T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM HATA KESTİRİM YAKLAŞIMLARI

ÖZKAN SARI

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
PROF. DR. OYA KALIPSIZ

İSTANBUL, 2015

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM HATA KESTİRİM YAKLAŞIMLARI

Özkan SARI tarafından hazırlanan tez çalışması 26/06/2015 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Prof. Dr. Oya KALIPSIZ

Yıldız Teknik Üniversitesi _____

Yrd.Doç.Dr.Yunus Emre SELÇUK

Yıldız Teknik Üniversitesi _____

Yrd.Doç.Dr.Zeynep ORMAN

İstanbul Üniversitesi _____

Bu çalışma, Provus Bilişim Hizmetleri A.Ş. tarafından desteklenmiştir.

ÖNSÖZ

Tez çalışmasının girişini oluşturan ilk bölümde yazılım hatalarıyla ve tezin çıkış noktası ile ilgili bilgiler verilmiş, literatür incelemelerine yer verilmiş, tezin amacı ve hipotez sunulmuştur. İkinci bölümde, metrik tabanlı yazılım hata kestirimi yaklaşımları diğer farklı yöntemlerle birlikte tanıtılmıştır. Bu bölümde, çalışmamızda yoğun olarak kullanılan metrikler hakkında da tanım ve bilgiler bulunmaktadır. Üçüncü bölümde çalışmamızla ilgili temel veri analizi kavramları sunulmuştur ve veri analizi yöntemlerinden regresyon analizi ve lojistik regresyon analizi açıklanmıştır. Bu bölümde ayrıca, çalışmamızda da sıklıkla değinilen, genel bazı istatistik terimleri de açıklanmıştır. Dördüncü bölümün konusu, kaynak veri setinden model oluşturma süreci planlamasıdır. Bu bölümde kullanılan kaynak hata kestirim veri tabanı tanıtılmış, bu veri tabanı kullanılarak hata kestirim modelinin nasıl oluşturulacağı belirtilmiştir. Beşinci bölümde, geliştirilen modelin uygulanacağı ATM İzleme ve Yönetim Sistemi Yazılımının yapısı ve bu yazılım kodunun incelenmesiyle modele girdi olarak sağlanacak metriklerin nasıl toplandığı ve hesaplandığı anlatılmıştır. 6. Bölüm, geliştirilen model üzerindeki deney ve doğrulama çalışmalarından oluşmaktadır. Yapılan farklı deney çalışmaları detaylı olarak, bu bölüm kapsamında anlatılmıştır. 7. Bölümde ise sonuçlar değerlendirilmiş, farklı yaklaşımlara ait yapılan deneylerin sonuçları karşılaştırılmış, çalışmadaki genel kazanımlara değinilmiş ve gelecekte yapılabilecek çalışmalardan bahsedilmiştir. Tezin ek bölümlerinde ise, oluşturulan deneysel hata kestirim veri tabanı, kaynak kodlar ve çalışmayla ilgili yayınlar verilmiştir.

Çalışmamı yürütürken benden yardımını esirgemeyen ve fikirleriyle bana yön veren danışmanım Prof. Dr. Oya KALIPSIZ'a, araştırmalarımda bana veri ve teknoloji bakımından destek veren *Provus Bilişim Sistemleri A.Ş.* firmasına, bana anlayış gösteren ve her daim bana manevi destekte bulunan hayat arkadaşım Derya SARI'ya ve beni yetiştirip bugünlere getiren aileme teşekkürlerimi sunarım.

Haziran, 2015

Özkan SARI

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

α	Yanlış pozitiflik oranı
β	Doğrusal regresyon analizindeki “β” katsayısı veya yanlış negatiflik oranı
χ^2	Anlamlılık (*Significance*) hesabındaki ki-kare değeri

KISALTMA LİSTESİ

ATM	Bankamatik
AGE	Kod Yaşı metriği
BUGCATG	Hata Kategorizasyonu metriği
BUGFIXES	Hata Veri tabanında Yer Alma Sayısı metriği
CBO	Nesneler Arası Bağlaşım metriği (*Coupling Between Objects*)
CHGSET	Değişiklik Boyutu metriği
CHURN	Kod Çalkantısı metriği
CSV	Microsoft Excel benzeri uygulamalarda kullanılmak üzere geliştirilmiş veri formatı. Veriler virgül gibi ayraçlarla bir birinden ayrılarak tutulurlar (*Comma Separated Variables*)
DIT	Miras Ağacı Derinliği metriği (*Depth of Inheritence Tree*)
FanIn	Referans Veren Sınıf Sayısı metriği
FanOut	Referans Verilen Sınıf Sayısı metriği
FN	Yanlış negatif değeri
FP	Yanlış pozitif değeri
İht.	İhtimali
LCOM	Yordamlarda Yapışkanlık Eksikliği metriği (*Lack of Cohesion in Methods*)
LINES	Eklenen/Çıkarılan Satır Sayısı metriği
LOC	Kod Satır Sayısı metriği (*Number of lines of code*)
N/A	Uygulanamaz veya uygun değil
NAUTH	Dosyada Değişiklik Yapan Kişi Sayısı metriği
NEXTBUGS	Sonraki Hata metriği
NFIX	Hata Düzeltmede Yer Alma Sayısı metriği
NOA	Değişken Sayısı metriği (*Number of attributes*)
NOPA	Dışa Açık Değişken Sayısı metriği (*Number of public attributes*)
NOAI	Miras Alınan Değişken Sayısı metriği (*Number of attributes inherited*)
NOM	Metot Sayısı metriği (*Number of methods*)
NOMI	Miras Alınan Metot Sayısı metriği (*Number of methods inherited*)
NOPM	Dışa Açık Metot Sayısı metriği (*Number of public methods*)
NOPRA	Gizli Değişken Sayısı metriği (*Number of private attributes*)
NOPRM	Gizli Metot Sayısı metriği (*Number of private methods*)
NR	Revizyon sayısı metriği (*Number of Revision*)
NREF	Yeniden düzenleme sayısı metriği (*Number of Refactoring*)
NSC	Miras Alan Sınıf Sayısı metriği (*Number Of Children*)
OR	Bahis oranı değeri (*Odds Ratio*)

Ör.	Örneğin
PREVBUGS	Önceki Hata metriği
RFC	Sınıf için potansiyel karşılık metriği (*Response For Class*)
Std.	Standart
TN	Doğru negatif değeri
TP	Doğru pozitif değeri
WMC	Ağırlıklı Metot Ortalaması metriği (*Weighted Method Count*)
XML	Genişletilebilir İşaretleme Dili (*Extensible Markup Language*) isimli evrensel olarak tanınan bir dosya standardı.

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

YAZILIM HATA KESTİRİM YAKLAŞIMLARI

Özkan SARI

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Prof. Dr. Oya KALIPSIZ

Geliştirilen yazılımların bazı hatalar içermesi doğaldır. Önemli olan bu hataların tespit edilebilmesidir. Hataların tespitinde testler önemli bir yer tutmaktadır ancak her zaman bu yeterli değildir. Bu nedenle yazılım hatalarının ve kusurlarının tespit edilebilmesi için etkin yöntemlere ihtiyaç vardır.

Yazılım hatalarının, yazılım geliştirme faaliyetlerinin erken safhalarında tespit edilmesinin daha az masraflı olduğu bilinmektedir. Araştırmalara göre, yazılımın teslimi sonrası bir hatanın bulunması ve düzeltilmesi, hatanın yazılımın gereksinim ve tasarım safhalarında bulunmasına kıyasla 100 kat daha masraflıdır. Yazılımdaki hataların ortalama %80'i, yazılım parçalarının (modüllerin) %20'sinde toplanmakta ve yazılımın %50'lik kısmında ise hiç hata bulunmamaktadır. Bu olguya dayanarak, koddaki hatalı olabilecek yerler kodun özellikleri incelenerek tespit edilebilirse hataların daha erken bulunması ve müdahale edilmesi mümkün olacaktır. Bu şekilde, hata eğilimli yazılım parçalarının tespit edilmesiyle test çabaları buralarda yoğunlaştırılabilecek ve eldeki kaynakların hataların bulunma ihtimali yüksek yerlere yönlendirilmesi ile hata tespitinde başarı ihtimali artacak ve hatanın erken tespiti ile büyük kazanç sağlamak mümkün olacaktır.

Hata eğilimli yazılım parçalarının tespiti ve hata kestirim faaliyetleri olarak adlandırılan bu çalışmalar bu amaca hizmet ederek, yazılımdaki hataların erken safhada tespitini amaçlamaktadır. Hata eğilimli yazılım parçalarının tespiti yönünde çok çeşitli

yaklaşımlar bulunmaktadır. Bu yaklaşımlar hem yöntem hem de başarım açısından farklılıklar göstermektedirler. Tez çalışmasında, belli başlı hata eğilimli yazılım parçalarının tespiti yöntemlerinin incelenmesi ve hata kestirimi için bir etkin bir model geliştirilmesi amaçlanmaktadır. Çalışmamızda lojistik regresyon analizi tabanlı bir yaklaşım benimsenmiş ve kaynak kod metrikleri üzerine yoğunlaşılmıştır.

Ortaya konulan model çeşitli halka açık veri setleri üzerinde ve Provus Bilişim Sistemleri A.Ş. bünyesinde geliştirilen yazılımlarda sınanmış ve sonuçlar paylaşılmıştır.

Anahtar Kelimeler: Yazılım hataları, Hata eğilimli yazılım parçaları, Hata kestirimi, Yazılım metrikleri, Yazılım güvenilirliği, Hata tahmini, Lojistik regresyon analizi, Kusur, Sezimleme, Ön kestirim modelleri

ABSTRACT

SOFTWARE DEFECT PREDICTION APPROACHES

Özkan SARI

Department of Computer Engineering

MSc. Thesis

Adviser: Prof. Dr. Oya KALIPSIZ

For the software being developed, it is natural to have some degree of defects. What is important is to be able to find these defects. Testing activities are essential but testing each fragment of the software is impossible and defects still occur even after several detailed test activities. Therefore, there is a need for effective methods to detect bugs in software.

It is well known that it's less expensive to find software defects early on in program. Research shows that finding and fixing a software problem after delivery is often 100 times more expensive than the requirements and design phase. About 80% of the defects come from 20% of the modules, and about half the modules are defect free. Considering these facts, if fault-prone modules are detected, software test efforts can be focused on these modules. By using the limited resources we have on the modules having the highest possibility to have defects, defect detection rate will increase and it will be possible to gain valuable resources by finding defects early.

Serving this purpose, activities named as detection of fault-prone modules or defect prediction, help to detect the presence of defects as early as possible in an automated fashion. These approaches differ according to their techniques and success rates. The main approaches in the field will be inspected and an effective model is aimed to be developed in order to predict software entities having bugs. In our study, an approach

based on logistic regression analysis and focusing on static software code metrics is utilized.

A public bug database and an ATM monitoring software source code which is developed by Provus Bilişim Hizmetleri A.Ş., are used for the creation of the model and to find the performance of the study.

Keywords: Software defects, Fault-prone modules, Defect prediction, Software metrics, Software reliability, Error estimation, Logistics regression analysis, Fault, Detection, Prediction models

BÖLÜM 1

GİRİŞ

Yazılımın teslimi sonrası bir hatanın bulunmasının ve düzeltilmesinin, hatanın yazılımın gereksinim ve tasarım safhalarında bulunmasına kıyasla çok daha masraflı olduğu bilinmektedir [1]. Hata eğilimli yazılım parçalarının tespit ile yazılımdaki hataların erken safhada tespiti amaçlamaktadır. Hata eğilimli yazılım parçalarının tespiti yönünde çok çeşitli yaklaşımlar bulunmaktadır. Bu yaklaşımlar hem yöntem hem de başarım açısından farklılıklar göstermektedirler [2].

Bilgisayar donanımları ve yazılımları birbirinden ayrılmaz bir bütündür. Bununla beraber, artık bilgisayar donanımları çok düşük maliyetlerle ve sıfıra yakın hata oranıyla üretilirken; yazılımlar için tam aksi bir durum söz konusudur. Yazılımların boyutları, kapasiteleri ve karmaşıklıkları arttıkça yazılım maliyetleri ve hata oranları yüksek seviyelere ulaşmıştır [3].

Yazılım maliyetlerinin büyük ölçüde geliştirme maliyetlerinden kaynaklandığı düşünülse de, yazılımların bakım masrafları zaman zaman daha büyük miktarlara ulaşabilmektedir. Tabi ki bu başarılı olan yazılım projeleri için geçerli bir ifadedir. Bununla beraber, yazılımların birçoğu ya geliştirilirken ya da geliştirildikten sonra ihtiyacı karşılamadığı için başarısızlıkla sonuçlanmaktadır. Amerikan ordusunun yazılım projeleri üzerine yaptığı bir araştırma göstermektedir ki, yazılım projelerinin %19'u başladıktan sonra iptal edilmekte, %47'si hiç kullanılmamakta, %29'u müşteri tarafından kabul görmemekte, sadece geri kalan %3'lik kısmı ufak değişiklikler görerek kullanılmakta ve ancak %2'lik kısmı teslim alındığı gibi kullanılmaktadır [3].

McKinsey tarafından 2012 yılında yapılan bir araştırmada, bütçesi 15 Milyon dolardan fazla olan 5400 adet büyük ölçekli bilişim sistemleri projesi incelenmiştir. Bulgulara

göre, projelerin %17'si o kadar kötü bir şekilde başarısız olmaktadır ki, firmanın varlığı dahi tehlikeye girmektedir. Ayrıca projelerin %45'i finansal bütçelerini aşmakta, %7'si zamanında tamamlanamamakta ve %56'si ise planlanandan daha az özellikte teslim edilmektedir [4].

Yazılımların başarısız olmasının birçok sebebi vardır ancak bunlardan en önemlilerinden birisi yazılım hataları ve dolayısıyla ortaya çıkan artan kaynak ve bütçe masraflarıdır.

Yazılım hatalarının, yazılım geliştirme faaliyetlerinin erken safhalarında tespit edilmesinin daha az masraflı olduğu bilinmektedir. Araştırmalara göre, yazılımın teslimi sonrası bir hatanın bulunması ve düzeltilmesi, hatanın yazılımın gereksinim ve tasarım safhalarında bulunmasına kıyasla 100 kat daha masraflıdır [1].

Geliştirilen yazılımların bazı hatalar içermesi doğaldır. Önemli olan bu hataların tespit edilebilmesidir. Hataların tespitinde testler önemli bir yer tutmaktadır ancak her zaman bu yeterli olmamaktadır. Testte tespit edilemeyen hataların sonradan tespit edilmesinde veya yazılımdaki hataları mümkün olduğunca erken tespit edebilmesi için etkin yöntemlere ihtiyaç vardır.

Yazılımların hataya açık olma eğiliminin tespiti, söz konusu yazılım birimlerinin hata içerme ihtimalinin tespit edilmesinde bilgisayar destekli bir yaklaşım kullanılmasına dayanır. Yazılım hatalarının çoğu, belirli ve az sayıda yazılım parçalarında toplandığı kabul edildiği için; bu hataya açık yazılım parçalarının tespit edilmesi ile test eforu buralarda yoğunlaştırılabilir ve yazılımın kalitesinin arttırılması sağlanır [5].

Yapılan çalışmanın uygulama alanı sadece nesne yönelimli programlama dilleri kullanılarak geliştirilen yazılımlarla sınırlı olmasa da çalışmanın pratik kısmında kullanılan metrikler, genelde nesne yönelimli yazılımları ilgilendiren kalıtım, kapsülleme, çok biçimlilik, uyumluluk, bağımlılık gibi ilkelere bağlıdır. Yazılım geliştirmenin popüler bir yolu olan nesne yönelimli programlama Java, C#, C++ gibi gözde programlama dilleri tarafından da desteklenmektedir. Yapısal programlamadan farklı yapıya sahip nesneye yönelik programlamada yapılan geliştirme, test, onarım ve bakım çalışmaları da farklılık arz etmekte ve bu yöntemle geliştirilen yazılımların farklı kalite ölçütlerine sahip olmasına yol açmaktadır.

1.1 Literatür Özeti

Yazılım mühendisliği alanında yapılmış çalışmalar incelendiğinde yazılım hata kestirimi üzerine birçok araştırma yapıldığı görülmektedir. Bu alanda kullanılan yöntemler incelendiğinde, istatistik veri analizi, makine öğrenmesi, genetik programlama [6], karar ağaçları[7], yapay sinir ağları [8], bulanık mantık [9] , yapay bağışıklık sistemleri [10], *Bayes* sınıflandırması ve bunun yanı sıra karma yöntemler olmak üzere çok çeşitli yöntemler kullanıldığı görülmektedir [11].

Hataya açık yazılım parçalarının tespiti, üzerinde yoğun olarak çalışılmış bir konudur [5,12,13,14]. Bu çalışmaların çoğu karmaşıklık, boyut, nesneye yönelik programlama (OOP) metrikleri gibi çeşitli yazılım metriklerinden faydalanmış ve sonuçta bir matematik modeli oluşturarak yazılım parçalarının hataya eğilimli olma ihtimallerini tespit etmişlerdir.

Lanubile ve diğerleri tarafından, 1995 yılında yapılan çalışmada, yazılım karmaşıklık ölçütleri kullanılarak, bileşenler hata içeren veya hata içermeyen şeklinde ayrıştırılmıştır. Yapılan çalışmada, temel bileşen analizi (PCA), diskriminant analizi, lojistik regresyon, mantıksal sınıflandırma modelleri, katmanlı sinir ağları ve holografik ağlar gibi çok çeşitli temel sınıflandırma modelleri incelendi ve sonuçlar karşılaştırılmıştır. Çalışma sonucunda, hiçbir modelin yeterince başarılı olamadığı iddia edilmiştir [15].

Khoshgoftaar ve diğerleri 2002 yılında yaptıkları çalışmada, yazılım parçalarını hataya açık ya da değil olarak sınıflandırmak için kaynak kod dosyalarında yapılan değişiklik sayısını temel alan bir çalışma gerçekleştirdi ve kaynak koda eklenen/çıkarılan satır sayısının yazılım parçaları bazındaki, gelecekteki kusurların tahmini açısından önemli olduğunu gösterdi [16].

Graves ve diğerleri ise 2000 yılındaki çalışmalarında, yazılım parçalarının gelecekteki hataları için en iyi öngörücünün (*prediktör*) geliştirilmesi amacıyla istatistiksel modellere dayanan bir yaklaşım geliştirdi ve en iyi öngörücünün yazılım parçalarına geçmişte yapılan katkıların toplamı olduğunu gösterdiler [17].

Giovanni ise 2000 yılındaki çalışmasında, statik kod metrikleri ile yazılım hatalarına açık olma eğilimi arasında bağlantı olduğunu tespit etmiştir [18].

Manasi Deodhar 2002 yılındaki çalışmasında, nesne yönelimli sistemlerde (OOP) hataya açık olma üzerine araştırmalarını ortaya koydu ve hata eğilimli sınıfları belirleyerek bunlar üzerinde test çabalarına odaklanılabileceğini belirtti [19]. Çalışmasında, her bir sınıfın yapısal kalitesini belirleyerek hata eğilimi kestiriminin yapılabileceğini ve sınıf büyüklük ölçütünün önemli bir etmen olduğunu ortaya koydu.

Briand ve diğerleri 2002 yılında yaptıkları çalışmada hata eğilimli olma tespiti modellerinin ekonomik olarak uygulanabilirliğini/yaşayabilirliğini ve farklı sistemler boyunca doğruluğunu/hassasiyetini koruyup korumadığını sorguladı [20]. Yapılan çalışma bir modeli farklı sistemlere uygulamanın, modelin doğruluğunu/hassasiyetini gerilettiğini ispatlamakla beraber, sıralamanın hala geçerli olduğunu ispatladı. Yani modelde bulunan sınıfın hatalı olma ihtimali yeni sistemde pek bir şey ifade etmemekle beraber diğer sınıflara göre daha az ya da daha fazla hatalı olma ihtimalinin hala geçerliliğini koruduğunu gösterdi. Bu çalışma, oluşturulacak modelin farklı yazılım ve sistemlerde kullanılabilirliği söz konusu olduğunda yol gösterici bir kaynak olmuştur.

Moser ve diğerleri, 2008 yılında yaptıkları çalışmada, farklı metrikleri bir araya getirerek *Eclipse* kaynak kodlarında hataların varlığı/yokluğunu tahmin için bir sınıflandırma tekniği önermişlerdir [21]. Yaptıkları çalışmada kod çalkantısı, geçmiş hatalar ve kod yenilemeleri (*refaktör*), geliştirici sayısı, dosya boyutu ve yaşı gibi çok farklı metrikleri bir arada kullanmışlardır.

Çalışmamıza kaynak oluşturan belki de en önemli kaynaklardan birisi de, D'Ambros ve diğerlerinin 2012 yılına ait “Hata Tahmini Yaklaşımlarının Detaylı Kıyaslanması” isimli çalışmasıdır [2]. Bu çalışmalarında yazarlar farklı hata kestirim faaliyetlerini inceleyerek bunları birbirleriyle kıyaslamışlardır.

1.2 Tezin Amacı

Yazılım kalite faaliyetleri içerisinde hataların giderilmesi için gerçekleştirilen test faaliyetleri önemli bir yer tutmaktadır. Bazı test faaliyetleri neticesinde dahi hataların tespit edilemediği durumlar olabilir. Bu nedenle yazılım hatalarının ve kusurlarının tespit edilebilmesi için etkin yöntemlere ihtiyaç vardır.

Koddaki hatalı olabilecek yerler kodun özellikleri incelenerek tespit edilebilirse hataların daha erken bulunması ve müdahale edilmesi mümkün olacaktır. Hata kestirim faaliyetleri de bu amaca hizmet ederek, yazılımdaki hataların otomatik bir şekilde ve erken safhada tespitini amaçlamaktadır.

Çalışmamızda, belli başlı hata kestirim yöntemleri incelenmiş ve hata kestirimi için bir etkin bir model geliştirilmesi amaçlanmıştır. Modelin oluşturulması için lojistik regresyon analizi tabanlı yöntemler üzerinde durulmuştur.

Bu çalışmanın amacı, yazılımdaki hataların kodun özellikleri incelenerek tespit edilmesi ve hatalı sınıf/dosyaların otomatik bir şekilde ayırt edilebilmesidir. Bu amaçla, kodların bir kısım metrikleri üzerinden lojistik regresyon analizi uygulanarak, metrik değerlerinden yazılım parçalarının hatalı olma ihtimali tespit edilecektir.

1.3 Hipotez

Kaynak, zaman, test elemanı gibi kısıtlar nedeniyle çoğunlukla yazılımın her yerinin yoğun bir şekilde test edilmesi mümkün olmamaktadır. Titiz bir şekilde ele alınmayan yazılım parçasının/sınıfın hata içermesi durumunda da bu hatalar çoğunlukla gözden kaçmaktadır. Bunun için test için ayrılan kaynakların hesaplı kullanılması önemlidir. Yazılım hatalarının çoğunluğu kodun yalnızca belli bölümlerinde bulunduğu kabul edilen bir gerçektir [5]. Bu yüzden, hataya eğilimli yazılım parçalarının tespit edilmesiyle test çabaları buralarda yoğunlaştırılabilir.

Yazılım kodundaki bir sınıfta hata bulunup bulunmadığı, kodun özellikleri incelenerek tespit edilebilir ve hataların erken tespit edilmesi mümkün olur. Hataların tespit edilebilmesi için kullanımı kolay ve uygulanabilir bir model oluşturulması önemlidir.

Yazılımdaki hataya eğilimli parçalar incelenirken, proje, paket, sınıf, metot gibi farklı büyüklüklerde yazılım birimleri ele alınabilir. Çalışmamızda yazılım hatalarının sınıf bazında incelenmesi uygun görülmüştür. Proje ve paket gibi büyük yazılım parçalarını incelemek, hem bazı ayrıntıların gözden kaçmasına neden olabileceği hem de proje ve paket bazında yazılımı inceleyen metriklerin kısıtlı olması nedeniyle tercih edilmemiştir. Metot ve kod satırı gibi yazılım parçalarının da değişken yapıları nedeniyle uygun olmadığı düşünülmüştür.

BÖLÜM 2

METRİK TABANLI YAZILIM HATA KESTİRİMİ YAKLAŞIMLARI

Yazılım faaliyetlerinin başarılı bir şekilde sonuçlanması, değerlendirilecek metriklerin doğru teşhisine ve yorumlanmasına bağlanmaktadır [22]. Yazılım kalitesinin sağlanmasında faydalanılan metrikler, yazılımların ölçüm ve geliştirilmesinde ön plana çıkmaktadır. Bu metrikler ile yazılımların hatalı parçalarının tespit edilmesi amacıyla kodun aşırı bağımlı, uyumsuz ve karmaşık kısımları tespit edilir. Bu sayede, yazılım parçalarından hangilerinin öncelikli olarak test edileceği, hangi kısımların daha yoğun test edilmesi gerektiği ve bakım faaliyetleri için ne kadar kaynak ayrılacağı gibi önemli bilgiler sağlanır [3].

Metrik tabanlı yazılım hata kestirimi konusunda yapılan çalışmalarda kullanılan farklı yaklaşımlar Marco D'Ambros ve diğerleri [23] tarafından beş ana gurupta toplanmıştır:

- Süreç Metrikleri Tabanlı Yaklaşım
- Önceki Kusurlardan Kestirim Yaklaşımı
- Kaynak Kod Metrikleri Tabanlı Yaklaşım
- Değişikliklerin Entropisi Tabanlı Yaklaşım
- Kod Çalkantısı (*Churn*) Kaynak Kod Metrikleri Tabanlı Yaklaşım

2.1 Süreç Metrikleri

Değişiklik metrikleri olarak da adlandırılırlar. Hatalara değişikliklerin sebep olduğu tezinden yola çıkarak değişikliklerin sayısı, son zamanlardaki değişikler vb. değerler

incelenir. Moser ve diğerlerinin [21] çalışmaları süreç metriklerine örnek olarak verilebilir. Yazılımdaki hatalar doğal olarak kodda yapılan bazı değişiklikler sonucunda ortaya çıkmaktadır. SVN gibi kod yapılandırma sistemlerinde tutulan kod sürümleri ve değişiklik tarihçeleri sayesinde kodda yapılan değişiklikler hakkında birçok farklı bilgi edinmek mümkündür. Bu sayede değişiklikler ile hatalar arasında bir ilişki belirlemek mümkün olur.

Belli başlı süreç metrikleri aşağıda verilmiştir:

Revizyon Sayısı (*NR, Number of Revision*): Koddaki yapılan değişiklik sayılarıdır. SVN gibi kod yapılandırma sistemlerinden çekilen bilgilerle elde edilir. Graves ve diğerlerinin çalışmalarında göstermiş olduğu üzere, hata kestirimi için geliştirilen en iyi lineer modeller, yapılan revizyon sayısı tabanlı oluşturulan modellerdir [17].

Yeniden Düzenleme Sayısı (*NREF, Number of Refactoring*): Bir yazılım parçasının kaç defa yapısal değişiklik/yeniden düzenleme çalışmasına tabi tutulduğunu gösterir. Yazılımı sürekli iyileştirmek ve yeni teknolojilere uyumlandırma amaçlı yapılan bu çalışmalar yazılımın kalitesini arttıran faaliyetler olmasına rağmen, dikkatli ve planlı yapılmayan bir yeniden düzenleme çalışması hatalara yol açabilir.

Dosyada Değişiklik Yapan Kişi Sayısı (*NAUTH, Number of authors who committed the file*): SVN gibi kod yapılandırma sistemleri sayesinde kodda yapılan değişikliklerin kimler tarafından yapıldığını bulmak mümkündür. Bir yazılım parçasında çok sayıda geliştiricinin değişiklik yapması, eğer bu geliştiriciler birbirinden habersiz olarak ve yazılım parçasının içyapısını ve ilişkilerini tam anlamadan değişiklik yapıyorlarsa hatalar ortaya çıkabilir.

Eklenen/Çıkarılan Satır Sayısı (*LINES, Lines added and removed*): Kodda yapılan her bir değişiklikte, eklenen ve çıkarılan satır sayısı bilgisi yapılan değişikliklerin boyutu konusunda faydalı bilgi vermektedir. Eklenen/Çıkarılan satır sayısı ile kodda oluşan hatalar arasında ilişki kurmak mümkündür.

Değişiklik Boyutu (*CHGSET, Change set size*): Kod değişiklik işleminin toplam boyu, yani eklenen-çıkarılan satır sayıları birlikte dikkate alınır.

Kod Çalkantısı (*CHURN, Codechurn*): Koddaki ciddi değişikliklerin tespitinde, koda yapılan ekleme ve çıkarmalara bakılmasından ziyade, matematiksel hesaplamalarla modellenerek kodun belirli aralıklarla örneği alınır ve örnekler arasındaki değişikliklerin boyutu (delta) hesaplanır.

Kod Yaşı (*AGE*): Kodun mevcut kesiti ile yapılan son değişiklikler arasındaki zaman hesaplanır. Örneğin, incelenen kod üzerinde 6 aydır değişiklik olmaması durumunda kodun yaşının 6 ay olması.

2.2 Önceki Kusurlar

Geçmişteki hatalardan yola çıkarak gelecekteki hataların tahmin edilebileceği düşüncesinden yola çıkılmıştır. Zimmermann ve diğerlerinin [24] çalışmaları bu yaklaşım için örnek teşkil eder. Bu yaklaşımda aşağıdaki metrikler kullanılır:

Hata Düzeltmede Yer Alma Sayısı (*NFIX, Number of times a file was involved in bug-fixing*): Örüntü işleme tabanlı sezgisel bir yaklaşımla, kod değişiklik (commit) yorumları incelenir. Yorumların "%hata%", "%kusur", "%fix", "%bug%" gibi örüntülerle eşleşip eşleşilmediğine bakılır.

Hata Veritabanında Yer Alma Sayısı (*BUGFIXES*): Yapılan değişikliklerin hata veritabanındaki karşılıklarının eşleştirilmesiyle ölçülen bir değerdir.

Hata Kategorilendirme (*BUG-CATG*): Geçmiş hataların kritik, yüksek öncelikli, önemli, basit gibi kriterlere ayırarak değerlendirmesiyle oluşur.

Bu metriklere ek olarak çalışmamızda faydalanılan metrikler aşağıda verilmiştir:

Önceki Hata (*PREVBUGS*): Kodun tahminde kullanılan kesitinde hata olup olmamasını belirtir.

Sonraki Hata (*NEXTBUGS*): Kodun tahminden sonraki kısmında hata olup olmamasını. Bu metrikle tahminin başarısı kontrol edilir.

2.3 Kaynak Kod Metrikleri

Karmaşık bileşenlerin değiştirilmesinin daha zor olduğu ve bu nedenle bu bileşenlerin hataya daha açık olduğu düşüncesine dayanır. Literatürdeki birçok çalışma CK

metriklerini kullanmaktadır. Buna ek olarak nesne tabanlı metrikler (*OOP*), kaynak kod satır sayısı (LOC) gibi çok farklı metriklerden de faydalanılmaktadır. Ek olarak bunların CK metrikleri ile birleşimi de hata kestiriminde tercih edilmektedir. CK metrikleri üzerinden hata kestirimi yapan çalışmalara, Chidamber & Kemerer modeli [25] ve Gyimothy ve diğerlerinin [26] kod satır sayısı (LOC) tabanlı hata kestirim yaklaşımı örnek olarak verilebilir.

Bu kategorideki metrikler aşağıdaki şekilde gruplanabilir:

- CK metrikleri
- Halstead metrikleri
- Tasarıma dayalı Metrikler
- Kod büyüklüğü ile ilgili metrikler
- Metotlarla alakalı metrikler
- Diğer metrikler

2.3.1 Chidamber & Kemerer (CK) Metrikleri

Ağırlıklı Metot Ortalaması (*WMC, Weighted Method Count*): Kodun karmaşıklığı (*cyclomatic complexity*) değerini ifade eder. Hesaplanmasında, metottaki toplam farklı yollar hesaba katılır. Buradaki değer incelenen birimdeki tüm metotların toplam WMC değerini ifade eder. Sınıftaki bütün metodların karmaşıklığı 1 ise, WMC değeri sınıftaki toplam metot sayısını verir. Ağırlıklı metot ortalaması değeri sınıfın geliştirilmesine ve bakımına ne kadar zaman harcanacağı hakkında fikir verir.

Miras Ağacı Derinliği (*DIT, Depth of Inheritance Tree*): Miras ağacındaki sınıfın seviyesini yani kök sınıfa uzaklığını belirtir. Kalıtım hiyerarşisinde daha derinde olan sınıflar, üst sınıflardan daha çok metot miras aldıkları için bunların davranışlarını tahmin etmek zorlaşır ve sınıf hataya açık hale gelir Hiç bir sınıftan türetilmemiş sınıflar için DIT değeri 0'dır. Örneğin, Java'da ara yüz (*interface*) tipindeki bir sınıfın miras ağacı derinliği 0'dır. Ayrıca her sınıfın atası konumunda bulunan *java.lang.Object* sınıfının da miras ağacı derinliği 0'dır. Java dilinde çoklu kalıtım yoktur ama çoklu kalıtıma izin veren dillerde DIT değeri hesaplanırken, en uzak köke olan derinlik dikkate alınır.

Ortalama Miras Ağacı Derinliği (*AvgDIT*): İncelenen kapsamdaki (paket/sınıf) tanımlı tiplerin ortalama miras ağacı derinliğidir.

Miras Alan Sınıf Sayısı (*NSC, Number Of Children*): Bir sınıftan doğrudan türetilen sınıfların sayısıdır.

Sınıf için potansiyel karşılık (*RFC, Response For Class*): Sınıfa gelen mesaja karşılık verilebilecek, potansiyel olarak çağrılabilecek metotların sayısının toplamıdır [27]. İncelenen sınıftaki bir nesnenin metodları çağrıldığında, bu nesnenin tetikleyebilceği tüm metodların toplam sayısı RFC değerini verir. Bu metrik sınıfın test maliyeti hakkında fikir vermektedir. Bir mesajın çok sayıda metot çağrısı tetiklemesi, sınıfın karmaşıklığının yüksek olduğunu gösterir ve bu da sınıftaki hataların ayıklanmasını zorlaştıracaktır [3].

Nesneler Arası Bağlaşım (*CBO, Coupling Between Objects*): Miras alınan sınıflar hariç, sınıfın çalışmak için ihtiyaç duyduğu sınıf sayısını gösterir.

Yordamlarda Yapışkanlık Eksikliği (*LCOM, Lack of Cohesion in Methods*): Yapışkanlık (cohesion), metotların uyumluluğu olarak da adlandırılır. Sınıf uyumluluğunun düşük olması, sınıfın daha fazla alt parçaya bölünmesi gerektiğini gösterir. Düşük uyumluluk, karmaşıklığı arttırdığından geliştirme aşamasında hata yapma ihtimalini yükseltir. Ayrıca metotlar arasındaki uyumsuzluk tasarımın kusurlu olduğuna dalalet eder [3]. Literatürde LCOM2, LCOM3, LCOM4 gibi farklı LCOM metrik tanımları vardır. Chidamber & Kemerer tarafından yapılan orjinal tanım şöyledir: C1 sınıfının M_1,M_2, … ,Mn metotlarına sahip olduğu ve $\{L_i\}$ kümesinin de Mi metodunda kullanılan nitelik değişkenleri kümesi olduğunu kabul edilirse, LCOM bu n adet kümenin kesişiminden oluşan ayrık kümelerin sayısıdır [25].

2.3.2 Halstead Metrikleri

Maurice Howard Halstead tarafından, 1977 yılında tanımlanan yazılım metrikleridir. Halstead'in amacı, maddelerin hacim, kütle, basınç vb. özellikleri gibi yazılımların da ölçülebilir özelliklerini ve aralarındaki ilişkiyi ortaya koymaktır. Halstead'a göre bir modülün okunurluğu ne kadar düşükse, modül o kadar hataya eğilimlidir [28].

Terim Sayısı (*NOOpd, Number of Operands*): Kapsamda kullanılan fonksiyon ismi, literal vb. terim sayısıdır.

Eşsiz Terim Sayısı (*NOUOpd, Number of Unique Operands*): Kapsamda kullanılan eşsiz terim sayısıdır.

Operatör Sayısı (*NOOpr, Number of Operators*): Aritmetik (+,-, vb.), eşitlik (<,>,vb.), atama (+=) gibi kapsamda kullanılan operatör sayısıdır.

Eşsiz Operatör Sayısı (*NOUOpr, Number of Unique Operators*): Kapsamda kullanılan eşsiz operatör sayısıdır.

Program Uzunluğu (*LEN, Program Length*): Program boyutunun uzunluk olarak tahminlenmesidir. Operatör ve terim sayısının toplamı şeklinde hesaplanır.

Kelime Haznesi (*VOC, Program Vocabulary*): Programı anlamak için gerekli kavramların yani programın kelime haznesinin tahminlenmesidir. Eşsiz Operatör ve Eşsiz terim sayılarının toplamı şeklinde hesaplanır.

Program Hacmi (*VOL, Program Volume*): Program boyutunun hacim olarak tahminlenmesidir. [Program uzunluğu] * log2([Kelime Haznesi]) şeklinde hesaplanır.

Zorluk (*DIFF, Difficulty*): ([Eşsiz Operatör Sayısı] / 2) * ([Terim Sayısı] / [Eşsiz Terim Sayısı]) şeklinde hesaplanır ve bulunan değer programın zorluğunu ifade eder.

Çaba (*EFF, Effort*): [Zorluk] * [Hacim] şeklinde hesaplanır ve programa harcanan çaba değerini ifade eder.

2.3.3 Tasarıma Dayalı Metrikler

Tasarıma dayalı metrikler, yazılım tasarım aşamasındaki gerçekleştirilen işlemlere göre sınıflandırmalardır [29].

İç içe Yuvalanmış Blok Derinliği (NBD, Nested Block Depth): Kod karmaşıklığının ölçümü için kullanılır. Koddaki if, for, while, do-while gibi kod blokların sadece içiçe yuvalanmış olanlarının, maksimum derinliğini sayısal olarak ölçerek elde edilen değerdir. Bu değerin yüksek olması, aynı anda dikkate alınması gereken koşulların çok olduğunu gösterdiği için, bu durum yazılımcıların kod üzerindeki kontrolünü azaltmakta ve mevcut hataların gözden kaçırılmasına sebep olmaktadır [30].

Ortalama Blok Derinliği (*Abd, Average Block Depth*): Sınıfta tanımlı metodların maksimum blok derinliği değerlerinin ortalamasıdır.

Çevrimsel Karmaşıklık (*VG, McCabe Cyclomatic Complexity*): McCabe tarafından tanımlanmıştır [31]. İncelenen birimdeki metotlardaki dallanmaların sayısı üzerinden kodun yapısal karmaşıklığını ölçen bir metriktir. Koddaki *for*, *if*, *while*, *do-while* blokları gibi akış kontrolleri ve karar noktaları temel alınarak bağımsız dallanmalar hesaplanır. Bu dallanmaların artması, karmaşıklığın artması anlamına gelir. Bu da kodun idaresinin zorlaşması ve hataların gizlenmesi anlamına gelmektedir. Ayrıca, yüksek çevrimsel karmaşıklık değerine sahip metotlarda, dallanmalar arttığı için yazılacak birim test sayısı da bu oranda artmakta ve kodun test edilmesi zorlaşmaktadır [30].

Ortalama Çevrimsel Karmaşıklık (*AvgCC, Average Cyclomatic Complexity*): İncelenen birimdeki her bir metodun ortalama çevrimsel karmaşık değeridir.

Soyutluk Oranı (*ABS, Abstractness Ratio*): İncelenen birimdeki soyut tiplerin (abstract class & interface) , tüm tiplere oranını ifade eder. Yüzdelik olarak yada 0-1 arasında bir değerle ifade edilir.

İç Eşleme (*Ca, Afferent Couplings/Fan In*): İncelenen tip dışında yer aldığı halde söz konusu tipe bağımlı elemanların sayısıdır (Referans veren sınıf sayısı). Tipin değişmesi halinde etkilenecek tip sayısını gösterir.

Dış Eşleme (*Ce, Efferent Couplings*): İncelenen tipin kendi dışında kaç tane tipe bağımlı olduğunu gösterir (referans verilen sınıf sayısı). Tipin yeniden kullanılabilirliğinin ölçüsüdür.

Kararsızlık (*INS, Instability*): [Dış Eşleme] / ([İç Eşleme] + [Dış Eşleme]) şeklinde hesaplanır. 0 ile 1 arasında değişen bu oran 1'e yaklaştıkça paketin kararlılığı azalır. (Yani paketin değişimi sistemdeki başka pek çok paket üzerinde değişiklik yapmayı gerektirir)

Uzaklık (*DIST, Distance*): [Soyutluk Oranı] + [Kararsızlık] - 1 şeklinde hesaplanır. Kararsızlık-Soyutluk grafiği çizildiğinde ideal ana çizgiye uzaklığı ifade eder.

2.3.4 Kod Büyüklüğü ile İlgili Metrikler

Büyüklük ifade eden metrikler şunlardır:

Kod Satır Sayısı (*LOC, Number of lines of code*): İncelenen birimdeki, boş satırların çıkarılması sonrası elde edilen asıl kod satırı sayısıdır.

Satır Sayısı (*NOLines, Number of Lines*): İncelenen birimdeki boş satırlar dâhil tüm satır sayılarıdır.

Metod Kod Satır Sayısı (*MLOC, Method Lines of Code*): İncelenen birimdeki sadece metodlara ait kod satır sayısı değeridir.

Ortalama Metod Kod Satır Sayısı (*AvgMLOC, Average Lines Of Code Per Method*): Sınıfta tanımlı metotların ortalam kod satır sayılarıdır.

Noktalı Virgül Sayısı (*NOSC, Number of Semicolons*): Kodların bitişini ifade eden noktalı virgüllerin incelenen birimdeki toplam sayısıdır. Bazen bir satıra birkaç işlem noktalı virgüllerle ayrılarak yazılabilir, amaç bunların toplam sayısını elde etmektir.

Karakter Sayısı (*NOChar, Number of Characters*): İncelenen birimdeki toplam karakter sayısıdır.

Değişkenlerle ilgili metrik değerleri şunlardır:

Değişken Sayısı (*NoA, Attributes*): İncelenen birimdeki toplam değişken sayısını ifade eder. Değişkenler, girdiğimiz değerleri alan ve atamayla değeri değişebilen veri tutuculardır

Tip Sayısı (*NoTy, Number of Types*): İncelenen birimdeki toplam tip sayısını ifade eder. Tip ile kasıt, Java sınıfları, C'deki struct gibi yapılardır.

Statik Değişken Sayısı (*NoStA, Number of Static Attributes*): İncelenen birimdeki statik (sınıf ortak değişkeni) olarak tanımlanan toplam değişken sayısını ifade eder.

Dışa Açık Değişken Sayısı (*NoPa, Number of public attributes*): İncelenen birimdeki dışa açık (*Public*), yani dışarıdan erişime açık toplam değişken sayısını ifade eder.

Gizli Değişken Sayısı (*NOPRA, Number of private attributes*): İncelenen birimdeki gizli (*Private*), yani dışarıdan erişime kapalı toplam değişken sayısını ifade eder.

Miras Alınan Değişken Sayısı (*NoAI, Number of attributes inherited*): İncelenen birimdeki miras alınan toplam değişken sayısını ifade eder.

Tip Başına Ortalama Değişken Sayısı (*AvgNoFPt, Average Number of Fields Per Type*): İncelenen birimdeki tanımlanan tip başına düşen ortalama değişken sayısıdır.

2.3.5 Metotlarla Alakalı Metrikler

Metotlarla ilgili metrikler şunlardır:

Metot Sayısı (*NOM, Number of methods*): İncelenen birimdeki toplam metod sayısıdır.

Dışa Açık Metod Sayısı (*NOPM, Number of public metho*ds): İncelenen birimdeki metodların dışarıdan erişilebilir olarak (public) tanımlanmış metod sayısı.

Statik Metot Sayısı (*NSM, Number of Static Methods*): İncelenen birimdeki metodların statik olarak (static) tanımlanmış metod sayısı.

Tekrar Tanımlanan Metod Sayısı (*NORM, Number of Overridden Methods*): Miras alan sınıf tarafından yeniden tanımlanmış metod sayısı.

Gizli Metod Sayısı (*NOPRM, Number of private me*thods): İncelenen birimdeki metodların dışarıdan erişilemez/gizli olarak (private) tanımlanmış metod sayısı.

Miras Alınan Metod Sayısı (*NOMI, Number of methods inherited*): İncelenen birimdeki kalıtım yoluyla miras alınan metodların sayısı.

Tip Başına Ortalama Metod Sayısı (*AvgNoMPt, Average Number of Methods Per Type*): İncelenen birimdeki tip başına tanımlı ortalama metod sayısı.

Yapılandırıcı Sayısı (*NOConstruct, Number of Constructors*): İncelenen birimdeki tanımlı toplam yapılandırıcı sayısını verir.

Tip Başına Ortalama Yapılandırıcı Sayısı (*AvgNoCPt, Average Number of Constructors Per Type*): İncelenen birimdeki tip başına düşen ortlama yapılandırıcı metod sayısıdır.

2.3.6 Diğer Metrikler

Diğer bazı metrikler şunlardır:

Alt Tip Sayısı (*AvgNoSt, Average Number of Subtypes*): İncelenen birimdeki kalıtımla miras alınan alt tip sayısıdır.

Yorum Sayısı (*NOComm, Number of Comments*): İncelenen birimdeki yorumların sayısı. Yazılan koddaki açıklanmış kısımları göstermesi açısından önemli kabul edilir.

Yorum Oranı (*CommR, Comments Ratio*): Yorumların kod satır sayısına oranını ifade eder.

Parametre Sayısı (*PAR, Number of Parameters*): Kapsamda geçen toplam parametre yani birimin çalışmasını değiştiren çalışma anında verilen değişken sayısıdır.

Ortalama Parametre Sayısı (*AvgPAR, Average Number of Parameters*): Kapsamdaki ortalama parametre sayısıdır.

Özelleşme (*SIX, Specialization Index*): Sınıf bazında [Tekrar Tanımlana Metot Sayısı] * [Miras Ağacı Derinliği] / [Metot Sayısı] şeklinde hesaplanır.

2.4 Değişikliklerin Entropisi

Kompleks değişikliklerin basit değişikliklere kıyasla hataya daha açık olduğu tezine dayanır. Hassan tarafından yapılan çalışma [32] bu yaklaşıma örnek olarak verilebilir. Bu fikrin temelinde, bir sistemdeki değişikliklerin ne kadar dağınık olduğunun belirli bir zaman aralığında ölçülmesi vardır. Tek bir dosyayı etkileyen değişiklik, birçok dosyayı etkileyen değişiklikten daha basit kabul edilir. Değişikliklerdeki yaygınlık arttıkça, karmaşıklık da artacaktır.

2.5 Kod Çalkantısı (Churn) Kaynak Kod Metrikleri

Bir koda yapılan eklemeler, silinmeler ve değişiklikler kodun hem ilgili hem de ilgisiz kısımlarında yan etkilere yol açabilme ihtimaline sahiptir. Bu yaklaşım, kaynak kod metriklerinin, kod çalkantısı (*churn*) tahmininde daha iyi bir ölçüt olduğu düşüncesine dayanmaktadır. Bu alanda yapılan ilk ve en iyi çalışmalardan biri Nikola ve diğerleri [33] tarafından yapılmış çalışmadır.

BÖLÜM 3

ÇALIŞMA İLE İLİŞKİLİ TEMEL VERİ ANALİZİ KAVRAMLARI

Yazılım hata kestiriminde kullanılan çeşitli yöntemlerden biri de veri analizidir. Çalışmamızda, veri analizi yöntemlerinden lojistik regresyon analizi üzerine odaklanılmıştır.

Lojistik regresyon analizinde temel amaç bir regresyon denklemi oluşturarak, bireylerin hangi grubun üyesi olduğunu kestirmektir [34].

3.1 Regresyon Analizi

Regresyon analizi, aralarında sebep-sonuç ilişkisi bulunan iki veya daha fazla değişken arasındaki ilişkiyi belirlemek ve ölçmek için kullanılır. Hayatın içindeki birçok olayda sebep sonuç ilişkisine rastlamak mümkündür. Örneğin, yaş ile boy, şehir nüfusu ile suç oranı, hayvana verilen yem miktarı ile alınan süt miktarı, çalışanın iş yükü ile stres gibi çeşitli değişkenler arasında ilişki ortaya koymak mümkündür. [35].

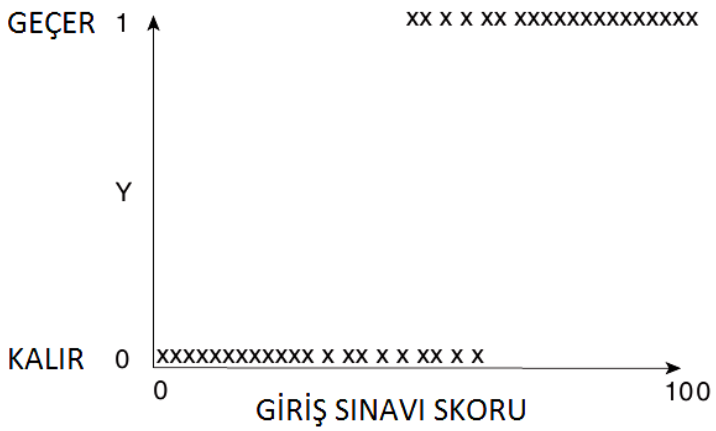
Değişkenler arasındaki bu ilişkiyi kullanarak o konu ile ilgili tahminler ya da kestirimler yapabilmek amacıyla regresyon analizi kullanılır.

3.2 Doğrusal Regresyon Analizi ile Lojistik Regresyon Analizinin Kıyaslanması

Doğrusal regresyon analizi ile Lojistik regresyon analizi değişkenlerin ölçüm biçimi yönünden farklılık gösterir. Doğrusal regresyon analizinde bağımlı değişken ve bağımsız değişkenler, sayısal (kesikli ya da sürekli) olarak belirtilir. Örneğin, yaş ile kan basıncı arasında bir ilişki aranacaksa; hem yaş, hem de kan basıncı sayısal olarak belirtilmelidir. Lojistik regresyonda ise bağımlı değişkenler nitelik ve kategorik bakımdan nitel olarak

değerlendirilirler. Cinsiyetin erkek ya da kadın olması buna örnek verilebilir. Lojistik regresyonda bağımsız değişkenler sayısal ya da nitel değerler alabilirler [36].

Aşağıdaki örnekte 200 öğrenci, bir kurs bitiminde geçti/kaldı şeklinde değerlendirilmiştir. Bu kursa girişte de bir ölçme sınavı uygulanmıştı. Kursa girişte uygulanan test skoru ile kurstan geçme arasındaki ilişkiyi bulmak için lojistik regresyondan faydalanılabilir. Bu örnekte doğrusal regresyon kullanmak olası değildir. Çünkü bağımlı değişken sonucu 1 ya da 0 değerini alabilir. Doğrusal regresyon ile sınırsız bir aralıkta sayısal değer elde edilebilir [37].



Şekil 3.1 Öğrenci giriş sınavı ile kurs geçme arasındaki ilişki araştırması örneği [37]

3.3 Lojistik Regresyon Analizi

Uygulamalı sosyal bilimlerde karşılaşılan ve araştırılan olaylara ilişkin elde edilen verilerde, çoğunlukla bağımlı değişkenin iki mümkün değerinden birine sahip olabileceği varsayılmaktadır. Örneğin bir kişi okuldan mezun olmuştur ya da olmamıştır, bir işçi çalışıyordur ya da işsizdir, bir kişi bir gruba üyedir ya da değildir, bir hasta tedaviye cevap verebilir ya da vermeyebilir. İki olası ve farklı değer içeren bu tür verilere iki değerli veriler denilmektedir. İki değerli veya ikili değişkenler literatürde (0;1) değişkenleri olarak da adlandırılmaktadır. İki değerli değişkenler ile çalışan bir araştırmacının hedefi, bağımsız değişkenlerin koşullu bir kümesine bağımlı olarak, başarı veya başarısızlık olasılığını kestirmektir[37].

Lojistik regresyon analizi, bağımlı değişkenin ölçüldüğü ölçek türüne ve bağımlı değişkenin seçenek sayısına göre üçe ayrılmaktadır [34,35]:

İkili (Binominal) Lojistik Regresyon Analizi: Bağımlı değişkenin iki düzeyi olduğunda kullanılır. Örneğin, hasta/sağlam, yaşıyor/öldü, etkili/etkisiz vb.

Sıralı (Ordinal) Lojistik Regresyon Analizi: Bağımlı değişken sıralı nitel veri tipinde olduğunda kullanılır. Örneğin, hafif/orta/şiddetli, çok etkili/orta derecede, etkili/etkisiz vb.

Çok Kategorili/Düzeyli İsimsel (Multinominal) Lojistik Regresyon Analizi: Bağımlı değişken ikiden çok düzeyli sıralı olmayan nitel veri tipinde olduğunda kullanılır. Örneğin, çalışıyor/çalışmıyor/emekli vb.

Lojistik regresyon, bağımsız değişkenin sayısına göre "tek değişkenli lojistik regresyon" ve "çok değişkenli lojistik regresyon" olarak da sınıflandırılmaktadır. Lojistik regresyon yönteminin hedefi, bağımlı değişkenin sonucunu tahmin edebilecek en sade modeli bulmaktır. Lojistik regresyon analizi sonucunda elde edilen modelin uygun olup olmadığı "model ki-kare" testiyle, her bir bağımsız değişkenin modelde varlığının anlamlı olup olmadığı ise Wald istatistiği ile sınanır [36].

3.4 Lojistik Regresyon Analizi Terimleri

Lojistik regresyon ile ilgili bazı terimler bu bölümde verilmiştir:

Bahis (Odds): Başarı ya da görülme olasılığının (p), başarısızlık ya da görülmeme olasılığına (1 - p) oranıdır.

Bahis Oranı (*Odds Ratio, OR*): İki sonuca ait bahis (odds) değerinin birbirine oranıdır.

Lojit: Bahis oranının doğal logaritmasıdır. Bahis oranı asimetriktir. Bu değerin doğal logaritması alınarak simetrik hale dönüştürülür Lojit katsayıları (lojit) doğrusal regresyon analizindeki "β" katsayısının karşılığıdır.

En Çok Olabilirlik Yöntemi (*Maksimum Likelihood Estimation*): Lojistik regresyonda model kestiriminde en küçük kareler (*ordinary least square*) yöntemi yerine, en çok olabilirlik (*maximum likelihood*) yöntemi kullanılır.

Lojistik regresyonun istatistiksel modeli aşağıda verilmiştir. “β”katsayılarının hesaplanması bölümün devamında anlatılmaktadır.

$$\log \frac{p(x)}{1-p(x)} = \beta_0 + x \cdot \beta \tag{3.1}$$

Buradan yola çıkarak p, yani bir kategoriye ait olma olasılığı aşağıdaki şekilde bulunur:

$$p(x;b,w) = \frac{e^{\beta_0 + x \cdot \beta}}{1 + e^{\beta_0 + x \cdot \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \cdot \beta)}} \tag{3.2}$$

Lojistik regresyon analizi uygulanırken aşağıdaki noktalara dikkat edilmelidir [36].

- Uygun tüm bağımsız değişkenler modele dâhil edilmelidir. Bazı değişkenlerin modele dâhil edilmemesi hata teriminin büyümesine ve modelin yetersizliğine neden olabilir.
- Uygun olmayan tüm bağımsız değişkenler de dışlanmalıdır. Nedensel olarak uygun olmayan değişkenlerin modele dâhil edilmesi; modeli karmaşık hale getirebilir, modelin yorumlanmasının zorlaştırabilir, bu değişkenlerin bağımlı değişken üzerinde pay sahibiymiş gibi yanlış izlenim vermesine neden olabilir.
- Aynı birey üzerinde bir kez gözlem yapılmalı, tekrarlayan ölçümler olmamalıdır.
- Bağımsız değişkenlerde ölçüm hatası küçük olmalıdır. Ölçüm hataları küçük olmalı, kayıp (eksik) veri olmamalıdır. Hatalar, katsayıların tahmininde yanlılığa ve modelin yetersizliğine neden olur.
- Bağımsız değişkenler arasında çoklu bağlantı (*Multicollinearity*) olmamalıdır. Bağımsız değişkenler birbirleriyle ilişkili olmamalıdır.
- Aşırı değerler olmamalıdır. Doğrusal regresyonda olduğu gibi, aşırı değerler sonucu önemli derecede etkileyebilir.
- Örneklem büyüklüğü yeterli olmalıdır. Az sayıda birey içeren örneklemde tahmin edilen değerlerin güvenilirliği azalır. Kural olarak, modeldeki her bağımsız değişken için en az 10 birey önerilmektedir.

- Bağımlı değişkenin beklenen varyansı ile gözlenen varyansı arasında büyük bir fark varsa modelin yetersiz olduğu ve yeniden tanımlanması gerekir. Bunun olası nedenleri, örneklemin rastgele yöntemle seçilmesi ya da araştırma düzeninde ciddi sorun olması olabilir.

Örnek olarak, yaşı 60 yaşından büyük olanları yaşlı, küçük olanları genç kabul ederek, hastalığa yakalanma ile yaşlılık arasındaki ilişkiyi Tablo 1'deki örnek veri setine göre inceleyebiliriz. Burada bağımlı değişken, hasta olup olmama, bağımsız değişken ise yaşlılık durumu olacaktır.

Çizelge 3.1 Yaşlılığa göre hastalığa yakalanma verisi

Yaşlılık (A)	**Hasta**	**Değil**	**Toplam**
Genç (<60)	22	51	73
Yaşlı (>60)	21	6	27

Bu veriden yola çıkarak aşağıdaki lojistik regresyon hesaplamaları yapılabilir:

$$Odds(genç)=22/51=0,431 \quad (3.3)$$

$$y=ln(0,431) = -0,841 \quad (3.4)$$

$$Odds(yaşlı)=21/6=3,5 \quad (3.5)$$

$$y=ln(3,5)=1,253 \quad (3.6)$$

$$Bahis\ Oranı =3,5/0,431=8,121 \quad (3.7)$$

Buradaki bahis oranı değerinin anlamı şudur; yaşı 60'tan büyük olanların hasta olma oranları, yaşı küçük olanlara göre 8,121 kat daha fazladır.

Modeli yaşlı olma parametresine göre kurarak A: yaşlı olup olmama (1/0) değerini alır.

$$b0= ln(0,431) = -0,841 \quad (3.8)$$

$$b1= ln(3,5)-ln(0,431) = 1,253 - (-0,841) =2,094 \quad (3.9)$$

$$y = -0,841+2,094A \quad (3.10)$$

Lojistik regresyon modeli bu durumda aşağıda gösterilmiştir:

$$p=exp(-0,841+2,094A) / (1 + exp(-0,841+2,094A)) \quad (3.11)$$

Örneğin A=1 yani yaşlı birisi, %77,8 oranında hasta olacaktır.

$$p=exp(-0,841+2,094 . 1) / (1 + exp(-0,841+2,094.1)) \quad (3.12)$$

$$p=3,5008 / 4,5008 = 0,778 \quad (3.13)$$

3.5 Genel İstatistik Terimleri

Bu bölümde çalışmada ve deneylerde kullanılacak ve sıkça referans verilecek belli başlı genel istatistik kavramları ve terimler açıklanacaktır.

3.5.1 Değişkenlerle İlgili Kavramlar

Metriklerin ifade edilmesinde genelde sayısal değerler kullanılsa da, bulunup bulunmama durumu gibi bazı metrikler veya değişkenler evet/hayır gibi değerler alırlar. Bu açıdan bakıldığında değişkenler nicel ve nitel olmak üzere iki tipe ayrılmaktadır [38].

Nicel (kantitatif) Değişkenler: Tanımlı olduğu aralıkta sadece tam sayı değerleri alabilen değişkenlerdir. Ör. Koddaki satır sayısı

Nitel (Kalitatif/Kategorik) Değişkenler: Ölçüm veya sayımla ifade edilemeyen değişkenlerdir. Kodlanarak sayısal hale dönüştürülebildikleri için kesikli değişkenlerin özel bir türü olarak düşünülebilir. Ör. Cinsiyet, başarılılık

3.5.2 Model Başarımı Kavramları ve Değerlendirme Kriterleri

Model başarımlarının değerlendirilmesi için öncelikle bazı istatistiksel kavramları bilmek gerekmektedir. İkili sınıflandırmalarda doğruluk ve kesinlik doğru pozitif (TP) yanlış pozitif (FP), yanlış negatif (FN) ve doğru negatif (TN) olmak üzere Çizelge 3.2'de gösterildiği üzere dört ana başlıkta incelenir [39,40]. Burada Yanlış Pozitif en dikkat

edilmesi gereken sınıflandırmadır, çünkü örnek pozitif sonuç verirken, testlerde negatiflik belirtilmesi sıkıntı yaratır.

Çizelge 3.2 Doğruluk sınıflandırması

		Öngörülen Sınıf (Test)	
		Pozitif	Negatif
Örnek Sınıfı	Pozitif	Doğru Pozitif (TP)	Yanlış Negatif (FN)
	Negatif	Yanlış Pozitif (FP)	Doğru Negatif (TN)

- **Doğru Pozitif (*TP, True Positive*):** Örnek şartı sağlamakta ve testler de pozitif sonuç vermekte. (Örnek = Test = Pozitif)
- **Doğru Negatif (*TN, True Negative*):** Örnek şartı sağlamıyor ve testler de negatif sonuç vermekte. (Örnek = Test = Negatif)
- **Yanlış Pozitif (*FP, False Positive*):** Örnek şartı sağlamıyor ama testler pozitif sonuç vermekte.
- **Yanlış Negatif (*FN, False Negative*):** Örnek şartı sağlıyor ama testler negatif sonuç vermekte. Örneğin yazılımda gerçekte hata varken, testlerin hata olmadığını söylemesi gibi.
- **Doğruluk Oranı:** Model başarımının ölçülmesindeki en popüler ve basit yöntemdir. Doğru sınıflandırılmış örnek sayılarının toplam örnek sayısına oranı, modele ait doğruluk oranını verir.

$$Doğruluk = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.14}$$

- **Hata Oranı:** Yanlış sınıflandırılmış örnek sayılarının toplam örnek sayısına oranı, modele ait hata oranını verir.

$$Hata\ Oranı = \frac{FP + FN}{TP + FP + FN + TN} \tag{3.15}$$

- **Kesinlik (*Positive Predictive Value*):** Doğru sınıflandırılmış pozitif örnek (TP) sayısının tüm pozitif örnek (TP+FP) sayılarına oranıdır. Bir başka deyişle, pozitif test sonuçlarının doğru olması ihtimalidir.

$$Kesinlik = \frac{TP}{TP + FP} \tag{3.16}$$

- **Negatif Kesinlik (*Negative Predictive Value*):** Doğru sınıflandırılmış negatif örnek (TN) sayısının, tüm negatif örnek (TN+FN) sayılarına oranıdır. Bir başka deyişle, negatif test sonuçlarının doğru olması ihtimalidir.

$$Negatif\ Kesinlik = \frac{TN}{TN + FN} \tag{3.17}$$

- **Duyarlılık (*Sensitivity/TP rate*) :** Doğru sınıflandırılmış pozitif örnek sayısının (TP), toplam pozitif örnek sayısına oranıdır (TP+FN). Bir başka deyişle, artı sağlayan örnekler arasında, test sonucunun pozitif olması yani doğru sınıf öngörümü yapılabilmesi ihtimalidir.

$$Duyarlılık = \frac{TP}{TP + FN} \tag{3.18}$$

- **Seçicilik (*Specificity/Selectivity/TN rate*) :** Doğru sınıflandırılmış negatif örnek sayısının (TN), toplam pozitif örnek sayısına oranıdır (TP+FN). Bir başka deyişle, negatif sağlayan örnekler arasında, test sonucunun negatif olması yani doğru sınıf öngörümü yapılabilmesi ihtimalidir.

$$Seçicilik = \frac{TN}{TN + FP} \tag{3.19}$$

- **Yanlış Pozitiflik Oranı (*Type 1 Error/α/ p-Value/ false positive rate*):** Koşula sahip olmayan örnekler için pozitif test çıkma şansı. (1-Seçicilik)

$$\alpha = \frac{FP}{FP + TN} \tag{3.20}$$

- **Yanlış Negatiflik Oranı (Type 2 Error/β/false negative rate):** Koşula sahip olan örnekler için negatif test çıkma şansı. (1-Duyarlılık) Negatif Test

Sınıflandırmalarının, Negatif ve Pozitif test sonuçlarına oranı şeklinde hesaplanır.

$$\beta = \frac{FN}{FN + TP} \tag{3.20}$$

- **Hatalı Keşif Oranı (q-value/ False Discovery Rate):** Pozitif teste sahip olan örneklerin koşula sahip olmaması şansı.

$$q\text{-}value = \frac{FP}{FP + TP} \tag{3.21}$$

- **Hatalı İhmal Oranı (False Omission Rate):** Negatif teste sahip olan örneklerin koşula sahip olması şansı.

$$Hatalı\ İhmal\ Oranı = \frac{FN}{(FN + TN)} \tag{3.22}$$

- **F-Ölçütü:** Kesinlik ve duyarlılık ölçütlerinin tek başına anlamlı olmayacağı ve birlikte değerlendirilmesi ile daha anlamlı bir karşılaştırma yapılabileceği düşünüldüğü için f-ölçütü kesinlik ve duyarlılığın harmonik ortalaması şeklinde tanımlanmıştır.

$$F-Ölçütü = \frac{2 \times Duyarlilik \times Kesinlik}{Duyarlılık + Kesinlik} \tag{3.23}$$

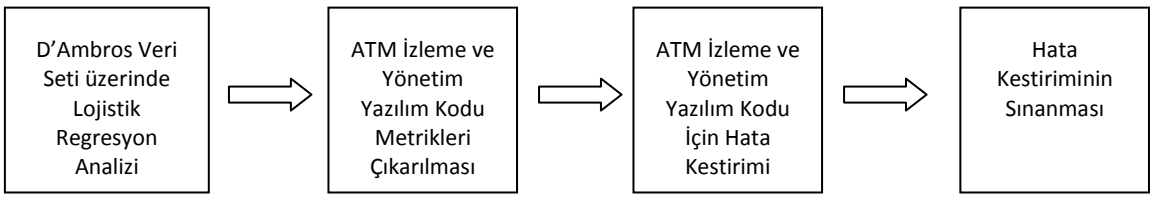
- **Anlamlılık (*Significance*):** Test edilen örnekler hakkında veriden ne çıkarım yapılabileceğine bakmaksızın, testin genel kalitesini ölçmek amacıyla anlamlılık değeri hesaplanabilir. Anlamlılık değeri sıfıra yaklaştıkça, anlamlılık artar.

$$\chi 2 = \frac{[\ (TP*TN - FP*FN)^2\ \ (TP+TN+FP+FN)\ ]}{[\ (TP+FP)(TP+FN)(TN+FP)(TN+FN)\ ]} \tag{3.24}$$

BÖLÜM 4

HATA KESTİRİM MODELİ OLUŞTURMA SÜRECİ

Hata kestirim modeli oluşturma sürecini oluşturan adımlar Şekil 4.1'de gösterilmektedir.



Şekil 4.1 Çalışma adımları

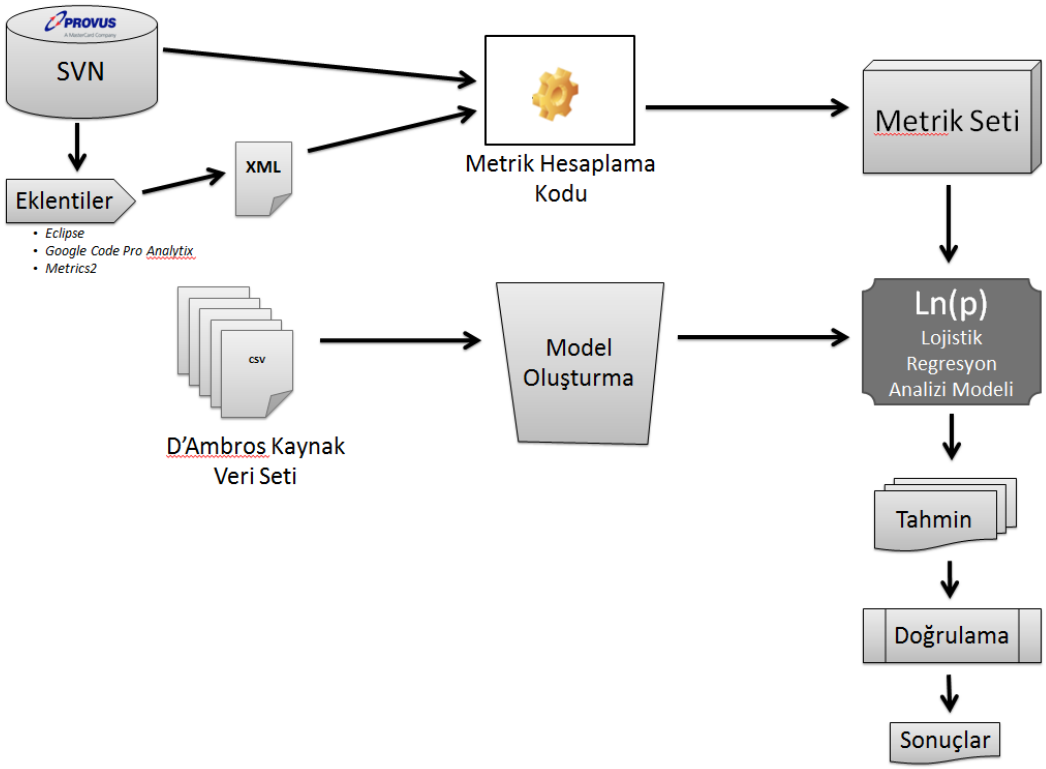
Çalışma kapsamında öncelikle Marco D'Ambros tarafından hazırlanmış hazır bir hata kestirimi veri tabanı ele alınmıştır [2]. Bu veri tabanındaki verilerin bir kısmı test için ayrıldıktan sonra, kalan veriler lojistik regresyon analizine tabi tutularak bir model oluşturulmuştur.

Çalışmamızın ikinci bölümünde ise, elde edilen bu modelin *Provus Bilişim Hizmetleri* bünyesinde geliştirilen *ATM İzleme ve Yönetim Yazılımı* kodları üzerine uygulanması hedeflenmiştir. Ancak bunu yapabilmek için öncelikle, söz konusu yazılım kodlarının belirlenen bir zaman dilimindeki (t zamanı) kesitinin analiz edilmesi ve metrik değerlerinin eldeki hata kestirim veri tabanına uygun bir şekilde çıkarılması gerekmiştir.

Bu adım sonrası, ilk bölümde elde edilen model, *ATM İzleme ve Yönetim Yazılımı* kodları metrikleri üzerine uygulanmış ve sınıflarda hata bulunma ihtimalleri tahminlenmiştir.

Son adımda da, yapılan bu tahminlerin başarısının ölçülmesi için kodun analiz edilen kesitinden sonraki bir zaman dilimindeki (t+1 zamanı) kesitine kadar, kodda hata tespit edilip edilmediği ölçümlenerek tahmininin başarımı sınanmıştır.

Hata kestirim modeli oluşturulması uygulama süreci modeli Şekil 4.2'de gösterilmektedir:



Şekil 4.2 Geliştirilen uygulama süreci modeli

4.1 Kullanılan Kaynak Veri Seti

Çalışma kapsamında öncelikle bir hata kestirim veri tabanı oluşturma üzerinde durulmuş ve modelin oluşturulmasında kaynak veri seti olarak Marco D'Ambros tarafından hazırlanmış hata kestirimi veri tabanı ele alınmıştır [2]. Öncelikle kullanılan kaynak veri seti hakkında bilgi verilecek ve daha sonra hata kestirim modelinin oluşturulmasına değinilecektir.

Çalışmamızın çıkış noktası olarak, Marco D'Ambros tarafından hazırlanmış hata kestirimi veri tabanından faydalanılmıştır. Bu veri seti Çizelge 4.1'de gösterildiği gibi 5 farklı projeden toplamda 5371 örneklem içermektedir. Verilerin elde edilmesinde

popüler açık kaynak kodlu bazı projelerin kodlarından faydalanılmıştır. Veri tabanında CK, nesneye dayalı programlama, değişim, entropi gibi birçok farklı metrikler ve dosyada/sınıfta hata bulunup bulunmadığı bilgisi verilmektedir. Bu veri tabanından sadece belli tipte metrikler seçilerek lojistik regresyon analizi modeli oluşturmada kullanılmıştır.

Çizelge 4.1 Veri setinin oluşturulmasında kullanılan kod kaynakları [23]

Sistem/Proje	**Versiyon**	**Zaman Dilimi**	**Sınıf Sayısı**
Eclipse JDT Core www.eclipse.org/jdt/core/	3.4	1.1.2005 - 17.06.2008	997
Eclipse PDE UI www.eclipse.org/pde/pde-ui/	3.4.1	1.1.2005 - 11.09.2008	1562
Equinox framework www.eclipse.org/equinox/	3.4	1.1.2005 - 25.06.2008	439
Mylyn www.eclipse.org/mylyn/	3.1	17.01.2005 - 17.03.2009	2196
Apache Lucene lucene.apache.org	2.4	1.1.2005 - 8.10.2008	691

4.2 Hata Kestirim Modelinin Oluşturulması

Söz konusu hata kestirimi veri tabanı R dili ve R-Studio aracı [41] kullanılarak lojistik regresyon analizine tabi tutulmuştur. 5371 örneklemden oluşan verinin %10'u test için ayrılmış, geri kalan kısmı lojistik regresyon modelinin oluşturulmasında kullanılmıştır.

Lojistik regresyon modeli, bir kod dosyasında hata olup olmama durumunun diğer niteliklerle bağlantısı üzerine oluşturulmuştur. Bu işlem sonucunda model katsayıları, katsayıların istatistiksel olarak anlamlı olup olmadığını test etmek için kullanılan Wald Z istatistik testi değeri (Z) ve bu değere karşılık gelen olasılık değeri yani Pr(>|z|) elde edilmiştir. Pr(>|z|) değerinin küçük olması istatistiğin anlamlı olduğunu gösterir. İstatistiğin anlamlılığı simgesel olarak (*) ile gösterilmiştir.

$$y = logit(p) \quad (4.1)$$

$$p = exp(y) / (1 + exp(y)) \quad (4.2)$$

Bu işlemler sonucunda, kodda hata bulunma ihtimalini veren lojistik regresyon denklemi elde edilmiştir.

4.3 İncelenen Yazılımdan Metriklerin Eldesi

Provus Bilişim Hizmetleri bünyesinde geliştirilen ATM İzleme ve Yönetim Sistemi kaynak kod dosyaları üzerinden metrik değerlerinin hesaplanması için, yazılımın belirli bir sürümü tahmin kesiti olarak seçilmiş ve bu sürüme ait kaynak kod dosyaları analiz edilmiştir.

Kaynak kod dosyası üzerinden metrik değerlerinin hesaplanması için çeşitli araçlardan faydalanılmıştır. Bu araçlar tarafından hesaplanan çok çeşitli metrik değerleri XML formatında çıktı olarak oluşmuş ve oluşan bu çıktı dosyalarının, lojistik regresyon analizinde kullanılmak üzere uygun bir formata dönüştürülmesi gerekmektedir. Bu iş için, java ortamında bir metrik hesaplama uygulaması geliştirilmiş ve çıktı dosyalarının işlenerek uygun formata dönüştürülmesi sağlanmıştır. Kod metriklerinin hesaplanması süreci, bölüm 5.4'de ayrıntılı olarak açıklanmıştır.

BÖLÜM 5

GELİŞTİRİLEN MODELİN UYGULANACAĞI YAZILIMIN YAPISININ ve METRİKLERİN İNCELENMESİ

5.1 ATM İzleme ve Yönetim Sistemi Yazılımı Kodunun Analizi

Çalışmamızda bir sonraki adım, modelin *Provus Bilişim Hizmetleri* bünyesinde geliştirilen yazılımların kodları üzerine uygulanmasıdır. Çalışmada kullanılan, hata veri tabanı Çizelge 4.1'de gösterilen popüler açık kaynak kodlu projelerin kodlarından derlenmiştir. Bu veri seti üzerinden oluşturulan modelin, tamamen farklı bir kod ortamında sınanması ve başarısının görülmesi önemlidir. Bu nedenle aktif olarak kullanılan bir yazılım projesi üzerinde uygulama gerçekleştirilmiştir.

5.2 Proje Özellikleri

Provus Bilişim Hizmetleri bünyesinde geliştirilmekte olan ATM yönetim sistemi yazılımı birçok farklı yazılım parçasından (modülden) oluşan büyük ölçekli bir yazılımdır. Bu yazılım 7-24 ATM'lerle iletişim halinde çalışan canlı bir sistemdir [42]. Bu yazılımı oluşturan proje kodları Java teknolojileri ile geliştirilmiştir ve 250 bin satırdan fazla kod içermektedir.

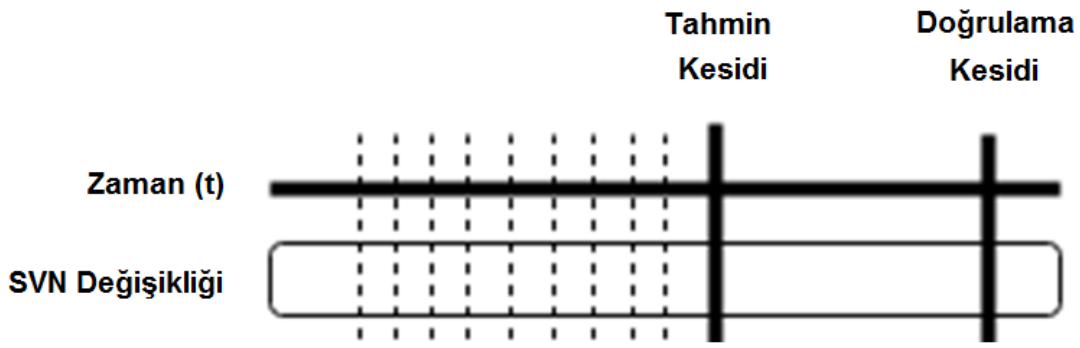
5.3 Kod Metriklerinin Toplanması

ATM yönetim sistemi yazılımı kodları, şirket dâhilindeki bir yapılandırma (konfigürasyon) yönetim sistemi sunucusunda tutulmaktadır. Konfigürasyon yönetim sistemi olarak Subversion (SVN) ürünü [43] kullanılmaktadır. Söz konusu yazılım

kodlarından metriklerin elde edilmesi için Eclipse Kod Geliştirme Ortamı [44], Google Code Pro Analytix kod kalitesi aracı [45,46] ve Metrics2 Eclipse eklentisi [47] araçlarından faydalanılmıştır.

SVN sisteminde proje ile ilgili dosyalar ve bu dosyalar üzerinde yapılmış değişiklik bilgileri tutulmaktadır. Bir değişiklik bilgisi, SVN sistemindeki dosyalarda yapılan değişiklik, ekleme veya silme işleminden oluşmaktadır. Her bir işlem sürüm numarası, tarih-saat, değişikliği yapan ve yorum bilgisi içerir.

Java uygulamalarının doğasında her bir kaynak kod dosyası tek bir ana sınıf ile birlikte, alt sınıflar da içerebilir. Kaynak kod setinde olduğu gibi, çalışma kapsamında da yalnızca ana sınıf dosyası dikkate alınmıştır. Çünkü metrikler dosya bazında hesaplanmaktadır ve bir dosyada birden fazla sınıf tanımlanması doğru bir yaklaşım kabul edilmez.

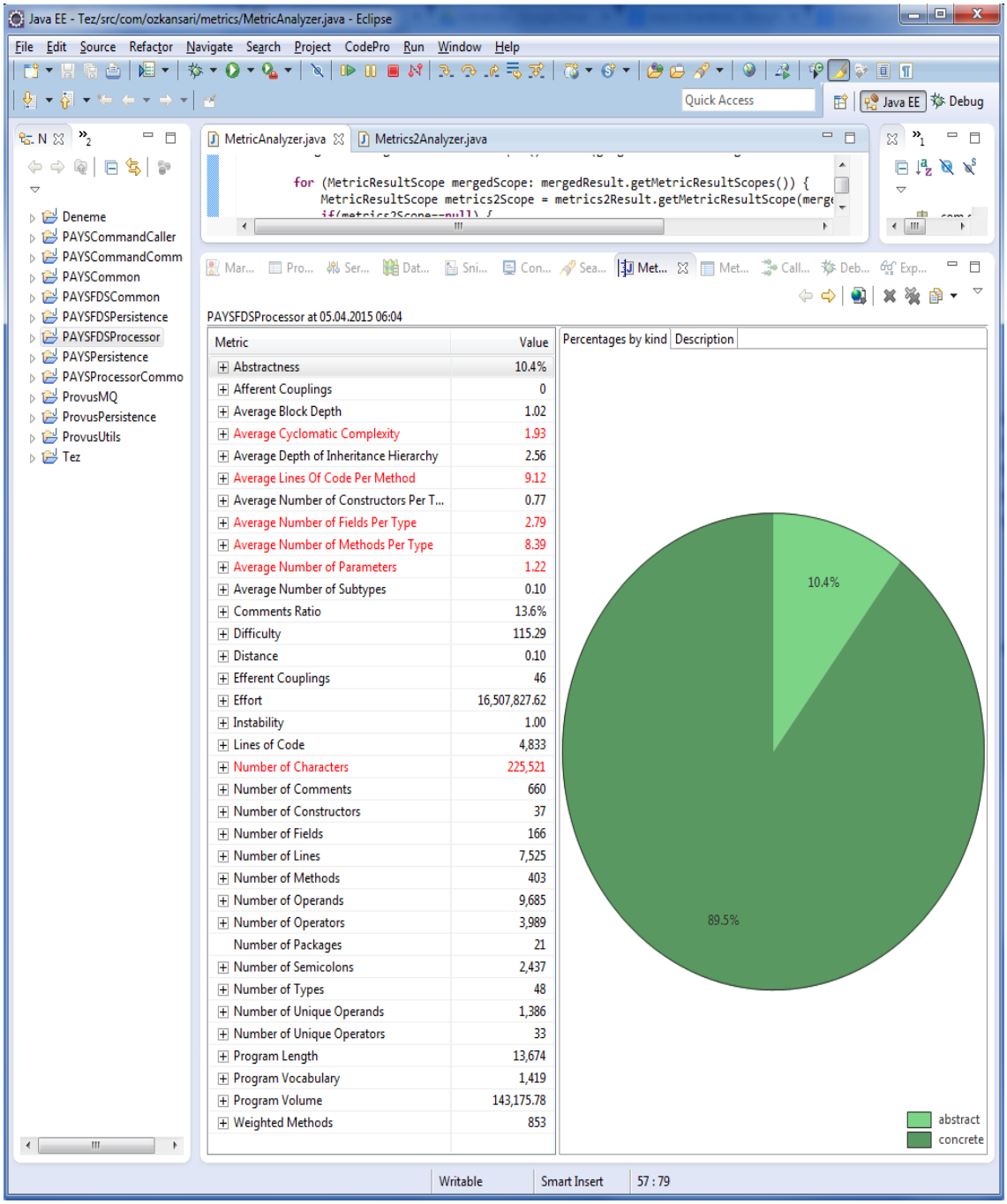


Şekil 5.1 SVN kesiti üzerinden metriklerin elde edilmesi

Kaynak kod dosyası üzerinden metrik değerlerinin hesaplanması için, belirli bir SVN sürümü tahmin kesiti olarak seçilmiş ve bu sürüme ait kaynak kod dosyaları analiz edilmiştir.

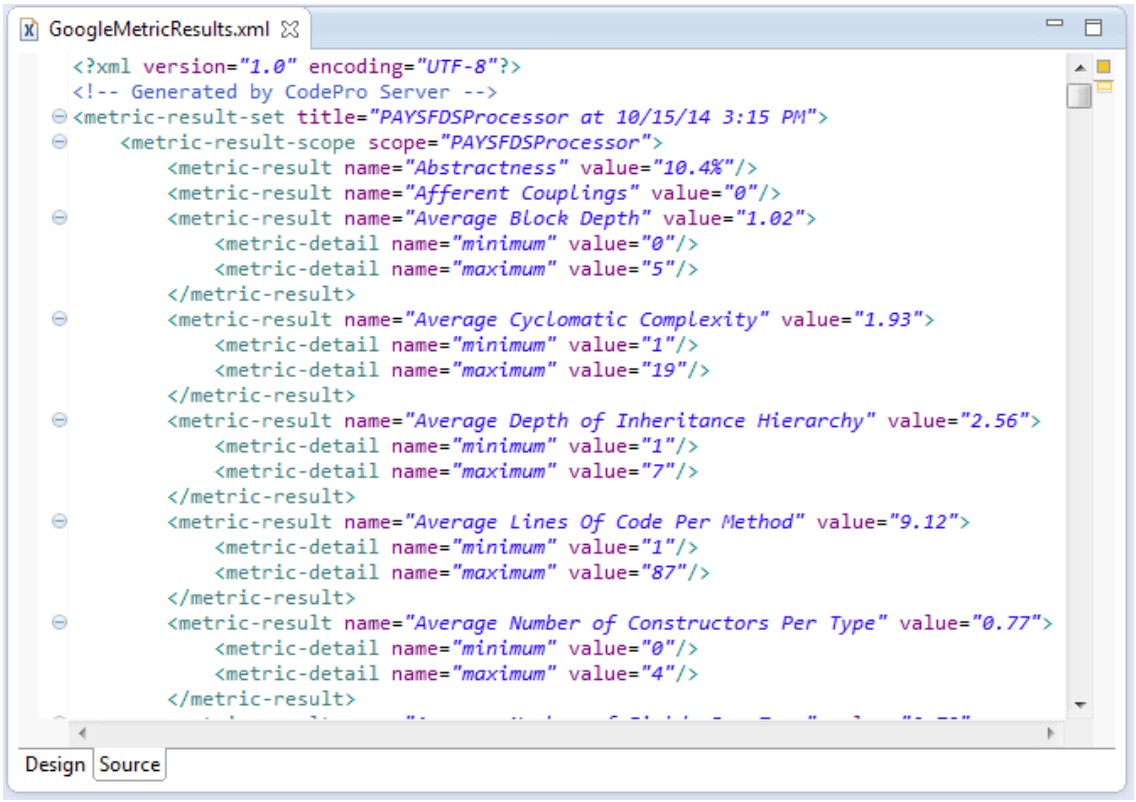
Kaynak kod dosyası üzerinden metrik değerlerinin hesaplanması için Eclipse Yazılım geliştirme ortamında kurulu, Google Code Pro Analytix kod kalitesi aracı ve Metrics2 Eclipse eklentisi kullanılmıştır. Bu araçlar tarafından hesaplanan çok çeşitli metrik değerleri kendilerine has formatta XML formatında çıktı olarak oluşmaktadır. Oluşan bu çıktı dosyalarının, lojistik regresyon analizinde kullanılmak üzere uygun bir formata dönüştürülmesi gerekmektedir. Bu iş için, çalışma kapsamında bir uygulama geliştirilmiş ve çıktı dosyalarının işlenerek uygun formata dönüştürülmesi sağlanmıştır.

Google Code Pro Analytix aracı, Eclipse Yazılım Geliştirme ortamıyla entegre bir şekilde çalışmaktadır. Çok çeşitli özelliklerinin yanı sıra, sunduğu kod analizi ve metrik değerlerinin proje, paket, sınıf ve metot bazında detaylı bir şekilde listelendiği bir ara yüz sunmaktadır. Şekil 5.2'de gösterilen metrik tablosu ara yüzü, çok çeşitli metrik değerlerini proje bazında göstermektedir.



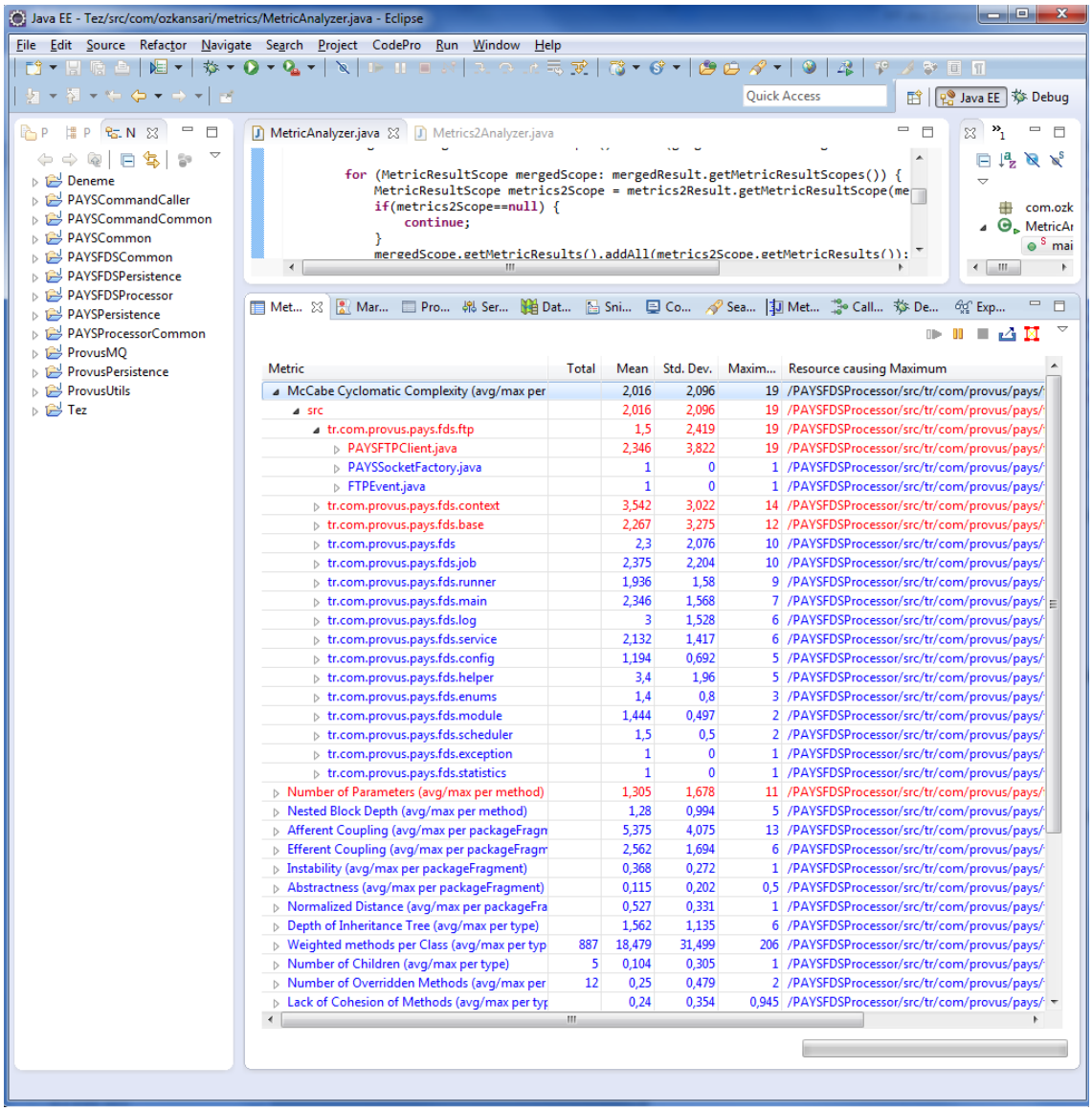
Şekil 5.2 "Google Code Pro Analytix" aracından metrik değerlerinin elde edilmesi

Ayrıca bu metrik değerleri XML formatı da dâhil olmak üzere çok çeşitli formatlarda çıktı olarak da alınabilir. Şekil 5.3'de örnek bir XML çıktısı gösterilmiştir. XML dosyası sadece ihtiyacımız olan sınıf bazında metrik değerleri dışında birçok farklı bilgi de içermektedir. Ayrıca buradaki değerlerin lojistik regresyon analizine girdi olarak verilebilmesi için, metrik değerlerinin sınıf bazında tek tek elde edilmesi gerekmektedir.



Şekil 5.3 "Google Code Pro Analytix" çıktı örneği

Benzer şekilde *Metrics2* aracı da Eclipse Kod Geliştirme ortamına bir eklenti olarak yüklenir. Bu araç sayesinde, Şekil 5.4'de gösterildiği gibi çeşitli metrik değerleri detaylı olarak paket, dosya ve metod bazında sunulur. *Google Code Pro Analytix* aracının sağladığı metrik değerleri ile *Metrics2* aracının metrik değerleri birbirinden ayrı olduğundan, *Google Code Pro Analytix* ile elde edilemeyen bazı metrik değerleri bu araç kullanılarak elde edilecektir.



Şekil 5.4 Metrics2 aracından metrik değerlerinin elde edilmesi

Metrics2 aracının sunduğu sonuçlar, benzer şekilde proje, paket, sınıf ve metot bazındadır. Bu araç Şekil 5.5'te gösterildiği gibi kendine has bir XML formatında da çıktı olarak alınabilir. Yine buradaki değerlerin lojistik regresyon analizinde kullanılmak üzere tablo formatına getirilmesi için belirli işlemlere tabi tutulması gerekmektedir.

Şekil 5.5 Metrics2 çıktı örneği

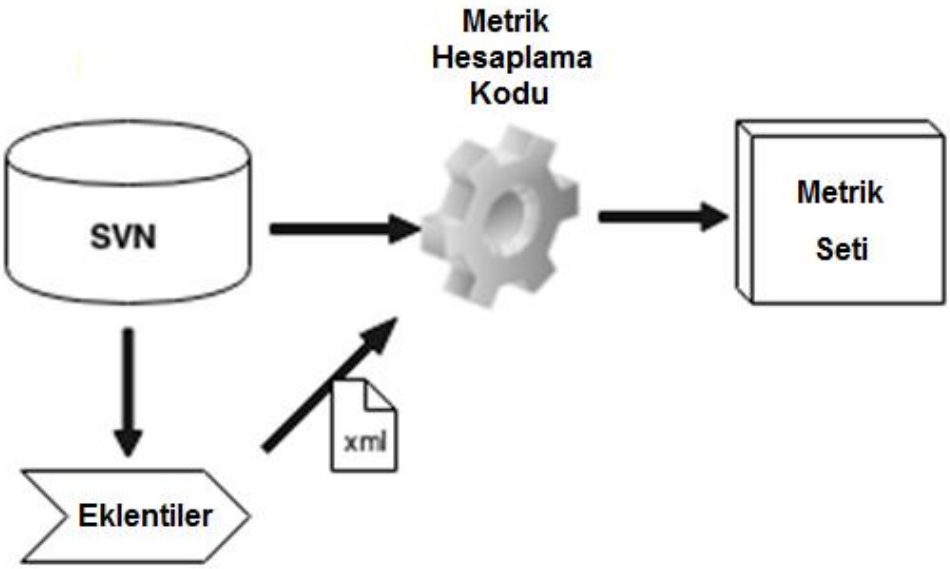
5.4 Kod Metriklerinin Hesaplanması

Google Code Pro Analytix ve *Metrics2* aracından elde edilen XML formatındaki sonuçların lojistik regresyon analizine sokulmak üzere uygun formata getirilebilmesi için bir metrik hesaplama kodu geliştirilmiştir. Bu kod temel olarak, elde edilen XML formatındaki dosya içeriklerini tek tek işler ve anlamlandırır. Kod Google Code Pro Analytix İşleyicisi, Metrics2 İşleyicisi, SVN İşleyicisi olmak üzere 3 parçadan oluşmaktadır. Uygulama algoritması Şekil 5.5'de verilmiştir:

```
Google Code Pro Analytix Dosyasını İşle
Metrics2 Dosyasını İşle
Sonuçları Birleştir
Döngü: Her bir sonuç için
  SVN sunucusuna bağlan ve SVN metriklerini bul
Sonuçları Formatla ve Dön
```

Şekil 5.6 Kod metrik hesaplayıcı uygulaması algoritması

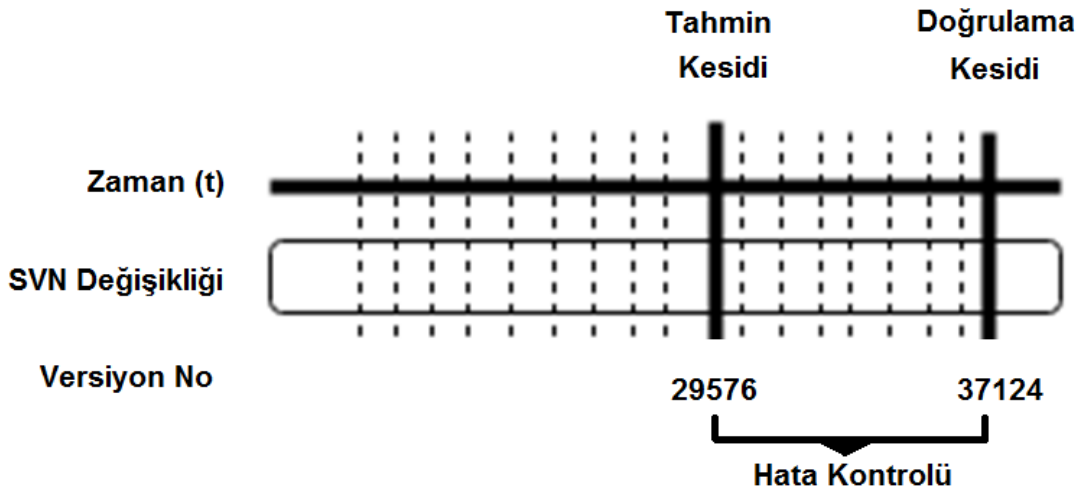
Şekil 5.7'de uygulamanın genel yapısı ve yapılan işlem adımları gösterilmiştir. Uygulama sonuç olarak, istenen şekle uygun bir metrik setini CVS formatında çıktı olarak sunar.



Şekil 5.7 Kod metriklerinin hesaplanması

5.5 Modelin ATM İzleme ve Yönetim Yazılımı Koduna Uygulanması

Kaynak veri setinden lojistik regresyon analizi modelinin oluşturulmasından ve *Provus Bilişim Hizmetleri* bünyesinde geliştirilmekte olan ATM İzleme ve Yönetim Sistemi yazılımına ait kaynak kodların belirlenen tahmin kesitinden metriklerin modele uygun bir şekilde elde edilmesinden sonra, her bir sınıfta hata bulunma oranı lojistik regresyon modeli sayesinde tahmin edilmiştir.

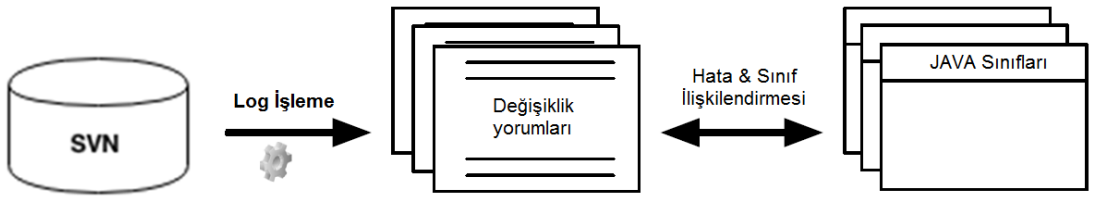


Şekil 5.8 Koddaki değişikliklerden hataların incelenmesi

Şekil 5.8'de yazılımın SVN'deki versiyonları dikey çizgilerle gösterilmiştir. Kullanılan tahmin kesidi ve doğrulama kesidi ise şekilde daha belirgin halde gösterilmiştir. Tahmin kesidi ile doğrulama kesidi arasında yapılan bir değişiklik nedeniyle hata oluşup oluşmadığı incelenir.

5.6 Koddaki Hataların Sınıflarla İlişkilendirilmesi

Bundan sonraki aşamada, hata kestirimlerinin başarımını ölçmek amacıyla, tahmin kesiti ile kodun doğrulama kesiti arasındaki değişikliklerde her hangi bir hata tespit edilip edilmediği kontrol edilmiştir. Koddaki hataların sınıflarla ilişkilendirilmesinde, Zimmermann ve diğerleri [24], Fischer ve diğerleri [48] ve Bird ve diğerlerinin [49] yaptığı çalışmalardan yola çıkarak benzer bir yaklaşım geliştirilmiştir. Yapılan kod değişimlerine yapılan yorumlardaki anahtar kelimeler incelenerek htaların sınıflarla ilişkilendirilmesi sağlanmıştır.



Şekil 5.9 Koddaki hataların sınıflarla ilişkilendirilmesi

Bir Java sınıfında bir kusur bulunup bulunmadığının tespit edilmesi için, söz konusu Java sınıfının ait olduğu dosyada yapılan SVN değişiklik (*commit*) bilgilerine ait geliştirici yorumları, geliştirilen uygulama (Bkz. Bölüm 5.4) sayesinde analiz edilmiştir. Yapılan değişikliklerdeki yorumlara bakılmış ve bu yorumların "hata", "düzeltme", "sorun", "bug", "fix", "defect" gibi çok çeşitli Türkçe ve İngilizce anahtar kelime setiyle uyuşup uyuşmadığına bakılmıştır.

Buna ek olarak, dosyadaki değişikliklerde adı geçen farklı geliştirici sayısı gibi farklı metrikler de oluşturulan veri tabanına eklenmiştir.

Bu çalışmalar sonucunda, hata kestirimi yapılırken faydalanılacak önceki hata değeri ve kestirimin doğrulanmasında kullanılacak sonraki hata gibi metriklerin yanı sıra çeşitli

metriklerin her bir sınıf için elde edilmesi işlemi tamamlanmıştır. Bu veri setinden örnek bir kesit Çizelge 5.1'de görülmektedir.

Çizelge 5.1 "ATM İzleme ve Yönetim Sistemi" metrik veri tabanından örnek veri

Sınıf	DIT	WMC	LCOM	NOA	NOM	NBD	PAR	ÖNCEKİ HATA	SONRAKİ HATA	…
PAYSFDSHelper.java	1	17	0	1	5	1,5	0,5	1	0	…
PAYSFDSScheduler.java	1	6	0	2	4	1	0	0	0	…
FDSMonitor.java	0	5	0	0	5	1,33	2,33	0	0	…
FDSJobExecutor.java	0	1	0	0	1	1,33	2	0	0	…
ObjectDefinition.java	1	19	0.571	3	7	1,33	2	0	1	…
RunnerComparator.java	1	9	0	0	1	1,043	0,65	0	0	…
…	…	…	…	…	…	…	…	…	…	…

Bölüm 5.3'te anlatılan "Kod Metriklerinin Toplanması" ve Bölüm 5.4'te anlatılan "Kod Metriklerinin Hesaplanması" adımlarının neticesinde ATM İzleme ve Yönetim Sistemi Yazılımı için bir hata veri tabanı ortaya çıkmıştır. Bu veri tabanı da aynen Bölüm 4.1'de anlatılan kaynak veri seti gibi hata kestirim modeli oluşturmada kullanılabilir.

BÖLÜM 6

GELİŞTİRİLEN MODEL ÜZERİNDEKİ DENEY ÇALIŞMALARI ve DOĞRULAMA

Yapılan çalışmalar kapsamında birçok deneyler yapılmış ve sonuçlarda iyileştirme elde etmek amacıyla sürekli farklı yöntemler denenmiştir. Bu bölümde yapılan bu deneyler sırasıyla verilmiştir.

6.1 Kaynak Veri Setinden Uygun Model Oluşturma Deneyleri

Bölüm 4.1'te açıklanan D'ambros kaynak veri seti, lojistik regresyon analizi deneylerine tabi tutularak çeşitli modeller oluşturulmuştur. Lojistik regresyon analizi için R dili ve *R-Studio* aracından faydalanılmıştır [41]. Öncelikle gerekli kütüphaneler yüklenmiş ve kaynak veri seti Şekil 6.1'de görüldüğü üzere, dosyadan, R-Studio sistemine aktarılmıştır.

```
> library(aod);
> library(ggplot2);
> kaynak_veri_seti <- read.csv( "C:/secilen_veri_seti.csv" );
```

Şekil 6.1 Kaynak veri setinin R-Studio ortamına aktarılması

Her bir deney sonucunda elde edilen değişkenler, katsayıları, standart hata değerleri, wald z istatistik değerleri, wald z olasılık değeri ve değişkenin anlamlılığı tablo halinde sunulmuştur. Verilen tablolardaki değerler Bölüm 3.3'de ve 4.2'de detaylı olarak açıklanmıştır.

6.1.1 Model Oluşturma Deneyi 1

Yapılan ilk deneme çalışmalarında kaynak veri setinden yalnızca belli başlı metrik değerleri ele alınmış ve model bu metrikler üzerinden kurulmuştur. Deneyin ikinci

aşamasında, ATM izleme ve Yönetim Sistemi kodları inceleneceği ve model bu proje kodları üzerine uygulanacağı için, projeden basitçe elde edilebilecek temel bazı metrikler seçilmiştir.

Çizelge 6.1'de, Çizelge 6.2'de ve Çizelge 6.3'de deney kapsamında elde edilen model katsayıları, anlamlılık değerleri, güven aralıkları ve bahis oranları görülmektedir.

Çizelge 6.1 Deney 1 - Lojistik regresyon modeli hesaplama sonuçları ve katsayıları

	Katsayı	**Std. Hata**	**Z değeri**	**Pr(>|z|)**	**Anlamlılık**
(Sabit Terimi)	-2,251056	0,058578	-38,428	< 2e-16	***
FAN-IN	0,007283	0,003001	2,426	0,01525	*
WMC	0,006744	0,001177	5,728	1,02e-08	***
NOA	0,025149	0,004061	6,193	5,91e-10	***
NOM	0,015507	0,005223	2,969	0,00299	**
Anlamlılık (*Significance*) Kodları: 0 '***' 0,001 '**' 0,01 '*' 0,05 '.' 0,1 ' ' 1					
Hata ~ FAN-IN + WMC + NOA + NOM					

Çizelge 6.2 Deney 1 – Lojistik model güven aralıkları

	% 2,5	**% 97,5**
(Sabit Terimi)	-2,365867297	-2,136243951
FAN-IN	0,001400130	0,013165179
WMC	0,004435944	0,009051245
NOA	0,017189900	0,033108933
NOM	0,005270617	0,025742899

Çizelge 6.3 Deney 1 – Lojistik model bahis oranı değerleri

(Sabit Terimi)	**FAN-IN**	**WMC**	**NOA**	**NOM**
0,105288	1,007309	1,006766	1,025468	1,015628

Sonuçta elde edilen lojistik model denklemi aşağıda verilmiştir. Burada logit, logaritma fonksiyonunu; p ise olasılık değerini ifade eder.

$$y = logit(p) = -2{,}251056 + 0{,}007283x\boldsymbol{FAN\text{-}IN} + 0{,}006744x\boldsymbol{WMC} + 0{,}025149x\boldsymbol{NOA} + 0{,}015507x\boldsymbol{NOM} \quad (6.1)$$

Çizelge 6.1'de verilen katsayı tahmin değerleri için R dilindeki confint fonksiyonu ile hesaplanmış, %95 güven aralığı değerleri Çizelge 6.2'de verilmiştir.

Katsayıların eksponansiyeli alınarak, bahis oranı (odds ratio, OR) değerleri de Çizelge 6.3'de verilmiştir. Buradaki sonuçlara bakarak, diğer değişkenler sabit tutulmak üzere değişkenlerdeki değişimin ihtimaller üzerindeki etkisi hakkında aşağıdaki yorumlar yapılabilir:

- FAN-IN değerindeki 1 birim artış, hata ihtimalini %0,73 arttırır.
- WMC değerindeki 1 birim artış, hata ihtimalini %0,67 arttırır.
- NOA değerindeki 1 birim artış, hata ihtimalini %2,55 arttırır.
- NOM değerindeki 1 birim artış, hata ihtimalini %1,56 arttırır.

Lojistik regresyon denkleminin elde edilmesinden sonra, test için ayrılan verinin modele göre nasıl sonuç verdiği kıyaslanmıştır. Test sonuçları Çizelge 6.4'de verilmiştir.

Çizelge 6.4 Deney 1 – Lojistik model test sonuçları

Test Veri Sayısı	573
Başarılı	486
Başarısız	87
Başarı Oranı	% 84,817

Deney 1 sonucunda elde edilen modelin %84 oranında başarılı olduğu görülmüştür.

6.1.2 Model Oluşturma Deneyi 2

ATM izleme ve Yönetim Sistemi kodlarının yapılandırma yönetim sistemi ile ilgili bazı metrik değerlerinin elde edilmesi sayesinde yeni bir lojistik regresyon modeli

oluşturulmuştur. İkinci deneyde, süreç metriklerinden koddaki yenileme (revizyon) sayısı (NR) metriğinin de hesaba katılmasıyla yeni bir çalışma yapılmıştır.

Çizelge 6.5 Deney 2 – Lojistik regresyon modeli hesaplama sonuçları ve katsayıları

	Katsayı Tahmin	**Std. Hata**	**Z değeri**	**Pr(>|z|)**	**Anlamlılık**
(Sabit Terimi)	-2,0960879	0,0842681	-24,874	< 2e-16	***
RFC	0,0072180	0,0011151	6,473	9.62e-11	***
CBO	0,0104307	0,0031917	3,268	0.00108	**
DIT	-0,1706357	0,0374747	-4,553	5.28e-06	***
NR	0,0171356	0,0018617	9,204	< 2e-16	***
LOC	-0,0008836	0,0002979	-2,966	0.00302	**
Anlamlılık (*Significance*) Kodları: 0 ‘***’ 0,001 ‘**’ 0,01 ‘*’ 0,05 ‘.’ 0,1 ‘ ’ 1					
Hata ~ RFC + CBO + DIT + NR + LOC					

Sonuçta elde edilen lojistik model denklemi aşağıda verilmiştir:

$$y = logit(p) = -2{,}0960879 + 0{,}0072180 \times \mathbf{RFC} + 0{,}0104307 \times \mathbf{CBO} - 0{,}1706357 \times \mathbf{DIT} + 0{,}0171356 \times \mathbf{NR} - 0{,}0008836 \times \mathbf{LOC} \quad (6.2)$$

Lojistik regresyon denkleminin elde edilmesinden sonra, test için ayrılan verinin modele göre nasıl sonuç verdiği kıyaslanmıştır.

Çizelge 6.6 Deney 2 – Lojistik model test sonuçları

Test Veri Sayısı	573
Başarılı	490
Başarısız	83
Başarı Oranı	% 85,515

Deney 2 sonucunda elde edilen modelin %86 oranında başarılı olduğu görülmüştür.

6.1.3 Model Oluşturma Deneyi 3

ATM İzleme ve Yönetim Sistemi kodlarından Bölüm 5.6'da açıklandığı gibi sınıf-hata ilişkisini de kurulmasıyla, buna uygun yeni bir model oluşturma ihtiyacı duyulmuştur.

Çizelge 6.7 Deney 3 – Lojistik regresyon modeli hesaplama sonuçları ve katsayıları

	Katsayı Tahmin	**Std. Hata**	**Z değeri**	**Pr(>|z|)**	**Anlamlılık**
(Sabit Terimi)	-3.4680919	0.2484656	-13.9580	< 2e-16	***
WMC	0.0062890	0.0011536	5.451000	5.00e-08	***
DIT	-0.0702946	0.0347372	-2.02400	0.04301	*
LCOM	-0.0001523	0.0000478	-3.18600	0.00144	**
NOM	0.0248885	0.0056447	4.409000	1.04e-05	***
NOA	0.0225958	0.0039736	5.686000	1.30e-08	***
PREVBUGS	1.4098361	0.2428095	5.806000	6.39e-09	***
Anlamlılık (*Significance*) Kodları: 0 '***' 0,001 '**' 0,01 '*' 0,05 '.' 0,1 ' ' 1					
Hata ~ WMC+DIT+LCOM+NOM+NOA+PREVBUGS					

Sonuçta elde edilen lojistik model denklemi aşağıda verilmiştir:

$$y = logit(p) = -3{,}4680919 + 0{,}0062890x\mathbf{WMC} - 0{,}0702946x\mathbf{DIT} - 0{,}0001523x\mathbf{LCOM} + 0{,}0248885x\mathbf{NOM} + 0{,}0225958x\mathbf{NOA} + 1{,}4098361x\mathbf{PREVBUGS} \quad (6.3)$$

Lojistik regresyon denkleminin elde edilmesinden sonra, test için ayrılan verinin modele göre nasıl sonuç verdiği kıyaslanmıştır.

Çizelge 6.8 Deney 3 – Lojistik model test sonuçları

Test Veri Sayısı	573
Başarılı	485
Başarısız	88
Başarı Oranı	% 84,642

Çizelge 6.8'de gösterildiği gibi Deney 3 sonucunda elde edilen modelin %85 oranında başarılı olduğu görülmüştür.

6.2 Modelin Uygulama ile Sınanması Üzerine Deneyler

Çalışmamızın bu adımında, oluşturulan lojistik regresyon modellerinin ATM İzleme ve Yönetim Sistemi kodlarında denenmesi anlatılmaktadır. Bölüm 4.3'de açıklandığı üzere, söz konusu sistemden metriklerin elde edilmesi sağlanmış ve bir hata veri tabanı oluşturulmuştur. Buradaki metrik değerlerinin, lojistik regresyon modelinde yerine konulmasıyla bir hata kestirimi gerçekleştirilmiştir. Yapılan hata kestiriminin sınanması için de, oluşturulan veri tabanındaki tahmin kesiti ile doğrulama kesiti arasında hata bulunup bulunmadığını gösteren *sonraki hata değeri* (NEXTBUGS) metriği kullanılmıştır.

6.2.1 Uygulama Deneyi 1

Yapılan ilk deney basit düzeyde olup, modelin gerçek sınıflar üzerindeki etkinliğini göz kararı görebilme amacıyla yapılmıştır. Çizelge 6.9'da özetlenen bu deneyde, sınırlı sayıda sınıfla çalışılmıştır. Bunun için yalnızca projede aktif olarak kullanılan bazı temel sınıflar seçilmiştir.

Çizelge 6.9 Uygulama deneyi 1 – Çalışma özet bilgileri

Çalışma Tarihi	05/2014
Kullanılan Model	y = logit(p) = -2,251056 + 0,007283x**FAN-IN** + 0,006744x**WMC** + 0,025149x**NOA** + 0,015507x**NOM**
İncelenen Sınıf Sayısı	8
Başarı Oranı	%87.5

İncelenen sınıflar ve projede gerçekleştirdikleri işlevler aşağıda verilmiştir:

FileUtil: Projenin ortak kütüphanesinde bulunan bir sınıftır. Genel dosya okuma ve yazma işlemleri bu sınıftan yönetilir.

MailMessageHelper: Tüm yazılım parçaları tarafından e-posta göndermek için ortak olarak kullanılır.

ProvusScheduledThreadPool: Paralel gerçekleştirilecek işlemlerin yönetilmesinde kullanılan, yoğun iş yüklerini yöneten temel sınıflardır.

ModuleInitializer: Modüllerin başlangıç işlemlerini yöneten ortak sınıflardır. Projenin ortak kütüphanesinde bulunur.

ConfigUtil: Projelerin kullandığı ayar dosyalarının okunması ve değerlerinin ortak bir hafıza alanında tutulmasını yöneten sınıftır.

EjournalParser: ATM'ler üzerinden alınan jurnal denilen kayıt dosyalarının işlenmesini yöneten ana sınıftır.

DieboldEJournalDownloadAtmRunner: Diebold marka ATM'lerdeki jurnal denilen kayıt dosyalarının ATM'den alınması işlemlerinin yönetildiği sınıftır.

PAYSFTPClient: ATM'ye FTP ile bağlanma gibi temel işlemleri yürüten sınıftır.

Elde edilen örnek veri seti ve modele göre hesaplanan hata ihtimali oranı Çizelge 6.10'da görülmektedir.

Çizelge 6.10 Uygulama deneyi 1 – Sonuçlar

Dosya/Sınıf İsmi	**FAN-IN**	**WMC**	**NOA**	**NOM**	**Hata İhtimali**	**Sonraki Hata**
FileUtil	95	143	2	44	%53.44	Evet
MailMessageHelper	2	22	0	3	%11.49	Hayır
ProvusScheduledThreadPool	2	20	5	9	%13.75	Hayır
ModuleInitializer	11	7	3	3	%11.90	Hayır
ConfigUtil	4	8	3	1	%11.14	Hayır
EjournalParser	1	185	24	37	%54.52	Evet
DieboldEJournalDownloadAtmRunner	1	61	12	15	%21.45	Hayır
PAYSFTPClient	17	52	12	26	%25.51	Evet

İncelenen her bir dosya için, verilen modele uygun metrik değerleri çıkartılmış ve modelin sonucunda tespit edilen hata ihtimali oranı ve gerçekte dosyada hata bulunup

bulunmadığı bilgisi verilmiştir. Çizelge 6.10'da verilen dosya/sınıflar için hata ihtimalleri incelendiğinde `FileUtil` ve `EjournalParser` sınıflarında diğerlerinden çok yüksek olarak %50 üzeri bir hata oranı tespit edilmiştir. Bu sınıflarda gerçekten de raporlanan hatalar mevcuttur. Bu iki sınıf dışında `PAYSFTPClient` sınıfında da raporlanmış bir hata olmasına rağmen, model tarafından hata ihtimali %25 gibi daha düşük bir değer olarak gösterilmiştir. Diğer sınıflar için modelin tespit ettiği düşük hata oranları, gerçek sonuçlarla uyumluluk göstermektedir.

Sonuç olarak, başlangıç aşamasında olan bu deney yetersiz olsa da sonraki çalışmalara yol gösterici niteliği ve modelin etkinliğini ticari bir uygulama üzerinde görmek açısından önemli olmuştur. Yapılan bu çalışma UYMS 2014 konferansına gönderilen bir bildiriyle paylaşılmıştır [40].

6.2.2 Uygulama Deneyi 2

Geliştirilen uygulama ile kodlardaki metriklerin otomatik olarak analiz edilmesiyle beraber daha fazla sınıf üzerinden çalışma yapabilme imkânı doğmuştur. Bu aşamada ATM İzleme ve Yönetim Sisteminin Dosya Transferi yazılım parçasını oluşturan PAYSFDSProcessor projesinin kodları üzerinde çalışma yapılmıştır (Çizelge 6.11).

Çizelge 6.11 Uygulama deneyi 2 – Çalışma özet bilgileri

Çalışma Tarihi	10/2014
Kullanılan Model	y = logit(p) = -3,4680919 + 0,0062890x**WMC** – 0,0702946x**DIT** – 0,0001523x**LCOM** + 0,0248885x**NOM** + 0,0225958x**NOA** + 1,4098361x**PREVBUGS**
İncelenen Sınıf Sayısı	48
Başarı Oranı	%66.6

Elde edilen 48 sınıfa ait örnek veri seti değerleri ve bu sınıfların modele göre hesaplanan hata ihtimali oranları Çizelge 6.12'de görülmektedir. Modelde kullanılan Önceki Hata (PREVBUGS), model sonucunda hesaplanan hata ihtimali ve sonuçların başarısını ölçmede kullanılan sonraki hata (NEXTBUGS) değerleri ait oldukları sınıflarla birlikte tabloda verilmiştir.

Çizelge 6.12 Uygulama deneyi 2 – Sonuçlar

Dosya/Sınıf İsmi	DIT	WMC	LCOM	NOA	NOM	Önceki Hata	Hata İht.	Sonraki Hata
PAYSFDSHelper.java	1	17	0	1	5	Evet	11,69%	Hayır
PAYSFDSScheduler.java	1	6	0	2	4	Hayır	2,93%	Hayır
FDSMonitor.java	0	5	0	0	5	Hayır	3,52%	Hayır
FDSJobExecutor.java	0	1	0	0	1	Hayır	3,12%	Hayır
ObjectDefinition.java	1	19	0,571	3	7	Hayır	4,10%	Evet
RunnerComparator.java	1	9	0	0	1	Hayır	3,06%	Hayır
PAYSSocketFactory.java	2	5	0,625	2	5	Hayır	3,21%	Hayır
FTPEvent.java	0	39	0	1	39	Hayır	9,52%	Hayır
PAYSFTPClient.java	6	61	0,875	10	25	Evet	22,34%	Evet
PAYSFDSStatisticsMBean.java	0	6	0	0	6	Hayır	3,62%	Hayır
PAYSFDSStatistics.java	2	9	0	1	7	Hayır	3,38%	Evet
PAYSFDSRequestHandler.java	1	10	0	2	5	Evet	13,08%	Evet
PAYSFDSJobExecutor.java	1	15	0	3	4	Evet	12,60%	Hayır
FDSConnectionMonitor.java	1	21	0	3	9	Evet	15,12%	Evet
FTPStrategy.java	2	7	0,667	3	4	Hayır	3,24%	Hayır
QuartzInitModule.java	2	5	0	1	3	Hayır	2,92%	Hayır
SPInitializationModule.java	2	3	0	1	2	Evet	10,62%	Hayır
ModuleManager.java	1	1	0	0	1	Hayır	2,91%	Hayır
JMXInitModule.java	2	4	0	2	3	Hayır	2,77%	Hayır
DefaultFDSService.java	1	65	0	2	22	Evet	23,20%	Hayır
FDSService.java	0	16	0	0	16	Evet	17,37%	Hayır
Main.java	1	14	0,5	3	6	Evet	12,78%	Evet
FDSProcessor.java	1	47	0	11	18	Evet	17,34%	Evet
ConfigurationInitializationException.java	4	4	0	1	0	Hayır	2,60%	Hayır

Çizelge 6.12 Uygulama deneyi 2 – Sonuçlar (devamı)

FTPEventException.java	3	4	0	1	0	Hayır	2,78%	Hayır
FDSJobExecutionException.java	3	4	0	1	0	Hayır	2,78%	Hayır
PAYSFDSUpdatePatchJob.java	2	7	0	2	4	Evet	11,58%	Hayır
PAYSFDSScriptSchedulerJob.java	2	4	0	1	2	Hayır	2,84%	Hayır
PAYSFDSGetCountersJob.java	2	14	0	2	4	Hayır	3,23%	Hayır
PAYSFDSJob.java	2	13	0,333	3	6	Hayır	3,45%	Hayır
FDSScriptContext.java	1	206	0,812	6	54	Evet	66,23%	Evet
FDSProtocolCommandListener.java	1	3	0	2	2	Hayır	3,23%	Hayır
ConnectionMonitorLog.java	1	2	0	2	1	Hayır	2,86%	Hayır
CounterCommandLog.java	1	2	0	2	1	Evet	10,76%	Evet
FailedCounterCommandLog.java	1	2	0	2	1	Hayır	2,86%	Evet
CommunicationLog.java	1	4	0	2	1	Hayır	2,89%	Evet
CommandAndReplyLog.java	1	6	0	3	1	Hayır	2,93%	Evet
FailedCommandLog.java	1	2	0	2	1	Evet	10,76%	Hayır
FDSConfigFactory.java	1	4	0	1	2	Hayır	2,89%	Evet
InstitutionConfiguration.java	1	13	0,909	6	12	Hayır	4,75%	Hayır
FTPConnectionConfiguration.java	1	15	0,923	7	14	Hayır	5,15%	Hayır
FDSProcessorConfiguration.java	1	36	0,945	11	25	Evet	26,32%	Evet
PortRange.java	1	6	0,5	2	4	Hayır	3,54%	Evet
FDSScriptRunner.java	3	54	0,882	15	31	Evet	31,17%	Evet
FDSUpdatePatchRunner.java	3	32	0,786	8	16	Evet	18,82%	Hayır
FDSGetCounterRunner.java	3	22	0,7	12	7	Evet	15,70%	Evet
FDSRestartRunner.java	3	20	0,708	9	8	Evet	14,95%	Hayır
FDSExternalScriptRunner.java	3	23	0,792	12	8	Evet	16,08%	Hayır

2. Uygulama Deneyi sonucunda elde edilen modelin %66,6 oranında başarılı olduğu görülmüştür. Genel olarak deneyin ve deney sonucunda oluşturulan modelin başarısı değerlendirmek gerekirse, hem değerlendirilen veri sayısının küçük olması hem de sonuçların yeterli başarıyı yakalayamaması nedeniyle başarının istenilen düzeyde olmadığı söylenebilir.

6.2.3 Uygulama Deneyi 3

Çizelge 6.13'de verilen çalışmada bir önceki uygulama deneyiyle aynı model kullanılmakla beraber, bu defa daha fazla veri için çalışma gerçekleştirilmiştir.

Çizelge 6.13 Uygulama deneyi 3 – Çalışma özet bilgileri

Çalışma Tarihi	02/2015
Kullanılan Model	y = logit(p) = -3,4680919 + 0,0062890x**WMC** – 0,0702946x**DIT** – 0,0001523x**LCOM** + 0,0248885x**NOM** + 0,0225958x**NOA** + 1,4098361x**PREVBUGS**
İncelenen Sınıf Sayısı	713
Başarı Oranı	%77,56

ATM İzleme ve Yönetim Sistemi yazılımının analiz edilmesiyle 713 sınıf için çıkartılan metrik değerlerinin, modelde yerine konulmasıyla bu sınıflarda hata tahminleri yapılmıştır.

Çizelge 6.14'de uygulamada gerçekleştirilen deneyin sonuçları, istatistiksel bilgileriyle birlikte verilmiştir. Görüldüğü üzere deney sonuçları %78'e yakın bir doğruluk oranı, %99 oranında bir seçicilik ve %29'a yakın bir kesinlik göstermektedir. Bununla beraber duyarlılık %1 gibi çok küçük bir oranda çıkmıştır. F-ölçütü ise %3'e yakın bir orandadır. Bu sonuçlar ışığında, modelin başarım oranının istenen seviyede olduğu söylenemz. Bu evrensel bir model oluşturmanın zorluğuna işaret etmekte yani D'ambros kaynak veri setinden elde edilen modelin tamamen farklı bir yazılıma uygulanması istenen başarı oranını göstermemiştir.

Çizelge 6.14 Uygulama deneyi 3 – İstatistiksel bilgiler

Doğru Pozitif Örnek Sayısı	2
Yanlış Pozitif Örnek Sayısı	5
Yanlış Negatif Örnek Sayısı	155
Doğru Negatif Örnek Sayısı	551
Doğruluk	%77,56
Hata Oranı	%22,44
Seçicilik	%99,1
Kesinlik	%28,6
Duyarlılık	%1,274
F-ölçütü	%2,44

6.3 Yazılım Verilerinin Kendi İçinde Değerlendirilmesi Üzerine Deneyler

Çalışmamızın bu aşamasında hem model oluşturmada hem de modelin uygulanmasında ATM İzleme ve Yönetim Sistemi kodlarının kullanılması gerçekleştirilmiştir. Çalışmada ayırt edilmeksizin tüm metrikler değerlendirilmiş ve anlamlılığı düşük metrikler modelden çıkarılarak sade ve etkili bir model oluşturma yoluna gidilmiştir.

Bu çalışmada kullanılan yöntem aşağıda özetlenmiştir:

1. Eldeki tüm metrikleri hesaba katan bir model oluştur

2. Modelden anlamlılığı düşük değerleri çıkar ve yeni bir model oluştur

3. Modeldeki tüm değişkenlerin anlamlılığı istenen seviyeye çıkana kadar tekrarla

6.3.1 Model Oluşturma Çalışmaları

Adım 1: Öncelikle elde edilen tüm metrik değerlerini içeren bir model oluşturulmuştur. Bununla metrik değerlerinin oluşan model üzerindeki etkisi ve anlamlılığının görülmesi amaçlanmıştır. Bazı metriklerin sonucu modelde uygulanamaz (N/A) olarak çıkmıştır. Tabloda anlamlılığı yeterli değişkenler, anlamlılık sırasına göre “***”, “**”, “*”, “.” Ve “,” ile ifade edilmiştir. Çizelge 6.15’de 1. Adım sonucunda elde edilen model ve katsayıları, anlamlılık değerleriyle birlikte görülmektedir.

Çizelge 6.15 Yazılım verilerinden model oluşturma 1. adımı

	Katsayı Tahmin	Std. Hata	Z değeri	Pr(>|z|)	Anlam-lılık
(Sabit Terimi)	-15,57	1455	-0,011	0,99147	
Soyutluk (Abs)	N/A	N/A	N/A	N/A	
İç Eşleme (Ca)	N/A	N/A	N/A	N/A	
Ort.Blok Derinliği (Abd)	3,887	1,263	3,078	0,00208	**
Ort. Çevrimsel Karmaşıklık (AvgCC)	-0,08082	0,4635	-0,174	0,86157	
Ort. Miras Ağacı Derinliği (AvgDIT)	-0,5043	0,1792	-2,815	0,00488	**
Ort. Metot Kod Satır Sayısı (AvgMLOC)	0,09183	0,195	0,471	0,63767	
Tip Başına Ort. Yapılandırıcı (AvgNoCPt)	15,89	1949	0,008	0,99349	
Tip Başına Ort. Değişken Sayısı (AvgNoFPt)	0,04026	0,17	0,237	0,81273	
Tip Başına Ort. Metod Sayısı (AvgNoMPt)	6,28	514,6	0,012	0,99026	
Ort. Parametre Sayısı (AvgPAR)	0,183	0,3093	0,592	0,55418	
Alt Tip Sayısı (AvgNoSt)	0,01262	0,02943	0,429	0,66813	
Yorum Oranı (CommR)	-3,353	2,196	-1,527	0,12683	
Zorluk (DIFF)	-0,1366	0,08781	-1,556	0,11968	
Uzaklık (DIST)	N/A	N/A	N/A	N/A	
Dış Eşleme (Ce)	N/A	N/A	N/A	N/A	
Efor (EFF)	0,000029	0,0000128	2,184	0,02896	*
Kararsızlık (INS)	N/A	N/A	N/A	N/A	
Kod Satır Sayısı (LOC)	0,04334	0,02282	1,899	0,05751	,
Karakter Sayısı (NoChar)	0,000126	0,0003324	0,379	0,70491	
Yorum Sayısı (NoComm)	0,1189	0,05162	2,303	0,02127	*
Yapılandırıcı Sayısı (NoConstruct)	-15,93	1949	-0,008	0,99348	
Değişken Sayısı (NoA)	0,03535	0,06606	0,535	0,59255	
Satır Sayısı (NoLines)	-0,01957	0,0126	-1,553	0,12042	
Metot Sayısı (NoM)	-6,622	514,6	-0,013	0,98973	

Çizelge 6.15 Yazılım verilerinden model oluşturma 1. adımı (devamı)

	Katsayı Tahmin	Std. Hata	Z değeri	Pr(>|z|)	Anlam-lılık
Terim Sayısı (NoOpd)	0,04443	0,01847	2,405	0,01615	*
Operatör Sayısı (NoOpr)	0,02954	0,02051	1,44	0,14981	
Paket Sayısı	N/A	N/A	N/A	N/A	
Noktalı Virgül Sayısı (NoSC)	-0,06916	0,0347	-1,993	0,04628	*
Tip Sayısı (NoTy)	12,29	1455	0,008	0,99327	
Eşsiz Terim Sayısı (NOUOpd)	0,03913	0,02399	1,631	0,10293	
Eşsiz Operatör Sayısı (NOUOpr)	0,3688	0,1232	2,994	0,00276	**
Program Uzunluğu (LEN)	N/A	N/A	N/A	N/A	
Kelime Haznesi (VOC)	N/A	N/A	N/A	N/A	
Program Hacmi (VOL)	-0,00726	0,002616	-2,775	0,00552	**
Ağırlıklı Metotlar	0,075	0,1341	0,559	0,57586	
Çevrimsel Karmaşıklık (VG)	0,008835	0,4648	0,019	0,98484	
Parametre Sayısı (PAR)	0,2454	0,3198	0,767	0,44284	
İç içe Yuvalanmış Blok Derinliği (NBD)	-3,431	1,331	-2,578	0,00995	**
Miras Ağacı Derinliği (DIT)	N/A	N/A	N/A	N/A	
Ağırlıklı Metod Ortalaması (WMC)	0,006061	0,1334	0,045	0,96376	
Miras Alan Sınıf Sayısı (NSC)	0,141	0,08554	1,649	0,09924	.
Tekrar Tanımlanan Metod Sayısı (NORM)	-0,06816	0,218	-0,313	0,75451	
Yordamlarda Yapışkanlık Eksikliği (LCOM)	-0,00118	0,0006288	-1,882	0,0599	.
Gizli Değişken Sayısı (NoPrA)	-0,08244	0,1665	-0,495	0,62062	
Statik Değişken Sayısı (NoStA)	N/A	N/A	N/A	N/A	
Metot Sayısı (NoM)	0,2768	0,1013	2,732	0,00629	**
Statik Metod Sayısı (NSM)	N/A	N/A	N/A	N/A	
Özelleşme (SIX)	-0,01043	0,01085	-0,961	0,33633	
Metod Kod Satır Sayısı (MLOC)	-0,1524	0,1854	-0,822	0,41126	

Çizelge 6.15 Yazılım verilerinden model oluşturma 1. adımı (devamı)

	Katsayı Tahmin	Std. Hata	Z değeri	Pr(>|z|)	Anlam-lılık
Değişiklik Yapan Kişi Sayısı (NAUTH)	0,1277	0,03133	4,075	0,000046	***
Önceki Hata (PREVBUGS)	-0,757	0,3421	-2,213	0,0269	*

Adım 2: Daha sonra modeldeki metriklerin anlamlılık değerleri göz önünde bulundurularak, anlamlılıkları düşük olan metrik değerleri modelden çıkarılmış ve kalan metrik değerleri ile yeni bir model üretilmiştir.

Çizelge 6.16 Yazılım verilerinden model oluşturma 2. adımı

	Katsayı Tahmin	Std. Hata	Z değeri	Pr(>|z|)	Anlamlılık
(Model Sabiti)	-2,56E+00	3,84E-01	-6,659	2,76E-11	***
Ortalama Blok Derinliği (Abd)	3,59E+00	9,95E-01	3,608	0,000308	***
Ortalama Miras Ağacı Derinliği (AvgDIT)	-2,70E-01	1,28E-01	-2,109	0,034953	*
Efor (EFF)	1,43E-05	6,64E-06	2,16	0,03076	*
Kod Satır Sayısı (LOC)	4,15E-02	1,49E-02	2,788	0,005301	**
Yorum Sayısı (NoComm)	-7,01E-04	2,13E-02	-0,033	0,973744	
Terim Sayısı (NoOpd)	2,55E-02	9,49E-03	2,684	0,007281	**
Noktalı Virgül Sayısı (NoSC)	-4,35E-02	2,42E-02	-1,8	0,071921	.
Eşsiz Operatör Sayısı (NOUOpr)	2,29E-01	4,41E-02	5,183	2,18E-07	***
Program Hacmi (VOL)	-3,84E-03	1,34E-03	-2,87	0,004111	**
İç içe Yuvalanmış Blok Derinliği (NBD)	-3,53E+00	1,08E+00	-3,259	0,001117	**
Miras Alan Sınıf Sayısı (NSC)	1,70E-01	6,80E-02	2,505	0,012262	*
Yordamlarda Yapışkanlık Eksikliği (LCOM)	-1,08E-03	5,22E-04	-2,066	0,038796	*
Metot Sayısı (NoM)	-2,78E-02	2,56E-02	-1,085	0,277742	
Dosyada Değişiklik Yapan Kişi Sayısı (NAUTH)	1,26E-01	2,64E-02	4,767	1,87E-06	***
Önceki Hata (PREVBUGS)	-4,06E-01	3,09E-01	-1,314	0,188679	

Adım 3: Oluşturulan yeni modeldeki anlamlılığı düşük olan değişkenler de çıkarılarak yeniden bir model oluşturulmuştur. Çizelge 6.17'de 3. Adım sonucunda elde edilen modele ait katsayı tahmin değerleri, hata değerleri ve anlamlılık değerleri verilmiştir.

Çizelge 6.17 Yazılım verilerinden model oluşturma 3. adımı

	Katsayı Tahmin	**Std. Hata**	**Z değeri**	**Pr(>|z|)**	**Anlamlılık**
(Model Sabiti)	-2,57E+00	3,829e-01	-6,718	1,84e-11	***
Ortalama Blok Derinliği (Abd)	3,54E+00	9,788e-01	3,615	0,00030	***
Ortalama Miras Ağacı Derinliği (AvgDIT)	-2,68E-01	1,279e-01	-2,097	0,03604	*
Efor (EFF)	1,39E-05	6,322e-06	2,206	0,02742	*
Kod Satır Sayısı (LOC)	3,20E-02	1,203e-02	2,663	0,00774	**
Terim Sayısı (NoOpd)	2,21E-02	8,303e-03	2,656	0,00790	**
Noktalı Virgül Sayısı (NoSC)	-3,96E-02	2,322e-02	-1,703	0,08850	.
Eşsiz Operatör Sayısı (NOUOpr)	2,27E-01	4,352e-02	5,212	1,87e-07	***
Program Hacmi (VOL)	-3,32E-03	1,187e-03	-2,799	0,00512	**
İç içe Yuvalanmış Blok Derinliği (NBD)	-3,37E+00	1,070e+00	-3,146	0,00166	**
Miras Alan Sınıf Sayısı (NSC)	1,73E-01	6,784e-02	2,554	0,01066	*
Yordamlarda Yapışkanlık Eksikliği (LCOM)	-1,22E-03	5,045e-04	-2,424	0,01536	*
Dosyada Değişiklik Yapan Kişi Sayısı (NAUTH)	1,05E-01	2,147e-02	4,908	9,19e-07	***

Adım 4: Adım 3 sonucunda elde edilen modeldeki anlamlılığı düşük olan "Noktalı virgül sayısı" isimli metrik de çıkarılarak yeni bir model oluşturulmuştur.

Adım 4 sonucunda elde edilen modele ait katsayı tahmin değerleri, hata değerleri ve anlamlılık değerleri Çizelge 6.18'de verilmiştir. Bu modelde 11 farklı bağımsız değişken bulunmaktadır. Bu adım sonucunda elde edilen modeldeki tüm bağımsız değişkenlerin anlamlılık değeri, en azından "*" olarak değerlendirildiği için yeterli olarak kabul edilmiş ve adımlar sonuçlandırılmıştır.

Çizelge 6.18 Yazılım verilerinden model oluşturma 4. adımı

	Katsayı Tahmin	Std. Hata	Z değeri	Pr(>|z|)	Anlamlılık
(Model Sabiti)	-2,626e+00	3,807e-01	-6,898	5,29e-12	***
Ortalama Blok Derinliği (Abd)	3,672e+00	9,690e-01	3,790	0,000151	***
Ortalama Miras Ağacı Derinliği (AvgDIT)	-2,574e-01	1,274e-01	-2,020	0,043332	*
Efor (EFF)	1,471e-05	6,098e-06	2,411	0,015890	*
Kod Satır Sayısı (LOC)	2,042e-02	9,268e-03	2,203	0,027597	*
Terim Sayısı (NoOpd)	1,958e-02	7,939e-03	2,466	0,013669	*
Eşsiz Operatör Sayısı (NOUOpr)	2,138e-01	4,202e-02	5,089	3,60e-07	***
Program Hacmi (VOL)	-3,429e-03	1,152e-03	-2,977	0,002910	**
İç içe Yuvalanmış Blok Derinliği (NBD)	-3,490e+00	1,058e+00	-3,300	0,000968	***
Miras Alan Sınıf Sayısı (NSC)	1,512e-01	6,751e-02	2,240	0,025103	*
Yordamlarda Yapışkanlık Eksikliği (LCOM)	-9,870e-04	4,782e-04	-2,064	0,039006	*
Dosyada Değişiklik Yapan Kişi Sayısı (NAUTH)	1,058e-01	2,116e-02	4,999	5,76e-07	***

Adım 4 sonucunda böylece aşağıdaki model elde edilmiştir.

$$y = logit(p) = -2{,}626 + 3{,}672*\mathbf{Abd} - 0{,}2574*\mathbf{AvgDIT} + 0{,}00001471*\mathbf{Eff} + 0{,}02042*\mathbf{LoC} + 0{,}01958*\mathbf{NoOpd} + 0{,}2138*\mathbf{NoUOpr} - 0{,}003429*\mathbf{Vol} - 3{,}490*\mathbf{NBD} + 0{,}1512*\mathbf{NSC} - 0{,}0009870*\mathbf{LCOM} + 0{,}1058*\mathbf{NAuth} \quad (6.4)$$

6.3.2 Modelin Uygulanması

Modelin uygulanmasında veri dosyasının %10 oranındaki kısmı test için ayrılmış ve bu test kümesi üzerinden modelin istatistiksel başarısı ölçülmüştür. Bunun yanı sıra, çapraz doğrulama (*k-fold)* benzeri bir yöntemle veri seti 10 parçaya ayrılmış ve sırayla her parçanın test kümesi olarak kullanılmasıyla modelin başarımının ölçülmesi yoluna gidilmiştir.

Yapılan bu doğrulama çalışmalarında test kümesi kısmı ayrıldıktan sonra kalan veri setinden model oluşturulmuştur. Çizelge 6.19'da her bir test kümesine karşılık gelen model değerleri görülmektedir.

Çizelge 6.19 Yazılım verilerinden model oluşturma deneyi katsayı değerleri

KATSAYI DEĞERLERİ											
	1	2	3	4	5	6	7	8	9	10	ORT
(Intercept)	-2,758000	-2,788000	-2,951000	-2,843000	-2,418000	-2,954000	-2,524000	-2,662000	-2,729000	-2,626000	**-2,725300**
Abd	3,701000	3,840000	3,227000	3,857000	3,665000	3,331000	3,057000	3,914000	3,459000	3,672000	**3,572300**
AvgDIT	-0,244100	-0,159700	-0,186400	-0,119500	-0,223700	-0,125100	-0,233300	-0,159800	-0,149600	-0,257400	**-0,185860**
Eff	0,000021	0,000014	0,000011	0,000012	0,000014	0,000013	0,000015	0,000013	0,000009	0,000015	**0,000014**
LoC	0,027310	0,021510	0,011680	0,020460	0,020960	0,020460	0,024440	0,017710	0,013540	0,020420	**0,019849**
NoOpd	0,025510	0,019090	0,023930	0,012670	0,014690	0,018030	0,015290	0,017570	0,011940	0,019580	**0,017830**
NoUOpr	0,207600	0,200500	0,167400	0,199800	0,191700	0,184600	0,187100	0,195700	0,181600	0,213800	**0,192980**
VOL	-0,004605	-0,003410	-0,003299	-0,002667	-0,002963	-0,003204	-0,003220	-0,003023	-0,002248	-0,003429	**-0,003207**
NBD	-3,596000	-3,782000	-2,981000	-3,836000	-3,596000	-3,132000	-3,054000	-3,894000	-3,250000	-3,490000	**-3,461100**
NSC.	0,051130	0,196300	0,166800	0,144000	0,175000	0,160900	0,136500	0,163300	0,162600	0,151200	**0,150773**
LCOM	-0,001254	-0,001034	-0,000597	-0,000639	-0,000887	-0,000943	-0,000820	-0,001010	-0,000821	-0,000987	**-0,000899**
NAUTH	0,131100	0,116400	0,116100	0,140700	0,108700	0,111800	0,107500	0,111400	0,121900	0,105800	**0,117140**

Çizelge 6.20'de ise her bir parçanın istatistiksel başarısı ve son sütunda da ortalamaları görülmektedir.

Çizelge 6.20 Yazılım verilerinden model oluşturma deneyi uygulama istatistikleri

	1	2	3	4	5	6	7	8	9	10	ORT
Doğruluk	0,873239	0,816901	0,661972	0,84507	0,84507	0,84507	0,84507	0,84507	0,84507	0,84507	**0,826761**
Hata Oranı	0,126761	0,183099	0,338028	0,15493	0,15493	0,15493	0,15493	0,15493	0,15493	0,15493	**0,173239**
Kesinlik	1	0,333333	0,75	0,25	0,25	0,25	0,25	0,25	0,25	0,25	**0,383333**
(-) KESİNLİK	0,869565	0,861538	0,650794	0,966102	0,966102	0,966102	0,966102	0,966102	0,966102	0,966102	**0,914461**
Duyarlılık	0,181818	0,181818	0,214286	0,6	0,6	0,6	0,6	0,6	0,6	0,6	**0,477792**
Seçicilik	1	0,933333	0,953488	0,863636	0,863636	0,863636	0,863636	0,863636	0,863636	0,863636	**0,893228**
Alfa	0	0,066667	0,046512	0,136364	0,136364	0,136364	0,136364	0,136364	0,136364	0,136364	**0,106772**
Beta	0,818182	0,818182	0,785714	0,4	0,4	0,4	0,4	0,4	0,4	0,4	**0,522208**
Q-value	0	0,666667	0,25	0,75	0,75	0,75	0,75	0,75	0,75	0,75	**0,616667**
Hatalı İhmal	0,130435	0,138462	0,349206	0,033898	0,033898	0,033898	0,033898	0,033898	0,033898	0,033898	**0,085539**
F-ölçütü	0,307692	0,235294	0,333333	0,352941	0,352941	0,352941	0,352941	0,352941	0,352941	0,352941	**0,334691**
Anlamlılık	0,008539	0,002405	0,001663	0,003759	0,003759	0,003759	0,003759	0,003759	0,003759	0,003759	**0,003892**

Çizelge 6.20'de verilen sonuçlar değerlendirildiğinde modelin doğruluk oranının ortalamada yaklaşık olarak %83 olduğu görülmektedir. Modelin pozitif kesinliği yani hatalı sınıfların tespit edilmesi oranı %38 dolaylarında kalırken, negatif kesinliği yani

hatasız sınıfların tespit oranının %91 gibi yüksek bir oranda olduğu görülmektedir. Modelin duyarlılık oranı %48 oranında kalmış ama seçiciliği %89 oranındadır.

BÖLÜM 7

SONUÇ ve ÖNERİLER

Çalışmamız genel olarak incelendiğinde içerisinde 3 ayrı çalışma barındırdığı görülmektedir:

1. Kaynak veri seti üzerinde hata kestirim modelinin oluşturulması ve modelin başarısının ölçümü

2. Modelin atm izleme ve yönetim yazılımı koduna uygulanması

3. Provus atm izleme ve yönetim sistemi yazılımı kodlarının analiz edilerek bir hata kestirim veri tabanı oluşturulması

Çalışmanın ilk kısmında, Marco D'Ambros tarafından hazırlanmış kaynak veri seti [2] üzerinde hata kestirim modelinin oluşturulması ve modelin ayrı bir yazılım üzerinde denenerek başarısının ölçümü gerçekleştirilmiştir. Bu şekilde evrensel bir model oluşturma çabasına gidilmiştir. Yapılan bu çalışma neticesinde oluşturulan nihai modelin %78 doğruluk oranı ve yüksek seçicilik ile sonuç verdiği görülse de, modelin duyarlılığı düşüktür.

Çalışmanın ikinci kısmında ise, hazır bir kaynak veri setinden yola çıkmak yerine modeli incelenen yazılımın kendisinden üretilmesi yoluna gidilmiştir. Bu şekilde oluşturulan model yazılma ait karakteristik özelliklere göre oluşturulduğu için daha başarılı bir sonuç üretebileceği düşünülmüştür. Gerçekten de, doğruluk oranı %83 ile kaynak veri setinden üretilen modele yakın bir sonuç verse de kesinlik, duyarlılık gibi kıstaslara göre çok daha iyi sonuç verdiği görülmüştür.

Bir veri seti üzerinden elde edilen hata kestirim modelinin, bir başka veri üzerine uygulanması bir dereceye kadar başarılı sonuç vermektedir ancak veri seti üzerinden

elde edilen modelin, aynı veri seti üzerine uygulanmasında daha başarılı sonuçlar elde edildiği görülmüştür. Bunda benzer özelliklere sahip yazılım kodlarından elde edilmiş modelin, yine benzer özelliklere sahip kodlar üzerine uygulanırsa modelin başarısının maksimize edileceği sonucuna varılmıştır. Bu durum uzun süre bakım aşaması sürdürülecek yazılımlar için avantaj sağlamaktadır.

7.1 Kazanımlar

1. Oluşturulan model sınıflarda hata bulunup bulunmadığının tespiti için önemli bir fayda sağlayacaktır. Yazılım kodundaki bir sınıfta hata bulunup bulunmadığı, kodun özellikleri incelenerek tespit edilebilecektir.

2. Yazılım hatalarının, yazılım geliştirme faaliyetlerinin erken safhalarında tespit edilmesinin daha az masraflı olduğu bilinmektedir. Araştırmalara göre, yazılımın teslimi sonrası bir hatanın bulunması ve düzeltilmesi, hatanın yazılımın gereksinim ve tasarım safhalarında bulunmasına kıyasla 100 kat daha masraflıdır. Çalışmamız, hataların erken tespit edilmesine katkı sağlayacaktır.

3. Hataya eğilimli yazılım parçalarının tespit edilmesiyle test eforu buralarda yoğunlaştırılabilir ve kısıtlı test kaynaklarının hem verimli kullanılması sağlanır hem de eldeki kaynakların hataların bulunma ihtimali yüksek yerlere yönlendirilmesiyle hata tespitinde başarı ihtimali artar. Ayrıca, yazılım hatalarının çoğunluğunun kodun yanlızca belli bölümlerinde bulunmaktadır. Yapılan bir araştırmaya göre, yazılımdaki hataların ortalama %80'i, yazılım parçalarının %20'sinde toplanmakta ve yazılımın %50'lik kısmında ise hiç hata bulunmamaktadır [1]. Asıl mesele, hataların toplandığı bu bölümleri tespit etmektir. Çalışmamız sayesinde kodun hata içerme ihtimali yüksek yerlerin tespiti mümkün olacaktır.

4. Hata kestiriminde kullanımı gibi bir faydanın yanı sıra, modelin verdiği sonuç yüzdelik bir puan olduğundan, yeni bir metrik değeriymişçesine, bu değer kodun kalitesini ölçen bir değer olarak kullanılabilir.

7.2 Gelecek Çalışmalar

Farklı hata kestirimi teknikleriyle kıyaslama: Çalışmamız kapsamında farklı hata kestirim yöntemleriyle bir kıyaslama yöntemine gidilmemiştir.

Daha fazla kod analiziyle veri setinin genişletilmesi: Yapılan çalışmalarda hem modelin oluşturulmasında, hem de test için eldeki veri setlerinin önemi kritiktir. Bunun için daha çeşitli projelerden kodların analiz edilerek bunların metrikleri değerlendirilmelidir. Ayrıca modelden edilen başarımın daha farklı yazılım hata kestirimi yöntemlerinin başarımlarıyla kıyaslanması faydalı olacaktır.

Otomatikleştirme (Otomatizasyon) ve Araçlarla Entegrasyon: Yapılacak çalışmanın neticesinde kodun analizi ve hata kestiriminin otomatik hale getirilmesi için, çalışmada ortaya konacak nihai modelin bir yazılım veya araç haline getirilmesi düşünülebilir. Bir başka ihtimal de mevcut kod kalitesi araçlarıyla bütünleştirmedir. Oluşturulan model SonarQube gibi araçlara veya Eclipse gibi kod geliştirme ortamlarına eklenti olarak tanıtılabilir.

KAYNAKLAR

[1] Boehm, B., Basili, V. (2001, Ocak). "Software Defect Reduction Top 10 List", 34(1): 135-137.

[2] D'Ambros, M., Lanza, M., Robbes, R. (2012). "Evaluating defect prediction approaches: a benchmark and an extensive comparison", 17: 531-577.

[3] Erdemir, U., Tekin, U., Buzluca, F. (2008). "Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi", Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, İstanbul, Türkiye.

[4] Bloch, M., Blumberg, S., Laartz, J. (2012). "Delivering large-scale IT projects on time, on budget, and on value", Londra, İngiltere: McKinsey & University of Oxford.

[5] Sharma, R., Budhija, N., Singh, B. (2012, February). "Study of Predicting Fault Prone Software Modules", International Journal of Advanced Research in Computer Science and Software Engineering, 2(2): 95-97.

[6] Evett, M., Khoshgoftaar, T., Chien, P., Allen, E. (1998). "GPbased software quality prediction", Proceedings of the 3rd Annual Genetic Programming Conference, San Francisco, CA, ABD, 60-65.

[7] Khoshgoftaar, T., Seliya, N. (2002). "Software quality classification modelling using the SPRINT decision tree algorithm", Proceedings of the 4th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, 365-374.

[8] Thwin, M., Quah, T. (2003). "Application of neural networks for software quality prediction using objectoriented metrics", Proceedings of the 19th International Conference on Software Maintenance, Amsterdam, Hollanda, 113-122.

[9] Yuan, X., Khoshgoftaar, T., Allen, E., Ganesan, K. (2000). "An application of fuzzy clustering to software quality prediction", Proceedings of the 3rd IEEE Symp. On Application-Specific Systems and Software Eng. Technology, IEEE Computer Society, Washington, DC, ABD, 85-90.

[10] Catal, C., Diri, B. (2007). "Software fault prediction with object-oriented metrics based artificial immune recognition system", Proceedings of the 8th International Conference on Product Focused Software Process Improvement, Lecture Notes in Computer Science 4589, Riga, Letonya, 300-314.

[11] Alan, O., Catal, C., Sevim, U., Diri, B. (2009). "Sınırlı Sayıda Kusur Verisiyle Yazılım Kusur Kestirim Aracı: YAKUT", 4. ULUSAL YAZILIM MÜHENDİSLİĞİ SEMPOZYUMU - UYMS'09, İstanbul, Türkiye.

[12] Seliya, N., Khoshgoftaar, T. (2004). "Comparative assessment of software quality classification techniques: An empirical study", Empirical Software Engineering, 9(3): 229-257.

[13] Bellini, P., Bruno, I., Nesi, P., Rogai, D. (2005). "Comparing faultproneness estimation models", Proc. of 10th IEEE International Conference on Engineering of Complex Computer Systems, Shanghai, Çin, 205–214.

[14] Menzies, T., Greenwald, J., Frank, A. (2007). "Data mining static code attributes to learn defect predictors", IEEE Trans.on Software Engineering, 1(33): 2-13.

[15] Lanubile, F., Lonigro, A., Visaggio, G. (1995). "Comparing Models for Identifying Fault-Prone Software Components", Proceedings of Seventh International Conference on Software Engineering and Knowledge Engineering, Rockville, Maryland, ABD, 12-19.

[16] Khoshgoftaar, T. M., Allen, E. B., Goel, N., Nandi, A., McMullan, J. (1996). "Detection of software modules with high debug code churn in a very large legacy system", In Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE 1996), White Plains, NY, ABD, 364–371.

[17] Graves, T., Karr, A., Marron, J., Siy., H. (2000). "Predicting fault incidence using software change history", IEEE Transactions on Software Engineering, 2(26): 653-661.

[18] Denaro, G. (2000). "Estimating Software Fault-Proneness for Tuning Testing", Proceedings of the 22nd International Conference on Software Engineering, Limerick, İrlanda.

[19] Deodhar, M. (2002). Prediction Model and the Size Factor for Fault-proneness of Object Oriented Systems, Yüksek Lisans Tezi, Michigan Technological University, Department of Computer Science, Michigan, ABD.

[20] Briand, L., Wust, J., Melo, W. (2002). "Assessing the applicability of fault-proneness models across object-oriented software projects", Software Engineering, IEEE Transactions, 28(7): 706-720.

[21] Moser, R., Pedrycz, W., Succi, G. (2008). "A comparative analysis of the effciency of change metrics and static code attributes for defect prediction", In Proceedings of ICSE 2008, Leipzig, Almanya, 181-190.

[22] Pressman , R., Maxim, B. (2005). "Software Engineering: A Practitioner's Approach", Singapur: Mc Graw Hill.

[23] D'Ambros, M., Lanza, M., Robbes, R. (2010). "An extensive comparison of bug prediction approaches", In MSR '10: Proceedings of the 7th International Working Conference on Mining Software Repositories, Cape Town, 31-41.

[24] Zimmermann, T., Premraj, a. A., Zeller, A. (2007). "Predicting defects for eclipse", In Proceedings of PROMISE 2007, Washington, DC, ABD, 76.

[25] Chidamber, S., Kemerer, C. (1994). "A metrics suite for object oriented design", Software Engineering, IEEE Transaction, 20(6): 476-493.

[26] Gyimothy, T., Ferenc, R., Siket, I. (2005). "Empirical validation of object oriented metrics on open source software for fault prediction", IEEE Trans. Software Eng., 31(10): 897-910.

[27] Arapidis, C. (2012). "The Response for Class metric. Sonar Code Quality Testing Essentials", Packt Publishing.

[28] Software metrics (2), http://www.win.tue.nl/~aserebre/2IS55/2010-2011/10.pdf, 5 Mayıs 2015.

[29] Martin, R. C. (1995). "Object Oriented Design Quality Metrics: An Analysis of dependencies", ROAD, 2(3).

[30] Yücetürk, A. C., Bektaş, M. A., Ekşioğlu, K. M. (2008). "Mobil Uygulama Yazılımlarında Yazılım Metriklerinin Kullanılması", YKGS2008, İstanbul, Türkiye.

[31] McCabe, T. (1976). "A Complexity Measure", IEEE Transactions on Software Engineering(2), 308-320.

[32] Hassan, A. E. (2009). "Predicting faults using the complexity of code changes", In Proceedings of ICSE 2009, Vancouver, BC, ABD, 78-88.

[33] Munson, J. C., Nikora, A. P. (2003). "Developing fault predictors for evolving software systems", In Proceedings of the 9th International Symposium on Software Metrics, IEEE Computer Society, 338-349.

[34] Larose, D. (2006). "Logistic Regression, in Data Mining Methods and Models", John Wiley & Sons, Inc., Hoboken, NJ, ABD.

[35] Türk İstatistik Web Sitesi, http://turkistatistik.net/upload/dosya/reganaliz.pdf, 9 Mayıs 2014.

[36] Sümbüloglu, K., Sümbüloğlu, V. (2007). "Biyoistatistik (16 b.)", Hatiboğlu Yayınevi, Ankara, Türkiye.

[37] Burns, R., (2009). Chapter 24. "Business Research Methods and Statistics Using SPSS", Sage Publications Ltd, Londra, İngiltere.

[38] Dokuz Eylül Üniversitesi İstatistik-1 Dersi Notları, http://kisi.deu.edu.tr//cenk.ozler/ist%201%20B%C3%B6l%201%20Tan%C4%B1mlar.pdf, 13 Nisan 2015.

[39] Coskun, C., Baykal, A. (2011). "Veri Madenciliğinde Sınıflandırma Algoritmalarının Bir Örnek Üzerinde Karşılaştırılması", Akademik Bilişim 2011, Malatya, Türkiye.

[40] Sari, Ö., Kalipsiz, O. (2014). "Yazılım Hata Kestirimi İçin Veri Analizi Yöntemlerinin Kullanılması", Proceedings of the 8th Turkish National Software Engineering Symposium, Güzelyurt, KKTC, Turkiye.

[41] R-Studio, Ürün Web Sitesi, https://www.rstudio.com/, 9 Mayıs 2014.

[42] Provus Bilişim Hizmetleri A.Ş., ATM Operasyonları, http://www.provus.com.tr/hizmetler/atm-operasyon.html, 9 Mayıs 2014.

[43] Subversion (SVN) Firması, Konfigürasyon Yönetim Sistemi Ürünü, https://subversion.apache.org/, 9 Mayıs 2014.

[44] Eclipse Projesi, Ürün Web Sitesi, http://www.eclipse.org/, 9 Mayıs 2014.

[45] Eclipse Marketplace Web Sitesi, CodePro AnalytiX Eklentisi,http://marketplace.eclipse.org/content/codepro-analytix, 9 Mayıs 2014.

[46] Google Code Pro Analytix Yazılım Kalitesi Aracı Web Sitesi, https://developers.google.com/java-dev-tools/codepro/, 9 Mayıs 2014.

[47] Metrics2 Eclipse Eklentisi Web Sitesi, http://metrics2.sourceforge.net, 9 Mayıs 2014.

[48] Fischer, M., Pinzger, M., Gall, H. (2003). "Populating a release history database from version control and bug tracking systems", In Proceedings of ICSM 2003, Amsterdam: IEEE CS.,23-32.

[49] Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V. ve diğerleri. (2009). "Fair and balanced?: bias in bug-fix datasets", In Proceedings of ESEC/FSE, New York, NY, USA,: ACM, 121-130.

EK-A

DENEYSEL HATA KESTİRİM VERİTABANI

Yapılan çalışmalar sonucunda, incelenen ATM İzleme ve Yönetim Sistemi yazılım kodlarından elde edilen sınıflar ve sınıflara ait metrik değerleri ile basit bir hata kestirim veri tabanı oluşturulmuştur. Bu veri tabanında incelenen 713 sınıf ve bu sınıflara ait 53 farklı metrik değeri bulunmaktadır.

Çizelge A. 1 Oluşturulan metrik veri tabanı özet bilgileri

Çalışma Tarihi	2014-2015
İncelenen Sınıf	713
Metrik Sayısı	53
Tahmin Kesiti Sürümü	29576
Doğrulama Kesiti Sürümü	37133

	File	Average Block Depth	Average Cyclomatic Complexity	Average Depth of Inheritance Hierarchy	Average Lines Of Code Per Method	Average Number of Constructors Per Type	Average Number of Fields Per Type	Average Number of Methods Per Type	Average Number of Parameters	Average Number of Subtypes	Comments Ratio	Difficulty	Effort	Lines of Code	Number of Characters	Number of Comments	Number of Constructors	Number of Fields	Number of Lines	Numt Metl
2	tr.com.provus.pays.command.caller.modules.SFInitializationModule.java	1,5	2,5	3	11	0	0	2	0,5	0	0,028	7,17	2728,08	35	1478	1	0	1	45	
3	tr.com.provus.pays.command.caller.modules.ModuleManager.java	1	1	2	5	0	0	1	0	0	0	3,11	218,73	12	443	0	0	0	17	
4	tr.com.provus.pays.command.caller.call.operation.PingCallOperation.java	1	1,66	3	11,66	1	1	2	0	0	0,061	10,78	9092,47	49	2144	3	1	2	65	
5	tr.com.provus.pays.command.caller.call.operation.GetAtmPortCallOperation.java	1,33	1,66	3	11,66	1	0	2	0	0	0,058	9,42	7574,42	51	2227	3	1	1	66	
6	tr.com.provus.pays.command.caller.call.operation.GetEjPortCallOperation.java	1,33	1,66	3	11,66	1	0	2	0	0	0,098	9,42	7574,42	51	2327	5	1	1	69	
7	tr.com.provus.pays.command.caller.call.operation.CommandCallOperation.java	0,75	1,21	2	5,47	1	9	22	0,4	3	0,07	15,16	38211,9	156	5755	11	1	10	210	
8	tr.com.provus.pays.command.caller.exception.CommandCallOperationException.java	1	1	4	3	4	0	0	0	0	0,062	2,66	267,52	16	526	1	4	1	26	
9	tr.com.provus.pays.command.caller.exception.CommandRunOperationException.java	1	1	4	3	4	0	0	0	0	0,062	2,66	267,52	16	519	1	4	1	25	
10	tr.com.provus.pays.command.caller.config.InstitutionConfiguration.java	0,7	1	2	3,14	1	3	6	0,5	0	0,137	3,64	625,38	29	1157	4	1	3	49	
11	tr.com.provus.pays.command.caller.config.RMIConnectionProperty.java	0,77	1	2	4	2	2	5	0,4	0	0,151	8,16	2508,73	33	1311	5	2	2	53	
12	tr.com.provus.pays.command.caller.config.PAYSCommandCallerConfig.java	0,81	1,37	2	4,37	1	3	7	0,42	0	0,163	10,19	5391,11	49	2196	8	1	4	78	
13	tr.com.provus.pays.command.caller.PAYSCommandCaller.java	1,8	2,82	2	20,55	1	4	33	5,36	0	0,14	50,45	832604,29	763	38400	107	1	5	943	
14	tr.com.provus.pays.command.caller.PAYSCommandCallerFactory.java	1	2	2	8,66	0	1	3	0,66	0	0,078	9,32	4273,24	38	1734	3	0	1	48	
15	tr.com.provus.pays.command.caller.run.operation.GoOutOfServiceRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1232	3	1	0	38	
16	tr.com.provus.pays.command.caller.run.operation.CheckAtmStatusRunOperation.java	0,5	1	5	8,33	1	3	2	0,5	0	0,023	5,41	2887,3	42	2059	1	1	4	57	
17	tr.com.provus.pays.command.caller.run.operation.RepairAtmPortRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1230	3	1	0	39	
18	tr.com.provus.pays.command.caller.run.operation.PingRunOperation.java	0,75	1	5	6	1	1	2	0,5	0	0,037	3,28	903,57	27	1131	1	1	1	36	
19	tr.com.provus.pays.command.caller.run.operation.CommandRunOperation.java	0,82	1,8	4	9,65	2	9	18	0,44	23	0,137	29,12	151839,37	233	10604	32	2	10	336	
20	tr.com.provus.pays.command.caller.run.operation.EjournalDataImportRunOperation.java	0,6	1	5	5,66	1	2	2	0,5	0	0,148	3,76	1128,2	27	1277	4	1	2	43	
21	tr.com.provus.pays.command.caller.run.operation.StopAtmPortRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1220	3	1	0	38	
22	tr.com.provus.pays.command.caller.run.operation.MarkIdleRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1206	3	1	0	37	
23	tr.com.provus.pays.command.caller.run.operation.GetEjournalRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1225	3	1	0	38	
24	tr.com.provus.pays.command.caller.run.operation.GetCounterRunOperation.java	0,75	1	5	6	1	1	2	0,5	0	0,103	3,22	992,76	29	1392	3	1	1	42	
25	tr.com.provus.pays.command.caller.run.operation.GetAtmPortRunOperation.java	0,75	1	5	6	1	1	2	0,5	0	0,142	3,2	944,23	28	1452	4	1	1	42	
26	tr.com.provus.pays.command.caller.run.operation.StartAtmPortRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,076	1,5	368,01	26	1200	2	1	0	36	
27	tr.com.provus.pays.command.caller.run.operation.DirtyFitnessRunOperation.java	0,6	1	5	6,66	1	2	2	0,5	0	0,121	3,54	1441,85	33	1605	4	1	2	49	
28	tr.com.provus.pays.command.caller.run.operation.StartEjPortRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1220	3	1	0	38	
29	tr.com.provus.pays.command.caller.run.operation.GetCashInCountersRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0	1,53	347,6	25	1095	0	1	0	32	
30	tr.com.provus.pays.command.caller.run.operation.RepairEjPortRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1224	3	1	0	38	
31	tr.com.provus.pays.command.caller.run.operation.GetEjPortRunOperation.java	0,83	1	5	5	1	1	4	0,5	0	0,114	3,43	1208,46	35	1718	4	1	1	51	
32	tr.com.provus.pays.command.caller.run.operation.GetAllPortRunOperation.java	0,75	1	5	5,33	1	1	2	0,5	0	0	2,84	642,42	26	1184	0	1	1	34	
33	tr.com.provus.pays.command.caller.run.operation.LoadRunOperation.java	0,75	1	5	6	1	1	2	0,5	0	0,068	3,31	1113,14	29	1325	2	1	1	40	
34	tr.com.provus.pays.command.caller.run.operation.ClearFitnessRunOperation.java	0,6	1	5	6,33	1	2	2	0,5	0	0,142	4,65	2118,83	35	1759	5	1	3	55	
35	tr.com.provus.pays.command.caller.run.operation.ClearStatusRunOperation.java	0,75	1	5	6	1	1	2	0,5	0	0,137	3,41	1140,02	29	1406	4	1	1	43	
36	tr.com.provus.pays.command.caller.run.operation.StopEjPortRunOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1216	3	1	0	38	
37	tr.com.provus.pays.command.caller.run.operation.GoInServiceOperation.java	1	1	5	5,66	1	0	2	0,5	0	0,115	1,5	368,01	26	1216	3	1	0	38	
38	tr.com.provus.pays.command.caller.run.operation.MarkForSchedulerCommandRunOperation.java	0,75	1	5	5,33	1	1	2	0,5	0	0	3,07	739,8	25	1154	0	1	1	32	
39	tr.com.provus.common.command.response.AtmState.java	0,8	1,45	2	4,81	2	4	9	0,44	0	0,066	16,06	13420,96	60	1915	4	2	4	86	
40	tr.com.provus.common.command.response.AtmPortState.java	0,71	1	2	4	2	6	13	0,46	0	0,083	10,27	10499,6	72	2440	6	2	7	109	
41	tr.com.provus.common.command.response.MbportcmdResponse.java	1	1,33	2	6,16	2	1	4	0,5	0	0,133	14,12	10726,14	45	1835	6	2	2	69	
42	tr.com.provus.common.command.response.CommandState.java	0,71	1,19	2	4	1	2	4	0,5	0	0	10,5	2408,57	26	907	0	1	2	36	
43	tr.com.provus.common.command.PAYSCommand.java	0	1	1	2,42	0	0	26	4,5	0	0,342	3,6	8722,37	76	10046	26	0	2	349	
44	tr.com.provus.common.command.enums.StatusDevice.java	0,8	1,25	3	3,75	1	2	3	0	0	0	6,54	936,4	20	420	0	1	2	28	
45	tr.com.provus.common.command.enums.PaysAtmCmd.java	1	1,37	3	6,37	1	4	7	0,14	0	0,083	12,38	13065,77	60	2247	5	1	4	84	
46	tr.com.provus.common.command.enums.AtmrkmCmd.java	1	1,33	3	3,66	1	7	8	0,12	0	0,046	8,35	4998,39	43	1466	2	1	7	66	
47	tr.com.provus.common.command.enums.MbPortCmd.java	1,05	1,87	3	4,87	1	12	15	0,06	0	0,097	7,94	10954,9	92	2955	9	1	12	140	
48	tr.com.provus.common.command.enums.LoadDevice.java	1	1,93	3	4,93	1	12	14	0,07	0	0,123	7,94	10954,9	89	2238	11	1	12	150	
49	tr.com.provus.common.command.enums.ProvusAtmCmd.java	1,14	2,35	3	6,7	1	19	19	0,36	0	0,025	13,93	47606,75	159	5506	4	1	19	204	
50	tr.com.provus.common.command.enums.DbAtmCmd.java	1,19	3,68	3	8,57	1	13	18	0,22	0	0	13,88	39378,11	179	5195	0	1	13	212	
51	tr.com.provus.common.command.enums.OasisEjournalDataImportStatus.java	1	1,42	3	3,85	1	6	6	0,16	0	0,138	7,75	3736,68	36	1167	5	1	6	65	
52	tr.com.provus.common.command.enums.OasisEjournalDataImportType.java	1	1,57	3	3,57	1	15	13	0,07	0	0,117	11,17	10567,07	69	2615	13	1	15	195	

Şekil A.1 Oluşturulan deneysel hata kestirim veri tabanından bir kesit

EK-B

KAYNAK KODLAR

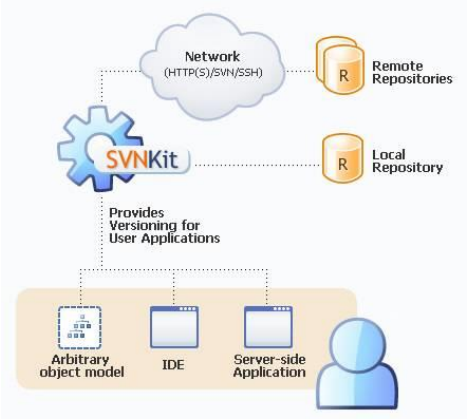
B.1 Metrik Hesaplayıcısı Uygulaması

Çalışma kapsamında, kaynak kod metriklerinin analiz edilip istenen formata getirilmesi ve SVN üzerindeki kaynak koddan bazı metriklerin otomatik olarak elde edilebilmesi için bir uygulama geliştirilmiştir. Geliştirilen uygulama java kodu ile geliştirilmiştir.

Metrik hesaplayıcı uygulaması, Bölüm 5.3'te ayrıntılı anlatılan Google Code Pro Analytix ve Metrics2 araçları ile hesaplanan metrik değerlerini içeren XML dosya çıktılarını işleyerek, çalışmamızda kullanabileceğimiz uygun formata getirerek lojistik regresyon analizine girdi olarak verilebilecek hale getirir.

Uygulamanın geliştirmesinde aşağıdaki kütüphanelerden faydalanılmıştır:

SVNKit: İncelenen yazılım kodları açık kaynak kodlu Subversion (SVN) kod yapılandırma sisteminde tutulduğu için SVN'e erişmek ve sorgulamalar için ise SVNKit (http://svnkit.com) kütüphanesinin 1.8.6 sürümü kullanılmıştır.



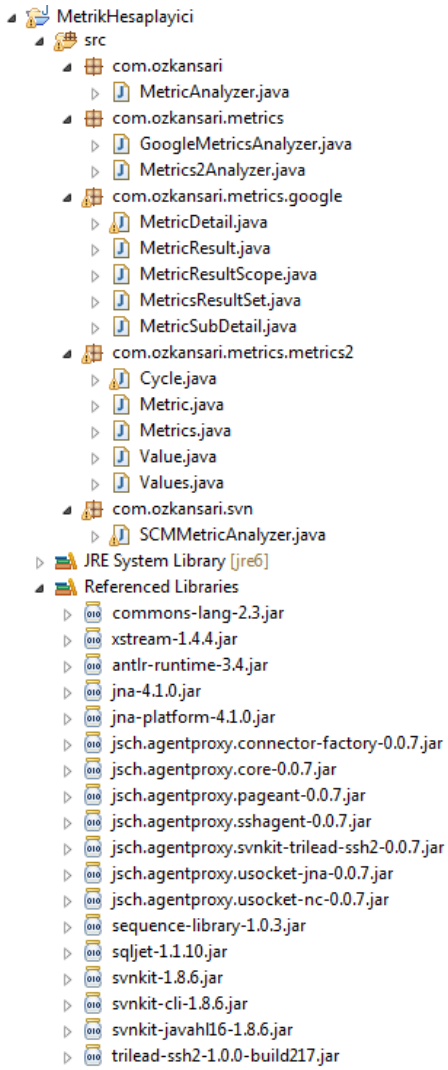
Şekil B.1 SVNKit kütüphanesi mimarisi

XStream: XML dosyalarının işlenmesinde XStream (http://xstream.codehaus.org/) kütüphanesinin 1.4.4 sürümü kullanılmıştır. XStream kütüphanesi sunduğu basit ara yüzlerle, XML dosya içeriklerinin kolay bir şekilde java sınıfları ile eşleştirilmesini sağladığı için tercih edilmiştir.

Apache Commons Lang: Projedeki basit yardımcı java programlama dilindeki yardımcı işlemler için Apache Foundation tarafından yayınlanan Commons Lang (https://commons.apache.org/proper/commons-lang/) kütüphanesinin 2.3 versiyonu kullanılmıştır.

B.2 Metrik Hesaplayıcısı Uygulaması Kaynak Kodu

Şekil B.2'de projedeki kaynak kodun genel yapısı görülmektedir.



Şekil B.2 Geliştirilen metrik hesaplayıcı projesi yapısı

MetricAnalyzer Ana Sınıfı:

```
package com.ozkansari;

import java.util.ArrayList;
import java.util.List;

import org.tmatesoft.svn.core.SVNException;
import org.tmatesoft.svn.core.io.SVNRepository;

import com.ozkansari.metrics.GoogleMetricsAnalyzer;
import com.ozkansari.metrics.Metrics2Analyzer;
import com.ozkansari.metrics.google.MetricResult;
import com.ozkansari.metrics.google.MetricResultScope;
import com.ozkansari.metrics.google.MetricsResultSet;
import com.ozkansari.svn.SCMMetricAnalyzer;

public class MetricAnalyzer {

    /**
     * @param args
     * @throws SVNException
     */
    public static void main(String[] args) throws SVNException {

        String projectName = args[0]; // "PAYSCommandCaller";
        MetricsResultSet googleMetricsResult = new GoogleMetricsAnalyzer()
                .parse("results/" + projectName + "/GoogleMetricResults.xml");

        MetricsResultSet metrics2Result = new Metrics2Analyzer()
                .parse("results/" + projectName + "/Metrics2.xml");

        MetricsResultSet mergedResult = new MetricsResultSet();
        mergedResult.getMetricResultScopes().addAll(
                googleMetricsResult.getMetricResultScopes());

        for (MetricResultScope mergedScope : mergedResult
                .getMetricResultScopes()) {
            MetricResultScope metrics2Scope = metrics2Result
                    .getMetricResultScope(mergedScope.getScope());
            if (metrics2Scope == null) {
                continue;
            }
            mergedScope.getMetricResults().addAll(
                    metrics2Scope.getMetricResults());

            String targetPath = mergedScope.getScope().replaceAll(".java", "")
                    .replaceAll("\\.", "/")
                    + ".java";

            SCMMetricAnalyzer scmMetricAnalyzer = new SCMMetricAnalyzer(
                    projectName, targetPath);
            SVNRepository repository = scmMetricAnalyzer.getRepository();
            // repository.getLatestRevision()
            long predictionVersion = repository
                    .getDatedRevision(SCMMetricAnalyzer.PREDICTION_VERSION_DATE); // 29576
            long validationVersion = repository
                    .getDatedRevision(SCMMetricAnalyzer.VALIDATION_VERSION_DATE); // 37124

            List<MetricResult> scmMetricsOld = scmMetricAnalyzer
                    .findSCMMetrics(0, predictionVersion);

            mergedScope.getMetricResults().addAll(scmMetricsOld);
            List<MetricResult> scmMetricsNew = scmMetricAnalyzer
                    .findSCMMetrics(predictionVersion, validationVersion);

            mergedScope.getMetricResults().addAll(scmMetricsNew);
        }
...
```

Şekil B.3 MetricAnalyzer ana sınıfı kodu

```
...
        // Calculate header lines
        List<String> metrics = new ArrayList<String>();
        metrics.add("File");
        for (MetricResultScope scope : mergedResult.getMetricResultScopes()) {
            for (MetricResult result : scope.getMetricResults()) {
                if (!metrics.contains(result.getName())) {
                    metrics.add(result.getName());
                }
            }
        }

        GoogleMetricsAnalyzer.print(mergedResult, metrics);
    }

}
```

Şekil B.3 MetricAnalyzer ana sınıfı kodu (devamı)

GoogleMetricsAnalyzer Sınıfı:

```
package com.ozkansari.metrics;

import java.io.File;
import java.util.List;

import com.ozkansari.metrics.google.MetricDetail;
import com.ozkansari.metrics.google.MetricResult;
import com.ozkansari.metrics.google.MetricResultScope;
import com.ozkansari.metrics.google.MetricSubDetail;
import com.ozkansari.metrics.google.MetricsResultSet;
import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;

/**
 *
 */
public class GoogleMetricsAnalyzer {

    /**
     *
     * @param fileFullPath
     * @return
     */
    public MetricsResultSet parse(String fileFullPath) {
        XStream xstream = new XStream(new DomDriver());

        // Class aliases
        xstream.alias("metric-result-set", MetricsResultSet.class);
        xstream.alias("metric-result-scope", MetricResultScope.class);
        xstream.alias("metric-result", MetricResult.class);
        xstream.alias("metric-detail", MetricDetail.class);
        xstream.alias("metric-sub-detail", MetricSubDetail.class);
        // MetricResult attributes
        xstream.aliasAttribute(MetricResult.class, "name", "name");
        xstream.aliasAttribute(MetricResult.class, "value", "value");
        // MetricResultScope attributes
        xstream.aliasAttribute(MetricResultScope.class, "scope", "scope");
        // Implicit Collections
        xstream.addImplicitCollection(MetricsResultSet.class,
                "metricResultScopes");
        xstream.addImplicitCollection(MetricResultScope.class, "metricResults");
        xstream.addImplicitCollection(MetricResult.class, "metricDetails");
        xstream.addImplicitCollection(MetricDetail.class, "metricSubDetails");

        return (MetricsResultSet) xstream.fromXML(new File(fileFullPath));
    }
...
```

Şekil B.4 GoogleMetricsAnalyzer sınıfı kodu

```
...
    /**
     * @param metricsResultSet
     * @param metrics
     */
    public static void print(MetricsResultSet metricsResultSet,
            List<String> metrics) {
        boolean headerLine = true;

        for (MetricResultScope scope : metricsResultSet.getMetricResultScopes()) {

            if (!scope.getScope().contains(".java"))
                continue;

            if (headerLine) {
                for (String m : metrics) {
                    System.out.print("" + m + " , ");
                }
                System.out.println("");
                headerLine = false;
            }

            System.out.print("" + scope.getScope() + " ,");
            for (String metricName : metrics) {
                if (metricName.equals("File")) {
                    continue;
                }
                MetricResult result = scope.getMetricResult(metricName);
                String value = "0";
                if (result != null) {
                    value = result.getCount() > 1 ? result.getAverage()
                            : String.valueOf(result.getDoubleValue());
                }
                System.out.print("" + value + " ,");
            }

            System.out.println("");
        }
    }
}
```

Şekil B.4 GoogleMetricsAnalyzer sınıfı kodu (devamı)

Metrics2Analyzer Sınıfı:

```
package com.ozkansari.metrics;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import com.ozkansari.metrics.google.MetricResult;
import com.ozkansari.metrics.google.MetricResultScope;
import com.ozkansari.metrics.google.MetricsResultSet;
import com.ozkansari.metrics.metrics2.Cycle;
import com.ozkansari.metrics.metrics2.Metric;
import com.ozkansari.metrics.metrics2.Metrics;
import com.ozkansari.metrics.metrics2.Value;
import com.ozkansari.metrics.metrics2.Values;
import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;

public class Metrics2Analyzer {

    private MetricsResultSet set;

...
```

Şekil B.5 Metrics2Analyzer sınıfı kodu

```
...
    public Metrics2Analyzer() {
        set = new MetricsResultSet();
        set.setMetricResultScopes(new ArrayList<MetricResultScope>());
    }

    public MetricsResultSet parse(String fileFullPath) {
        XStream xstream = new XStream(new DomDriver());

        // Introduce Metrics
        xstream.alias("Metrics", Metrics.class);
        xstream.addImplicitCollection(Metrics.class, "metricList", Metric.class);
        xstream.aliasAttribute(Metrics.class, "scope", "scope");
        xstream.aliasAttribute(Metrics.class, "type", "type");
        xstream.aliasAttribute(Metrics.class, "date", "date");
        xstream.aliasAttribute(Metrics.class, "xmlns", "xmlns");

        // Introduce Cycle
        xstream.alias("Cycle", Cycle.class);
        xstream.addImplicitCollection(Cycle.class, "packageList");

        // Introduce Package
        xstream.alias("Package", String.class);

        // Introduce Metrics
        xstream.alias("Metric", Metric.class);
        xstream.aliasAttribute(Metric.class, "id", "id");
        xstream.aliasAttribute(Metric.class, "description", "description");
        xstream.aliasAttribute(Metric.class, "max", "max");
        xstream.aliasAttribute(Metric.class, "hint", "hint");

        // Introduce Values
        xstream.alias("Values", Values.class);
        xstream.aliasAttribute(Values.class, "per", "per");
        xstream.aliasAttribute(Values.class, "avg", "avg");
        xstream.aliasAttribute(Values.class, "stddev", "stddev");
        xstream.aliasAttribute(Values.class, "max", "max");
        xstream.addImplicitCollection(Values.class, "valueList");

        // Introduce Value
        xstream.alias("Value", Value.class);
        xstream.aliasAttribute(Value.class, "name", "name");
        xstream.aliasAttribute(Value.class, "source", "source");
        xstream.aliasAttribute(Value.class, "packageName", "package");
        xstream.aliasAttribute(Value.class, "value", "value");

        Metrics metrics = (Metrics) xstream.fromXML(new File(fileFullPath));

        for (Metric metric : metrics.getMetricList()) {

            if (metric.getValues() != null) {

                if (metric.getValues().getPer().equals("method")) {
                    // sinifdaki metodlar bazinda
                    // sinif toplami ve ortalamasi bulunmali

                } else if (metric.getValues().getPer().equals("type")) {
                    // sinif bazinda

                }

                for (Value value : metric.getValues().getValueList()) {
                    MetricResultScope scope = addToMetricResultScope(
                            metric.getDescription() + " (" + metric.getId()+ ")",
                            value.getPackageName() + "." + value.getSource(),
                            value.getValue(), metric.getValues().getPer());
                    set.addMetricResultScope(scope);
                }
            }
...
```

Şekil B.5 Metrics2Analyzer sınıfı kodu (devamı)

```
...
            else if (metric.getValue() != null) {

                // Bir sey yapma
                // Degerler sinif bazinda degil
            }
        }

        return set;
    }

    /**
     *
     * @param metric
     *            metric name
     * @param scope
     *            full source name (package+class)
     * @param value
     *            metric value
     * @param per
     *            method/type/packageFragment
     *
     * @return
     */
    private MetricResultScope addToMetricResultScope(
            String metric,
            String scope,
            String value,
            String per)
    {

        // Set MetricResultScope
        MetricResultScope metricResultScope = set.getMetricResultScope(scope);

        if (metricResultScope == null) {
            metricResultScope = new MetricResultScope(scope);
            set.addMetricResultScope(metricResultScope);
        }

        // Set MetricResult List
        MetricResult metricResult = null;

        List<MetricResult> metricResults = metricResultScope.getMetricResults();

        if (metricResults == null || metricResults.isEmpty()) {
            metricResults = new ArrayList<MetricResult>();
            metricResultScope.setMetricResults(metricResults);
        } else {
            metricResult = metricResultScope.getMetricResult(metric);
        }

        // Set MetricResult
        if (metricResult == null) {
            metricResult = new MetricResult(metric, value);
            metricResultScope.getMetricResults().add(metricResult);
        } else {
            double totalValue = Double.parseDouble(metricResult.getValue())
                    + Double.parseDouble(value);
            metricResult.setValue(String.valueOf(totalValue));
            metricResult.incrementCount();
        }

        return metricResultScope;
    }

}
```

Şekil B.5 Metrics2Analyzer sınıfı kodu (devamı)

MetricDetail Sınıfı:

```
package com.ozkansari.metrics.google;

import java.util.List;

public class MetricDetail {
    private List<MetricSubDetail> metricSubDetails;
}
```

Şekil B.6 MetricDetail sınıfı kodu

MetricResult Sınıfı:

```
package com.ozkansari.metrics.google;

import java.util.List;
import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;

public class MetricResult {

    private List<MetricDetail> metricDetails;
    private String name;
    private String value;
    private int count = 0;

    public MetricResult() { }

    public MetricResult(String theName, String theValue) {
        this.name = theName;
        this.value = theValue;
        this.count = 1;
    }

    public List<MetricDetail> getMetricDetails() {
        return metricDetails;
    }

    public String getName() {
        return name;
    }

    public String getValue() {
        return value;
    }

    public double getDoubleValue() {

        String valueDoubleStr = value;
        if (valueDoubleStr.contains(",")) {
            valueDoubleStr = valueDoubleStr.replaceAll(",", "");
        }

        double valueDouble;
        if (valueDoubleStr.contains("%")) {
            valueDouble = Double
                    .parseDouble(valueDoubleStr.replaceAll("%", "")) / 100.0;
        } else {
            valueDouble = Double.parseDouble(valueDoubleStr);
        }
        return valueDouble;
    }
...
```

Şekil B.7 MetricResult sınıfı kodu

```
...
    @Override
    public boolean equals(Object obj) {

        if (!(obj instanceof MetricResult)) {
            return false;
        }

        if (obj == this) {
            return true;
        }
        MetricResult currentResult = (MetricResult) obj;

        return new EqualsBuilder().
                append(name, currentResult.getName()).isEquals();    }

    @Override
    public int hashCode() {
        return new HashCodeBuilder(17, 31).
                append(name).toHashCode();
    }

    public void setValue(String value) {
        this.value = value;
    }

    public void incrementCount() {
        this.count++;
    }

    public int getCount() {
        if (count == 0) {
            return 1; // must be 1, if not set before
        }
        return count;
    }

    public String getAverage() {
        double avg = getDoubleValue() / count;
        return String.valueOf(avg);
    }
}
```

Şekil B.7 MetricResult sınıfı kodu (devamı)

MetricResultScope Sınıfı:

```
package com.ozkansari.metrics.google;

import java.util.List;

import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;

public class MetricResultScope {
    private List<MetricResult> metricResults;

    private String scope;

    public MetricResultScope() {

    }

    public MetricResultScope(String theScope) {
        this.scope = theScope;
    }
...
```

Şekil B.8 MetricResultScope sınıfı kodu

```
...
    public List<MetricResult> getMetricResults() {
        return metricResults;
    }

    public void setMetricResults(List<MetricResult> metricResults) {
        this.metricResults = metricResults;
    }

    public String getScope() {
        return scope;
    }

    public MetricResult getMetricResult(String metric) {
        if (metricResults == null) {
            return null;
        }
        int index = metricResults.indexOf(new MetricResult(metric, null));
        if (index >= 0) {
            return metricResults.get(index);
        }
        return null;
    }

    @Override
    public boolean equals(Object obj) {

        if (!(obj instanceof MetricResultScope)) {
            return false;
        }
        if (obj == this) {
            return true;
        }
        MetricResultScope currentScope = (MetricResultScope) obj;

        return new EqualsBuilder().
                append(scope, currentScope.getScope()).isEquals();
    }

    @Override
    public int hashCode() {
        return new HashCodeBuilder(17, 31).
                append(scope).toHashCode();
    }

}
```

Şekil B.8 MetricResultScope sınıfı kodu (devamı)

MetricsResultSet Sınıfı:

```
package com.ozkansari.metrics.google;

import java.util.ArrayList;
import java.util.List;

public class MetricsResultSet {

    private List<MetricResultScope> metricResultScopes;

    public MetricsResultSet() {
        metricResultScopes = new ArrayList<MetricResultScope>();
    }

    public List<MetricResultScope> getMetricResultScopes() {
        return metricResultScopes;
    }
...
```

Şekil B.9 MetricResultSet sınıfı kodu

```
...
    public void setMetricResultScopes(List<MetricResultScope> metricResultScopes) {
        this.metricResultScopes = metricResultScopes;
    }

    public void addMetricResultScope(MetricResultScope scope) {
        if (metricResultScopes == null) {
            metricResultScopes = new ArrayList<MetricResultScope>();
        }

        if (metricResultScopes.contains(scope)) {
            return;
        }

        metricResultScopes.add(scope);
    }

    public MetricResultScope getMetricResultScope(String scope) {

        if (metricResultScopes == null) {
            return null;
        }

        int index = metricResultScopes.indexOf(new MetricResultScope(scope));

        if (index >= 0) {
            return metricResultScopes.get(index);
        }

        return null;
    }
}
```

Şekil B.9 MetricResultSet sınıfı kodu (devamı)

MetricSubDetail Sınıfı:

```
package com.ozkansari.metrics.google;

public class MetricSubDetail {

    private String name;

    private String value;

    public MetricSubDetail() {

    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

Şekil B.10 MetricSubDetail sınıfı kodu

Cycle Sınıfı:

```
package com.ozkansari.metrics.metrics2;

import java.util.List;

public class Cycle {

    private List<String> packageList;

}
```

Şekil B.11 Cycle sınıfı kodu

Metric Sınıfı:

```
package com.ozkansari.metrics.metrics2;

/**
 * <Metric
 *  id = "VG"
 *  description ="McCabe Cyclomatic Complexity"
 *  max ="10"
 *  hint="use Extract-method to split the method up"
 * >
 */
public class Metric {

    private String id;

    private String description;

    private String max;

    private String hint;

    private Values Values;

    private Value Value;

    public String getId() {
        return id;
    }

    public String getDescription() {
        return description;
    }

    public String getMax() {
        return max;
    }

    public String getHint() {
        return hint;
    }

    public Values getValues() {
        return Values;
    }

    public Value getValue() {
        return Value;
    }
}
```

Şekil B.12 Metric sınıfı kodu

Metrics Sınıfı:

```
package com.ozkansari.metrics.metrics2;

import java.util.List;

/**
 * Scope bazındaki Metric listesini iceren, ayrica date, type gibi ek bilgileri içeren siniftir.
 *
 * <Metrics
 *  scope="MyColumnist"
 *  type="Project"
 *  date="2014-10-14"
 *  xmlns=http://metrics.sourceforge.net/2003/Metrics-First-Flat
 * >
 *
 */
public class Metrics {

    private String scope;

    private String type;

    private String date;

    private String xmlns;

    private Cycle Cycle;

    /**
     * com.ozkansari.metrics.metrics2. Metric sinifi listesi
     */
    private List<Metric> metricList;

    /**
     * Kapsam bilgisi
     */
    public String getScope() {
        return scope;
    }

    /**
     * Tip bilgisi
     */
    public String getType() {
        return type;
    }

    public String getDate() {
        return date;
    }

    public String getXmlns() {
        return xmlns;
    }

    public Cycle getCycle() {
        return Cycle;
    }

    public List<Metric> getMetricList() {
        return metricList;
    }

}
```

Şekil B.13 Metrics sınıfı kodu

Value Sınıfı:

```
package com.ozkansari.metrics.metrics2;

/**
 * <Value name="checkCookies" source ="CookieManager.java"
 *  package="com.mycolumnist.model" value ="4"/>
 */
public class Value {
    private String name;
    private String source;
    private String packageName;
    private String value;

    public String getName() {
        return name;
    }

    public String getSource() {
        return source;
    }

    public String getPackageName() {
        return packageName;
    }

    public String getValue() {
        return value;
    }
}
```

Şekil B.14 Value sınıfı kodu

Values Sınıfı:

```
package com.ozkansari.metrics.metrics2;
import java.util.List;
/**
 * <Values per = "method" avg = "1.447" stddev = "1" max = "9">
 */
public class Values {
    private String per;
    private String avg;
    private String stddev;
    private String max;
    private List<Value> valueList;

    public String getPer() {
        return per;
    }

    public String getAvg() {
        return avg;
    }

    public String getStddev() {
        return stddev;
    }

    public String getMax() {
        return max;
    }

    public List<Value> getValueList() {
        return valueList;
    }
}
```

Şekil B.15 Values sınıfı kodu

GoogleMetricsAnalyzer Sınıfı:

```
package com.ozkansari.svn;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.regex.Pattern;
import org.tmatesoft.svn.core.SVNDirEntry;
import org.tmatesoft.svn.core.SVNException;
import org.tmatesoft.svn.core.SVNLogEntry;
import org.tmatesoft.svn.core.SVNNodeKind;
import org.tmatesoft.svn.core.SVNURL;
import org.tmatesoft.svn.core.auth.ISVNAuthenticationManager;
import org.tmatesoft.svn.core.internal.io.dav.DAVRepositoryFactory;
import org.tmatesoft.svn.core.io.SVNRepository;
import org.tmatesoft.svn.core.io.SVNRepositoryFactory;
import org.tmatesoft.svn.core.wc.SVNWCUtil;

import com.ozkansari.metrics.google.MetricResult;

public class SCMMetricAnalyzer {

    public static final Date PREDICTION_VERSION_DATE = new GregorianCalendar(
            2012, 4, 8, 17, 57, 0).getTime(); // Version 29576
    public static final Date VALIDATION_VERSION_DATE = new GregorianCalendar(
            2014, 10, 25, 15, 10, 0).getTime(); // Version 37124
    public static final int HEAD_VERSION = -1;
    private static final String[] ERROR_LOGS = new String[] { "hata", "bug",
            "duzelt", "düzelt", "fix", "sorun" };

    private static final String NAME = "ozkans";
    private static final String PWD = "2231559";
    private static final boolean DEBUG = false;

    private SVNRepository repository = null;

    private String projectName;

    private String targetPath;

    public SCMMetricAnalyzer(String projectName, String targetPath) {
        this.projectName = projectName;
        this.targetPath = targetPath;

        DAVRepositoryFactory.setup();
        try {
            String svnPath = "http://10.81.136.240:8090/svn/provus/java/"
                    + projectName + "/trunk/src/" + targetPath;
            SVNURL svnURL = SVNURL.parseURIEncoded(svnPath);
            repository = SVNRepositoryFactory.create(svnURL);
            ISVNAuthenticationManager authManager =
                    SVNWCUtil.createDefaultAuthenticationManager(NAME, PWD);
            repository.setAuthenticationManager(authManager);
        } catch (SVNException e) {
            e.printStackTrace();
        }
    }

    public SVNRepository getRepository() {
        return repository;
    }
...
```

Şekil B.16 GoogleMetricsAnalyzer sınıfı kodu

```
...
    public static void main(String[] args) {
        try {
            new SCMMetricAnalyzer(args[0], "").listEntries("");
        } catch (SVNException e) {
            e.printStackTrace();
        }
    }

    public List<MetricResult> findSCMMetrics(long startRevision, long endRevision) {

        List<MetricResult> metricResults = new ArrayList<MetricResult>();

        Collection<SVNLogEntry> logEntries = null;

        boolean hasBugsUntil = false;
        int authorCount = 0;
        Set<String> authors = new HashSet<String>();

        try {
            logEntries = repository.log(new String[] { "" }, null, 0, -1,false, false);
        } catch (SVNException e) {
            // No log found
            MetricResult authorCountMetricsResult = new MetricResult(
                    "AUTHOR COUNT UNTIL" + endRevision, "0");
            metricResults.add(authorCountMetricsResult);
            MetricResult hasBugsUntilMetricsResult = new MetricResult(
                    "HAS BUGS UNTIL" + endRevision, "0");
            metricResults.add(hasBugsUntilMetricsResult);
            return metricResults;
        }

        // Check all revisions
        for (Iterator<SVNLogEntry> entries = logEntries.iterator(); entries
                .hasNext();) {
            SVNLogEntry logEntry = (SVNLogEntry) entries.next();

            if (logEntry.getRevision() < startRevision
                    || logEntry.getRevision() > endRevision) {
                continue;
            }

            String logMsg = logEntry.getMessage();
            if (logMsg == null || logMsg.trim().length() == 0) {
                continue;
            }
            for (String errorLog : ERROR_LOGS) {

                if (Pattern
                        .compile("(" + errorLog + ")", Pattern.CASE_INSENSITIVE)
                        .matcher(logMsg).find()) {
                    hasBugsUntil = true;
                    break;
                }
            }

            if (!authors.contains(logEntry.getAuthor())) {
                authorCount++;
            }

        }

        MetricResult authorCountMetricsResult = new MetricResult(
                "AUTHOR COUNT UNTIL" + endRevision, String.valueOf(authorCount));
        metricResults.add(authorCountMetricsResult);
        MetricResult hasBugsUntilMetricsResult = new MetricResult(
                "HAS BUGS UNTIL" + endRevision, hasBugsUntil ? "1" : "0");
        metricResults.add(hasBugsUntilMetricsResult);
        return metricResults;
    }
...
```

Şekil B.16 GoogleMetricsAnalyzer sınıfı kodu (devamı)

```
...
    private void listEntries(String path) throws SVNException {
        Collection<SVNDirEntry> entries = repository.getDir(path, -1, null,
                (Collection<SVNDirEntry>) null);
        Iterator<SVNDirEntry> iterator = entries.iterator();
        while (iterator.hasNext()) {
            SVNDirEntry entry = (SVNDirEntry) iterator.next();

            if (entry.getKind() == SVNNodeKind.FILE) {

                if (DEBUG) {
                    System.out.print("/" + (path.equals("") ? "" : path + "/")
                            + entry.getName() + " ( author: '"
                            + entry.getAuthor() + "'; revision: "
                            + entry.getRevision() + "; date: "
                            + entry.getDate() + ")");
                }

                long predictionVersion = repository
                        .getDatedRevision(PREDICTION_VERSION_DATE);
                findSCMMetrics(0, predictionVersion);

            } else if (entry.getKind() == SVNNodeKind.DIR) {
                listEntries((path.equals("")) ? entry.getName() : path + "/"
                        + entry.getName());
            }
        }
    }
}
```

Şekil B.16 GoogleMetricsAnalyzer sınıfı kodu (devamı)

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Özkan SARI

Doğum Tarihi ve Yeri : 01.01.1984 , Kütahya

Yabancı Dili : İngilizce

E-posta : ozkansari@gmail.com

ÖĞRENİM DURUMU

Derece	**Alan**	**Okul/Üniversite**	**Mezuniyet Yılı**
Lisans	Bilgisayar Mühendisliği	Yeditepe Üniversitesi	2007
Lise	Fen Bölümü	Kütahya Lisesi	2002

İŞ TECRÜBESİ

Yıl	**Firma/Kurum**	**Görevi**
2015-……..	Provus Bilişim Hizmetleri A.Ş.	Yazılım Müdürü
2014- 2015	Provus Bilişim Hizmetleri A.Ş.	Yazılım Müdür Yardımcısı
2012-2014	Provus Bilişim Hizmetleri A.Ş.	Yazılım Şefi
2009-2012	Provus Bilişim Hizmetleri A.Ş.	Uzman Yazılımcı
2007-2008	Overteam Technologies	Uygulama Geliştirici

YAYINLARI

Bildiri

1. Sari, Ö., Kalipsiz, O. (2014). "Bug Prediction for an ATM Monitoring Software: Use of Logistic Regression Analysis for Bug Prediction", Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS), Barselona, İspanya.

2. Sari, Ö., Kalipsiz, O. (2014). "Yazılım Hata Kestirimi İçin Veri Analizi Yöntemlerinin Kullanılması", Proceedings of the 8th Turkish National Software Engineering Symposium, Güzelyurt, KKTC, Turkiye.

3. Sari, Ö., Kalipsiz, O. (2014). " ATM Yerleşim Noktası Karar Destek Sistemi", Proceedings of the 8th Turkish National Software Engineering Symposium, Güzelyurt, KKTC, Turkiye.

4. Sari, Ö. (2013). “Scrum Çevik Süreçlerinin Ar-Ge Yazılım Projelerinde Kullanımı”, ÇEYA 2013 İzmir, Türkiye

5. Gokturk,M., Sari, Ö.,Akkus, S.,Gecici, K. (2013). “An AHP-Based Interactive ATM Location Decision Support System”, Proceedings of the EWG-DSS Thessaloniki-2013 Workshop, Selanik, Yunanistan

Proje

1. TEYDEB Prj. ATM Servis Noktası Yerleşimi Karar Destek Sistemi - 2014

2. Lisans Tezi Gümrükler İçin Konteyner Kod Tanıma Sistemi - 2007