranışsal durumlarını etkilemektedir [29]. Günümüzde yazlımın davranışsal kısmını oluşturan fonksiyonel gereksinimlere ek olarak, yapısal kısma ait fonksiyonel olmayan gereksinimler de çok büyük önem kazanmıştır. Yapısal kısımdaki modüllerin genel yapı kalitesi, sistem kalitesiyle doğru orantılı durumdadır [4].

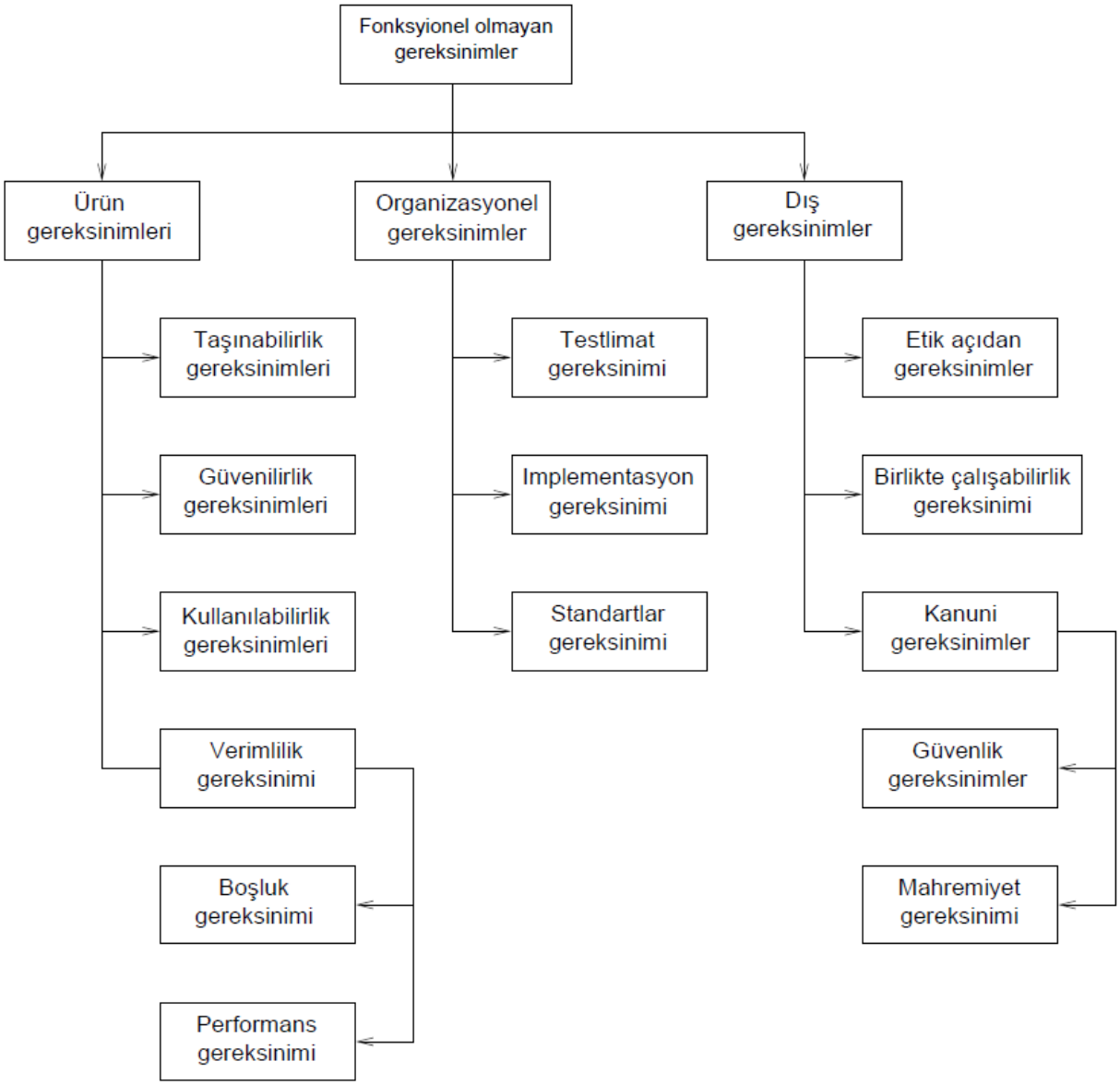
Yazılımın fonksiyonel gereksinimleri;

- Yazılımın fonksiyonelliğini ya da sistemin işleyişini anlatır,
- Yazılımın ne tür bir sistemde, hangi kullanıcılar tarafından kullanılacağını tarif eder (yazılım tipine bağlı olarak),
- Yazılım sisteminin nasıl çalışacağını detaylıca açıklar.

Yazılımın fonksiyonel olmayan gereksinimleri Şekil 2.2'de [31] gösterilmektedir. Bu gereksinimler özetle;

- Ürün gereksinimleri (Ürünün tam olarak nasıl davranması gerektiğini tarif eder; çalışma hızı, güvenilirlik vb. gibi)
- Organizasyonel gereksinimler (Uygulama gereksinimleri, kullanılacak süreç standartları vb. gibi)
- Dış gereksinimler (Sistemi ve geliştirme sürecini dışardan etkileyebilecek gereksinimler; yasal gereksinimler, birlikte çalışabilirlik gereksinimleri vb. gibi)

olarak sıralanabilirler.



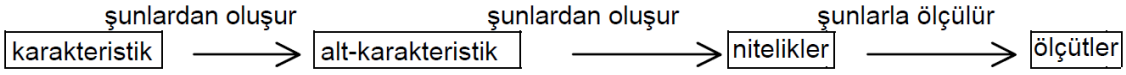
Şekil 2.2 Fonksiyonel olmayan gereksinimler [31]

2.2.1 Yazılım kalite modeli

ISO 9126-1 standardına göre kalite; bir ürünün ya da hizmetin, belirtilen veya ima edilen ihtiyaçları karşılama yeteneği olarak tanımlanmaktadır [21]. Farklı perspektiflerden de kalite şu şekillerde değerlendirilebilmektedir [32];

- Kullanıcı açısından; son ürünün kalitesi,
- Geliştirici açısından; yazılım geliştirme sürecinde farklı paydaşlar tarafından üretilen ara ürünlerin kalitesi;
- "Son-kullanıcı Yöneticisi" açısından; pazarlama gereksinimlerinin karşılanması ve kalitesi.

Kalite modeli ve model elemanları arasında Şekil 2.3'teki gibi bir bağ vardır;



Şekil 2.3 Kalite modeli ve kalite elemanları arasındaki ilişki

Bu bağlamda ISO 9126-1 standardı karakteristikleri, alt karakteristikleri ve nitelikleri Şekil 2.4'te gösterilmektedir [31].

Karakteristik	Alt-Karakteristik	Özellik	Karakteristik	Alt-Karakteristik	Özellik
Fonksiyonellik	Uygunluk	*İşlevsellik durumu*	**Verimlilik**	Zaman davranışı	*Zaman*
	Kesinlik	*Boyut*		Kaynak kullanımı	*Boyut*
	Birlikte çalışma	*Düzen durumu*			
	Güvenlik	*Düzen durumu*	**Bakım – onarım**	Değişkenlik, kararlılık, test edilebilirlik	*Boyut (Karmaşıklık)*
	Uyumluluk	*Standart uygulanma durumu*			
Güvenilirlik	Olgunluk	*Zaman*	**Taşınabilirlik**	Uyumluluk	*Düzen durumu*
	Hata toleransı	*Düzen durumu*		Kurulabilirlik	*Düzen durumu*
	Kurtarılabilirlik	*Düzen performans durumu*		Aynı anda çalışabilme	*Düzen durumu*
				Değiştirilebilirlik	*Değişebilir parça durumu*

Şekil 2.4 ISO 9126 Modeli; karakteristik, alt karakteristik ve nitelikleri

Literatür taraması çalışmasında, ISO 9126-1 kalite standardını, yazılım mimarisini ölçebilecek şekilde uyarlamaya çalışan araştırmalarla karşılaşılmıştır. Fitrisia and Hendradjaya, [7] yaptığı çalışmada ISO 9126-1 standardını bir envanter bilgi sisteminde kullanmak üzere OO ölçütlerini kullanarak uyarlamayı hedeflemişlerdir.

Çalışmada, yazılımın iç değerlendirmesine odaklanılarak Politeknik Caltex Riau adlı envanter bilgi sistemi için OO ölçütleriyle ölçümler yapılmıştır. Tüm sınıflar için sonuçlar değerlendirildiğinde kalıtım, bağımlılık ve boyut göstergelerinin iyi sonuçlar verdiği gözlemlenmiştir ve yöntemin yazılım kalitesi değerlendirmesinde kullanılabilir olduğu söylenmektedir.

Çalışma dahilinde tez konusunu da ilgilendiren en önemli kısım, ISO 9126 kalite standardı karakteristiklerinin ve alt karakteristiklerin tanımlanması ve detaylandırılması olmuştur. Her bir kalite faktörü için analiz yapılmaktadır.

2.2.1.1 Fonksiyonellik

Fonksiyonelliğin alt karakteristikleri;

- Suitability (Uygunluk)

Fonksiyonelliğin ne kadar uygulanabilir olduğunu tarif eder. Envanter bilgi sisteminden örnekle; veri kaydı, envanterdeki bir maddeyi yeniden konumlandırma, bakım, veri çıkarmadaki uygunluk durumudur. Gereksinimlerle gelen sınıf sayısının toplam gereksinimlere oranıyla ölçülebilir.

- Accuracy (Doğruluk)

Bilginin tutarlılığıdır. Envanter bilgi sisteminden örnekle; envanterin doğru numaralandırılması, girdileri doğru kategoride yer alması, bilginin güncelliği vb. gibi durumların tutarlılığıdır. Bilgi tutarlılığını sağlayan sınıf yüzdesi, bilgi doğruluğu yüzdesi gibi oranlarla ölçülür.

- Security (Güvenlik)

Kullanıcı doğrulama işlemleri gibi güvenlik işlemlerinin tutarlı olmasıdır. Şifre ve kullanıcı adı kullanımı, hesap işlemleri ile veri kullanımına göre kayıt tutulması, veri değişiklikleri ya da bakım-onarım işlemleri için yetkilendirme vb. gibi güvenlik prosedürlerinin olgunluğudur. Güvenlik doğrulamasını yöneten sınıfların yüzdesi ile ölçülür.

- Interoperability (Birlikte çalışabilirlik)

Farklı sistemlerin paralel şekilde çalışabilmesi durumudur. Envanter bilgi sisteminden örnekle; finans sistemi, envanter sistemi ve tedarik sisteminin eş zamanlı olarak birlikte çalışabilmesi durumudur. Birlikte çalışabilme metotlarını kullanan sınıfların yüzdesi ile ölçülür.

- Compliance (Şartları sağlama)

İstenilen şartların yüzde oranı ile ne kadar sağlandığı durumudur. Envanter bilgi sisteminden örnekle; envanter yazılım standartlarının ne kadarının uyarlandığı durumudur. OO ölçütleri ile ölçülemez.

2.2.1.2 Güvenilirlik

- Maturity (Olgunluk)

Yazılımın versiyon numarasına bakılarak anlaşılmaktadır. OO ölçütleri ile ölçülemez.

- Fault tolerance (Hata toleransı)

Yazılımın kullanımı esnasında; veri kaydı, bakım-onarım işlemi, veri tabanı kullanımı, donanımsal yapı gibi süreçlerde ortaya çıkan bozukluk sayısıdır. Karmaşıklık, kalıtım, boyut, uyumluluk ve bağımlılık gibi niteliklerle ölçülür.

- Recoverability (Kurtarılabilirlik)

Sistemin genelinde çıkan sorunlarda veri kaybına sebep olabilecek durumların kurtarılabilme durumudur. OO ölçütleri ile ölçülemez.

- Compliance (Şartları sağlama)

- Efficiency (Verimlilik)

Yazılımın performansının yüksek, kaynak kullanımının optimal seviyede olması durumudur. OO ölçütleri ile ölçülemez.

2.2.1.3 Bakım kolaylığı

- Analyzability (Çözümlenebilirlik)

Yazılım geliştirme dokümanlarına, kaynak koda vb. bakılarak sistemin analiz edilebilme durumudur. Kalıtım, karmaşıklık, boyut, uyumluluk ve bağımlılık nitelikleri ile ölçülür.

- Changeability (Değişebilirlik)

Yazılım geliştirme dokümanlarına, gereksinim analizi çalışmalarına bakılarak sistem üzerinde gerekli görülen değişikliklere ve modifikasyonlara uygunluk durumudur. Kalıtım, karmaşıklık, boyut, uyumluluk ve bağımlılık nitelikleri ile ölçülür.

- Stability (Kararlılık)

Sistemin genel olarak operasyonel haldeyken ve değişiklikler sonrasında kararlılık durumudur. Kalıtım, karmaşıklık, boyut, uyumluluk ve bağımlılık nitelikleri ile ölçülür.

- Testability (Test edilebilirlik)

Sistemin; test planı, test açıklamaları, test sonuçları gibi girdilere ve çıktılara olan yatkınlığıdır. OO ölçütleri ile ölçülemez.

- Compliance (Şartları sağlama)

2.2.1.4 Taşınabilirlik

- Adaptability (Uyarlanabilirlik)

Çoklu-platform için yazılımın çalışır şekilde uyarlanabilme durumudur. Kalıtım, karmaşıklık, boyut, uyumluluk ve bağımlılık nitelikleri ile ölçülür.

- Instalability, co-existance, replaceability (Kurulabilirlik, birliktelik, değiştirilebilirlik)

Yazılımın çalışacağı platforma kurulabilmesi, aynı anda başka bir yerde paralel çalışabilmesi gibi durumlardır. Direk olarak OO ölçütleri ile ölçülemez.

- Compliance (Şartları sağlama)

2.2.1.5 Kullanılabilirlik

- Understandability (Anlaşılabilirlik)

Sistemin dokümanlarla, tanıtım içerikleri ile, ürün açıklamaları ile anlaşılma ve kullanılır hale gelme durumudur. Aynı zamanda sistemin gelen ve giden mesajlarının anlaşılırlık durumudur [30]. OO ölçütleri ile ölçülemez.

- Learnability (Öğrenilebilirlik)

Kullanım kılavuzu gibi dokümanlarla sistemin işleyişinin öğrenilebilme durumudur. Toplam kullanım zamanı ve deneyim kazanma zamanı ile ölçülebilir [30]. OO ölçütleri ile ölçülemez.

- Operability (İşlerlik)

Sistemin girdi ve çıktılarının özelleştirilebilir olması, rahat müdahale edilebilir olması durumudur. OO ölçütleri ile ölçülemez.

- Attractiveness (İlgi çekicilik)

Yazılım sisteminin kullanıcıların ilgisini çekebilme ve kendini kullanılır hale getirebilme durumudur. OO ölçütleri ile ölçülemez.

- Compliance (Şartları sağlama)

Fitrisia and Hendradjaya, [7]'nin kendi çalışmaları için çıkardığı bu tanımlamalar dahilinde envanter bilgi sistemleri için ölçümler yapılarak "domain-specific", alana özel, bir kalite modeli uyarlaması yapılmaktadır. Çalışma sonucunda yapılan Pearson korelasyon analizi ile kalıtım, uyumluluk, bağımlılık, boyut ve karmaşıklık ölçütleri arasında bağlantı olduğu tespit edilmiştir (böyle olması da beklenmektedir).

2.3 Literatür Taraması Sonuçları ve Karşılaştırma

Tez çalışması öncesi yapılan literatür çalışmasında (2. Bölüm'de ele alınan ve alınmayan araştırmalarda); OO ölçüt değerlerinin olası kötü kod durumları ile olan bağlantıları [11], tasarım aşamasında OO ölçütleri kullanıldığı takdirde yazılım büyüklüğünün nasıl etkilendiği [19], OO ölçütleri ile yazılım kalite modelleri arasındaki bağlar [20; 24], yazılım ölçütleri için geniş çaplı bir haritalandırma-eşleştirme çalışması [28], kalite modellerinin değerlendirilmesi ve uyarlanması [21] gibi araştırmalar yapıldığı gözlenmiştir. Bu araştırmalar dahilinde geniş çaplı bilgi edinimi sağlanmıştır.

Tez kapsamında yürütülen çalışmada ise farklı olarak;

- Öncelikle, yapısal ve OO yazılım ölçütleri araştırılıp tanımlanmaktadır.

- Literatür çalışmasında da görüldüğü üzere, çok sık kullanılan ve önemsenen CK ölçüt kümesinin tanımlaması yapılmaktadır.
Kıral ve Ayyıldız, [33]'ün yaptığı çalışmada en çok dikkat edilmesi gereken CK ölçütleri belirlenip, önem sırasına göre analiz edilmektedir.

- LK ve McCabe's Complexity Suite ölçüt kümeleri incelenerek bir sonraki kısımda hangilerinin kullanıldığı tarif edilmektedir.

- ISO 9126 kalite standardı tekrar tarif edilip, bakım kolaylığı karakteristiğinin öneminden bahsedilmektedir.

- Son olarak, Kıral ve Ayyıldız, [34]'ün yaptığı çalışma ile bakım kolaylığı kalite karakteristiği ölçümü için yeni bir ölçüt kümesi önerisi anlatılmaktadır. Çalışma kapsamında kullanılan araçlardan, veri setinden, yöntemlerden bahsedilmektedir.

3 YÜRÜTÜLEN ÇALIŞMA

3.1 Yazılım Kalite Ölçütleri

Yazılım ölçütleri ilk olarak 1970'lerde önerilmiş olup hala üzerinde çalışmalar sürdürülmektedir [35]. Ölçütler, OO ölçütler ve yapısal ölçütler olarak ikiye ayrılabilirler. OO ölçütler sadece OO programlama dillerinde kullanılırken, yapısal ölçütler yapısal programlama dilleri ve OO programlama dillerinin her ikisi için de kullanılabilmektedirler [35].

Yazılım kalite ölçütleri, literatür çalışmasında da çok kez bahsedildiği gibi, yazılım kalite karakteristiklerinin ölçülmesinde kullanılmaktadırlar ve yazılım projesi ürünün başarılı olmasında en temel elemanlardır. Erken aşamalarda yazılım kalite ölçütlerinin doğru yorumlanmasıyla, ileri safhalarda gerçekleşebilecek geri dönüşü mümkün olmayan hataların önüne geçilebilmektedir.

3.2 Yapısal Yazılım Kalite Ölçütleri

Sık kullanılan yapısal kalite ölçütlerinden bazıları şöyledir;

- Line of Code (LOC); kaynak kod içeren satır sayısıdır.
- Comment Lines; açıklama (yorum) içeren satır sayısıdır.
- Blank Lines; kaynak kod ve açıklama içermeyen satır sayısıdır.
- Ratio Comment/Code; kaynak kod satırı başına açıklama satırı oranıdır.
- Functions; yazılımdaki fonksiyon sayısıdır.
- Inputs (FANIN); çağrılan program parçacıkları ve okunan değişkenlerin sayısı toplamıdır.
- Declarative Statements; bildirim yapan ifade sayısıdır.
- Executable Statements; uygulayıcı ifade sayısıdır.
- Total Operator; toplam operatör sayısıdır.
- Total Operands; toplam operand sayısıdır.

- Halstead Metrics Suite; eşsiz operatör & operand sayıları ve toplam operatör & operand sayıları kullanılarak karmaşıklık analizi yapılan bir ölçüt kümesidir [36].

- McCabe's Complexity Suite; yazılımın karmaşıklığını ölçen geniş bir ölçüt kümesidir. En çok kullanılan ölçütü CC, modülün karar yapısının karmaşıklığını gösterir. Ölçüt değeri büyüdükçe o modülün bakım yapılabilirliğinin düştüğü ve kusur eğiliminin git gide arttığı gözlemlenmektedir [37].

3.3 OO Yazılım Kalite Ölçütleri

OO yazılım kalite ölçütlerinden en sık kullanılan ölçüt kümesi 1994 yılında Chidamber-Kemerer adlı araştırmacılar tarafından önerilen CK ölçüt kümesidir. Ayrıca MOOD, QMOOD, LK gibi yine OO yazılım kalitesi ölçen ölçüt kümeleri de literatürde kullanılmaktadır.

3.3.1 CK ölçüt kümesi

OO tasarım için kullanılan bu ölçüt kümesi, bir sistemin bütün olarak değerlendirilmesinden ziyade, yazılımdaki sınıfları değerlendirmektedir. 6 adet ölçüt tanımlanmıştır;

- WMC: Bir sınıftaki tüm metotların karmaşıklığının toplamı olarak açıklanmaktadır. Buradaki sorun, karmaşıklık pek çok ölçütle ölçülebilmektedir (CC gibi). Burada hangi ölçüt değerinin daha önemli olduğu kararı geliştiriciye kalmaktadır.

WMC erken tasarım aşamasında çok iyi bilgi toplayamaz çünkü sınıf içeresindeki metotlar tanımlanmış ama henüz gerçekleştirilmemiş olabilir. Böyle durumlarda WMC değeri için sınıftaki tanımlı tüm metotların sayısını karmaşıklık değeri gibi varsaymak önerisi sunulmuştur. Bu şekilde kullanıldığında WMC artık karmaşıklık değil boyut ölçütü gibi davranmaktadır [38].

- RFC: Yerel metotların çağrılması durumunda, nesnenin tetikleyebileceği tüm metotların sayısıdır [38; 24]. Sınıfın test maliyeti hakkında fikir verir. Bir mesaj çok sayıda metodu tetikliyorsa, hata ayıklaması ve sınıf testi zorlaşıyor demektir. Sınıf karmaşıklığının da yüksek olduğunun bir göstergesidir [20].

- LCOM: Sınıf içerisindeki metotların uyumunun eksiklik oranını ölçen bir ölçüttür. Bu ölçütün çok tanımı vardır ancak genel olarak, benzerlik gösteren metot çiftlerini sıfır kabul eder ve benzer olmayan metot çiftlerinin değerlerini sıfırdan çıkararak ölçüm yapar. LCOM, metotların benzeşmeme değeri gibi hareket ettiğinden, sınıf tasarımındaki kusurların tespit edilmesine yardımcı olabilmektedir. Sınıf içindeki uyumluluk, kapsülleme özelliğini destekleyip karmaşıklığı azalttığı için aranılan bir özelliktir [38]. Literatürde LCOM2 [8], LCOM3 [8], LCOM4 [39;20] adlarıyla farklı LCOM ölçüt tanımlamaları da mevcuttur.

- CBO: Bir sınıfın bağlı olduğu diğer farklı sınıfların sayısıdır. Buradaki bağ, sınıfa ait metotları ya da nitelikleri kaç tane farklı sınıfın kullanıyor olduğudur [38]. Verimliliği ve yeniden kullanılabilirliği ölçmede kullanılmaktadır [24].

- DIT: Eğer bir sınıf başka bir sınıfın özellik veya davranışını paylaşıyorsa kalıtım var demektir. Alt bir sınıf, sadece bir üst sınıftan kalıtıma sahipse buna tekli kalıtım, eğer birden çok üst sınıftan kalıtımsa sahipse buna çoklu kalıtım denmektedir. Bir sınıf çoklu kalıtım içeriyorsa, DIT bu kalıtım ağacında kök noktadan en uzak noktaya kadarki uzunluktur [38]. DIT değerinin yüksek olması, test edilebilirliği azaltır, değerin düşük olması ise OO özelliklerden fazla yararlanılmadığını gösterir [40]. DIT ölçütü, yeniden kullanılabilirlik, verimlilik, anlaşılabilirlik, test edilebilirlik ölçümünde kullanılmaktadır.

- NOC: Bir sınıftan direk türetilmiş alt sınıfların sayısıdır. Eğer bir sınıf çok fazla alt sınıfa sahipse kalıtım özelliğinin yanlış kullanıldığını gösteriyor denilebilir. [20].

3.3.2 CK ölçütleri için önem sırası çalışması

OO yazılım kalite ölçütlerinden en sık kullanılanın CK ölçüt kümesi olduğundan bahsedilmişti. Kıral ve Erçelebi Ayyıldız, [33]'ün yaptığı yazılım kalite ölçütleri kıyaslama çalışmasında, güvenliğin çok kritik öneme sahip olduğu uzay ve havacılık alanından [66] 16 adet açık kaynak kodlu ve orta ölçekli OO yazılım projesi seçilerek, Understand statik kod analiz aracı yardımıyla CK ölçüt değerleri analizi yapılmıştır. Yazılım projeleri için özel şirketlerden, kamu kurum ve kuruluşlarından kaynak kod talebinde bulunulmuş ancak olumlu geri dönüş alınamamıştır. Bu nedenle çalışma kapsamındaki yazılım projeleri açık kaynak

kodlu olarak; NASA Open Source Software kütüphanesinden [54], ve GitHUB üzerinden edinilmiştir. Çalışma tutarlılığı açısından aynı programlama dilinde yazılan yazılım projelerinin seçilmesi planlanmıştır. Proje sayısı fazlalığı sebebiyle JAVA programlama dilindeki yazılımlar tercih edilmiştir.

Projeler ve projelere ait KLOC (Kilo Lines of Code) değerleri Çizelge 3.1'de gösterilmektedir.

Çizelge 3.1 16 adet JAVA açık kaynak kodlu OO yazılım projesi KLOC değerleri

Proje	KLOC	Proje	KLOC
CCSDS_MO_GraphicalEditor	11K	DAVEtools-master	15K
CCSDS_MO_TESTBEDS	25K	DERT-master	42K
CCSDS_MO_TRANS	14K	FEI-master	63K
GUSTO	13K	JavaGenes	32K
nmf-core	30K	jpf-core-master	114K
NMF_MISSION_OPS-SAT	32K	symbc-master	199K
NMF_MISSION_SOFTWARE_SI	12K	mct-master	124K
CCDD	75K	SpaceLaunchNow-Android-master	25K

3.3.2.1 Understand statik kod analiz aracı

Scientific Tools INC. tarafından geliştirilen Understand, kaynak kod analizi yaparak; detaylı ölçüt raporu, akış diyagramları, bağımlılık analizi tabloları gibi önemli bilgileri çıktı olarak sunmaktadır. Sektördeki önemli firmaların tercih etmesi, kod analizindeki hızı, kullanıcı dostu oluşu gibi sebepler göz önüne alınarak Understand statik kod analiz aracının çalışma kapsamında kullanımına karar verilmiştir [41].

Hızlı ve otomatik şekilde; fonksiyonlar, sınıflar, değişkenler gibi bileşenler için nasıl kullanıldıkları, nasıl çağırıldıkları, nasıl uyarlandıkları hakkında bilgiler verebilmektedir. Çağırım ağaçları, ölçüt değerleri, referanslar vb. gibi, kaynak kodla alakalı bilinmesi istenen bilgileri kullanıcıya sunabilmektedir. [41]

CK ölçütleri önem sıralaması çalışmasında kullanılan JAVA projelerinden bir tanesi için Understand statik kod analiz aracını tarif eden ekran görüntüleri adım adım EK-1'de sunulmaktadır.

Çalışma süresince kullanılacak ölçütler Understand kod analiz aracında kendi isimleri ile bulunmamaktadır. Bu nedenle aracın internet sitesinde, ölçütlerin Understand'deki karşılıkları açıklanmaktadır [41]. Ölçüt karşılıkları Çizelge 3.2'de gösterilmektedir.

Çizelge 3.2 Understand'de ölçütlerin karşılıkları

Ölçüt ismi	**Understand karşılığı**
CBO	CountClassCoupled
NOC	CountClassDerived
RFC	CountDeclMethodAll
DIT	MaxInheritanceTree
WMC	CountDeclMethod
LCOM	PercentLackOfCohesion
CC	Cyclomatic
Essential Complexity (EC)	Essential
Number of Inherited Methods (NIM)	CountDeclInstanceMethod
Number of Inherited Variables (NIV)	CountDeclInstanceVariable

CK ölçütleri önem sıralaması çalışması için Understand statik kod analiz aracından çıkacak değerlerin yorumlanabilmesi gerekmektedir. Bu sebeple ölçütlerin eşik-değerlerine ihtiyaç vardır.

CK ölçütleri için literatürde çeşitli eşik-değer önerileri yapılmıştır. Çizelge 3.3 Literatürde CK ölçütleri için eşik-değer önerilerine ilişkin çalışmaları ve bu çalışmalara ait eşit değerleri göstermektedir.

Çizelge 3.3 Literatürde CK ölçütleri için eşik-değer önerileri

İlişkili Çalışmalar	LCOM	DIT	CBO	NOC	RFC	WMC
[42]	3	6	9	3	36	30
[43]	1	6	8	6	35	15
[44]	Düşük	4	8	6	35	11
[45]	20	2	14	2	44	20
[46]	Düşük	4	94	5	10	108

Görüldüğü gibi herkes tarafından kabul görmüş genel bir eşik-değer bulabilmek çok güçtür. İlişkili çalışmalarda CK ölçütleri için proje bazlı olarak eşik-değer üretildiği gözlenmiştir. Singh and Kaur, [46] yaptıkları çalışmada, ölçüt değerlerinden elde edilen sonuçların ortalamasının ve standart sapmasının bulunup toplanmasıyla ilgili proje için eşik değer türeten bir yöntem sunmaktadırlar. Kullanılan formül Denklem 3.1'de verilmiştir.

T eşik değer, μ ortalama değer, Ω standart sapma değerlerini göstermektedir

$$T = \mu + \Omega \quad [46] \tag{3.1}$$

Kıral ve Erçelebi Ayyıldız, [33]'ün CK ölçütleri için önem sırası çalışmasında tek bir proje yerine 16 adet proje olduğundan, her bir proje için Denklem 3.1 ile elde edilen eşik değerlere μ + Ω işlemi tekrar uygulanarak uzay ve havacılık alanı için kullanılır olan genel bir eşik-değer kümesi üretimi sağlanmıştır [33]. Elde edilen değerler Çizelge 3.3 ile birleştirilerek Çizelge 3.4'te tekrar verilmektedir.

Understand statik kod analizi aracında LCOM ölçütü için yüzdelik bir oran kullanılmaktadır. Bu nedenle Çizelge 3.4'te son bulunan LCOM eşik-değeri için yüzdelik oranda bir limit belirlenmiştir.

Çizelge 3.4 CK ölçütleri için eşik değer üretimi [33]

İlişkili Çalışmalar	LCOM	DIT	CBO	NOC	RFC	WMC
[42]	3	6	9	3	36	30
[43]	1	6	8	6	35	15
[44]	Düşük	4	8	6	35	11
[45]	20	2	14	2	44	20
[46]	Düşük	4	94	5	10	108
[33]	%86.8	2,9	10,8	14,8	62,4	25,6

Elde edilen eşik-değerler kullanılarak 16 adet açık kaynak kodlu JAVA projesi için eşik değerleri aşan sınıfların frekansları ve proje içerisindeki tüm sınıf sayısına oranları bulunmuştur (Çizelge 3.5).

Çizelge 3.5 16 adet açık kaynak kodlu JAVA projesi için eşik-değer aşma frekansları

İlişkili Çalışmalar	LCOM	DIT	CBO	NOC	RFC	WMC	#Class
CCSDS_MO_Graphi.	13 (%14)	45 (%48)	11 (%12)	2 (%2)	30 (%32)	9 (%9)	93
CCSDS_MO_TESTBED	30 (%13)	18 (%8)	9 (%4)	1 (%0)	8 (%3)	18 (%8)	218
CCSDS_MO_TRANS	21 (%13)	37 (%24)	4 (%2)	0 (%0)	8 (%5)	7 (%5)	155
GUSTO	15 (%7)	4 (%2)	7 (%3)	0 (%0)	0 (%0)	9 (%4)	217
Nmf	14 (%4)	8 (%2)	25 (%8)	1 (%0)	1 (%0)	6 (%2)	332
NMF_MISSION_OPS-sa	42 (%9)	1 (%0)	5 (%1)	0 (%0)	4 (%1)	12 (%2)	482
NMF_MISSION_SOFT_	9 (%6)	0 (%0)	2 (%1)	0 (%0)	1 (%1)	7 (%5)	158
CCDD	92 (%10)	39 (%5)	82 (%10)	5 (%1)	9 (%1)	23 (%3)	842
DAVEtools	8 (%7)	15 (%14)	9 (%8)	0 (%0)	17 (%15)	8 (%7)	110
DERT	49 (%7)	132 (%19)	48 (%7)	4 (%1)	6 (%1)	22 (%3)	705
FEI	43 (%7)	2 (%0)	21 (%3)	0 (%0)	9 (%1)	33 (%5)	631
JavaGenes	24 (%4)	154 (%21)	73 (%10)	1 (%0)	8 (%1)	19 (%3)	717
Jpf-core	118 (%7)	232 (%13)	139 (%8)	9 (%0)	503 (%28)	81 (%4)	1810
symbc	242 (%10)	441 (%17)	241 (%10)	11 (%0)	675 (%26)	127 (%5)	2630
mct	249 (%10)	249 (%10)	155 (%6)	7 (%0)	75 (%3)	43 (%2)	2442
SpaceLaunchNow-	38 (%8)	23 (%5)	9 (%2)	0 (%0)	2 (%0)	10 (%2)	506

Çizelge 3.5'te görüldüğü gibi özellikle yüksek uyum, düşük bağımlılık fikrine dikkat edilmesi gerektiği sonucuna varılabilmektedir.

Kıral ve Erçelebi Ayyıldız, [33]'ün yaptığı çalışmada DIT eşik değeri 2,9 olarak belirlenmiştir ancak kalıtım derinliği için genelde 4-6 aralığında bir değer alınmakta olup bu ölçüt özelinde bir eşik-değer hesaplaması yapılmamaktadır. İncelenen 16 adet JAVA projesi için DIT değerinin çok nadiren 5 değerine çıktığı gözlemlenmiştir. Proje bazlı oluşturulan 2,9 eşik-değeri göz ardı edilip genelde

olduğu gibi 4-6 aralığında bir değer olan 5 değeri, DIT için eşik-değer olarak uygulanmıştır.

RFC ve WMC ölçütleri için proje bazlı olarak çok yüksek veya çok düşük eşik-değerler kullanıldığı Çizelge 3.4'te görülmektedir. 16 Java projesinden elde edilen ölçümlerde eşik-değerler genelde çok fazla aşılmamaktadır. Bazı tekil projeler yüksek aşım frekanslarına sahiptirler ve bu durum projeye özgü olarak değerlendirilebilir.

Bu istisnai durumlar LCOM ve CBO için geçerli olmamaktadır.

NOC ölçütü için, projeler genelinde çok sık alt sınıf türetilmediği gözlemlenmiştir. Çok aktif kullanımı olmadığı için ölçüm frekansları düşük çıktığı düşünülmektedir.

Sonuçlar değerlendirildiğinde, yazılım projesinde dikkat edilmesi gereken CK ölçütleri sıralaması; LCOM, CBO, DIT, WMC, RFC ve NOC olmaktadır. (Şekil 3.1)

Şekil 3.1 CK ölçütleri önem sırası

Yazılım geliştiriciler ve proje yöneticileri kod yazım esnasında en çok hata yapılan kısımları bildikleri takdirde erken safhada önlem alabilmektedirler. Bu durumun maddi manevi proje başarısını etkileyeceği düşünüldüğünden, bu çalışmanın araştırmacılara yol göstereceği düşünülmektedir.

3.3.3 LK ölçüt kümesi

Lorenz and Kidd ölçüt kümesi, yazılım tasarımının statik karakteristiğini ölçmek için tanımlanmıştır [47]. Bu karakteristikler; kalıtım, sınıfın sorumlulukları vb. gibidir [48].

Genero et al., [50], 5 adet gerçek hayat projesine LK ölçütlerini uygulayarak aşağıdaki gibi önemli önermeler elde etmiştir;

- Çok sığ ya da derin kalıtım hiyerarşisi kalite sorunlarının habercisidir.

- Hiç metot örneği oluşturulmaması ya da metodun çok fazla örneğinin oluşturulması optimal olmayan sorumluluk yüklenmesi demektir (NIM ölçütünden yola çıkılarak).

- Çok sayıda değişken örneği oluşturulması diğer sınıflarla bağımlılığı artırmakta ve tekrar kullanılabilirliği düşürmektedir. (NIV ölçütünden yola çıkılarak)

- Ortalama sınıf değişkeni sayısı düşük olmalıdır.

- Sınıf metodunun fazlalığı, sınıfların örnekler yerine çalışmasını doğurmaktadır. Yaratılan metot örnekleri ya da değişken örnekleri yerine sınıflar kendi başlarına çalışırlar.

- Metot bindirmesi yapmak – özellikle hiyerarşinin derin bölümlerinde – kötü alt sınıflandırma belirtisidir.

LK ölçüt kümesi elemanları Çizelge 3.6'da gösterilmiştir. [47]

Çizelge 3.6 LK ölçüt kümesi

	Ölçüt	**Açıklama**
Sınıf boyutu ölçütleri	PIM	Public Instance Methods; sınıftaki açık -public- yaratılan metot örneklerinin sayısını ölçer.
	NIM	Sınıfa ait açık, özel -private-, korumalı -protected- yaratılan tüm metot örnekleri sayısını ölçer.
	NIV	Sınıf içerisindeki açık, özel, korumalı tüm değişken örneklerinin sayısını ölçer.
	NCM	Sınıftaki tüm sınıf metotlarının toplam sayısını ölçer.
	NCV	Sınıftaki tüm sınıf değişkenlerinin toplam sayısını ölçer.
Sınıf kalıtım ölçütleri	NMO	Bir alt sınıf tarafından üstüne bindirim yapılan metot sayısını ölçer
	NMI	Alt sınıfa kalıtım yoluyla geçen metot sayısını ölçer.
	NMA	Alt sınıf tarafından tanımlanan metot sayısını ölçer.
	SIX	Her bir sınıf için şu şekilde hesaplanır. (NumberOfOverridenMethods * HierarchyNestingLevel) / TotalNumberOfMethods

3.3.4 McCabe's complexity suite ölçüt kümesi

Yazılım karmaşıklık ölçümleri için sunulan ölçüt kümesidir [37].

- Cyclomatic Complexity (CC)

Bu ölçüt setinden en çok kullanılan ve çevrimsel karmaşıklığı ölçen CC ölçütüdür. Yazılımda test edilebilirliği ve güvenilirliği düşük olan modüllerin bulunmasında kullanılmaktadır [10]. Graf üzerindeki düğüm, kenar ve modül sayıları kullanılarak aşağıdaki Denklem 3.2 ile hesaplanır [10].

$$v(G) = E - N + (2 * P) \tag{3.2}$$

v(G): CC ölçütü değeri

E: Kenar (edge) sayısı,

N: Düğüm (node) sayısı,

P: Modül sayısıdır.

- Essential Complexity (EC)

Bir diğer sık kullanılan McCabe ölçütü ise EC'dir. Ölçüt değeri, modül içerisindeki "unstructured construct" ların sayısıdır.

Denklem 3.3'te gösterildiği gibi hesaplanmaktadır [10; 49].

$$ev(G) = v(G) - M \tag{3.3}$$

eV(G): EC ölçütü değeri,

v(G): CC ölçütü değeri,

M: Modüldeki "structured construct" sayısıdır.

3.4 Bakım Kolaylığı Özelliği Ölçümü İçin Yazılım Ölçütleri Önerisi

Teknoloji sektörünün günlük yaşamın büyük bir parçası haline gelmesi ile kullanılan yazılımlar da kod ve insan gücü olarak sürekli genişlemektedir. Yazılım projesindeki büyüme beraberinde bakım-onarım masrafları, proje maliyeti, yazılım

geliştirme zamanı gibi çok önemli faktörün de artmasına yol açmaktadır. Bu ve benzeri faktörler en başından doğru öngörülemez ve yürütülemezse, sonradan düzeltilmesi mümkün olamayacak problemlerle karşılaşılmak kaçınılmazdır.

Önceki bölümlerde bahsedildiği üzere; müşteriler tarafından reddedilen, kullanılamayan, artan maliyetler yüzünden iptal edilen ya da yüksek bakım-onarım maliyetleri gerektiren yazılım projeleri ekonomiye büyük miktarda zarar vermektedirler [1; 2].

Yazılım kalitesi ölçülerek; yazılımın müşteri ihtiyaçlarını karşılama oranı, geliştiriciler için yazılımın anlaşılırlığı, yazılımın yapısal kalitesi, amaca uygun hazırlanmış yazılımın maliyet ve fiyat dengesi gibi faktörler erken safhalarda hesaplanıp uygulanması gereken önlemler için erken karar alınabilmektedir.

3.4.1 ISO 9126 kalite standardı

ISO 9126 kalite standardı ilk olarak 1991 yılında yazılım kalitesi değerlendirmek üzere ortaya çıkmıştır [51]. Orijinal doküman 13 sayfadan oluşmaktadır ve standardın temel halidir. ISO 9126 aşağıdaki kalite elementleri ile bilinmektedir [50];

- Verimlilik; bütünlük ve kesinlik özellikleriyle yazılımın müşteri hedeflerine uygunluğudur.

- Üretkenlik; müşteri hedefleri sağlanırken insan, bütçe gibi kaynakların gereksiz kullanılmaması durumudur.

- Güvenlik; yazılım, mülkiyet, iş, suç gibi faktörler için risklerden korunma seviyesidir.

- Memnuniyet; yazılımı kullanan bireyler için kullanıcı memnuniyetidir.

ISO 9126 standardı, 6 adet karakteristik ve 27 adet alt karakteristikten oluşmaktadır (Çizelge 3.7).

Çizelge 3.7 ISO 9126 kalite standardı

Karakteristik	Alt karakteristik
Fonksiyonellik	Uygunluk, doğruluk, birlikte çalışabilirlik, güvenlik vb.
Güvenilirlik	Olgunluk, hata toleransı, kurtarılabilirlik vb.
Kullanılabilirlik	Anlaşılabilirlik, işlerlik, ilgi çekicilik vb.
Verimlilik	Zaman kullanımı, kaynak kullanımı vb.
Bakım kolaylığı	Çözümlenebilirlik, değiştirebilirlik, tutarlılık, test edilebilirlik vb.
Taşınabilirlik	Uyarlanabilirlik, kurulabilirlik, yer değiştirilebilirlik vb.

Literatür taraması çalışmasında ISO 9126 alt karakteristikleri tarif edilmektedir. Bu kısımda ise ana karakteristiklerin kısa tarifleri sunulmaktadır [50].

- Fonksiyonellik: Yazılımın, kullanıcı ihtiyaçlarına göre belirlenen yazılım fonksiyonlarını sağlayabilmesidir.
- Güvenilirlik: Yazılımın, performans seviyesine önem verebilmesidir.
- Kullanılabilirlik: Yazılımın kullanılabilme kolaylığıdır.
- Verimlilik: Yazılımın çalışması esnasında fiziksel kaynakları etkin biçimde kullanabilmesidir.
- Bakım kolaylığı: Yazılımda değişiklik yapabilme kolaylığıdır.
- Taşınabilirlik: Yazılımın, gönderildiği farklı ortamlarla bağlantı kurabilmesidir. Direk olarak yazılımın çalışabilmesinden ziyade ne kadar uyum sağlayabildiği durumudur. Tam uyumlu olunan durumda, sistem tamamıyla çalışıyor denilebilir.

3.4.1.1 Bakım kolaylığı ölçümü neden önemli?

Yazılım, şirketlerin; teknolojik, rekabet gerektiren, organizasyonel ve pazarsal alanlardaki artan değişim oranlarına hem ayak uydurmasını sağlamaktadır hem de şirketleri bu değişim oranları için son derece zorlamaktadır [52]. Değişime ayak

uydurabilmek için bakım kolaylığı olan bir yazılım projesi üretmiş olmak hayli önem arz etmektedir.

Kollanus and Koskinen, [53]'ün yaptığı çalışmaya göre başarısız yazılım projelerinin ortalama %80'i, bakım-onarım faaliyetlerde bütçelerini aşmalarından dolayı projelerini sonlandırmaktadırlar. [53]

Bakım kolaylığı özelliği iyi yürütülmüş bir yazılım projesi ile [52];

- Yazılımın hangi parçalarının gerektiğinde tekrar kullanılabileceğine rahat karar verilebilmektedir.

- Yazılımın hangi parçalarının ne zaman ve ne şekilde bakıma alınacağı ya da geliştirileceğine rahat karar verilebilmektedir.

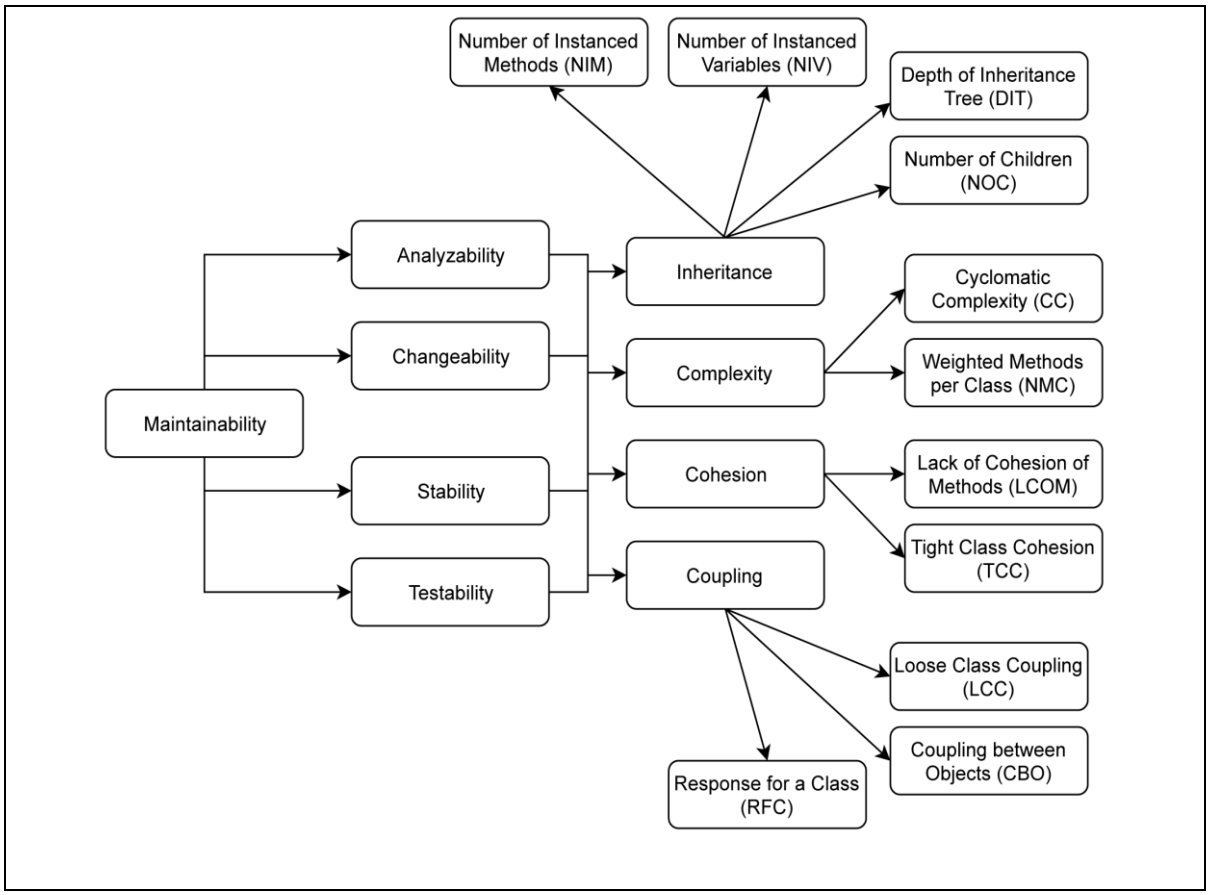
- Bakım-onarım faaliyetleri sırasında harcanması gereken efor rahatça kestirilebilmektedir.

- Modüllerin tekrar kullanılması sırasında gereken verilere daha rahat şekilde karar verilebilmekte ve ulaşılabilmektedir.

- Toplam yazılım maliyetinin ve bütçesinin ayarlanması kolaylaşmaktadır. Ayrıca projenin devamı için öngörülen bütçenin altına inme imkanları doğabilmektedir.

3.4.2 Bakım kolaylığı kalite karakteristiği ölçümü için önerilen ölçütler

Literatürde bakım kolaylığı ölçümü için Fitrisia and Hendradjaya, [7] yaptıkları çalışmada CK ölçüt kümesinin tamamını ve ek olarak Tight Class Cohesion (TCC), Loose Class Coupling (LCC) ölçütlerini kapsayan bir ölçüt seti önermişlerdir. (Şekil 3.2)



Şekil 3.2 Bakım kolaylığı ve OO ölçütleri

Kulkarni et al., [38] ise yaptığı çalışmada bakım kolaylığı ölçümü için CK ölçüt kümesinden WMC ve CBO ölçütlerinin kullanımını önermiştir. (Çizelge 3.8)

Çizelge 3.8 Kalite özelliklerini ölçen CK ölçütleri

Kalite Özelliği	**CK Ölçütleri**
Bakım kolaylığı	WMC, CBO
Tekrar kullanılabilirlik	WMC, CBO, DIT, NOC
Test edilebilirlik	RFC, CBO, NOC
Anlaşılabilirlik	RFC, CBO, DIT
Geliştirme Eforu	WMC, LCOM

Bu bilgiler ışığında ISO 9126 kalite standardının bakım kolaylığı karakteristiği ölçümü için WMC ve CBO ölçütleri ön planda tutularak bir sınıflandırma yapılacaktır. Bu sınıflandırma kullanılarak WEKA makine öğrenmesi aracı ile makine öğrenmesi yöntemi kullanılarak diğer ölçütlerin bakım kolaylığı analizi üzerine etkileri araştırılacaktır. Makine öğrenmesi yöntemi vasıtasıyla, yüksek öneme sahip olduğu tespit edilen ölçütler ile yeni bir bakım kolaylığı ölçüt kümesi önerisi yapılacaktır.

Literatür taraması çalışması kapsamında bu yeni ölçütlerden bazılarının DIT, NOC RFC gibi ölçütler olması öngörülmektedir.

3.4.3 Veri seti ölçümleri ve eşik-değer üretimi

Çalışma kapsamında 40 adet açık kaynak kodlu, JAVA programlama dilinde yazılmış, uzay ve havacılık alanına ait OO yazılım projesi, NASA Open Source Software Library [54], ve GitHUB üzerinden indirilmiştir. Projelerin tamamı 10-250 KLOC aralığında olup orta ölçeklidir. Çizelge 3.9'da yazılım projelerine ait KLOC değerleri gösterilmektedir.

Daha önce bahsedildiği gibi (Bölüm 3.3.2) güvenlik kritikliğinin yüksek öneme sahip olması sebebiyle [66], uzay ve havacılık alanında çalışma yapılması kararlaştırılmıştır. Yazılım projelerini incelemek ve üzerinde çalışmak maksadıyla özel şirketlerden, kamu kurum ve kuruluşlarından kaynak kod talebinde bulunulmuştur fakat olumsuz dönüşler alınmıştır. Bu sebeple açık kaynak kodlu kütüphaneler olan; NASA Open Source Software kütüphanesinden [54], ve GitHUB üzerinden yazılım projelerine erişim sağlanmıştır. JAVA dilinde yazılan projelerin sayıca fazla olması sebebiyle bu dilde yazılmış projeler edinilmiştir.

Çizelge 3.9 40 JAVA OO uzay ve havacılık projesi ve KLOC değerleri

#	KLOC	#	KLOC	#	KLOC	#	KLOC	#	KLOC
1	13K	9	52K	17	11K	25	114K	33	25K
2	21K	10	29K	18	44K	26	199K	34	78K
3	11K	11	17K	19	10K	27	124K	35	166K
4	36K	12	49K	20	75K	28	50K	36	20K
5	22K	13	49K	21	15K	29	119K	37	27K
6	54K	14	19K	22	15K	30	68K	38	42K
7	12K	15	14K	23	42K	31	115K	39	10K
8	16K	16	18K	24	63K	32	134K	40	32K

Yazılım projeleri, Kıral ve Erçelebi Ayyıldız, [33]'ün yaptığı çalışma ile aynı alandan seçilmiştir. Çünkü ilgili çalışmada CK ölçütleri için eşik değer önerisi yapılmaktadır. Bu çalışmada da uzay ve havacılık alanı projeleriyle çalışıldığı için ilgili eşik-değerlerden faydalanılacaktır.

3.4.3.1 CK, LK ve McCabe karmaşıklık ölçütleri için eşik-değer üretimi

Yazılım kalitesi ölçümünde kullanılan ölçütler için yapılan çalışmalarda farklı farklı eşik-değerler üretilip kullanılmaktadır.

CK ölçütleri için önerilen bazı eşik-değer önerileri Çizelge 3.10'da gösterilmektedir.

Çizelge 3.10 CK ölçüt kümesi için eşik-değer örnekleri

	CK Ölçütleri					
İlgili Çalışmalar	**LCOM**	**DIT**	**CBO**	**NOC**	**RFC**	**WMC**
[42]	3	6	9	3	6	30
[43]	1	6	8	6	35	15
[44]	Düşük	4	8	6	35	11
[45]	20	2	14	2	44	20
[46]	Düşük	4	94	5	10	108
[33]	(%85.6)	6	10	14	62	25

Bahsedildiği gibi, bu çalışmada CK ölçütleri için Çizelge 3.10'un en alt satırındaki (uzay ve havacılık alanı projeleri) yapılan çalışmaya ait CK eşik-değerleri kullanılacaktır [33].

McCabe's Complexity Suite ölçüt kümesinden CC ve EC ölçütleri için, sırasıyla 10 ve 4 eşik-değerleri önerilmektedir [55]. Yine McCabe ölçüt kümesinden Ratio C/C ölçütü için eşik-değer 0,16 tavsiye edilmiştir [56].

LK ölçüt kümesinden, sınıf boyutu ölçütleri olan NIV ve NIM ölçütleri seçilmiştir [48]. NIM eşik-değeri için 0,8 ve NIV eşik-değeri için 9 önerilmiştir [57].

Yürütülen çalışmada kullanılacak ölçütler ve eşik-değerleri toplu olarak Çizelge 3.11'de sunulmaktadır.

Çizelge 3.11 Bakım kolaylığı çalışmasında kullanılacak ölçütler ve eşik-değerleri

Ölçüt	**LCOM**	**DIT**	**CBO**	**NOC**	**RFC**	**WMC**	**CC**	**EC**	**C/C**	**NIM**	**NIV**
Eşik-Değer	%85,6	6	10	14	62	25	10	4	0,16	0,8	9

3.4.3.2 Yazılım projelerinde eşik-değerleri aşan sınıf frekansları

Çizelge 3.11'de gösterilen eşik-değerler için 40 adet açık kaynak kodlu, JAVA dilinde yazılmış, uzay ve havacılık alanına ait OO yazılım projesi, Understand

statik kod analiz aracıyla değerlendirilmiştir. Sınıfların eşik-değerleri aşma frekanslarının yüzdelik değerleri Çizelge 3.12'de verilmektedir.

Kıral ve Erçelebi Ayyıldız, [33]'ün yaptığı çalışmada aynı alana ait projelerde eşik-değerleri aşan sınıf frekanslarının %10 u çok fazla aşmadığı gözlemlenmiştir. Bu nedenle bakım kolaylığı çalışması dahilinde daha önce belirlenen ve Çizelge 3.8'de gösterilen WMC, CBO ölçütleri için; frekansı %10'un altında olan projeler 0, yani "başarılı"; %10'un üstünde olan projeler iste 1, yani "başarısız" olarak sınıflandırılmıştır. İlgili sınıflandırma WEKA makine öğrenme yazılımında kullanılmak üzere Çizelge 3.12'de gösterilmektedir. 40 adet projenin 27 tanesi başarısız, 13 tanesi ise başarılı olarak işaretlenmiştir.

Çizelge 3.12 Eşik-değerleri aşan sınıflar için frekans değerleri ve sınıflandırma

Proje #	CBO	NOC	RFC	DIT	LCOM	WMC	CC	EC	Ratio C/C	NIM	NIV	Sınıf
1	0.05	0.00	0.00	0.00	0.06	0.05	0.04	0.01	0.86	0.15	0.21	1
2	0.02	0.00	0.00	0.00	0.06	0.08	0.02	0.02	0.32	0.14	0.13	1
3	0.08	0.04	0.32	0.00	0.14	0.13	0.02	0.02	0.83	0.01	0.11	0
4	0.06	0.00	0.04	0.00	0.13	0.15	0.01	0.02	0.81	0.13	0.14	0
5	0.03	0.00	0.05	0.00	0.13	0.08	0.02	0.02	0.59	0.05	0.10	1
6	0.04	0.00	0.02	0.00	0.06	0.06	0.00	0.02	0.76	0.16	0.05	1
7	0.02	0.00	0.00	0.00	0.02	0.03	0.02	0.02	0.66	0.31	0.06	1
8	0.05	0.00	0.00	0.00	0.11	0.05	0.01	0.03	0.84	0.19	0.13	1
9	0.12	0.00	0.01	0.00	0.06	0.08	0.00	0.02	0.81	0.13	0.05	0
10	0.03	0.00	0.00	0.00	0.07	0.08	0.01	0.01	0.21	0.13	0.07	1
11	0.02	0.00	0.00	0.00	0.07	0.02	0.03	0.02	0.88	0.24	0.12	1
12	0.09	0.00	0.00	0.00	0.04	0.07	0.04	0.07	0.54	0.14	0.10	1
13	0.02	0.00	0.01	0.00	0.09	0.06	0.01	0.03	0.69	0.04	0.10	1
14	0.01	0.00	0.01	0.00	0.08	0.10	0.02	0.02	0.71	0.08	0.12	0
15	0.01	0.00	0.00	0.00	0.08	0.05	0.02	0.03	0.88	0.30	0.05	1
16	0.05	0.00	0.00	0.00	0.07	0.05	0.01	0.02	0.94	0.14	0.07	1
17	0.01	0.00	0.00	0.00	0.09	0.07	0.04	0.04	0.77	0.23	0.20	1
18	0.06	0.01	0.00	0.00	0.05	0.05	0.01	0.02	0.84	0.13	0.05	1
19	0.03	0.00	0.00	0.00	0.09	0.06	0.02	0.03	0.87	0.20	0.09	1
20	0.12	0.01	0.01	0.00	0.11	0.07	0.04	0.04	0.09	0.04	0.15	0
21	0.01	0.00	0.00	0.00	0.12	0.08	0.01	0.01	0.45	0.38	0.10	1
22	0.10	0.00	0.15	0.00	0.07	0.10	0.05	0.01	0.37	0.05	0.18	0
23	0.08	0.01	0.01	0.00	0.07	0.08	0.03	0.05	0.48	0.06	0.18	1
24	0.04	0.00	0.01	0.00	0.08	0.10	0.04	0.06	0.31	0.12	0.15	1
25	0.12	0.00	0.01	0.00	0.04	0.04	0.00	0.02	0.71	0.10	0.04	0
26	0.09	0.01	0.25	0.00	0.06	0.07	0.01	0.04	0.79	0.08	0.05	1
27	0.11	0.01	0.24	0.01	0.09	0.07	0.02	0.05	0.79	0.21	0.04	0
28	0.08	0.00	0.03	0.00	0.12	0.06	0.01	0.02	0.62	0.07	0.07	1
29	0.10	0.00	0.03	0.00	0.14	0.11	0.06	0.06	0.55	0.12	0.12	0
30	0.03	0.00	0.01	0.00	0.10	0.09	0.05	0.05	0.70	0.18	0.23	1
31	0.03	0.01	0.00	0.00	0.08	0.05	0.03	0.03	0.78	0.14	0.08	1
32	0.02	0.00	0.01	0.00	0.11	0.08	0.02	0.02	0.76	0.16	0.15	1
33	0.03	0.00	0.01	0.00	0.07	0.07	0.01	0.02	0.88	0.10	0.10	1
34	0.02	0.00	0.00	0.00	0.08	0.06	0.03	0.01	0.94	0.11	0.14	1
35	0.20	0.01	0.01	0.00	0.10	0.06	0.01	0.01	0.33	0.07	0.12	0
36	0.15	0.00	0.01	0.00	0.07	0.06	0.01	0.01	0.41	0.04	0.08	0
37	0.02	0.00	0.02	0.00	0.09	0.10	0.02	0.02	0.59	0.16	0.18	0
38	0.00	0.00	0.00	0.00	0.23	0.02	0.00	0.00	0.96	0.02	0.01	1
39	0.07	0.01	0.09	0.00	0.14	0.11	0.03	0.06	0.30	0.10	0.22	0
40	0.01	0.00	0.00	0.00	0.02	0.01	0.00	0.01	0.10	0.02	0.26	1

3.4.4 Bakım kolaylığı ölçümüne yeni ölçüt kümesi önerisi

40 adet açık kaynak kodlu, JAVA programlama dilinde yazılmış, uzay ve havacılık alanına ait OO yazılım projesi, Çizelge 3.12'de WMC ve CBO eşik-değerlerini aşma durumlarına göre sınıflandırılmıştır. Frekansı %10'un altında olan projeler 0,

yani “başarılı”, %10’un üstünde olan projeler iste 1, yani “başarısız” olarak sınıflandırılmıştır.

Bu sınıflandırma kullanılarak makine öğrenmesi çalışması yapılması planlanmıştır.

3.4.4.1 WEKA Machine Learning Software

WEKA, veri madenciliği görevleri için makine öğrenmesi algoritmalarının bir araya getirilmesiyle oluşan bir yazılımdır. Veri hazırlama, sınıflandırma, regresyon, kümeleme, veri madenciliği ve görselleştirme için araçlar içermektedir. [58]

Weka, makine öğrenme algoritmalarının kullanımı için çok hızlı ve kolay çözümler sunmaktadır. Analiz sonuçlarının da yorucu olmadan ve olabildiğince sorunsuz şekilde elde edilmesini sağlamaktadır. [67] Analizciler için çok çeşitli görselleştirme metotlarının da uygulanabilmesi kabiliyeti sonucu çalışma kapsamında WEKA makine öğrenmesi aracının kullanımına karar verilmiştir.

Yürütülen çalışma kapsamında kullanılan WEKA modülleri şu şekildedir.

- Preprocess; verinin hazırlandığı yerdir. WEKA’ya eklenen veri tablosundan sütun ekleme-çıkarma, sınıf sütunu belirleme gibi işlemler yapılabilmektedir. Bunların haricinde istenilen sütunlar için kapsamlı modifikasyonlar yapılabilmektedir. Örnek olarak, yürütülen çalışma kapsamında hazırlanan Çizelge 3.12’deki verilerin tamamı nümerik değerlerdir. Ancak sınıf sütunu normalde nümerik bir değerden ziyade “True” ya da “False” durumunu anlatmak amacıyla kullanılmaktadır. Bu nedenle WEKA “Filter” kısmından, “unsupervised” ve “attribute” başlıkları seçilerek, “NumericToNominal” filtrelemesi – sadece sınıf sütunu için ayarlanarak – uygulanmıştır. Böylece sınıf sütunu kategorizasyona uygun hale getirilmiştir.

- Classify; belirli sınıflandırma algoritması kullanılarak makine öğrenmesi testlerinin yapıldığı bölümdür. Proje veri setleri için öğrenme ve sonrasında öğrendiği sistematiği sınıflandırarak test etme işlemleri yapılmaktadır.

- Select attributes; sınıflandırma kısmında da olduğu gibi veri setinin öğrenilmesi ve test edilmesi aşamasında kullanılan özellikler için, içlerinden

gereksiz ya da daha yüksek öneme sahip olanların analizinin ve değerlemesinin yapılabildiği bölümdür.

WEKA arayüzüne ait görseller EK-2'de sunulmaktadır.

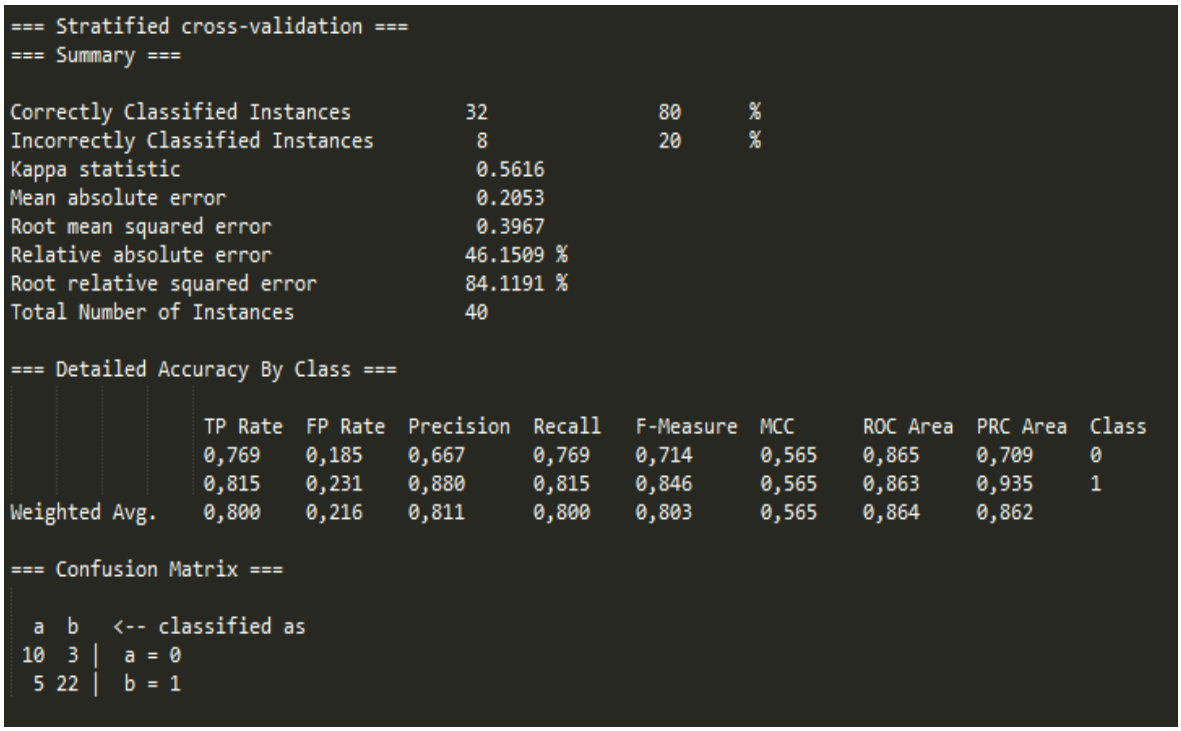
3.4.4.2 WEKA'da ham verinin yol gösterici olması için sınıflandırma testi

Bakım kolaylığı ölçümü için önerilen WMC ve CBO ölçütlerine göre sınıflandırması yapılan proje frekansları veri seti, baz bir değer alabilmek adına WEKA'da "Naive Bayes" algoritmasıyla test edilmiştir. Naive Bayes algoritması aşağıda tarif edilmektedir.

- Naive Bayes algoritması; sık kullanılan sınıflandırma algoritmalarındandır. Özetle bir olayın herhangi bir kategoriye ait olmasının olasılığıdır. Nitelikler birbirinden bağımsız olmalıdır. [59]

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}. \tag{3.4}$$

Çizelge 3.12'de gösterilen veri seti, "Naive Bayes" sınıflandırma algoritması ile WEKA'ya sunulup, önce veri öğrenmesi sona veri testi işlemleri gerçekleştirilmiştir. Sonuçlar Şekil 3.3'te gösterilmektedir.



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          32               80      %
Incorrectly Classified Instances         8               20      %
Kappa statistic                          0.5616
Mean absolute error                      0.2053
Root mean squared error                  0.3967
Relative absolute error                 46.1509 %
Root relative squared error             84.1191 %
Total Number of Instances               40

=== Detailed Accuracy By Class ===

               TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
               0,769    0,185    0,667      0,769    0,714      0,565    0,865     0,709     0
               0,815    0,231    0,880      0,815    0,846      0,565    0,863     0,935     1
Weighted Avg.  0,800    0,216    0,811      0,800    0,803      0,565    0,864     0,862

=== Confusion Matrix ===

  a  b   <-- classified as
 10  3 |  a = 0
  5 22 |  b = 1
```

Şekil 3.3 Naive Bayes algoritması ile veri seti sınıflandırma testi sonuçları

Yüzde 80 oranında veriler doğru sınıflandırılmıştır. Bu oran bile tek başına başarısız sayılabilir durumdadır. Yanlış sınıflandırılan verilerden 3 tanesi bakım kolaylığı açısından “başarılı” olmasına rağmen “başarısız” sayılmıştır, 5 tanesi ise “başarısız” olduğu halde “başarılı” sayılmıştır.

Burada “başarısız” ve “başarılı” olma durumları daha önce de belirtildiği üzere; sınıfların, WMC ve CBO ölçütlerine göre bakım kolaylığı yönünden hangi kategoride yer aldığıdır.

Yeni önerilecek ölçütler ile sınıflandırma oranında ilerleme kaydedilip kaydedilemeyeceğine bakılmak üzere sonraki aşamalar değerlendirilmektedir.

3.4.4.3 WEKA özellik seçimi işlemleri

Çizelge 3.12’de oluşturulan frekans değerleri, veri seti olarak WEKA’ya sunulmuştur. “Select attributes” bölümünden korelasyona göre özellik değerlendirmesi için “CorrelationAttributeEval” seçeneği seçilmiştir. “CorrelationAttributeEval” özelliği aşağıda tarif edilmektedir.

- CorrelationAttributeEval; Özellikler ve sınıflar arasında önemli bir bağ olup olmadığını Peason Korelasyon ölçümü yaparak değerleyen seçenektir.

Veri seti hazırlanırken WMC ve CBO özellikleri baz alınarak sınıflandırma yapıldığı için, diğer ölçütlerde de benzeri bir desen olup olmadığı araştırılmak istenmiştir. “CorrelationAttributeEval” değerlemesi seçeneği bu sebeple seçilmiştir.

İlgili özellik seçimi işleminin WEKA çıktısı Şekil 3.4‘te gösterilmiştir.

```
Attribute Evaluator (supervised, Class (nominal): 12 maint):
    Correlation Ranking Filter
Ranked attributes:
 0.637    1 cbo
 0.503    6 wmc
 0.37     3 rfc
 0.306    2 noc
 0.303   10 NIM
 0.237    9 ratio c/c

Selected attributes: 1,6,3,2,10,9 : 6
```

Şekil 3.4 Korelasyon sıralamasına göre önerilen ölçütler.

WEKA, özellik seçimi işlemleri sonrasında ön görüleceği üzere ilk iki sırada CBO ve WMC’yi önermektedir. Ancak ilişkili olabilme ihtimali olan başka ölçütlerin de olabileceğini söylemektedir. Bunlar önem sıralarına göre RFC, NOC, NIM ve Ratio C/C’dir. Sonraki aşamada elde edilen bu 6 ölçüt kullanılarak bir sınıflandırma yapıldığında başarım oranları araştırılacaktır.

3.4.4.4 WEKA’da elde edilen yeni ölçütlerle Naive Bayes algoritması kullanılarak veri setinin öğrenilmesi ve testi

Önceki aşamalardaki çalışmalarla elde edilen, bakım kolaylığı ölçümü için anlamlılık ihtimali bulunan yeni ölçüt seti WMC, CBO, RFC, NOC, NIM ve Ratio C/C kullanılarak, veri seti tekrar “Naive Bayes” algoritmasıyla sınıflandırılmıştır. Sonuçlar Şekil 3.5’te gösterilmektedir.

```
Correctly Classified Instances          36               90      %
Incorrectly Classified Instances         4               10      %
Kappa statistic                          0.7721
Mean absolute error                      0.1352
Root mean squared error                  0.288
Relative absolute error                 30.6223 %
Root relative squared error             61.4892 %
Total Number of Instances               40

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,846    0,074    0,846      0,846    0,846      0,772    0,943     0,872     0
                 0,926    0,154    0,926      0,926    0,926      0,772    0,943     0,975     1
Weighted Avg.    0,900    0,128    0,900      0,900    0,900      0,772    0,943     0,942

=== Confusion Matrix ===

  a  b   <-- classified as
 11  2 |  a = 0
  2 25 |  b = 1
```

Şekil 3.5 Yeni ölçüt seti kullanılarak Naive Bayes sınıflandırması sonuçları

Doğru sınıflandırma oranının %90'a çıktığı görülmektedir. %10'luk bu artış ile daha önce bakım kolaylığı ölçümü için Kulkarni et al., [38] tarafından önerilen WMC ve CBO ölçütlerinden daha başarılı bir sonuç elde edildiği ilk etapta görülmektedir.

Daha detaylı sonuç analizi 5. Bölüm'de, Sonuç kısmında incelenmektedir.

4 TARTIŞMA

40 adet, JAVA programlama dilinde yazılmış uzay ve havacılık alanına ait OO yazılım projesi için, bakım kolaylığı ölçümünü iyileştirmek üzere ölçüt değerleri belirlenmiştir. Literatürde bu ölçütler için eşik-değer önerileri araştırıp çalışmada kullanılmak üzere eşik-değer seti toparlanmıştır.

Bu ölçütlere göre Understand statik kod analiz aracıyla her bir proje için eşik-değerleri geçen sınıfların frekansları bulunarak bir veri seti elde edilmiştir. Bu veri seti, literatürde bakım kolaylığı ölçümü için önerilen çalışmalar kapsamında WMC ve CBO ölçütlerine göre “başarısız” ya da “başarılı” olarak sınıflandırılmıştır. 40 adet projenin 27 tanesi başarısız, 13 tanesi başarılı olarak işaretlenmiştir.

İlgili sınıflandırmalar için veri seti, WEKA makine öğrenme yazılımı yardımıyla ve “Naive Bayes” algoritması kullanılarak test edilmiştir. WMC ve CBO ölçütleri baz alınarak yapılan bu çalışmada %80 oranında başarılı sınıflandırma yapılmıştır.

Veri seti bir sonraki adımda WEKA özellik seçimi (attribute selection) işlemine tabi tutulmuştur. Özellik seçimi sonrası WEKA, WMC ve CBO’ya ek olarak RFC, NOC, NIM ve Ratio C/C ölçütlerinin de anlamlı olabileceği sonucunu vermiştir.

Anlamlı olabileceği öngörülen ölçütler kullanılarak tekrar “Naive Bayes” algoritması ile sınıflandırma yapıldığında, başarılı sınıflandırma oranı %90’a çıkmıştır.

%10’luk başarı artışı bulunmaktadır. Bakım kolaylığı kalite karakteristiğinin ölçümü için WMC, CBO, RFC, NOC, NIM ve Ratio C/C ölçütlerinin kullanılabilirliği mümkün olarak görülmektedir.

Çalışmanın tutarlılığını test etmek amacıyla, WEKA özellik seçimi (attribute selection) işleminin hassasiyeti düşürülerek 8 adet ölçüt seçmesi sağlanmıştır. WMC, CBO, RFC, NOC, NIM ve Ratio C/C’ye ek olarak DIT ve LCOM ölçütlerinin de hesaba katıldığı Naive Bayes sınıflandırması sonucunda başarı oranının %87.5’e düştüğü gözlenmiştir. Özellik seçimi sonuçları Şekil 4.1’de ve Naive Bayes sınıflandırması sonuçları Şekil 4.1’de gösterilmektedir.

```
Attribute Evaluator (supervised, Class (nominal): 12 maint):
    Correlation Ranking Filter
Ranked attributes:
 0.637    1 cbo freq
 0.503    6 wmc
 0.37     3 rfc freq
 0.306    2 noc freq
 0.303   10 NIM
 0.237    9 ratio c/c
 0.231    4 dit freq
 0.177    5 Lcom

Selected attributes: 1,6,3,2,10,9,4,5 : 8
```

Şekil 4.1 İki yeni ölçütün özellik seçimi ile elde edilmesi

```
Correctly Classified Instances          35               87.5    %
Incorrectly Classified Instances         5               12.5    %
Kappa statistic                          0.7093
Mean absolute error                      0.1379
Root mean squared error                  0.2938
Relative absolute error                 31.2151 %
Root relative squared error             62.7086 %
Total Number of Instances               40

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,769    0,074    0,833      0,769    0,800      0,711    0,946     0,873     0
                 0,926    0,231    0,893      0,926    0,909      0,711    0,946     0,977     1
Weighted Avg.    0,875    0,180    0,874      0,875    0,874      0,711    0,946     0,943

=== Confusion Matrix ===

  a  b   <-- classified as
 10  3 |  a = 0
  2 25 |  b = 1
```

Şekil 4.2 Seçile 8 adet ölçüt ile Naive Bayes sınıflandırması

5 SONUÇ

Çalışma sonucu olarak, %90 başarım ile WEKA'nın doğru sınıflandırma yaptığı görülmektedir. Burada en önemli nokta, yanlış sınıflandırılan projelerdir.

İlk sınıflandırma sonucunda, 3 proje başarılı olduğu halde başarısız ve 5 proje başarısız olduğu halde başarılı olarak sınıflandırılmıştı. Bu durum ikinci sınıflandırmada değişti ve 2 proje başarılıyken başarısız, 2 proje ise başarısızken başarılı olarak sınıflandırıldı.

Bu durum sağlık sektöründen bir senaryoyla ilişkilendirilirse daha anlaşılır olabilmektedir.

Hasta değilken, hasta olarak sınıflandırılan kişiler (sayıları çok yüksek olmadığı sürece) çok büyük sorun teşkil etmemektedirler. Çünkü çelişkili bu gibi durumlarda hastalar yapılan bir muayene sonucunda doğru sınıflandırmaya dahil edilebilirler. Bu tip yanlış sınıflandırmalar (sayısı mümkün olduğunca az olduğu sürece) riske girmemek adına kabul edilebilir bir durum sayılabilirler.

Ancak hastayken, hasta değil şeklinde sınıflandırılan kişiler için aynı durum geçerli olmamalıdır. Hasta olan bir kişi sınıflandırma yoluyla hasta değil raporu alırsa, hastalığı kontrol edilemeden süreç o kişi için sonlanacaktır. İdeal olarak bu yanlış sınıflandırmanın sıfır olması istenmektedir. Ancak sıfır olamadığı durumlar için bu yanlış sınıflandırma sayısı ne kadar azsa o kadar iyidir.

Tekrar ilgili çalışmaya dönülecek olursa, bakım kolaylığı açısından başarılı olup başarısız denilen proje sayısı 2'dir. Bu sayı aslında yeterince düşüktür, ilgili 2 proje tekrar gözden geçirilebilir. Yanlış sınıflandırmaya sebep olan durumlar varsa, belki bu durumların düzeltilmesi sağlanır ve projeler kendi sınıflarına eklenirler.

Başarısız olup başarılı denilen 2 proje ise, bu çalışmadaki "açık" noktadır. Bu yanlış sınıflandırma daha önce belirtildiği gibi idealde sıfır olmalıdır. Ancak ilk "Naive Bayes" ve İkinci "Naive Bayes" uygulaması arasında da bu yanlış sınıflandırma için iyileşme olduğu görülmektedir. Sayının sıfıra yakın (2) oluşu da ele alındığında genel bir başarımdan bahsedilebilmektedir.

Bakım kolaylığı kalite ölçümü için literatürde önerilen WMC ve CBO ölçütleriyle elde edilen %80 başarım, bu çalışma ile önerilen ölçütler kullanılarak %90'a çıkarılmış ve sınıflandırma hatalarında da iyileşmeler gözlemlenmiştir.

İleri aşamalarda proje sayıları artırılarak başarım oranındaki istikrarın korunması ya da başarım miktarının yükseltilmesi hedeflenebilir olarak görülmektedir. Bakım kolaylığı çalışması için önerilen ölçütlere eklemeler ya da çıkarmalar yapılıp yapılamayacağı da araştırılabilir konular arasında düşünülmektedir.

- Araştırma Sorusu Cevabı 1: Literatür taraması ve Kıral ve Erçelebi, [33]'ün CK ölçütleri ile ilgili yaptığı çalışma sonucunda, CK ölçütlerinin OO yazılım kalitesindeki önemi ve sıklıkla kullanıldığı anlaşılmıştır.

- Araştırma Sorusu Cevabı 2: Yazılım bakım kolaylığı kalite özelliği için yürütülen çalışma neticesinde, yeni bir ölçüt kümesi önerilmiştir ve literatürde kullanılması önerilen yöntemle kıyaslama yapılmıştır. Yeni önerilen ölçüt kümesinin verdiği sonuçlar tutarlı ve kullanılır görülmektedir.

Çizelge 5.1'de iki Naive Bayes uygulaması sonuçları gösterilmektedir.

Çizelge 5.1 Literatür ve yürütülen çalışma karşılaştırması

WMC ve CBO için Naive Bayes			**Yeni ölçüt kümesi için Naive Bayes**		
Doğru Sınıflandırma		%80	**Doğru Sınıflandırma**		%90
Confusion Matrix	**Başarılı**	**Başarısız**	**Confusion Matrix**	**Başarılı**	**Başarısız**
Başarılı	10	3	**Başarılı**	11	2
Başarısız	5	22	**Başarısız**	2	25

Gelecek çalışmalarda öncelikli olarak; aynı dilde yazılmış uzay ve havacılık alanına ait proje sayısının artırılması hedeflenmektedir. Proje sayısının artırılmasını takiben, bu çalışmada uygulanan metotlarla yeni ölçüt kümeleri bulunup bulunamayacağının araştırılması ve çalışmada önerilen ölçütlerle kıyaslama yapılması planlanmaktadır.

Ayrıca bu çalışma kapsamında ele alınan ISO 9126 kalite standardına ek olarak, IEEE 1219-1998 Yazılım Bakım Standardı [68] ve ISO/IEC/IEEE 14764-2006 Uluslararası Yazılım Mühendisliği – Yazılım Yaşam Döngüsü Süreçleri / Yazılım Bakım Standardı [69] incelenerek, bu standartlar kapsamında yazılım bakım-onarım ölçümü metotlarının araştırılması planlanmaktadır.

KAYNAKLAR LİSTESİ

[1] Internet: nist.gov, 2002,
'The Economic impacts of inadequate infrastructure for software testing'. https://www.nist.gov/sites/default/files/documents/director/planning/report02-3.pdf. [Accessed: 10- Oct- 2018].

[2] Internet: Software Testing Tools for Continuous Testing Webpage, 2017, https://www.tricentis.com/wp-content/uploads/2018/01/20180119_Software-Fails-Watch_Small_Web.pdf. [Accessed: 07- Oct- 2018].

[3] KAN, Stephen H., Metrics and Models in Software Quality Engineering, 2nd Edition, Addison-Wesley Longman Publishing Co., Boston, s.22, 2004.

[4] LOSAVIO, F. and CHIRINOS, L. and MATTEO, A. and LEVY, N. and RAMDANE-CHERIF, A., ISO quality standards for measuring architectures, Journal of Systems and Software, vol.72, s.209-223, 2004.

[5] SUNDAY, D.A., Software Maintainability - a new "ility", Northrop Corp., Proceedings, Annual R&M Symposium, 1989, s.50-51, 1989.

[6] ZHUO, F. and LOWTHER, B and OMAN, P and HAGEMEISTER, J., Constructing and testing software maintainability assesment models, First International Software Metrics Symposium, 1993 Proceedings, s.61-70, 1993.

[7] FITRISIA, Y. and HENDRADJAYA, B., Implementation of ISO 9126-1 quality model for asset inventory information system by utilizing object oriented metrics, IEEE International Conference on Electrical Engineering and Computer Science, Bali, Indonesia, s.229-234, 2014.

[8] CHIDAMBER, S. and KEMERER, C., A metrics suite for object-oriented design, IEEE Trans. Software Eng., vol. 20, no. 6, s.476-493, 1994.

[9] BANSIYA, J. and DAVIS, C., A hierarchical model for object-oriented design quality assesment, IEEE Trans. Software Eng., 28(1), s.4-17, 2002.

[10] McCABE, T. J., A complexity measure, In ICSE '76: Proceedings of the 2nd international conference on software engineering, IEEE Computer Society Press, 1976.

[11] FONTANA, A. and FERME, V. and MARINO, A. and WALTER, B. and MARTENKA, P., Investigating the impact of code smells on system's quality: an empirical study on systems of different application domains, 2013 IEEE International Conference on Software Maintenance, Eindhoven, s. 260-269, 2013.

[12] TEMPERO, E. et al, The qualitas corpus: a curated collection of java code for empirical studies, Proc. of 17th Asia Pasific Software Engineering Conference (ASPEC 10), IEEE, s.336-345, 2010.

[13] Internet: iPlasma, http://loose.upt.ro/iplasma/index.html. [Accessed: 20- Aug- 2018].

[14] Internet: Google Code Pro Analytix, http://code.google.com/intl/en-EN/javadevtools/codepro/doc/index.html. [Accessed: 20- Aug- 2018].

[15] Internet: Eclipse metrics plugin, http://eclipse-metrics.sourceforge.net. [Accessed: 20- Aug- 2018].

[16] ROY, C. And CORDY, J., NiCad: Accurate detection of near-miss intentional clones using flexible pretty-painting and code normalization, Proc. of the 16th IEEE International Conference on Program Comprehension (ICPC 2011), s.172-181, 2011.

[17] ABREU, F.B.E and CARAPUCA, R., Object-oriented software engineering: measuring and controlling the development process, in: Proceedings of the 4th international conference on software quality, 1994.

[18] MARTIN, R.C., Agile software development, principles, patterns and practices, Prentice Hall, 2002.

[19] TANRIÖVER, Ö. Ö. and ERYIĞIT, R., An emprical analysis of early object oriented design metrics in relation to code size, 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, s.62-65, 2015.

[20] ERDEMİR, U. ve TEKİN, U. ve BUZLUCA, F., Nesneye dayalı yazılım metrileri ve yazılım kalitesi, Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS08), s.249- 258, İstanbul, 2008.

[21] ISO/IEC 9126-1: Information Technology – Software Product Quality – Part 1: Quality Mode. ISO/IEC JTC1/SC7/WG6 (1999).

[22] BOOCH, G., Object oriented design with applications, Redwood City, CA: Benjamin/Cummings, s.547, 1991.

[23] ABREU, F.B.E and PEREIRA, G. And SOURSA, P., Coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems, Proc. Euromicro Conf. Software Maintenance and Reeng., s.13-22, 2000.

[24] CALP, H. M. ve ARICI, N., Nesne yönelimli tasarım metrikleri ve kalite özellikleriyle ilişkisi, Politeknik Dergisi Cilt: 14 Sayı: 1, s.9-14, 2011.

[25] JETTER, A., Assessing software quailty attributes with source code metrics, Diploma Thesis, University of Zurich Department of Informatics, Zurih, 2006.

[26] MUSTAFA, K. And KHAN, R.A., Software testing: concept and practices, India, Lucknow, 2007.

[27] KAUR, J. and VERMA, P. And THAPAR, S., Software quality metrics for object-oriented environments, Proceedings of National Conference on Challenges & Oppurtunities in Information Technology (COIT-2007), RIMT-IET, Mandi Gobindgarh, s.13-16, 2007.

[28] ARVANITOU, E. M. and AMPATZOGLOU, A. and CHATZIGEORGIOU, A. and GALSTER, M. and AVGERIOU, P., A mapping study on design-time quality attributes and metrics, Journal of Systems and Software, vol.127, s.52-77, 2017.

[29] BRITTON, C., Architectures and middleware. Addison-Wesley, Boston, 2001.

[30] SANTOS, C. and NOVAIS, T. and FERREIRA, M. and ALBUQUERQUE, C. and de FARIAS, I. H. and FURTADO, A. P. C., Metrics focused on usability ISO 9126 based, 2016 11th Iberian Conference on Information Systems and Technologies (CISTI), Las Palmas, s.1-3., 2016.

[31] Internet: inf.ed.ac.uk, The University of Edinburgh, 2002, 'CS2 Software Engineering note 2, CS2Ah Autumn 2004'. http://www.inf.ed.ac.uk/teaching/courses/cs2/LectureNotes/CS2Ah/SoftEng/se02.pdf. [Accessed: 24- Nov- 2018].

[32] SANTOS, C. and NOVAIS, T. and Ferreira, M. and ALBUQUERQUE C. and FARIAS de I. H. and FURTADO A.P.C., Metrics focused on usability ISO 9126 based, 2016 11th Iberian Conference on Information Systems and Technologies (CISTI), Las Palmas, s. 1-3., 2016.

[33] KIRAL, A. ve ERÇELEBİ AYYILDIZ, T., Yazılım kalite metriklerinin kıyaslanması: örnek bir olay incelemesi, 12. Ulusal Yazılım Mühendisliği Sempozyumu (UYMS'18), İstanbul/Türkiye, 2018.

[34] KIRAL, A. ve ERÇELEBİ AYYILDIZ, T., SOFTWARE METRICS PROPOSAL to measure maintainability., 3rd International Conference on Theoretical and Applied Computer Science and Engineering, Ankara, Turkey, 2018.

[35] ÇATAL, Ç. ve DİRİ, B., Yazılım metriklerini kullanarak düşük kalitesi / yüksek riskli modüllerin otomatik tespiti, YKGS2008: Yazılım Geliştirme Araçları, İstanbul, 2008.

[36] HALSTEAD, M., Elements of software science, Elsevier North-Holland, New York, 1977.

[37] Internet: McCabe IQ – Software Metrics Glossary, http://www.mccabe.com/iq_research_metrics.htm. [Accessed: 14-Oct-2018].

[38] KULKARNI, U. L. and KALSHETTY, Y. R. and ARDE, V. G., Validation of ck metrics for object oriented design measurement, 2010 3rd International Conference on Emerging Trends in Engineering and Technology, Goa, s.646-651, 2010.

[39] HITZ, M. and MONTAZERI, B., Chidamber and kemerer's metric suite: a measurement theory perspective, IEEE Transactions on Software Engineering, Vol. 4, s.267-271, 1996.

[40] Internet: McCabe Software, Using code quality metrics in management of outsourced development and maintenance. http://www.mccabe.com/pdf/McCabeCodeQualityMetrics-OutsourcedDev.pdf. [Accessed: 14-Jan-2019].

[41] Internet: SciTools.com, Understand Code Analysis Tool, https://scitools.com/. [Accessed: 30-Aug-2018].

[42] ANTONY, P., Predicting reliability on software using thresholds of ck metrics, International Journal of Advanced Networking, s.1778-1785, 2013.

[43] ZHOU, Y. and LEUNG, H., Empirical analysis of object-oriented design metrics for predicting high and low severity faults, IEEE Transactions on Software Engineering, s.771–789, 2006.

[44] MAGO, J. and KAUR, P., Analysis of quality of the design of the object oriented software using fuzzy logic. International Conference on Recent Advances and Future Trends in Information Technology (iRAFIT2012) Proceedings Published in International Journal of Computer Applications® (IJCA), s.21–25, 2012.

[45] BAKAR, D. and SULTAN, A. and ZULZALIL, H. and DIN, J., Predicting maintainability of object-oriented software using metric threshold. Information Technology Journal, vol. 13, s.1540-1547, 2014.

[46] SINGH, S. and KAUR, M., Deriving and validating software metrics threshold values for design errors, International Journal of Engineering Technology and Scientific Research, vol. 2, 2016.

[47] LORENZ, M. and KIDD, J., Object-oriented software metrics: a practical guide, Prentice Hall Englewood Cliffs. New Jersey, 1994.

[48] GENERO, M. and PIATTINI, M. and CALERO, C., A survey of metrics for UML class diagrams, Journal of Object Technology, vol. 4, no. 9, Zurich, 2005.

[49] Internet: L09 McCabe and Complexity – argo49/SOEN384-Fall2014 Wiki - GitHUB https://github.com/argo49/SOEN384-Fall2014/wiki/L09-McCabe-and-Complexity. [Accessed: 20-Aug-2018].

[50] SUGIYANTO and ROCHIMAH, S., Integration of DEMATEL and ANP methods for calculate the weight of characteristics software quality based model ISO 9126, 2013 International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, 2013, s.143-148, 2013.

[51] BEHKAMAL, B. and KAHANI, M. and AKBARI, M., Customizing ISO 9126 quality model for evaluation of B2B applications, Elsevier Journal Information and Software Technology, 2009.

[52] CHEN, C. and ALFAYEZ, R. and SRISOPHA, K. and BOEHM, B. and SHI, L., Why is it important to measure maintainability and what are the best ways to do it? , 2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, s.377-378, 2017.

[53] KOLLANUS, S. and KOSKINEN, J, Survey of software inspection research, The Open Software Engineering Journal, vol.3, s.15-34, 2009.

[54] Internet: NASA Open Source Software, https://code.nasa.gov/. [Accessed: 14-Oct-2018].

[55] YAMASHITA et al., Threholds for size and complexity metrics: a case study from the perspective of defect density, 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), s.191-201, Vienna, 2016.

[56] Internet: Mccabe.com, All metrics thresholds in McCabe iq, http://www.mccabe.com/pdf/McCabe%20IQ%20Metrics.pdf. [Accessed: 7- Oct- 2018].

[57] DEMEYER, S. and DUCASSE, S. and LANZA, M., A hybrid reverse engineering approach combining metrics and program visualisation, Sixth

Working Conference on Reverse Engineering (Cat. No.PR00303), s.175-186, Atlanta, GA, USA, 1999.

[58] Internet: Machine Learning Project at the University of Waikato in New Zealand, https://www.cs.waikato.ac.nz/ml/index.html. [Accesed: 15-Oct-2018].

[59] JOHN, G. H. and LANGLEY, P., Estimating continuous distributions in Bayesian classifiers, In Proceedings of the Eleventh conference on Uncertainty in artificial intelligence (UAI'95), Philippe Besnard and Steve Hanks (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, s.338-345, 1995.

[60] Internet: FindBugs, http://findbugs.sourceforge.net/index.html, [Accessed: 2- Feb- 2019].

[61] Internet: Checkstyle, http://checkstyle.sourceforge.net/index.html, [Accessed: 2- Feb- 2019].

[62] Internet: SDmetrics, http://www.sdmetrics.com, [Accessed: 2- Feb- 2019].

[63] Internet: Metrics, http://metrics.sourceforge.net/index.html, [Accessed: 2-Feb- 2019].

[64] Internet: PMD, https://pmd.github.io/, [Accesed: 2- Feb- 2019].

[65] Internet: Coverlipse, http://coverlipse.sourceforge.net/index.php, [Accessed: 2-Feb- 2019].

[66] BELCASTRO, M. C., Validation of safety-critical systems for aircraft loss-of-control prevention and recovery, AIAA Guidance Navigation and Control Conference, 2012.

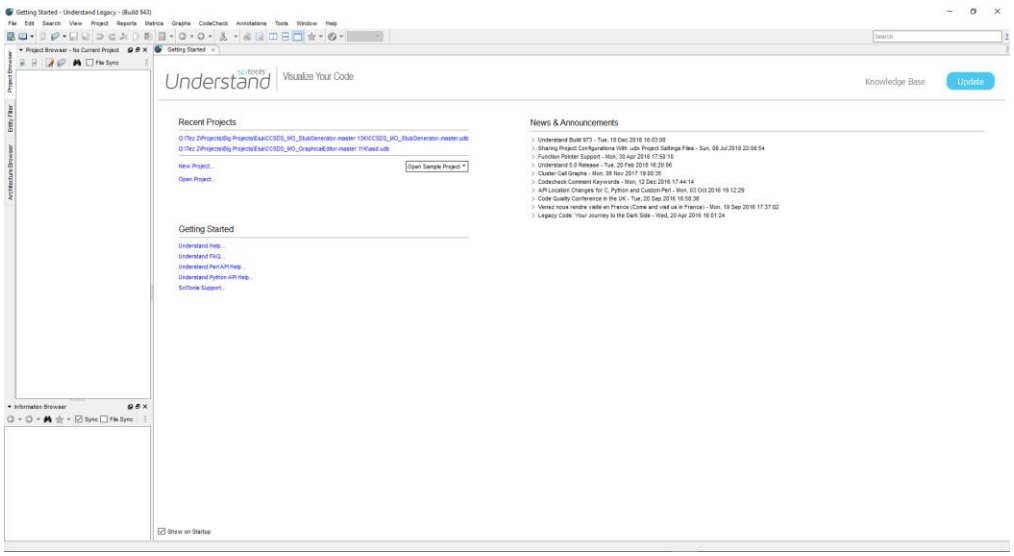
[67] Internet: Linkedin.com, https://www.linkedin.com/pulse/why-its-good-idea-data-scientists-use-weka-al-yazdani, [Accessed: 1-Feb-2019]

[68] Internet: IEEE 1219-1998, https://standards.ieee.org/standard/1219-1998.html, [Accessed: 5-Feb-2019].

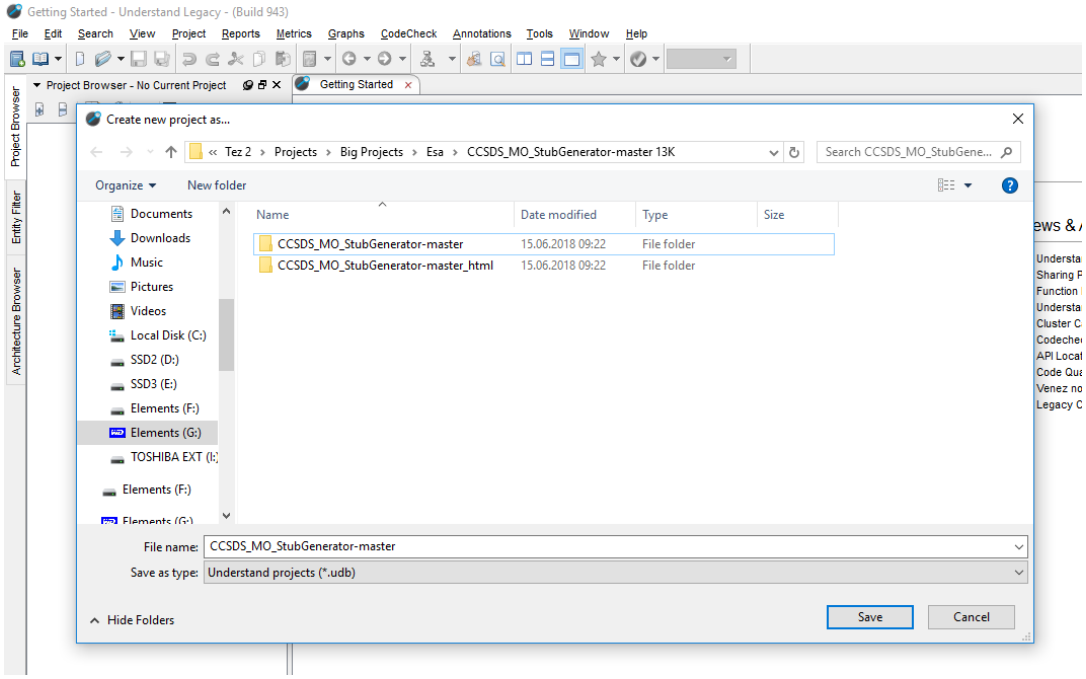
[69] Internet: IEEE 14764-2006, https://standards.ieee.org/standard/14764-2006.html, [Accessed: 5-Feb-2019].

EKLER

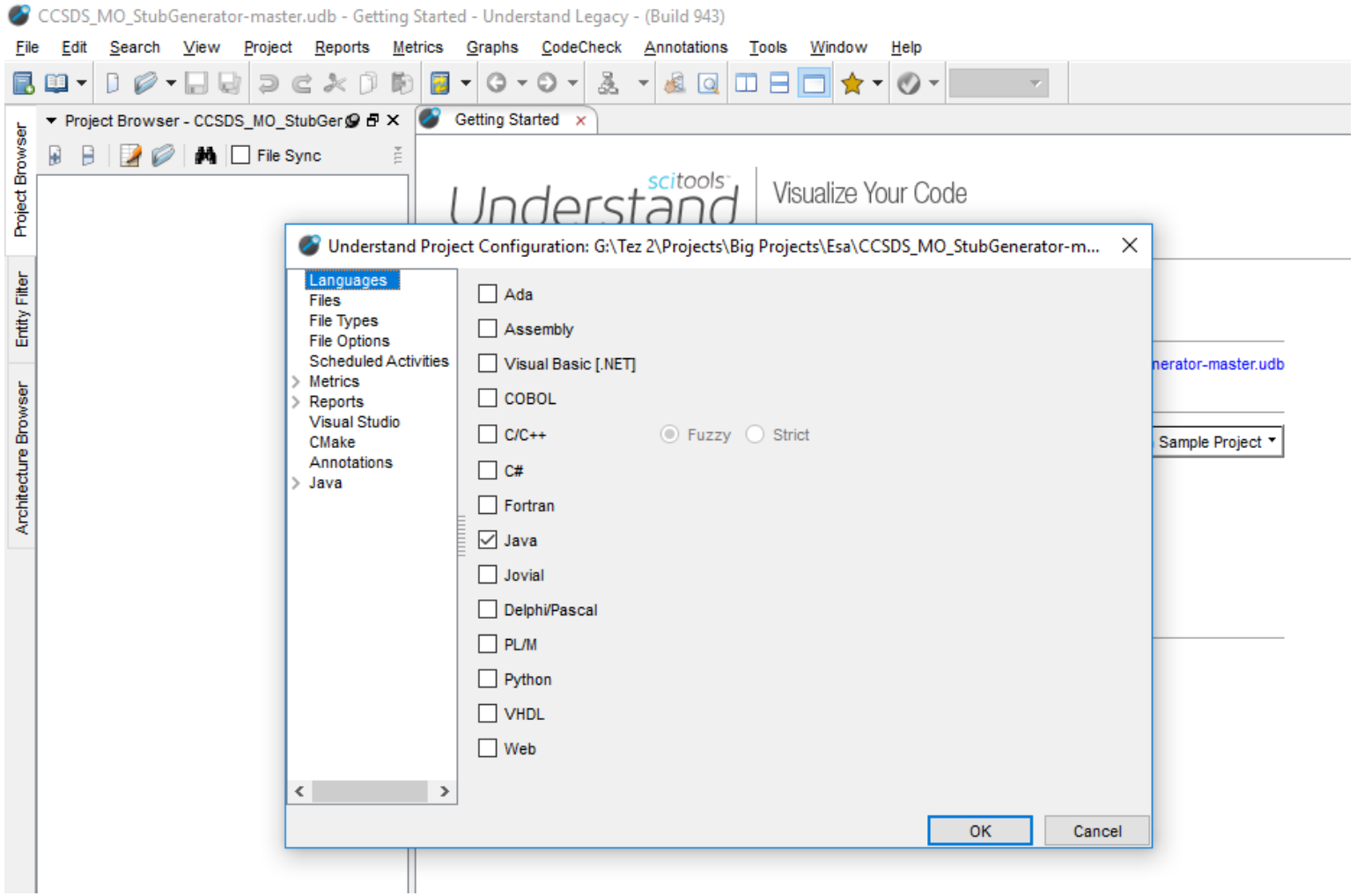
EK 1 UNDERSTAND EKRAN GÖRÜNTÜLERİ



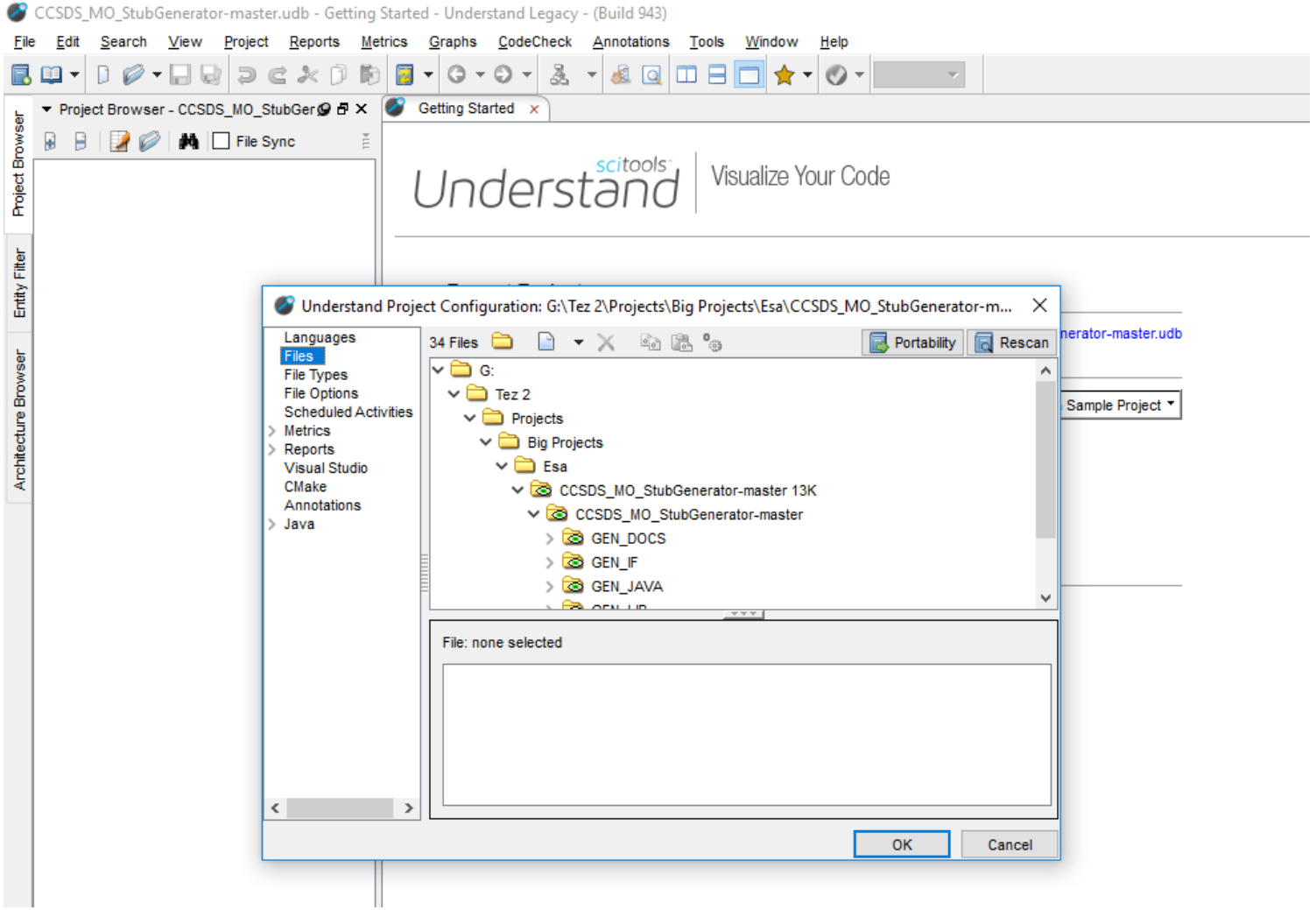
Görsel 1-1 Understand açılış ekranı



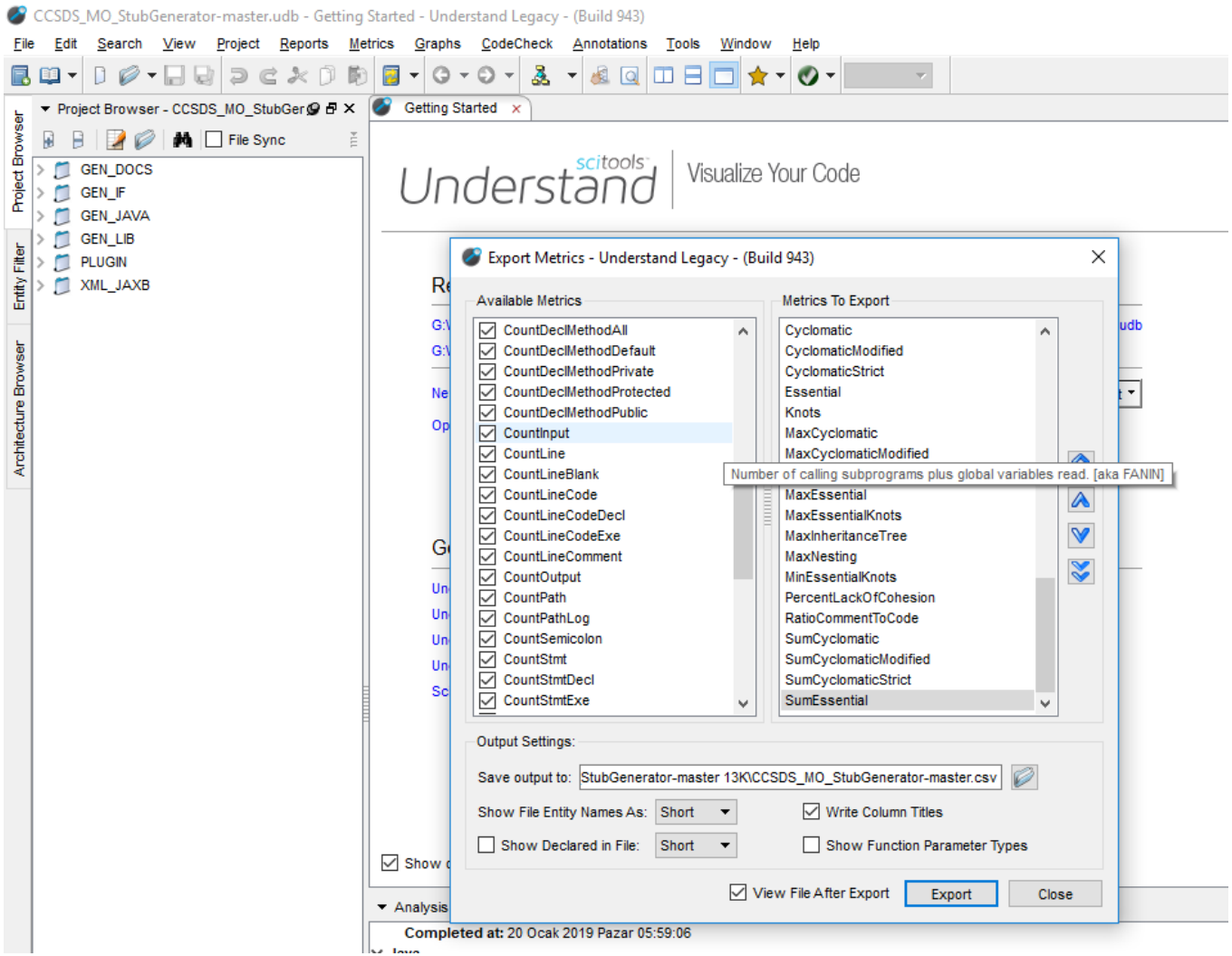
Görsel 1-2 Understand'de yeni çalışma oluşturma



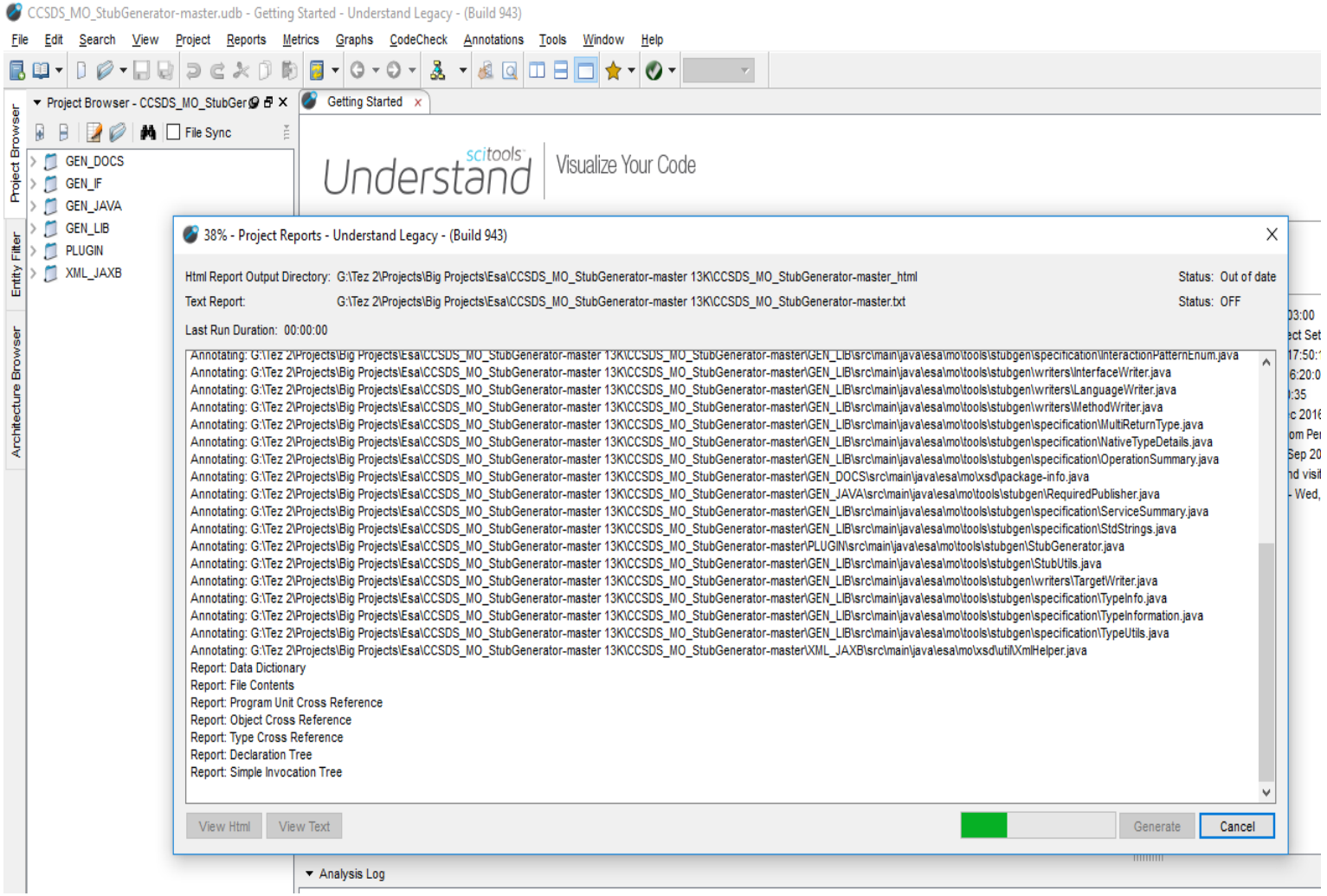
Görsel 1-3 Projenin kodlandığı dilin seçimi



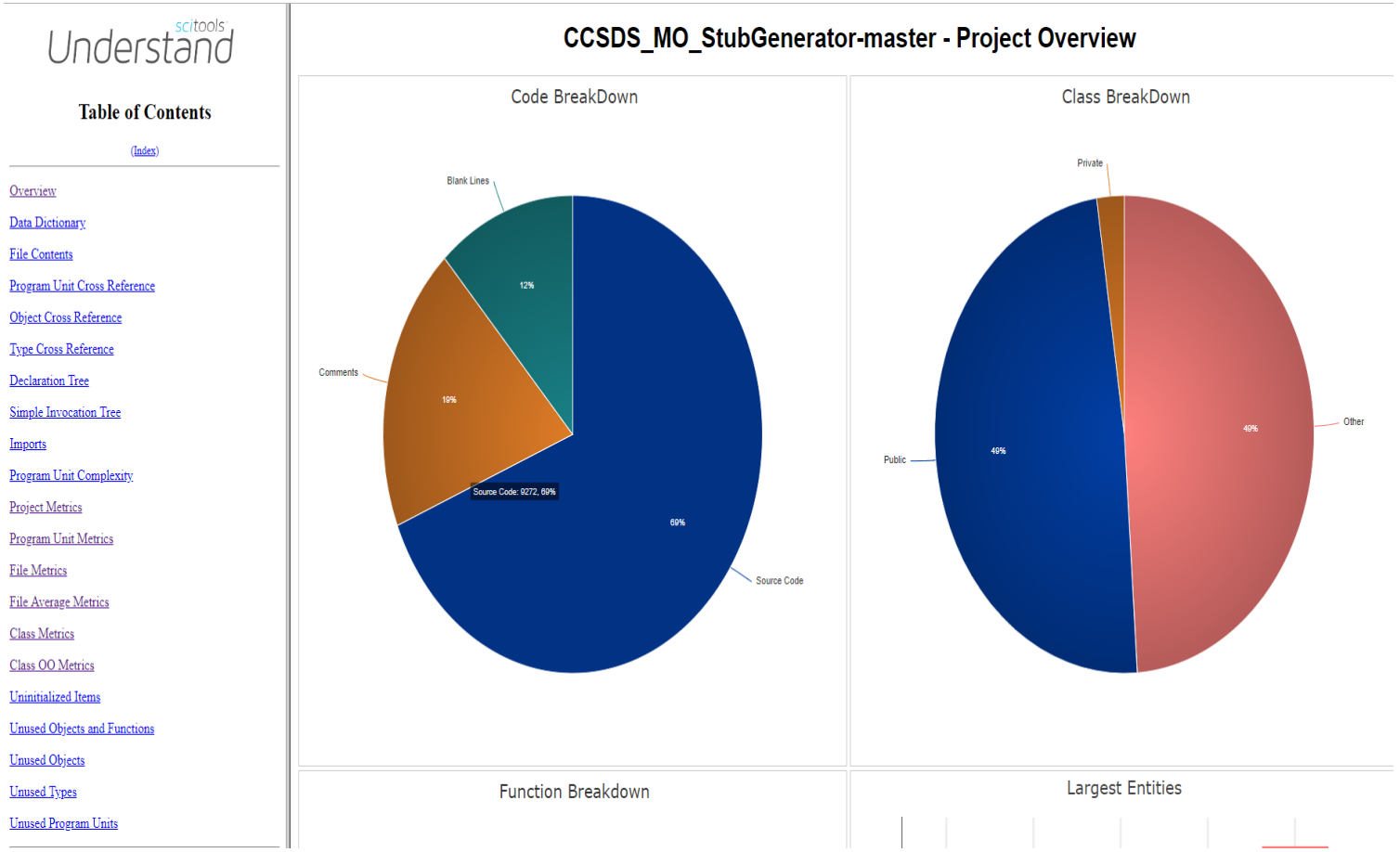
Görsel 1-4 Projenin alt klasörlerinin seçimi



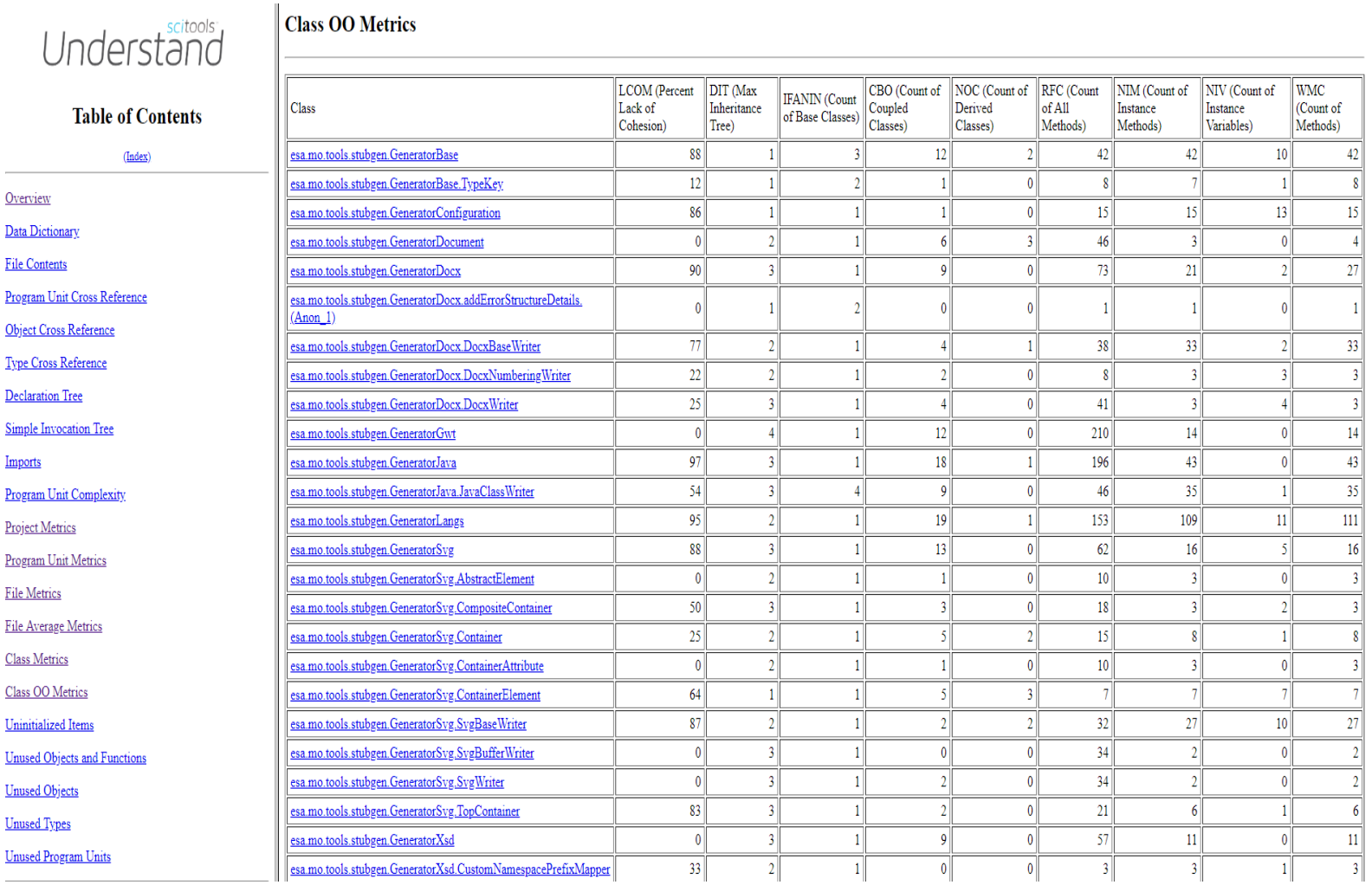
Görsel 1-5 Projenin analizi için kullanılacak ölçütlerin seçimi



Görsel 1-6 Projenin analiz ediliş süreci



Görsel 1-7 Projenin analiz çıktısı, genel özet sayfası



scitools Understand

Table of Contents

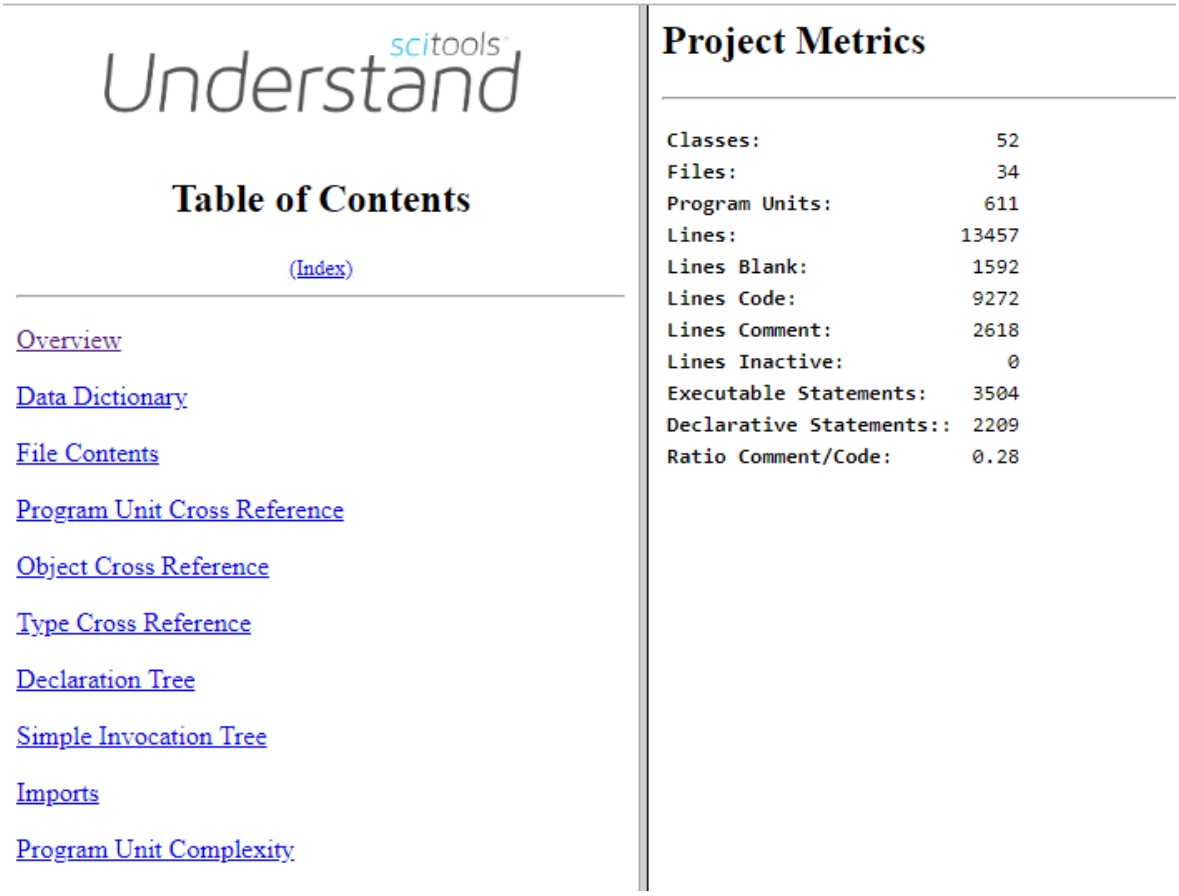
(Index)

Overview
Data Dictionary
File Contents
Program Unit Cross Reference
Object Cross Reference
Type Cross Reference
Declaration Tree
Simple Invocation Tree
Imports
Program Unit Complexity
Project Metrics
Program Unit Metrics
File Metrics
File Average Metrics
Class Metrics
Class OO Metrics
Uninitialized Items
Unused Objects and Functions
Unused Objects
Unused Types
Unused Program Units

Class OO Metrics

Class	LCOM (Percent Lack of Cohesion)	DIT (Max Inheritance Tree)	IFANIN (Count of Base Classes)	CBO (Count of Coupled Classes)	NOC (Count of Derived Classes)	RFC (Count of All Methods)	NIM (Count of Instance Methods)	NIV (Count of Instance Variables)	WMC (Count of Methods)
esa.mo.tools.stubgen.GeneratorBase	88	1	3	12	2	42	42	10	42
esa.mo.tools.stubgen.GeneratorBase.TypeKey	12	1	2	1	0	8	7	1	8
esa.mo.tools.stubgen.GeneratorConfiguration	86	1	1	1	0	15	15	13	15
esa.mo.tools.stubgen.GeneratorDocument	0	2	1	6	3	46	3	0	4
esa.mo.tools.stubgen.GeneratorDocx	90	3	1	9	0	73	21	2	27
esa.mo.tools.stubgen.GeneratorDocx.addErrorStructureDetails.(Anon_1)	0	1	2	0	0	1	1	0	1
esa.mo.tools.stubgen.GeneratorDocx.DocxBaseWriter	77	2	1	4	1	38	33	2	33
esa.mo.tools.stubgen.GeneratorDocx.DocxNumberingWriter	22	2	1	2	0	8	3	3	3
esa.mo.tools.stubgen.GeneratorDocx.DocxWriter	25	3	1	4	0	41	3	4	3
esa.mo.tools.stubgen.GeneratorGwt	0	4	1	12	0	210	14	0	14
esa.mo.tools.stubgen.GeneratorJava	97	3	1	18	1	196	43	0	43
esa.mo.tools.stubgen.GeneratorJava.JavaClassWriter	54	3	4	9	0	46	35	1	35
esa.mo.tools.stubgen.GeneratorLangs	95	2	1	19	1	153	109	11	111
esa.mo.tools.stubgen.GeneratorSvg	88	3	1	13	0	62	16	5	16
esa.mo.tools.stubgen.GeneratorSvg.AbstractElement	0	2	1	1	0	10	3	0	3
esa.mo.tools.stubgen.GeneratorSvg.CompositeContainer	50	3	1	3	0	18	3	2	3
esa.mo.tools.stubgen.GeneratorSvg.Container	25	2	1	5	2	15	8	1	8
esa.mo.tools.stubgen.GeneratorSvg.ContainerAttribute	0	2	1	1	0	10	3	0	3
esa.mo.tools.stubgen.GeneratorSvg.ContainerElement	64	1	1	5	3	7	7	7	7
esa.mo.tools.stubgen.GeneratorSvg.SvgBaseWriter	87	2	1	2	2	32	27	10	27
esa.mo.tools.stubgen.GeneratorSvg.SvgBufferWriter	0	3	1	0	0	34	2	0	2
esa.mo.tools.stubgen.GeneratorSvg.SvgWriter	0	3	1	2	0	34	2	0	2
esa.mo.tools.stubgen.GeneratorSvg.TopContainer	83	3	1	2	0	21	6	1	6
esa.mo.tools.stubgen.GeneratorXsd	0	3	1	9	0	57	11	0	11
esa.mo.tools.stubgen.GeneratorXsd.CustomNamespacePrefixMapper	33	2	1	0	0	3	3	1	3

Görsel 1-8 Projenin analiz çıktısı, sınıflar için OO ölçümleri



scitools Understand

Table of Contents

(Index)

Overview

Data Dictionary

File Contents

Program Unit Cross Reference

Object Cross Reference

Type Cross Reference

Declaration Tree

Simple Invocation Tree

Imports

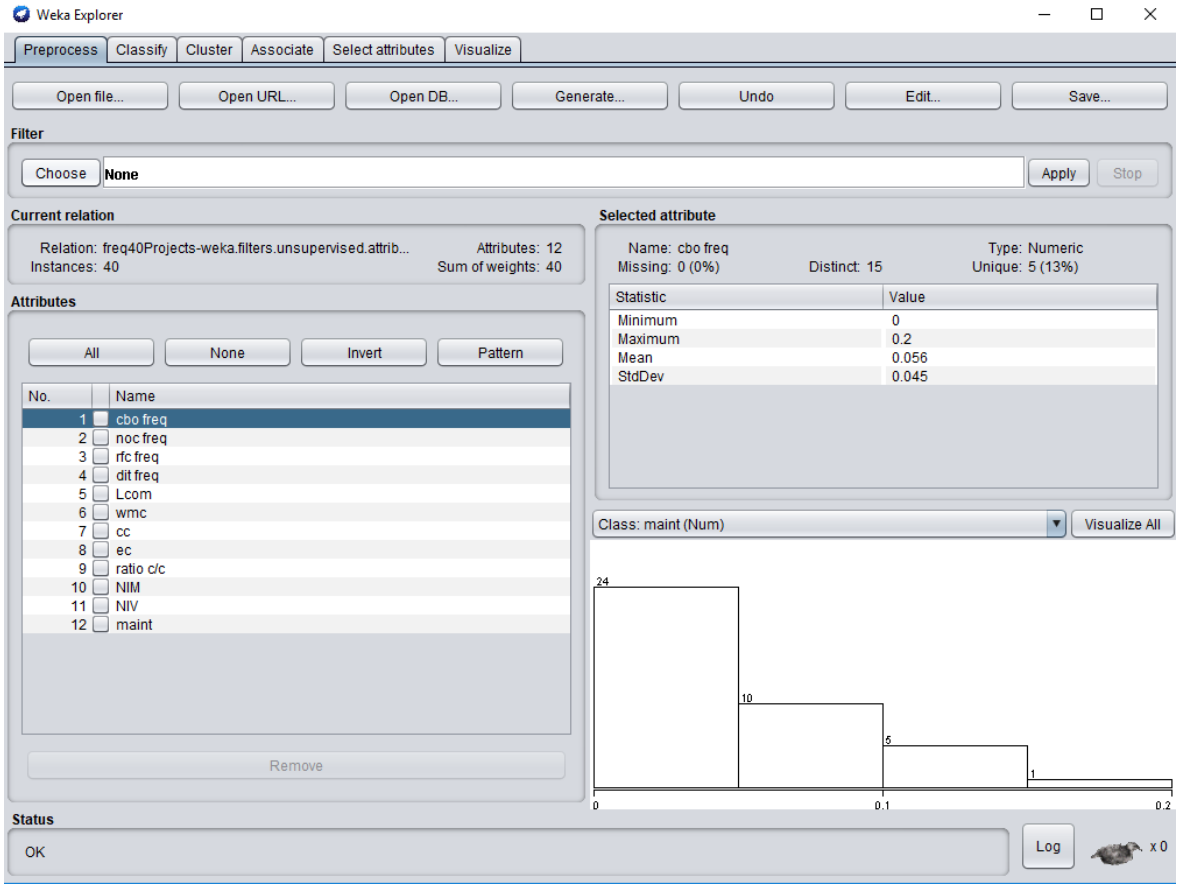
Program Unit Complexity

Project Metrics

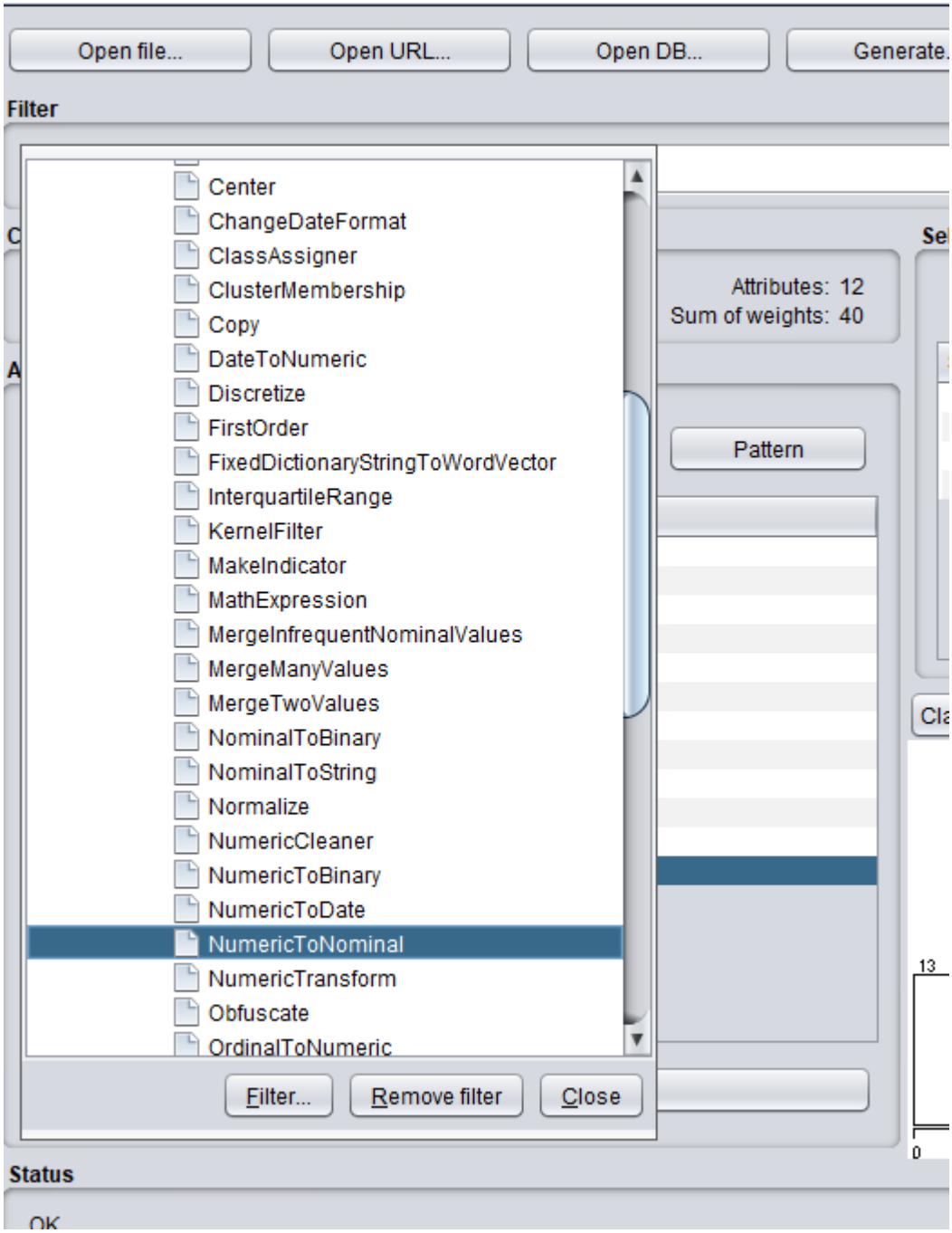
Classes:	52
Files:	34
Program Units:	611
Lines:	13457
Lines Blank:	1592
Lines Code:	9272
Lines Comment:	2618
Lines Inactive:	0
Executable Statements:	3504
Declarative Statements::	2209
Ratio Comment/Code:	0.28

Görsel 1-9 Projenin yapısal ölçümleri

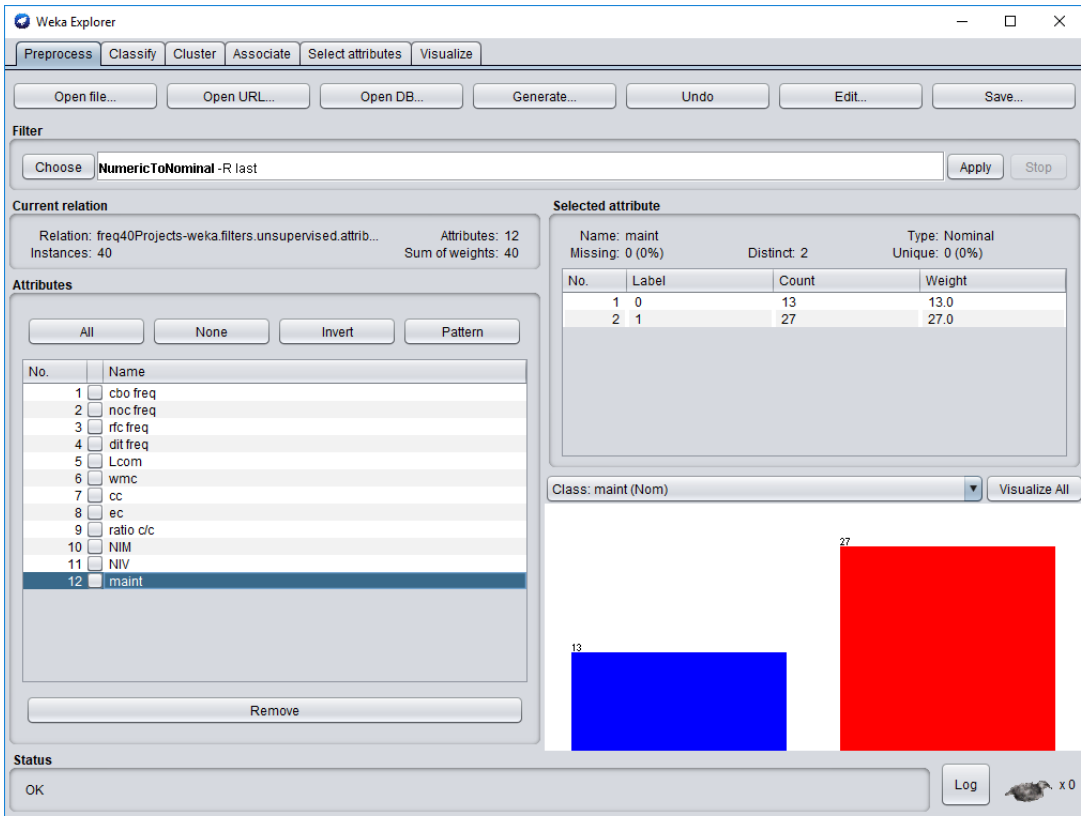
EK-2 WEKA EKRAN GÖRÜNTÜLERİ



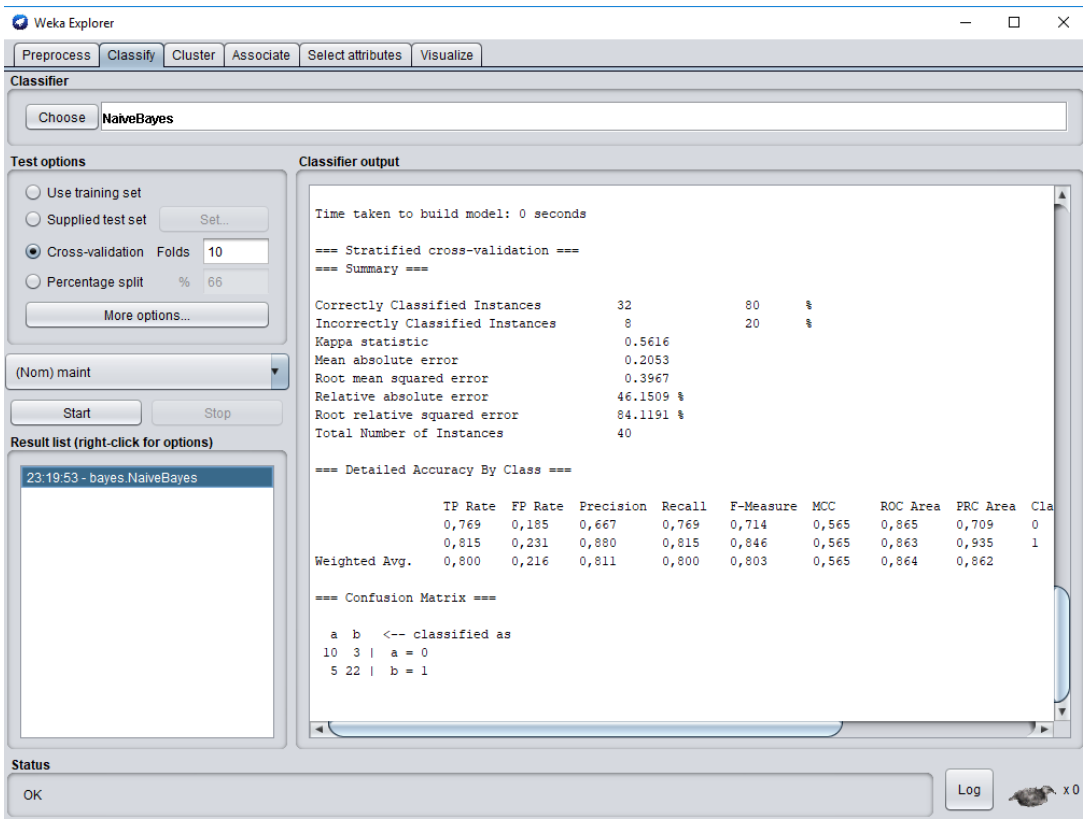
Görsel 2-1 Weka preprocess ekranı



Görsel 2-2 Weka veri seti için filtreleme seçenekleri



Görsel 2-3 Sınıf veri sütununun nümerikten nominal değere dönüştüğü kısım



Görsel 2-4 Sınıflandırma bölümü, örnek bir sınıflandırma çıktısı görüntüsü