İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

NESNEYE DAYALI YAZILIMLARIN TASARIM KALİTESİNİ ÖLÇMEK İÇİN ÖĞRENME TABANLI BİR YÖNTEM

YÜKSEK LİSANS TEZİ

Nurdan CANBAZ

Bilgisayar ve Bilişim Fakültesi

Bilgisayar Bilimleri Ana Bilim Dalı

Tez Danışmanı: Doç. Feza BUZLUCA

Mayıs, 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

NESNEYE DAYALI YAZILIMLARIN TASARIM KALİTESİNİ ÖLÇMEK İÇİN ÖĞRENME TABANLI BİR YÖNTEM

YÜKSEK LİSANS TEZİ

Nurdan CANBAZ
(704101008)

Bilgisayar ve Bilişim Fakültesi

Bilgisayar Bilimleri Ana Bilim Dalı

Tez Danışmanı: Doç. Dr. Feza BUZLUCA

Mayıs, 2015

İTÜ, Bilişim Enstitüsü'nün 704101008 numaralı Yüksek LisansNurdan CANBAZ, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "Nesneye Dayalı Yazılımların Tasarım Kalitesini Ölçmek İçin Öğrenme Tabanlı Bir Yöntem**"**başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Feza BUZLUCA** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Banu DİRİ** Yıldız Teknik Üniversitesi	.....
	Y.Doç. Dr. Ayşe TOSUN MISIRLI İstanbul Teknik Üniversitesi	.....

Teslim Tarihi : **05 Mayıs 2015**
Savunma Tarihi : **27 Mayıs 2015**

Aileme,

ÖNSÖZ

Tez çalısmalarım süresince yardımlarından ve katkılarından dolayı tez danışmanım Sayın Doç. Dr. Feza BUZLUCA başta olmak üzere, hayatımın her döneminde olduğu gibi tez çalışmalarım boyunca da destek olan annem, babam ve abime sonsuz teşekkür eder, saygılarımı sunarım.

Mayıs 2015 Nurdan CANBAZ

İÇİNDEKİLER

Sayfa

KISALTMALAR

MOOD	: Metrics for Object Oriented Design
QMOOD	:Quality Model for Object Oriented Design
JFlex	: The Fast Scanner Generator for Java
SEI	: Software Engineering Institute
CMMI	: Capability Maturity Model Integration
WMC	: Weighted Methods per Class
DIT	: Depth of Inheritance Tree
NOC	: Number of Children
CBO	: Coupling Between Object Classes
RFC	: Response for a class
LCOM	: Lack of cohesion in methods
ART	: Adaptive Resonance Theory
SOM	: Self Organizing Map
VTBK	: Veri Transferi Bilgi Keşfi
CHAID	: Chi- Squared Automatic Interaction Detector
CRT	: Classification and Regression Trees
MARS	: Multivariate Adaptive Regression Splines
QUEST	: Quick, Unbiased, Efficient Statistical Tree
SLIQ	: Supervised Learning in Quest
SPRINT	: Scalable Paralleizable Induction of Decision Trees

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

NESNEYE DAYALI YAZILIMLARIN TASARIM KALİTESİNİ ÖLÇMEK İÇİN ÖĞRENME TABANLI BİR YÖNTEM

ÖZET

Bilgisayar yazılımlarının insan hayatına her geçen gün daha çok girmesi sonucunda kaliteli yazılım geliştirmek ve bunun paralelinde yazılım kalitesini ölçebilmek önem kazanmaya başlamıştır.

Yazılım projelerinde kalite kavramı projenin başından itibaren değerlendirilmesi gereken bir olgudur. Bununla birlikte yazılım kalitesi ölçülmesinin de proje başından itibaren belirlenen kalite özelliklerine uygun şekilde yapılmalıdır. Çalışmamızda ISO/IEC 25010:2011[28] kalite modeli temel alınarak projeye uygun kalite özelliklerinin proje başında seçilmesi ve buna uygun metrikler belirlenerek projenin başından itibaren kalite özelliklerinin nasıl değiştiğinin takip edilmesi ve ölçülebilir kalitesi olan bir ürünün müşteriye sunulması hedeflenmiştir.

Yazılım sektöründe en temel sorunlardan biri müşteri gereksinimlerindeki değişikliktir. Bu sorunun çözümü olarak çevik yazılım geliştirme metotları ortaya çıkmıştır. Günümüzde Scrum, Unified Process, XP gibi çevik yazılım geliştirme metodolojileri yazılım firmalar tarafından kullanılmaktadır. Bu metodolojilerle, ürün gereksinimlerini bölmek, ürün tam olarak bitmeden önce müşteriden geri bildirim almak ve bu geri bildirimlerle değişen gereksinimlere ayak uydurabilmek ya da gereksinimlerin değişmesini asgari seviyede tutmak amaçlanır. Ancak müşterinin her zaman ne istediğini tam olarak anlatamamasından, yazılımcının alan bilgisi eksikliğinden dolayı yinelemeli olarak ilerleyen bir süreç sonunda bile gereksinim değişikliği talebi gelmektedir. Bu yüzden bakımı kolay yazılım geliştirmek yazılım firmalarının çalışan ve zaman maliyetini düşürmek için önem teşkil etmektedir.

Çalışmada, amaçlanan tanı eksikliğine bağlı olarak yazılımda düzenlenecek bölümlerin belirlenmesinde, düzeltilmesinde ve test edilmesinde gereken çabanın bulunmasıdır. Bunun için bunların karşılığı olan ISO/IEC 25010:2011[28] kalite modelinde belirlenen bakım kolaylığı temel alınmıştır.

Bu modelde, bakım kolaylığının ISO/IEC 25010:2011[28] kalite modelinde bulunan alt kriterlerinden Analiz edilebilirlik, Yeniden kullanılabilirlik, Test edilebilirlik ve Değiştirilebilirlik seçilmiştir.

YazılımdaAnaliz edilebilirlik, Yeniden kullanılabilirlik, Test edilebilirlik ve Değiştirilebilirliğin nasıl değiştiğinin takip edilebilmesi, proje yürütücüleri için projenin ilerleyişinin izlenebilmesi, geliştiriciler için ortaya çıkabilecek sorunların önceden tahmin edilmesi açısından önem taşımaktadır. Bu kapsamda yazılım kalitesini ölçmeye dayalı metrikler ve istatistiksel analizler kullanılarak yazılım sınıflarının kalitesine ait değerlendirmeler yapılmıştır.

Değerlendirmeler neticesinde kalite özelliklerini karşılayan metrik kümeleri oluşturulmuştur. Metrik kümeleri oluşturulurken, yazılımın statik analizinden çıkan

metrik kümelerini dışında, proje ekibinde çalışan kişilerin yazılım ile ilgili alan bilgileri ve tecrübelerini içeren metrikler eklenmiştir. Ayrıca her sürümde çıkan hata sayısı ve hataların çözülme süreleri de metrik kümelerine dahil edilmiştir.

Veri kümeleri kullanılarak hazırlanan veri seti karar ağacı ve Naive Bayes yöntemlerine tabi tutulmuş ve ilerleyen sürümlerde alt kalite özellikleri temel alınarak bakım kolaylığının nasıl değişebileceğinin tahmin edilmesi sağlanmıştır.

Çalışmanın sonucunda, projenin önceki sürümlerine ait sınıf bilgilerinin kullanılması ile projenin bakım kolaylığı alt özelliklerinin gelecek sürümlerinde nasıl değişeceğinin tahmin edilmesine yönelik başarılı sonuçlar alınmıştır. Sonuçlar yazılım ekibi ve yöneticileri ile de değerlendirildiğinde sonuçları başarılı olduğu gözlenmiştir.

İlerleyen aşamalarda farklı veri madenciliği ve istatiksel yöntemler kullanılarak analiz yönteminin geliştirilmesi planlanmaktadır.

A LEARNING-BASED MEASUREMENT METHOD FOR DESIGN QUALITY OF OBJECT ORIENTED SOFTWARE SYSTEMS

SUMMARY

Since computer software has earned a more prominent place in our lives day by day, it has become increasingly important to develop quality software, and correspondingly, to measure software quality. Even low-scale software cannot be developed by a single person, but rather by a software team. This fact complicates the design, development and management of the software systems.

Software measurement studies are complemented by software engineering concepts, both of which aim to encourage the increase in software development efforts and quality improvement. There has been great interest in the measurement of software development processes in the last two decades; however, dealing with the measurement of software products also has equal importance.

In software projects, it is a must to evaluate the concept of quality starting from the beginning of the project.

Additionally, software quality measurement should be conducted according to the quality criteria, which are set at the beginning of the project.

In this work, we aimed to set the applicable criteria for the project based on the quality model ISO/IEC 25010:2011[28] at the beginning of the project, to monitor changes in quality criteria as of the start of the project by specifying relevant metrics and to present a product of measurable quality to the customer.

One of the basic problems in software sector is changing customer requirements. As a solution, agile software development methods have emerged.

Today, software companies use agile software development methodologies such as Scrum, Unified Process and XP. By using these methodologies, it is aimed to divide all product requirements, to receive feedback from customers before completing the product and to respond to changing requirements through customer feedback or to minimize such changes.

Even at the end of a recursive process due to inconveniences such as customers' inability to explain exactly what they want, and software specialist's lack of knowledge/expertise in his field, requirement changes are requested. Therefore, it is crucial to develop software that is easy to maintain to reduce employee and time costs in software companies.

The purpose of this work is to determine how much effort will be required to detect the relevant parts needs editing in software due to lack of diagnosis and to perform necessary edition and testing. In order to do so, as the equivalent of these steps in ISO/IEC 25010:2011[28] quality model, maintainability is taken as the basis of this work.

In this model, Analyzability, Reusability, Testability and Modifiability are selected from the sub criteria of maintainability in ISO/IEC 25010:2011[28] quality model.

Monitoring how Analyzability, Reusability, Testability and Modifiability change in software is important for project coordinators to track the progress of the project and for developers to predict probable problems in advance.

Within this scope, the quality of software classes is evaluated by using metrics and statistical analyses relating to software quality measurement.

As the result of these evaluations, metric sets - corresponding to the quality criteria - are created.

Apart from those concluded as the result of software statistical analyses, metrics including field knowledge and expertise of the project team members are also added while creating metric sets.

Additionally, the number of errors occurring in each version and time spent to fix these errors are also included in metric sets.

Following general analysis of the program, it was studied whether or not the Decision Tree method, which is one of the machine learning methods, would be compatible with quality properties (Maintainability, Flexibility and Reusability) specified for future versions.

This dataset is trained with decision tree and Naive Bayes methods. By means of this training model, it is shown that changes in maintainability can be predicted by considering the sub quality criteria in following versions.

In the decision tree learning, a tree structure is created to reflect class labels with tree leaves, and processes on properties with branches going to such leaves and coming out of the beginning. 4In our study we determined 2 criteria to classify quality properties (Maintainability, Reusability and Flexibility).

Quality criteria; Good,Bad (Needs refactoring)According to the criteria specified, software developers were asked to define one of the classification criteria specified to individual quality properties of different versions of each class. This dataset is trained with decision tree and Naive Bayes methods.

By means of this training model, it is shown that changes in maintainability can be predicted by considering the sub quality criteria in following versions.Basic concepts used in evaluation of success of the model are error rate, precision, recall and F-measure.

Success of the model is related to quantity of number of examples assigned to correct class and number of examples assigned to wrong class. Success information of results obtained from the test can be reflected with a complexity matrix. In complexity matrix, lines reflect actual numbers of examples in test set while columns reflect estimation of the model.

As the result of this work, we obtained successful results in predicting how the sub criteria of project maintainability may change in following versions by means of using class information pertaining to the previous versions of the project. The results are proved to be successful after being evaluated by software team and managers.

The study carried out has enabled to determine that quality properties to be specified in projects by using data mining techniques can be predicted in future versions and such predictions match the actual world. It is targeted to produce higher quality products by addressing to quality models specified in the project at development stage of the project.

In the forthcoming phases, it is beingplanned to develop analysis method by using different data mining and statistical methods.

The conclusion of the study reveals that, estimates constructed on the model for the project studied are consistent with the results in the real world.

During the flow of the study, it is intended to develop a computer vs aided software tool capable of automatically analyzing by means of different machine learning techniques, and scoring the quality attributes defined for the project. Hence, it is proposed to ensure that both the project coordinators can readily have inhand a full picture of the project, and also that the customer can track the current progress of the software.

1. GİRİŞ

Günümüzde yazılım endüstrisindeki gelişmeler ile birlikte kaliteli yazılım geliştirmenin önemi artmıştır. Küçük ölçekli yazılımlar bile artık tek kişi tarafından geliştirilmemekte bir yazılım ekibi tarafından oluşturulmaktadır. Bu da yazılımın tasarlanmasını, geliştirilmesini ve yönetilmesini zorlaştırmaktadır. Uygun yöntemler kullanılmadan ve kalite kriterleri dikkate alınmadan geliştirilen yazılımlar, firmaları ve yazılımları satın almak isteyen müşterileri zarara uğratmaktadır. Bu doğrultuda yazılım kalitesi farklı açılardan değerlendirilmelidir. Bunlar, geliştirici ekip, proje yönetimi ve yazılım ürününü satın alacak müşteridir. Müşteri açısından incelendiğinde, kaliteli bir yazılımın temel özellikleri işlevsel gereksinimlere uygunluk, kullanım kolaylığı, güvenilirlik (hata sıklığının az olması) ve güncelleme kolaylığıdır Yazılım ürünün kalitesini önceden bilmek müşterinin ürünü ile ilgili ortaya çıkacak sorunları önceden kestirmesine yardımcı olacaktır. Böylece müşteri pazardaki alternatif ürünlerden uygun olanı seçebilecektir. Geliştirici ve proje yönetimi açısından bakıldığında ise çıkabilecek hataları veya ilerde sorun yaratabilecek bileşenleri önceden tespit edebilmek, maliyeti azaltmak ve daha başarılı bir projeye imza atabilmek için önemli rol teşkil eder.

Yazılım projelerinde kalite halen soyut bir kavram olma özelliğini taşımaktadır; ancak yapılan çalışmalarla yazılım kalitesinin ölçülebilmesi için farklı metrikler geliştirilmiştir. Bu metrik değerleri ile birlikte yazılımın kalitesi somut değerlere çevrilebilmektedir [1][2]. Yazılım projelerinde kullanılan metrik ölçme araçları ile birlikte sayısal veriler elde edilebilmektedir. Ancak bu veriler üzerinden anlam çıkarabilmek ve çıkarılan bu anlamın doğruluğunu teyit edebilmek hem çaba hem de uzun zaman gerektirmektedir. Literatürde bu verilerden anlam çıkarmaya yönelik araştırmalar hâlihazırda mevcut olmasına karşılık yeni araştırmaların yapılmasına ihtiyaç duyulmaktadır. Bizim çalışmamız, proje yöneticilerine ve geliştiricilere nesneye dayalı geliştirilen yazılım projelerindeki kodlama safhasının nasıl ilerlediğinin gösterilmesini hedeflemektedir.

Çalışmada, projenin başında belirlenen kalite özelliklerine uygun olan metrik kümeleri belirlenmiştir. Karar ağacı ve NaiveBayes yöntemi ile projenin gelişimi aşamasındaki sürümlerden oluşturulan veri kümesi ile projenin ilerleyen aşamalarında belirlenen kalite özelliklerine uygun olup olmadığı belirlenmiştir. Proje geliştiricileri ile yapılan görüşmelerde sınıfların bakım kolaylığı ve test edilebilirlikleri için yapılan tahminler değerlendirilmiş çıkan sonuçların geliştiricilerden alınan geribildirimlere uygun olduğu gözlenmiştir.

1.1 Tezin Amacı

Yazılım projelerinde iyileştirmeler proje yaşam döngüsünde ne kadar erken yapılırsa proje yönetimi kriterleri açısından daha başarılı sonuçlar elde edilmiş olur. Tezde amaçlanan, proje yöneticilerine ve geliştiricilere projenin ilerleyişi sırasında projenin belirlenen kalite kriterlerini ne kadar sağladığı hakkında fikir vermektir.

Diğer bir deyişle çalışmamızda amaçlanan, yazılımların sürümleri kullanılarak yazılımın sınıflarının beraberinde projenin bütünün ilerlemesi hakkında ISO/IEC 25010:2011[28] kalite modelinde belirlenen bakım kolaylığı kriteleri doğrultusunda karar verilip verilemeyeceğinin bulunmasıdır.

Sınıfların zamanla nasıl değiştiğinin takip edilebilmesi, proje yürütücüleri için projenin zaman içinde ne şekilde ilerlediğinin izlenebilmesi, geliştiriciler için sınıflarla ilgili ortaya çıkabilecek sorunların önceden tahmin edilmesi açısından önem taşımaktadır. Bu kapsamda yazılım kalitesini ölçmeye dayalı metrikler ve istatistiksel analizler kullanılarak yazılım sınıflarının kalitesine ait değerlendirmeler yapılmıştır. Değerlendirmeler neticesinde elde edilen veriler kullanılarak hazırlanan veri seti, karar ağacı ve Naive Bayes yöntemlerine tabi tutulmuş ve ilerleyen sürümlerde bakım kolaylığının nasıl değişebileceğinin tahmin edilmesi sağlanmıştır.

Çalışmada nesneye dayalı metriklerin yanında, Durumsal liderlik modeli baz alınarak çalışanların yetkinlikleri, sınıfların ait oldukları paketlerde çıkan hatalar ve bu hataların çözülme süreleri kullanılmıştır.

Yapılan deneysel çalışmalarda bunların etkileri araştırılarak yazılımın belirlenen kalite özelliğine ne kadar etki ettiği değerlendirilmiş, yazılım ekibinden bilgiler toplanmıştır.

Bu çalışmanın neticesinde, projenin önceki sürümlerine ait sınıf bilgilerinin kullanılması ile projenin bakım kolaylığı alt özelliklerinin gelecek sürümlerinde nasıl değişeceğinin tahmin edilmesi sağlandı.

1.2 Tezin Katkıları

Çalışmamızda, nesnye dayalı tasarım metriklerininbir yazılım projesinin bakım kolaylığının tespitinde kullanılıp kullanılmayacağını araştırdık. Bu değerlendirmeyi yapabilmek için gerekli olan uygun metrik kümesi yaşayan projeler incelerek seçildi. Bakım kolaylığının önceden tespit edilip bu doğrultuda proje devam edilmesi projelerin maliyetlerinde azalma, çalışan performanslarında artmaya sebep olmaktadır. Çünkü proje yaşayan bir olgudur ve sürekli yeni istekler, düzeltilmesi gereken yerler ortaya çıkmaktadır. Eğer yazılım projenin başından itibaren bu bakım kolaylığına sahip olacak şekilde ilerlerse müşteri, çalışan ve yönetici memnuniyeti artacaktır.

Projede nesneye dayalı metriklerin yanında çalışanlar yetkinlikleri, sınıfların bulunduğu pakette çıkan hata sayısı, hataların giderilme süreleri eklenmiştir. Çalışanların yetkinlikleri durumsal liderlik baz alınarak yöneticilerle yapılan görüşmeler neticesinde eklenmiştir. Bunun yanında kullanılan proje yönetim portalından yararlanılarak hata sayıları ve hataların giderilme süreleri ele alınmıştır.

Çalışanlarla yapılan görüşmelerde sürekli değerlendirme ve ölçme yapılması ile çalışanların da daha özenli davrandıkları, bunun proje üzerinde olumlu etkiler yarattığı gözlenmiştir.

1.3 Literatür Araştırması

Bu bölümde, hata eğilimli sınıfların bulunması ve veri madenciliği tekniklerinin yazılım ölçmede kullanılmasına ilişkin literatürdeki araştırmalar incelenmiştir. Karar ağaçları, yapay sinir ağları ve destek vektör makineleri hata eğilimli sınıfları bulmakta kullanılmıştır[3]. Burada kullanılan model ve veri setinin hata eğilimli sınıfları bulmakta performansı etkilediği belirlenmiştir [4][5].

Li ve Leung tarafından yapılan çalışmada, hataya yatkınlığın bulunması için denetimsiz öğrenme modelinin geliştirilmesi amaçlanmaktadır [3].

Çalışmadaki çıkış noktası, aynı metrik kümelerinde bulunan bileşenlerin benzer hata yatkınlıklarının olmasıdır.

Kullanılan veri seti NASA'dan temin edilen 12 farklı projeye ait hata kayıtlarını ve kaynak kodları içermektedir. Elde edilen veriler üzerinde ön işleme yapılarak veriler normalleştirilmiş ve metrik değerleri hesaplanmıştır.

En Yakın Komşu algoritması kullanılarak oluşturulan model ile hata yatkınlıklarının bulunması hedeflenmiş ve %70 başarı elde etmişlerdir.

J.Osbeck, Waverly tarafından yapılan çalışmada, veri madenciliği teknikleri kullanılarak yazılımların kalitesinin önceden tahmin edilebilmesini sağlamak amacı ile yapılmıştır[6].Bu çalışmayı yapan araştırmacılar daha önceki çalışmalarında C4.5 algoritmasını [7]kullanarak yaptıkları çalışmayı J48 [8] algoritmasını kullanarak yapmışlardır.

Projenin kalitesinin tahmin edilmesi için QMOOD (Quality Model for Object Oriented Design) [9] model araştırmacılar tarafından tercih edilmiştir.Burada QMOOD modelinin yapısında bulunan "Çok Biçimlilik, Karmaşıklık, Kalıtım vb." için "İyi, Kötü, Çok İyi" şeklinde tanımlayıcılar atanmış ve Weka açık kaynak kodlu veri madenciliği aracı kullanılarak tahminler yapmaya çalışılmıştır.

Tahminlerin %75 başarılı olduğu gözlemlenmiştir. Bu çalışmada yazılım ürününün sürümleri arasında bir kıyaslama yapılmamakta sadece tek bir sürüm üzerinde inceleme yapılmaktadır. Makine öğrenmesi ve veri madenciliği teknikleri kullanılarak yapılan benzer bir çalışma, kodda programlama kurallarının çıkarılması, kopyala-yapıştır kısımların ve API kullanımlarının araştırıldığı çalışmadır [10]. Bu çalışmanın amacı, kopyala- yapıştır ve API kullanımının bir arada bulunduğu bir yöntem bulmaktır. Çünkü araştırmacılara göre kopyala-yapıştır ile yazılan kodlarda hatalının görülme olasılığı fazladır ve hata tüm kodlarda olacağından bu tarz kodların bakımı zordur.

Veri madenciliği teknikleri dışında bulanık mantık gibi yapay zeka teknikleri de son dönemdeki araştırmalarda kullanılmıştır.

Aljahdali ve Sheta yaptığı çalışmada yazılımın geçmiş hatalarına bakılarak projede çıkabilecek hataları bakım safhasına geçmeden tahmin edilmesi amaçlayan model (Software Reliability Growth Model) bulanık mantık kullanılarak geliştirilmiştir.

Bu modelin oluşturulmasında 16 projenin hata kayıtları incelenmiş, testi için ise gerçek zamanlı, askeri ve işletim sistemi olmak üzere üç proje kullanılmıştır [11].

Benzer bir çalışma yazılımcıların ISO/IEC 9126 kalite modelindeki anlaşılabilirliğin (intelligibility) bulunmasını makine öğrenmesi kullanarak bulunmasını amaçlayan çalışmadır. Makine öğrenmesi tekniklerinden olan Bayes Ağları, Olay Tabanlı Öğrenme, Yapay Sinir Ağlarından Weka açık kaynak kodlu yazılım aracılığı ile bu çalışmada yararlanılmıştır.

Lounis ve arkadaşlarının yaptıkları bu çalışmadaki asıl amaç aynı metriklerin farklı modellerle nasıl sonuç verdiğinin incelenmesi ve anlaşılabilirliğe olan etkilerinin araştırılmasıdır [12]. Yapılan diğer bir çalışma test adımlarından biri olan geleneksel kod gözden geçirmelerinin yerine makine öğrenmesi tekniklerinin kullanılmasının önerildiği çalışmadır [13]. Nesneye dayalı programlarda sınıf değişimlerinin önceden tahmin edilmesinde makine öğrenmesinin kullanıldığı çalışmalar yapılmıştır [14].

Bu çalışmada kaynak kod yerine tersine mühendislik yapılarak çıkarılan UML diyagramlar kullanılmıştır. Amaç, sürümde sınıfın ne kadar değişeceğini bulmaktır. Bunun için JFlex (The Fast Scanner Generator for Java) isimli 58 sınıflı açık kaynak kodlu proje incelenmiştir.

Metriklerin gruplamasına yönelik çalışmalar da literatürde mevcuttur. Metriklerin gruplanması ve farklı projelerin sonuçlarının incelenmesine yönelik çalışmalardan biri C++' da nesneye dayalı metriklerin incelenmesine yöneliktir [15].Bu çalışmada iki tane proje nesneye dayalı metrikler aracılığı ile karşılaştırılmıştır.

Karşılaştırılan projelerden biri bir kolejin 9 sınıftan oluşan kütüphane yönetim sistemi, diğeri 8 sınıftan oluşan grafik editör yazılımıdır. Hata bulmaya yönelik bir çalışma 2 milyon satırlık kodun test verileri ile yapılmıştır [16].Bu çalışmada amaçlanan, müşteriye teslim edilmeden önce yapılan sağlamlık ve kararlılık testlerinde bulunan hatalı verilerin kullanılarak bundan sonra oluşacak hataların önceden tahmin edilmesini sağlamaktır.

Araştırmacıların uzun süren testlerden önce hatayı kısa sürede tahmin eden bir model geliştirmek için poison dağılımından yararlanmışlar ve başarılı sonuçlar elde etmişlerdir.

Geçmiş verilerin yazılım kalitesinin belirlenmesinde kullanıldığı bir diğer çalışma ise programda bulunan kalıpların hata bulmada kullanıldığı araştırmadır [17]. Bu araştırmada hata kodları ve hata raporlarının girilerek gelecekte oluşacak hatalı kodların program kalıpları ile otomatik olarak bulunabileceği savunulmaktadır.

2. YAZILIM KALİTESİ

2.1 Yazılımda Kalite Kavramı

Kalite (Qualites) Latince "nasıl oluştuğu" anlamına gelen "qualis" kelimesinden gelmektedir. Buna göre kalite hangi ürün veya hizmet için kullanılıyorsa, onun ne olduğunu ifade etmeğe yöneliktir.[23]

1960'lı yıllardan itibaren kalite anlayışı değişerek daha insan odaklı olmaya başlamıştır. Kalitenin sadece üretim kontrolüyle görevli kişilerin değil tüm çalışanların sorumluluğunda olduğu ön plana çıkarılmıştır.Bununla birlikte kalite üzerine değişik bakış açıları geliştirmiş ve farklı ekoller tarafından farklı kalite tanımları yapılmıştır. Deming, 1986 yılında kaliteyi "ihtiyaçları karşılama yeteneği" olarak tanımlamıştır. Philip B. Crosby'nin bakış açısına göre ise kalite; sistemin şartlara ve talebe uygunluğudur. Bu kavramları baz alarak yazılım projeleri açısından kaliteye bakıldığında,kalitenin daha soyut bir kavram olarak karşımıza çıktığını görürüz.

Yazılım projelerinin temel hedefi, müşteri gereksinimlerini karşılayan, öngörülmüş bütçeyi aşmayacak şekilde zamanında teslim edilen hatasız ürün geliştirmektir. Bu yüzden yazılım projeleri bünyelerinde ürün olma özelliklerini, süreç olma özelliklerini ve ayrıca geliştirme aşamasında birbirinden farklı birçok disiplini (teknoloji yönetimi, proje yönetimi, süreç yönetimi, kalite yönetimi, insan kaynakları yönetimi vb.) kullandığından kalite arayışı bulunmaktadır. [23]

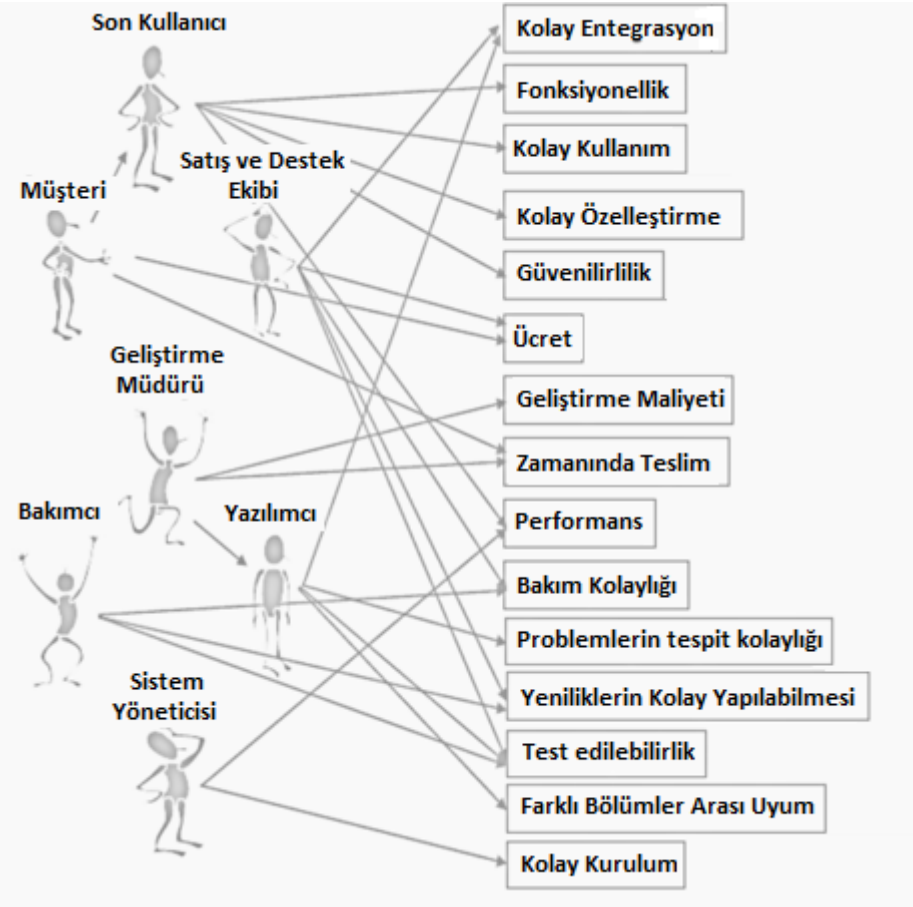
Yazılım projelerinde kalite kavramı ürün geliştirme hatlarındaki kalite anlayışına oranla daha soyut kalmaktadır ve kalite kavramı projedeki taraflara göre değişmektedir.

Müşteri açısından yaklaşıldığında, kaliteli bir yazılımın temel özellikleri işlevsel gereksinimlere uygunluk, kullanım kolaylığı, güvenilirlik (hata sıklığının az olması) ve güncelleme kolaylığıdır.[29] Yazılım ürünün kalitesini önceden bilmek müşterinin ürünü ile ilgili ortaya çıkacak sorunları önceden kestirmesine yardımcı olacaktır. Böylece müşteri pazardaki alternatif ürünlerden uygun olanı seçebilecektir.

Geliştiriciaçısından yaklaşıldığında, kolay genişletilebilir, kod okunabilirliği olan ve bakımı kolay bir yazılım daha kaliteli olarak nitelendirilebilir. Örneğin,

Baldassari ve arkadaşları tarafından yazılım kalitesi "Geliştiricinin bakış açısıyla ya da içsel bakış açısı ile kalite, maliyet ve gecikmenin doğru tahminine, daha kolay teste, daha iyi bakıma giden yol" olarak tanımlanmıştır [18]

Proje yönetimi açısından yaklaşıldığında ise çıkabilecek hataları veya ilerde sorun yaratabilecek bileşenleri önceden tespit edebilmek, maliyeti azaltmak ve daha başarılı bir projeye imza atabilmek için önemli rol teşkil eder.



Şekil 2.1: Yazılım projesi taraflarına göre kalite beklentisi[19]

Şekil 2.1'de yazılım projesinden tarafların beklentisinin özet şekli gösterilmiştir.[19]

Her ne kadar yazılım projelerinde kalite kavramı net bir tanım bulmasa da kaliteyi iyileştirmek için çalışmalar yapılmaktadır. Yazılım projelerinde kaliteyi iyileştirmek için yapılması gereken sürecin doğru belirlenmesidir. Yazılım kalitesinin iyileştirilmesi için projeye uygun standartlar seçilmeli, süreç boyunca ölçümler yapılarak bu sonuçlara uygun şekilde iyileştirmeler yapılmalıdır.

Yazılım geliştirme faaliyetlerinde süreç çok önemli bir kavram taşımaktadır. Süreç, amacı bir iş yapış biçiminde standart oluşturmak; değişkenliği azaltmak ve bu şekilde iş yapış biçiminde iyileşme sağlamak olan bir iş yapma yöntemidir. Yazılımın kalitesinde sürecin önemi yıllar önce ortaya çıkmış ve bu amaçla ISO 9001 ile başlayan süreç ISO/IEC 12207, ISO/IEC 15504 (SPICE) gibi modellerle ilerlemiştir. Günümüzde ise, tüm bu modellerin yanında Carnegie Mellon Üniversitesi bünyesinde faaliyet gösteren Yazılım Mühendisliği Enstitüsü'nün (Software Engineering Institute - SEI) ortaya koyduğu CMMI (Capability Maturity Model Integration) genel kabul görmüş durumdadır.

2.2 Yazılımda Ölçme

Yazılım projelerinde kalite kavramı halen soyut bir kavram olma özelliği taşımaktadır; ancak literatürde yazılımın kalitesini ölçmek için geliştirilen yöntemler bulunmaktadır ve yazılımda ölçüm yöntemlerinin kullanılması, yazılım sektöründe gittikçe önem kazanır olmuştur.

Kurumlar üç ana amaçla yazılımda ölçümü gündemlerine almaktadırlar:

- Yazılım projesini anlamak ve modellemek,
- Yazılım projelerinin yönetilmesine yol göstermek,
- Yazılım süreç geliştirme ve iyileştirme çalışmalarını yön vermek,

Tom DeMarco,"Ölçülemeyen şeyler denetlenemezler" demiştir.[20] Eğer yazılım projesinin kalitesini sayısal değer ile ilişkilendiremezsek projenin başarılı olup olmadığını bilemeyiz. Hatta sayısallaştırma işlemini projenin ilerlemesi aşamasında yapılması projenin maliyetleri, müşteri memnuniyeti, çalışanların verimi açısından daha başarılı sonuçlar elde edilebilir.

Projelerin gelişim sürecinde düzeltmeler yapmak oldukça zordur. 2004 yılında, Standish Group International'ın yapmış olduğu bir çalışma, yazılım projelerinin %53'ünün gecikmiş ya da bütçesini aşmış, %18'inin tamamlanamamış, ya da değiştirilmiş olduğunu ortaya koymuştur.

Projelerin yalnızca %29'u zamanında ve ayrılan bütçesine uygun tamamlanmıştır. Yazılım projelerinde iyi yönetim başarısızlık olasılığını azaltmak için bir anahtardır.[21]

Ölçme (measurement), çeşitli ölçütlerle (metrics) ve bu ölçütleri içeren modellerle sürekli olarak yapılan ve geliştirilen bir süreçtir. Yazılım mühendisliğinde de diğer mühendislik dallarında olduğu gibi ölçmenin önemi yadsınamaz.

Yazılımın ölçülmesi, belirlenen kalite özelliklerinin yazılım yaşam döngüsü boyunca karşılanıp karşılanmadığının belirlenmesidir. Bu şekilde yazılımın ölçümü kişilere bağımlı olmak yerine daha somut değerlendirmelerle yapılabilmektedir.[22]

Yazılımda ölçmenin hedefleri ve faydaları aşağıdaki gibidir;[22]

- Yazılım projesinin kalite düzeyini belirler.
- Hataların önlenmesine olanak sağlar.
- Proje yöneticilerinin süreci değerlendirmesine olanak sağlar.
- Müşterilerin ürün hakkında somut bir fikri olmasını sağlar.
- Yazılım firmalarına kendini iyileştirme olanağı sunar.
- Proje yöneticilerinin çalışanlarını analiz etmeye ve iş dağıtımını buna göre yaparak verimi arttırmasına olanak sağlar.

ISO/IEC 15939, yazılım endüstrisinde ölçüm sürecine yönelik etkinlik ve işleri tanımlayan uluslararası bir standarttır [24]. ISO/IEC 15939 standardına göre ölçüm süreci bilgiye gereksinimi olan kişiler veya firma tarafından yapılmaktadır.

Ölçüm süreci, işletmenin bilgi gereksinimi ile ilgilidir ve her gereksinim sonucunda bilgi üretilmekte ve bu bilgi de işletmenin karar süreciyle ilgili gereksinimi karşılamaktadır. Model yinelemeli dört ana süreçten oluşmaktadır. Şekil 2.2'de sürece ait özet bilgi verilmiştir.

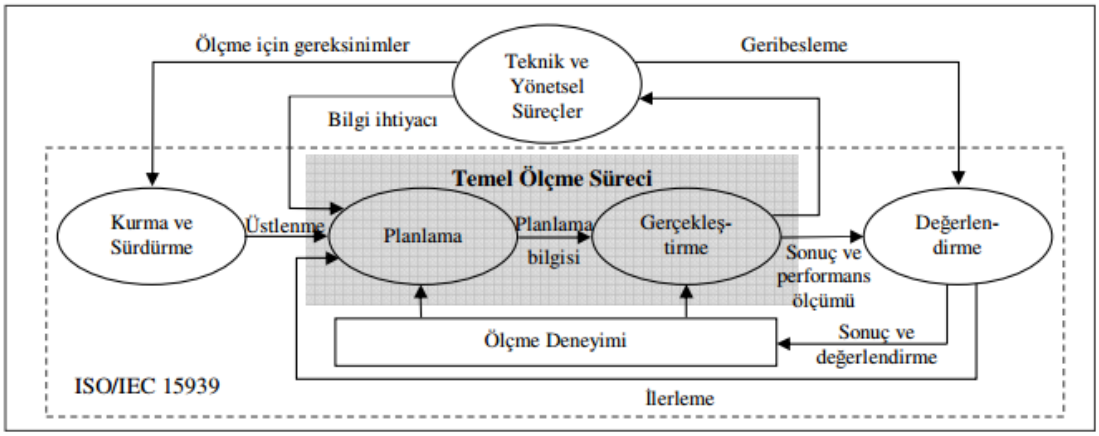
Bu süreçler:

1. Ölçmenin kurulması ve sürdürülmesi. Bu süreçte, ölçmenin amacının tanımlanması ve yönetim isteklerine göre sorumlulukların tespit edilmesi daha sonra da ölçmeye ilişkin görevlerin ve kaynakların atanması yer almaktadır.

2. Ölçme sürecinin planlanması. Bu süreçte hangi bilgilere gereksinim duyulacağının tespit edilmesi, bunların önceliklendirilmesi, ölçümün tanımlanması, ölçütlerin seçimi, değerlendirme özelliklerinin saptanması, ölçmenin planlanması ve gerekli araçların elde dilmesi ve kurulması bulunmaktadır.

3. Ölçme sürecinin gerçekleştirilmesi. Bu süreç, verilerin toplanması, sürecin ve verilerin doğrulanması, verilerin analizi ve anlamlandırılması, analiz sonuçlarının dokümante edilmesi ve faydalanıcılarla olan iletişimi içermektedir.

4. Sonuçların değerlendirilmesi. Ölçme sürecinin verimliliğinin ve ölçütlerin değerlendirilmesi, yapılmaktadır.



Şekil 2.2 :ISO/EIC 15939 Ölçme Süreci Modeli[24]

2.3 Kalite Modeli

Yazılım kalitesinin, yazılım projelerini etkilemesi sebebi ile yazılım kalitesini ölçmeye yönelik kalite modelleri oluşmuştur.

Kalite modelleri yazılım ürününün kalitesini ya da kodun kalitesini ölçmek ve değerlendirmek için oluşturulmuştur.

Örneğin, literatürde McCall, ISO/IEC 9126, Dromey, Bansiya ve NASA'nın Yazılım Güvence Teknoloji Merkezi (The Software Assurance Technology Center's-SATC) tarafından geliştirilen yazılım kalite modelleri mevcuttur.

McConnell'ın geliştirdiği kalite modeline göre, yazılım kalite sisteminde yazılım kalite ölçütleri ikiye ayrılmaktadır.

Bu modele göre; yazılımın kalitesini, hem yazılımı geliştiren kişiler ve yazılım firmaları hem de yazılımı kullanan kişiler açısından değerlendirilmeli ve yazılımın kalitesini artırmak için çaba gösterilmelidir.

- **Dış Kalite ölçütleri**

  Yazılım kullanıcılarını ilgilendiren ölçütlerdir.

- **Doğruluk**(Correctness): Yazılımın hatalar içermemesi, gereksinimlerde belirtildiği şekilde çalışması.
- **Etkinlik**(Efficiency): Bellek ve işlemci gibi sistem kaynaklarının en az oranda kullanımı.
- **Güvenilirlik**(Reliability): Sistemin her koşulda istenildiği gibi çalışması, hatalar arasındaki ortalama zaman aralığının (MTBF) yüksek olması.
- **Güvenlik**(Security): İzinsiz ve yetkisiz işlemler mümkün olmamalı. Bütünlük(Integrity): Veriler ve işlemler arasındaki tutarlılığın korunması.
- **Uyarlanabilirlik**(Adaptability): Sistemin değişik uygulamalar veya ortamlarda kullanılabilmesi için mümkün olduğunca az değişiklik gerektirmesi.
- **Hassaslık** (Accuracy): Sistemin kendisinden beklenen işi mümkün olduğunca iyi yapabilmesi.
- **Sağlamlık**(Robustness): Aykırı girişlere veya güç çalışma ortamlarına karşılık sistemin çalışmayı sürdürebilmesi.
- **Kullanılabilirlik**(Usability): Yazılımın kolay kullanılabilir olması

- **İç kalite ölçütleri**

  Yazılımı geliştirenleri ilgilendiren ölçütler.

  - **Yeniden kullanılabilirlik**(Reusability): Sistemin parçalarının başka sistemlerde kullanılabilmesinin kolay olması.
  - **Bakım kolaylığı** (Maintainability): Yazılıma yeni yetenekler eklemenin, yazılımdaki hataları gidermenin veya yazılımın başarımını attırmanın mümkün olduğunca kolay olması.
  - **Esneklik**(Flexibility): Yazılımın orijinal olarak tasarlandığı uygulamanın dışında çalışabilmesi için gerekli olan değişikliklerin olduğunca az olması.
  - **Taşınabilirlik**(Portability): Yazılımın farklı donanım ve işletim sistemleri gibi değişik çalışma ortamlarına kolaylıkla aktarılabilmesi.

- **Okunabilirlik**(Readability): Kodun kaynak kodunun incelenmesinin kolay olması.
- **Analiz Edilebilirlik**(Understandablility): Yazılımın sistem, bileşen ve kod düzeylerinde anlaşılabilirliğinin mümkün olduğunca kolay olması. Okunabilirlik sadece kod düzeyinde anlaşılabilirliği sağlar.
- **Sınanabilirlik**(Testability): Sistemin istenen gereksinimleri karşılayıp karşılamadığının sınanabilmesinin bileşen ve tüm sistem çapında mümkün olduğunca kolay olması.

Farklı bir kalite modeli McCall kalite modeli olarak bilinmektedir.

Bu modelde yazılımcı ile kullanıcı arasındaki ile anlaşmayı kolaylaştırmayı amaçlar. Bu modelin felsefesine göre tüm etkenlerin bir araya gelmesi ile yazılım kalitesinin resminin ortaya konulacağıdır.

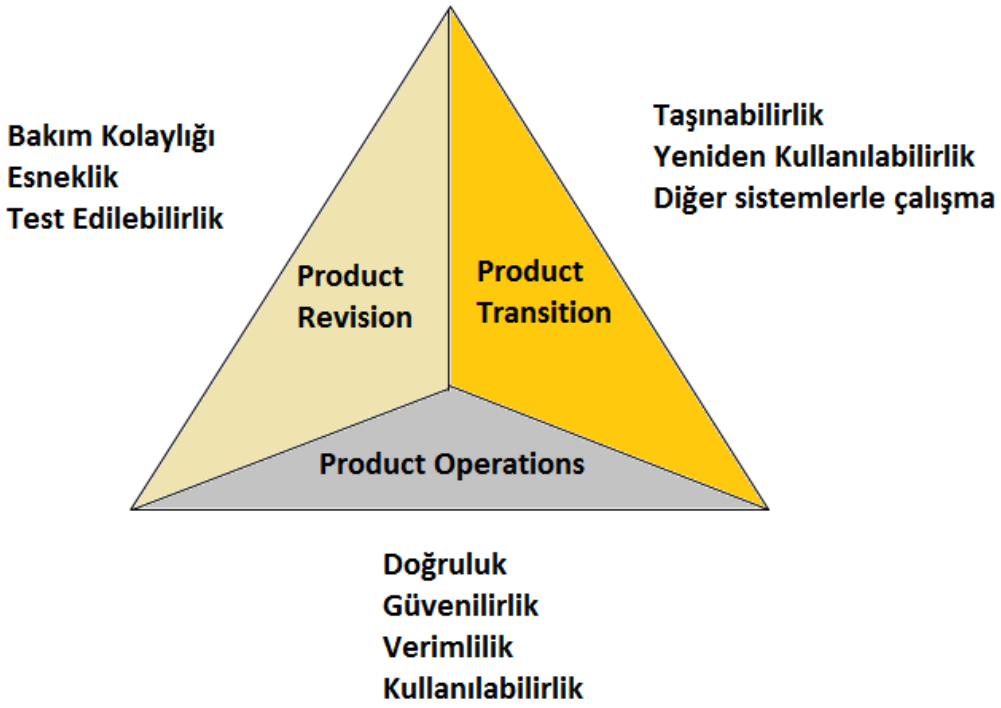
Yazılımı üç temel bakış açısı ile ele alır.Şekil 2.3'te McCall kalite modelinin bakış açısının özeti verilmiştir. McCall kalite modeli genelden özele ilerleyen bir kalite modelidir.

Her bakış açısına göre yazılım kalitesini belirleyen etkenler(faktörler) vardır. Bu etkenler kullanıcı tarafından gözlemlenen bilen durumlardır. Bu etkenleri belirleyen iç kalite özelliklerine bu modelde kriter denilmiştir.

Kriterler, yazılımın içeriden görünen özellikleridir. Kriterleri ölçmek için tanımlanan somut verilere ise metrik denilmiştir. Bu modellere göre; kriterlerden etkenlere(faktörler), etkenlerden de kaliteye geçilmektedir.

Modelde genelden özele bir yaklaşım belirlenmiştir. Etkenler, etkenleri oluşturan kriterler ve kriterlerin değerlendirilebilmesi için gerekli metrikler belirlenmiştir.

Hiyerarşik model türü olan Faktör/Kriter/Metrik olarak adlandırılan bu modelin özet şekli Şekil 2.4'te verilmiştir.



Şekil 2.3 : McCall Kalite Modeli[27]

McCall kalite modelinde bakış açılarına ait detayları 3 başlıkta toplayabiliriz. Bunlar;

- Product revision: Yazılımın bakımının yapılabilmesi, değişen isteklere göre güncellebilmesi.

  Faktörler:

  - Bakım Kolaylığı
  - Esneklik
  - Test Edilebilirlik

- Product transition: Yazılımın yeni ortamlara taşınabilmesi, eski modullerin yeni yazılımlarla kullabilmesi

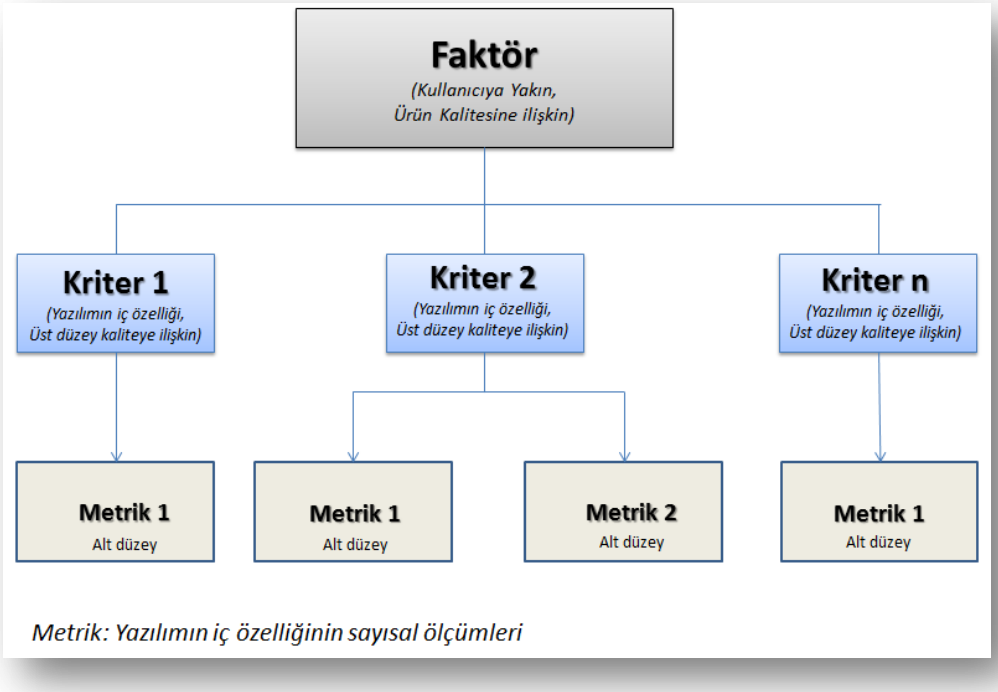
Faktörler:

- Taşınabilirlik
- Yeniden Kullanılabilirlik
- Başka sistemlerle birlikte biçimde çalışabilme

- Product operation:Yazılımın kullanılması sırasındaki kalitesi ile ilgilenir.

Faktörler:

- Doğruluk
- Güvenilirlik
- Verimlilik
- Kullanılabilirlik



Şekil 2.4 : Hiyerarşik Faktör/Kriter/Metrik Modeli[24]

ISO/IEC 25010:2011yazılım kalitesinin değerlendirilmesi için gereken bileşenleri içeren uluslararası bir standarttır. ISO/IEC 25010:2011 [28] standardına göre yazılım kalitesini modellemekte içsel kalite özellikleri ve dışsal kalite özellikleri (kullanım kalitesi) kullanılmaktadır.

Bu modelde kullanılan kriterler uygunluk, tutarlılık, değiştirilebilirlik, uyarlanabilirlik, verimlilik, bakım kolaylığı, güvenilirlik, olgunluk gibi kriterlerdir. Yazımın kalitesi tüm kalite kriterlerinin birleşiminden oluşmaktadır.

Standartta yazılım kalitesinin tanımlanması için altı ürün karakteristiğini ve bunlara bağlı alt karakteristiklerikapsamaktadır.

Çizelge 2.1'de ISO/IEC 25010:2011'ya ait kalite karakteristikleri verilmiştir.

Çizelge 2.1 : Yazılım projesi taraflarına göre kalite beklentisi

Karakteristikler	Alt karakteristikler
Fonksiyonellik	Uygunluk, doğruluk, karşılıklı işlerlik, uyum, güvenlik
Güvenilirlik	Olgunluk, hata toleransı, kurtarılabilirlik
Kullanılabilirlik	Analiz Edilebilirlik, öğrenebilirlik, işlerlik
Verimlilik	Zaman davranışı, kaynak yararlanımı
Bakım yeteneği ve korunabilirlik	Analiz edilebilirlik, yeniden kullanılabilirlik, durağanlık, test edilebilirlik, değiştirilebilirlik
Taşınabilirlik	Uyum yeteneği, kurulum kolaylığı, uygunluk, değiştirilebilirlik

- **Fonksiyonellik:** Yazılımın belirli şartlar altında, tanımlanan ihtiyaçları karşılamak için bir araya gelmiş olan özellikler olarak tanımlanmaktadır. Uygunluk, doğruluk, birlikte çalışabilirlik, uyum ve güvenlik konuları bu kategori altında incelenmektedir.
    - **Uygunluk:** Belirlenmiş fonksiyonlar setini yazılımın sahip olması ve buna fonksiyonlar setine uygunluk becerisidir.
    - **Doğruluk:** Yazılımın belirtilmiş gereksinimleri karşılaması
    - Karşılıklı işlerlilik: Yazılımın belirlenmiş sistemlerle etkileşim içinde olabilme becerisidir.
    - **Güvenlik:** Hatayla veya kasten programlara ve verilere yetkilendirilmemiş erişimi önleyebilme özelliğidir.
    - **Uyum:** Yazılımın uygulama ile ilişkili standartlara, anlaşmalara veya kanuni düzenlemelere ve dengi talimatlara uygun olması becerisidir.
- **Güvenilirlik:** Yazılımın belirlenen şartlar altında, düzgün çalışma halini muhafaza edebilmesi olarak tanımlanmaktadır. Olgunluk, hata toleransı ve geri kurtarma konuları bu kategori altında incelenmektedir.

- **Olgunluk:** Yazılımdaki hatalar nedeniyle yazılımın çökme frekans sıklığının düşük olması becerisidir.
- **Hata toleransı:** Yazılımdaki hatalar nedeniyle yazılımın çökme frekans sıklığının düşük olması becerisidir.
- **Kurtarılabilirlik:** Sistemin göçmesi durumunda doğrudan etkilenen verinin geri yüklenebilme ve performans seviyesinin yeniden oluşturulabilme becerisidir.

- **Kullanılabilirlik:** Yazılımın kullanım kolaylığısağlamak için kolay öğrenilebilme, anlaşılabilme özelliklerini ifade etmektedir. Öğrenebilme, anlaşılabilirdik, işletilebilirdik ve kullanıcı etkileşimi konuları bu kategori altında incelenmektedir.
  - **Analiz Edilebilirlik:** Yazılımın mantıksal yapısının ve onun uygulanabilirliğinin kullanıcı tarafından tanınabilmesi için harcanması gereken kullanıcı çabası ile ilgili yazılım becerisidir.
  - **Öğrenebilirlik:** Yazılım uygulamalarının kullanıcı tarafından öğrenilebilmesi için harcanması gereken kullanıcı çabası ile ilgili yazılım becerisidir.
  - **İşlerlik:** Operasyon ve operasyon kontrolü için gereken kullanıcı çabası ile ilgilenen yazılım becerisidir.
- **Verimlilik:** Yazılımın ihtiyaç duyulan ölçüde yeterli performansla çalışabilme becerisi olarak tanımlanmaktadır. Zaman ve kaynak kullanımı konuları bu kategori altında incelenmektedir.
  - **Zaman davranışı**:Üretilen iş oranı ve işlem zamanları fonksiyonunu yerine getirirken talebin yanıtlaması ile ilgili yazılım becerisidir.
  - **Kaynak yararlanımı:** Kaynak kullanımı ve kaynak kullanım süresinin fonksiyonları yerine getirirken kullanılan miktarı ile ilgilenen yazılım becerisidir.
- **Bakılabilirlik:** Yazılımın yeni isteklere uyum sağlama, değişiklik veya düzeltme yapmaya cevap verme yeteneği olarak tanımlanmaktadır. Değiştirilebilirdik, sınanabilirdik, analiz edilebilirlik ve bağışıklık konuları bu kategori altında incelenmektedir.

- Burada tanımlanan sınanabilirdik, yazılımın sınanabilmesi, yapılan değişikliklerindoğrulanabilmesi kabiliyetidir.
    - **Analiz edilebilirlik:** Tanı eksikliği ve çökme sebepleri veya düzenlenecek parçaların belirlenmesinde tanımlama için gereken çaba ile ilgilenen yazılım becerisidir.
    - **Yeniden kullanılabilirlik:** Yazılımın başka bir yazılım ortamında kullanılabilmesi için fırsat sağlama ve harcanması gereken çaba ile ilgili yazılım becerisidir.
    - **Durağanlık:** Modifikasyon sonucunda yazılımın beklenmeyen etki doğurması riski ile ilgilenen yazılım becerisidir.
    - **Test edilebilirlik:**Uyarlanan yazılımın onaylanması için gereken çaba ile ilgilenen yazılım becerisidir.
    - **Değiştirilebilirlik:** Hata giderme, uyarlama veya çevresel değişiklik için gereken çaba ile ilgilenen yazılım becerisidir.
- **Taşınabilirlik:** Yazılımın farklı çalışma ortamlarına uyum sağlayabilmeyeteneği olarak tanımlanmaktadır. Adaptasyon yeteneği, yüklenebilirdik özellikleri, ortam değiştirme imkânı ve diğer yazılımlarla uyum konuları bu kategori altında incelenmektedir.
    - **Uyum yeteneği:** Göz önüne alınan yazılım için belirlenen amacın sağlanabilmesinde değişiklik gerçekleştirmeden yazılımın belirli bir farklı çevrede adapte edilebilme becerisidir.
    - **Kurulum kolaylığı:** Tanımlanmış çevrede yazılımın kurulumu için gereken çaba ile ilgili yazılım becerisidir.
    - **Uygunluk:** Taşınabilirlik ile ilgili anlaşmalar veya standartlara bağlı yazılım geliştirilmesi ile ilgili yazılım becerisidir.

Yazılım ürününün yukarıda tanımlanan kalite karakteristikleri aynı zamanda ölçülebilir büyüklüklerdir. Bu özelliklerinden dolayı yazılımın ölçülmesinde kullanılan ölçütler arasındadırlar.

2.4 Yazılım Ölçütleri

Yazılım ürününün belirlenen kalite karakteristlikleri aynı zamanda ölçülebilir büyüklüklerdir. Yazılım ölçütleri konusunda yapılan çalışmalar, yazılım kalitesi, yazılım ölçümü ve yazılım kalite standartları ile yakından ilişkilidir. Yazılım ölçütleri Kan tarafından üç başlık altında sınıflandırılmaktadır.[29]

- Ürün ölçütleri; yazılımın büyüklüğü, yazılım tasarımı, performansı açısından ele alınır. Ürün ölçütleri genellikle ürün kalitesi ile ilişkili olup, ürünün kalite ölçütlerine ve müşteri gereksinimleri karşılanmasına göre iki seviyeyi içermektedir. Ürün büyüklüğü alanında ölçüm yapıldığında, veritabanı büyüklüğü, fonksiyonel değişim iş yükü, gereksinim büyüklüğü ürün ölçütlerine örnek olarak verilebilir. Benzer şekilde kaynak ve maliyetler alanında ölçüm yapıldığında, çaba, deneyim, maliyet, kaynak uygunluğu ürün ölçütlerine örnektir.
- Süreç ölçütleri; yazılımın geliştirilme sürecinde ve bakımının yapılamasında kullanılır. Hata ve eksikliklerin giderilmesinin yazılımın veriminin artırılmasında önemi olması süreç ölçütlerini ilgilendirir.
- Proje ölçütleri; proje yöneticileri açısından önemlidir. Projenin yöneticiler tarafından nasıl yönetileceğini ve projenin özelliklerini tanımlamaktadır.

2.5 Nesneye Dayalı Ölçütler

Nesne yönelimli metrikler, yazılım test sürecinin etkililiğinin önemli bir göstergesidir.

Goodman yazılım metriklerini [30]; "Ölçüm temelli tekniklerin yazılım geliştirme sürecine sürekli uygulanması, süreç ve sürece ait ürünlerinin gelişmesi için bu tekniklerin kullanılmasıyla birlikte, anlamlı ve zamanında bilgi yönetimini sağlamak" olarak tanımlar.

Geleneksel tasarımda kullanılan metrikler, nesneye dayalı programlarda doğrudan uygulanamamaktadır. Bu yüzden, nesneye dayalı yazılımların ölçülmesini sağlamak amacı ile yapısal programlarda kullanılan metriklerden farklı metrik kümeleri geliştirilmiştir.

Literatürde; Chidamber&Kemerer, Brito e Abreu (Metrics for Object Oriented Design-MOOD) ve Bansiya&Davis QMOOD (Quality Model for ObjectOriented Design) gibi kabul görmüş nesne yönelimli birçok yazılım metrik kümesi bulunmaktadır.[26]

- **Chidamber & Kemerer'in Nesne Yönelimli Metrik Kümesi** (Chidamber & Kemerer object-oriented set of metric)

  Nesneye yönelimli yazılımlarda kullanılan bu metrik grubu yazılımı oluşturan sınıfları değerlendirmeyi amaçlar. Sonuçlar sadece sınıf hakkında bilgi verdiğinden bu metrik kümesi ile yazılımın tümüne ait bir sonuç alınmamaktadır. Chidamber & Kemerer, nesne yönelimli tasarım için altı adet metrik tanımlar aşağıda verilmiştir.

  - **Sınıfın ağırlıklı metot sayısı** (Weighted Methods per Class - WMC);

  Bir sınıftaki metotların karmaşıklık derecesi veya sayısıdır. Bir sınıfın metotlarının karmaşıklığı ve sayısı, sınıfın geliştirilmesine ve bakımına harcanacak zaman-çaba hakkında fikir verir [26]. Bu metrik, bir sistemdeki sınıfların ortalaması hesaplanarak ölçülür [33]. WMC; nesne yönelimli bir yazılımın anlaşılırlığını, yeniden kullanılabilirliğini ve dayanıklılığını/bakılabilirliğini ölçmede kullanılır [31]. Bir sınıfın n adet metodu varsa ve ci sınıfın i'inci metodun karmaşıklığı ise WMC formülü Şekil 2.5'te verilmiştir.

$$WMC = \sum_{i=1}^{n} c_i$$

Şekil 2.5 : Sınıf ağırlıklı metot sayısı[29]

  - **Kalıtım ağacının derinliği** (Depth of Inheritance Tree - DIT)

Sınıfın kalıtım ağacının köküne uzaklığıdır [26]. Bu değerin çok yüksek olması test edilebilirliğin çok düşük olduğunu, aksi halde nesne yönelim ilkelerinin fazla kullanılmadığını gösterir.[32]Bu metrik; verimliliği, yeniden kullanımı, anlaşılırlığı ve test edilebilirliği ölçer [31].

  - **Alt sınıf sayısı** (Number of Children - NOC)

Bir sınıftan direk türetilmiş alt sınıfların sayısıdır.

NOC, kalıtımlı ifadeler dikkate alınarak hesaplanır. Bir sınıf hiyerarşisinin genişliğini ölçer. Alt sınıf sayısının fazla olması; yeniden kullanımın yüksek olduğunu, daha çok hatanın oluşabileceğini [34], test esnasında harcanacak zamanı-çabayı ve kalıtımın yanlış kullanıldığını gösterir. NOC; verimlilik, yeniden kullanılabilirlik ve test edilebilirlik düzeyini ölçer [31].

- **Nesne sınıfları arasındaki bağımlılık** (Coupling Between Object Classes - CBO)

  Bir sınıf içindeki özellik (attribute) ya da metotların (method) diğer sınıfta kullanılması ve sınıflar arasında kalıtımın olmaması durumunda iki sınıf arasında bağımlılıktan bahsedilebilir [34]. CBO; verimliliği ve yeniden kullanılabilirliği ölçmede kullanılır [31].

- **Sınıfın tetiklediği metot sayısı** (Response for a class - RFC)

  Bir sınıftan bir nesnenin metotları çağrılması durumunda, bu nesnenin tetikleyebileceği tüm metotların sayısı RFC değerini verir. Yani, bir sınıfta yazılan ve çağırılan toplam metot sayısıdır (9). Bu metrik; sınıf seviye tasarım metriklerinden olup [35]; anlaşılabilirliği, dayanıklılığı, karmaşıklığı ve test edilebilirliği ölçmede kullanılır [31].

- **Metotlardaki uyum eksikliği** (Lack of cohesion in methods - LCOM)

  Uyumluluk nesne yönelimli programlamada oldukça önemli bir kavramdır. Bir sınıf elemanlarının birlikteliğinin derecesini gösterir. İyi bir tasarım yüksek uyumlulukta sınıflar sunmalıdır.

  Uyumluluk bir sınıfın birden fazla soyutlama yapmamasını gerektirir. Aksi durumda sınıfın başka sınıflara bölünmesi gereklidir.

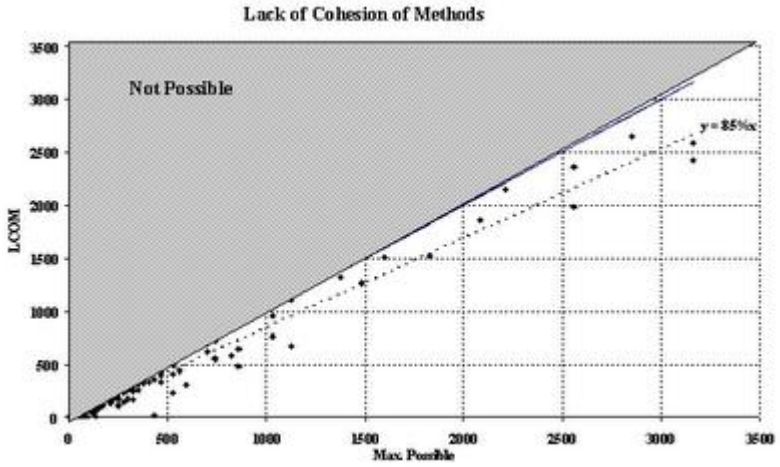
  Chidamber ve Kemerer'a göre sınıfın aynı değişkenlerine erişen birden fazla method varsa sınıfta bir uyumluluktan bahsedilebilir. LCOM bir sınıfın içindeki hiçbir ortak alanı paylaşmayan method toplamından, en az bir ortak alanı paylaşan method toplamının farkı ile bulunur.Bulunan değer 0'dan küçükse 0 kabul edilir.[34]

Methodların uyumsuzsuzluğu sınıfın iyi kapsulleme (encapsulation) sunamadığının göstergesidir. Uyumsuzluğu yüksek sınıfların iki ya da daha fazla alt sınıfa bölünmeleri gözden geçirilmelidir

Methodlar arası farklılık, sınıf tasarımlarındaki çatlakların bulunmasında yardımcı olur.

Uyumsuzluk karmaşıklığı arttırır ve geliştirme sırasında hata yapılma ihtimalini arttırır

LCOM değeri sınıfın method sayısına bağlı olarak değişir. COM değerinin alabileceği sınır değer sınıfın metot sayısıdır.Şekil 2.6'te sınıfların LCOM / NOM değerlerine göre dağılımı gösterilmiştir.



Şekil 2.6 : LCOM / NOM değerlerine göre dağılımı[29]

LCOM, metriği test ediciye; verimlilik ve yeniden kullanılabilirlik derecesi hakkında bilgi verir.

2.6 Veri Madenciliği Yöntemleri

Makine Öğrenmesi, verilen bir problemi probleme ait ortamdan edinilen veriye göre modelleyen bilgisayar algoritmalarının genel adıdır. [43]

Yoğun çalışılan bir konu olduğu için önerilmiş birçok yaklaşım ve algoritma mevcuttur. Bu yaklaşımların bir kısmı tahmin (prediction) ve kestirim (estimation) bir kısmı da sınıflandırma (classification) yapabilme yeteneğine sahiptir.

Tahmin (prediction): Veriden öğrenen modellerde sistem çıkışının nicel olması durumunda kullanılan yöntemlerin ürettiği değerlerdir.

Sınıflandırma (classification): Giriş verisine ait çıkışların nitel olduğu durumlarda kullanılan yöntemlerin her veri örneğinin hangi sınıfa ait olduğunu belirlemesidir.

Makine Öğrenmesi yöntemleri verinin yapısına göre ikiye ayrılır.

Danışmanlı (supervised) Öğrenme: Veri, etkiye tepki prensibiyle çalışan sistemlerden alınır ve giriş-çıkış düzeninde organize edilir.

Danışmansız (unsupervised) Öğrenme: Sınıf bilgisi olmayan veya verilmeyen veri içerisindeki grupları keşfetmeyi hedefler.

Veriden eğitim için önerilen her algoritma farklı bir öğrenme kuralı olsa da genel eğilim algoritmaları ortak özelliklerine göre gruplamak yönündedir. Öğrenme algoritmaları dört grupta toplanabilir:

- Hebb

  1949'da geliştirilen ilk öğrenme kuralıdır. "Bir hücre, bağlı olduğu diğer hücreleri etkiler" prensibine dayalıdır. Bu öğrenme kuralı geliştirilerek farklı öğrenme kuralları geliştirilmiştir.

- Delta

  Beklenen sonuç ve hesaplanan sonuç arasındaki karesel fark, sistemin hatasıdır. Bu hatayı indirgemek için hücreler arasındaki bağlantılar sürekli değiştirilir. Çok katmanlı algılayıcı ağlar bu kurala göre eğitilirler.

- Hopfield

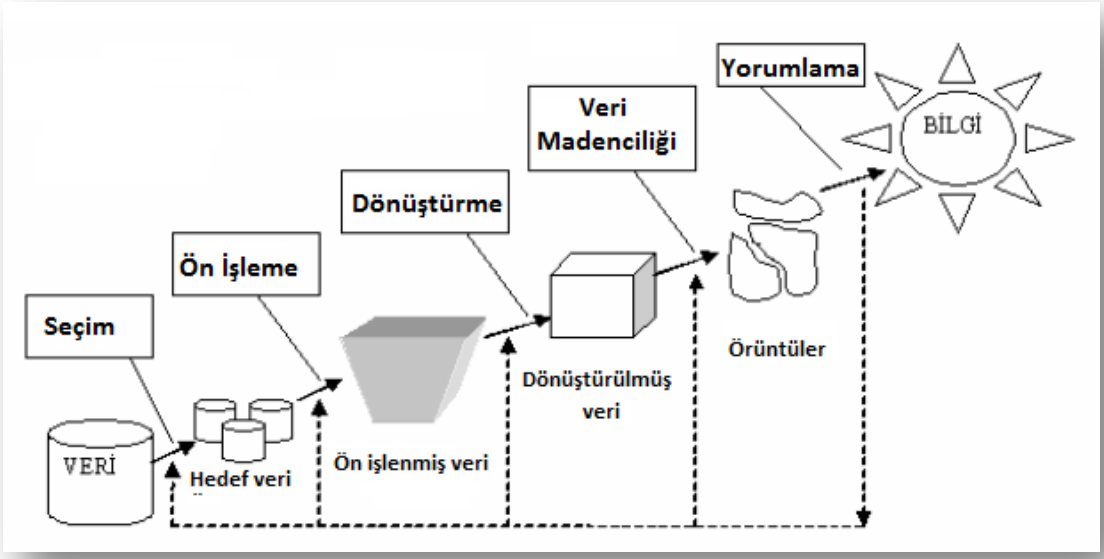
  Beklenen sonuç ile hesaplanan sonuç aynı ise hücreler arası bağlar belirli bir oranda güçlendirilir, aksi durumda bağlar zayıflatılır. Recurrent ve Elman ağları bu kural ile eğitilirler.

- Kohonen

  Bu danışmansız modelde hücreler yarış halindedir. En büyük sonucu üreten hücre yarışı kazanır. Kazanan hücrenin ve komşularının bağları güçlendirilir. ART (Adaptive Resonance Theory) ve Kohonen tarafından geliştirilen SOM (Self Organizing Map) ağları örnek olarak verilebilir.

Veri madenciliği için yapılan farklı tanımlardan bazıları şu şekildedir: Veri madenciliği, istatistiksel ve matematiksel teknikler ile örüntü tanıma teknolojilerinin kullanılarak, depolama ortamlarında sıkışmış bulunan büyük miktardaki verinin elenmesi ile anlamlı yeni korelasyon, örüntü ve eğilimlerin keşfedilmesi sürecidir [40]. Veri madenciliği, büyük miktardaki veriden, anlamlı örüntüler ve kurallar keşfetme sürecidir [41].

En basit tanımıyla veri madenciliği, veri içerisindeki yeni, gizli kalmış veya beklenmeyen örüntüleri bulmak için kullanılan faaliyetler bütünüdür [42]. Veri madenciliği, daha büyük bir süreç olarak adlandırılan bilgi keşfi sürecinin bir bölümüdür [43]. Şekil 2.7'da VTBK (Veri Transferi Bilgi Keşfi) süreci ve bu sürecin bir parçası olan veri madenciliğine yer verilmiştir.



Şekil 2.7 : Veri Transferi Bilgi Keşfi süreci[42]

Veri Madenciliği altı adımlı bir süreç olarak incelenebilir:

- Araştırma Probleminin Tanımlanması (Business Understanding)

  Bu aşama veri madenciliği sürecinin en önemli aşamasıdır. Araştırma probleminin (konusunun) tanımlanması aşaması araştırmanın amacını, mevcut durumun değerlendirilmesini, veri madenciliğinin amaçlarını ve proje planlama sürecinin belirlenmesini kapsamaktadır.

- Verileri Tanıma Aşaması (Data Understanding)

Veri anlama aşaması veri toplamakla başlamaktadır. Daha sonra benzer verileri bir araya getirme, veri niteliklerini tanımlama, verileri keşfetme, gizli bilgileri sınıflandırma ile sürece devam etmektedir.

- Veri Hazırlama Aşaması (Data Preperation)

Veri hazırlama aşaması, ham veriden başlayarak son veriye kadar yapılması gereken bütün düzenlemeleri içermektedir.

Veri hazırlama, tablo, kayıt, veri dönüşümü ve modelleme araçları için veri temizleme gibi özellikleri içermektedir.

- Modelleme Aşaması (Modelling)

Bu aşamada, verilerden bilgi çekmek için ileri çözümleme yöntemleri kullanıldığından veri madenciliği sürecinin en gösterişli aşamasıdır. Bu aşama uygun modelleme tekniğinin seçimi, test tasarımının üretimi, model geliştirme ve tahmin işlemlerini içermektedir. Uygun modellerin seçilip uygulanmasıyla birlikte parametreler en uygun değişkenlere dönüştürülmektedir. Veri madenciliği, her problem tipi için farklı yöntemler içermektedir. Bazı yöntemler, veri tipi için uygun değildir ya da özel tanımlamalar gerektirmektedir. Bu nedenle gerekli olduğunda 3. aşama olan veri hazırlama aşamasına geri dönülür.

- Değerlendirme Aşaması (Evaluation)

Değerlendirme aşamasında, uygun model ya da modeller kurulduktan sonra, veri madenciliği sonuçlarının araştırma probleminin amaçlarını gerçekleştirip gerçekleştirmediği değerlendirilir. Bu aşama sonuçların değerlendirilmesi, veri madenciliği sürecinin gözden geçirilmesi ve sonraki adımların ne olacağı hususlarını içermektedir. Bu aşamanın sonunda veri madenciliği sonuçlarının kullanımı üzerindeki karara varılmaktadır.

- Uygulama Aşaması (Deployment)

Son aşama olan uygulama aşaması, araştırmacının tüm emeklerinin karşılığını aldığı bir aşamadır. Bu aşamada veri Mmadenciliği süreciyle üretilen bilgiler, pratik işletme problemlerinin çözümünde kullanılmaktadır. Bu aşamada elde edilen bilgilerin uygulanabilmesine yönelik bir plan hazırlama, gözden geçirme ve bakım faaliyetlerini içerir. Ayrıca bu aşamada nihai araştırma raporunun yazılması ve projenin gözden geçirilmesi işlemleri yer almaktadır.

2.7 Karar Ağaçları

Karar ağaçları, sınıfları bilinen örnek veriden tümevarım yöntemiyle öğrenilen ağaç şekilli bir karar yapısı çeşididir. [36]

Bir karar ağacı, basit karar verme adımları uygulanarak, büyük miktarlardaki kayıtları, çok küçük kayıt gruplarına bölerek kullanılan bir yapıdır.

Her başarılı bölme işlemiyle, sonuç gruplarının üyeleri bir diğeriyle çok daha benzer hale gelmektedir.[37]

Büyük veri tabanlarının kullanıldığı pek çok sınıflama probleminde ve karmaşık ya da hata içeren bilgilerde karar ağaçları yararlı bir çözüm olmaktadır. [38] Tahmin edici ve tanımlayıcı özelliklere sahip olan karar ağaçları, veri madenciliğinde kuruluşlarının kolay olması, yorumlanmalarının kolay olması, veri tabanı sistemlerine kolayca entegre edilebilmeleri, güvenilirliklerinin daha iyi olması nedenleri ile sınıflama modelleri içerisinde en yaygın kullanıma sahip olan bir tekniktir.

Karar ağacı temelli analizlerin yaygın olarak kullanıldığı alanlar şunlardır:

- Belirli bir sınıfın olası üyesi olacak elemanların belirlenmesi,
- Çeşitli vakaların yüksek, orta, düşük risk grupları gibi çeşitli kategorilere ayrılması,
- Parametrik modellerin kurulmasında kullanılmak üzere çok sayıdaki değişkenden en önemlilerinin seçilmesi,
- Gelecekteki olayların tahmin edilebilmesi için kurallar oluşturulması, • Sadece belirli alt gruplara özgü olan ilişkilerin tanımlanması,
- Kategorilerin birleştirilmesi ve sürekli değişkenlerin kesikli değişkenlere dönüştürülmesidir.

Karar ağacı oluşturmak için geliştirilen bu algoritmalar arasında

- CHAID (Chi- Squared Automatic Interaction Detector),
- Exhaustive CHAID,
- CRT (Classification and Regression Trees),
- ID3,

- C4.5,
- MARS (Multivariate Adaptive Regression Splines),
- QUEST (Quick, Unbiased, Efficient Statistical Tree),
- C5.0,
- SLIQ (Supervised Learning in Quest),
- SPRINT (Scalable Paralleizable Induction of Decision Trees) yer almaktadır.

Karar ağaçları akış şemalarına benzeyen yapılardır. Her bir nitelik bir düğüm tarafından temsil edilir. Dallar ve yapraklar ağaç yapısının elemanlarıdır. En son yapı ”yaprak”, en üst yapı “kök” ve bunların arasında kalan yapı ise “dal” olarak adlandırılır. Karar ağaçları sınıflama algoritmasını uygulayabilmek için uygun bir alt yapı sağlamaktadır.

Karar ağacı oluşturmak için birçok yöntem geliştirilmiştir.

Bunlar temel olarak:

1. Entropiye dayalı algoritmalar

2. Sınıflandırma ve Resresyon araçları

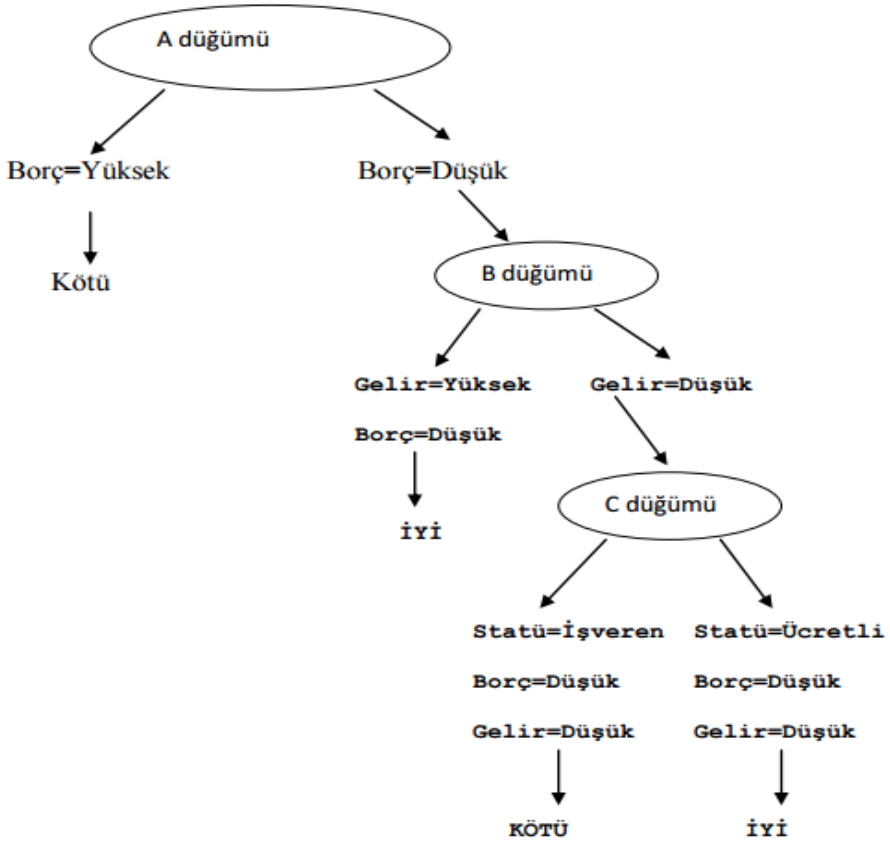
3. Bellek tabanlı sınıflandırma modelleri

Örnek:

Borç	Gelir	Statü	Risk
Yüksek	Yüksek	İşveren	Kötü
Yüksek	Yüksek	Ücretli	Kötü
Yüksek	Düşük	Ücretli	Kötü
Düşük	Düşük	Ücretli	İyi
Düşük	Düşük	İşveren	Kötü
Düşük	Yüksek	İşveren	İyi
Düşük	Yüksek	Ücretli	İyi
Düşük	Düşük	Ücretli	İyi
Düşük	Düşük	İşveren	Kötü
Düşük	Yüksek	İşveren	İyi

Şekil 2.8 : Örnek veri seti

Şekil 2.8'de verilen örnek veri setinden karar ağacı oluşturulur. Karar ağacı oluşturulduktan sonra karar kuralları oluşturulur.



Şekil 2.9 : Karar ağacı yapısı

Şekil 2.9'de oluşan karar ağacı verilmiştir. Karar ağacına uygun oluşan kurallar:
Kural.1:Borç Yüksek ise Risk Kötü

Kural.2:Borç Düşük ve Gelir=Yüksek ise Risk=İyi Kural.

3:Borç Düşük ve Gelir=Düşük ve Statü=İşveren ise Risk=Kötü Kural.

4:Borç Düşük ve Gelir=Düşük ve Statü=İşveren ise Risk=Kötü

2.8 Model Başarım Ölçütleri

Model başarımını değerlendirirken kullanılan temel kavramlar hata oranı, kesinlik, duyarlılık ve F-ölçütüdür. Modelin başarısı, doğru sınıfa atanan örnek sayısı ve yanlış sınıfa atılan örnek sayısı nicelikleriyle alakalıdır. Test sonucunda ulaşılan sonuçların başarım bilgileri karışıklık matrisi ile ifade edilebilir. Karışıklık matrisinde satırlar test kümesindeki örneklere ait gerçek sayıları, kolonlar ise modelin tahminle meşini ifade eder.Şekil 2.10'da karışıklık matrisine ait şekil verilmiştir.

		Öngörülen Sınıf	
		Sınıf=1	Sınıf=0
Doğru Sınıf	Sınıf=1	a	b
	Sınıf=0	c	d

a: TP (True Pozitif) c: FP (False Pozitif)
b: FN (False Negatif) d: TN (True Negatif)

Şekil 2.10 : Karışıklık Matrisi[39]

- Doğruluk – Hata oranı

Model başarımının ölçülmesinde kullanılan en popüler ve basit yöntem, modele ait doğruluk oranıdır. Doğru sınıflandırılmış örnek sayısının (TP +TN), toplam örnek sayısına (TP+TN+FP+FN) oranıdır. Hata oranı ise bu değerin 1'e tamlayanıdır. Diğer bir ifadeyle yanlış sınıflandırılmış örnek sayısının (FP+FN), toplam örnek sayısına (TP+TN+FP+FN) oranıdır.Şekil 2.11'da doğruluk ve hata oranı formülü verilmiştir.

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

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

Şekil 2.11 : Doğruluk - Hata Oranı[39]

- Kesinlik

Kesinlik, sınıfı 1 olarak tahminlenmiş True Pozitif örnek sayısının, sınıfı 1 olarak tahminlenmiş tüm örnek sayısına oranıdır.Şekil 2.12'de kesinlik hesaplamasına ait formül verilmiştir.

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

Şekil 2.12 : Kesinlik hesaplaması[39]

- Duyarlılık

Doğru sınıflandırılmış pozitif örnek sayısının toplam pozitif örnek sayısına oranıdır.Şekil 2.13' de duyarlılık hesaplamasına ait formül verilmiştir.

$$Duyarlılık = \frac{TP}{TP + FN}$$

Şekil 2.13 : Duyarlılık hesaplaması[39]

- F-Ölçütü

Kesinlik ve duyarlılık ölçütleri tek başına anlamlı bir karşılaştırma sonucu çıkarmamıza yeterli değildir. Her iki ölçütü beraber değerlendirmek daha doğru sonuçlar verir. Bunun için f-ölçütü tanımlanmıştır. F-ölçütü, kesinlik ve duyarlılığın harmonik ortalamasıdır.Şekil 2.14'te F-Ölçütü hesaplamasına ait örnek verilmiştir.

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

Şekil 2.14:F-Ölçütü Hesaplaması[39]

3. ÖĞRENME TABANLI KALİTE ÖLÇME YÖNTEMİ

3.1 Amaç

Yapılan literatür taramasında hataya yönelik araştırmalarda veri madenciliği tekniklerinin yoğun olarak kullanıldığı görülmüştür. Bu çalışmada yeni bir yaklaşım olarak, sınıfların gelecek sürümlerdeki durumlarına ilişkin kestirimlerin sınıfların proje yaşam döngüsü boyunca değişimlerinden çıkarılması hedeflenmiştir. ISO/IEC 25010:2011[28]kalite modeline uygun olacak şekilde bakım kolaylığını etkileyen alt kalite özelliklerinden seçilen kalite özelliklerine uygun olan metrikler belirlenmiştir.

Belirlenen nesneye dayalı metriklere ek olarak durumsal liderlik modelinde tanımlanan çalışan yetkinliklerine uygun olarak yazılım ekibi elemanları yetkinlikleri doğrultusunda değerlendirilmiş ve bu kriterler metrik olarak eklenmiştir. Bununla birlikte sınıfın ait olduğu pakette bulunan hata sayısı ve bu hataların yazılımcı tarafından ne kadar sürede çözülmüş olduğu da metrik olarak ele alınmıştır.

Çalışmada, yazılımların sürümleri kullanılarak yazılımın sınıflarının ve beraberinde projenin bütünün ilerlemesi hakkındaISO/IEC 25010:2011[28] kalite modelinde belirlenen bakım kolaylığı kriteleri doğrultusunda karar verilip verilemeyeceğinin bulunması amaçlanmıştır.

3.2 Öğrenme Tabanlı Kalite Ölçme Modeli

Birçok yazılım firması, zaman kısıtı, maddi öncelikler, bilgi eksikliği gibi nedenlerden dolayı yazılım kalitesine yeteri kadar önem vermemekte, bazı firmalar ise ürün bittikten sonra yazılım kalitesi ile ilgili bazı çalışmalar yapmaktadır.

Yazılım projelerinde kalite kavramı projenin başından itibaren değerlendirilmesi gereken bir olgudur. Bununla birlikte yazılım kalitesinin ölçülmesinin de proje başından itibaren belirlenen kalite özelliklerine uygun şekilde yapılmalıdır.

Çalışmamızda ISO/IEC 25010:2011[28] kalite modeli baz alınarak projeye uygun kalite özelliklerinin proje başlangıcında seçilmesi ve uygun metrikler belirlenerek projenin başından itibaren kalite özelliklerin nasıl değiştiğinin takip edilmesi ve ölçülebilir kalitesi olan bir ürünün müşteriye sunulması hedeflenmiştir.

Yazılım sektöründe en temel sorunlardan biri müşteri gereksinimlerindeki değişikliktir. Bu sorunun çözümü olarak çevik yazılım geliştirme metotları ortaya çıkmıştır. Günümüzde Scrum, Unified Process, XP gibi çevik yazılım geliştirme metodolojileri yazılım firmalar tarafından kullanılmaktadır. Bu metodolojilerle, ürün gereksinimlerini bölmek, ürün tam olarak bitmeden önce müşteriden geri bildirim almak ve bu geri bildirimlerle değişen gereksinimlere ayak uydurabilmek ya da gereksinimlerin değişmesini asgari seviyede tutmak amaçlanır. Ancak müşterinin her zaman ne istediğini tam olarak anlatamamasından, yazılımcının alan bilgisi eksikliğinden dolayı yinelemeli olarak ilerleyen bir süreç sonunda bile gereksinim değişikliği talebi gelmektedir. Bu yüzden bakımı kolay yazılım geliştirmek yazılım firmalarının çalışan ve zaman maliyetini düşürmek için önem teşkil etmektedir.

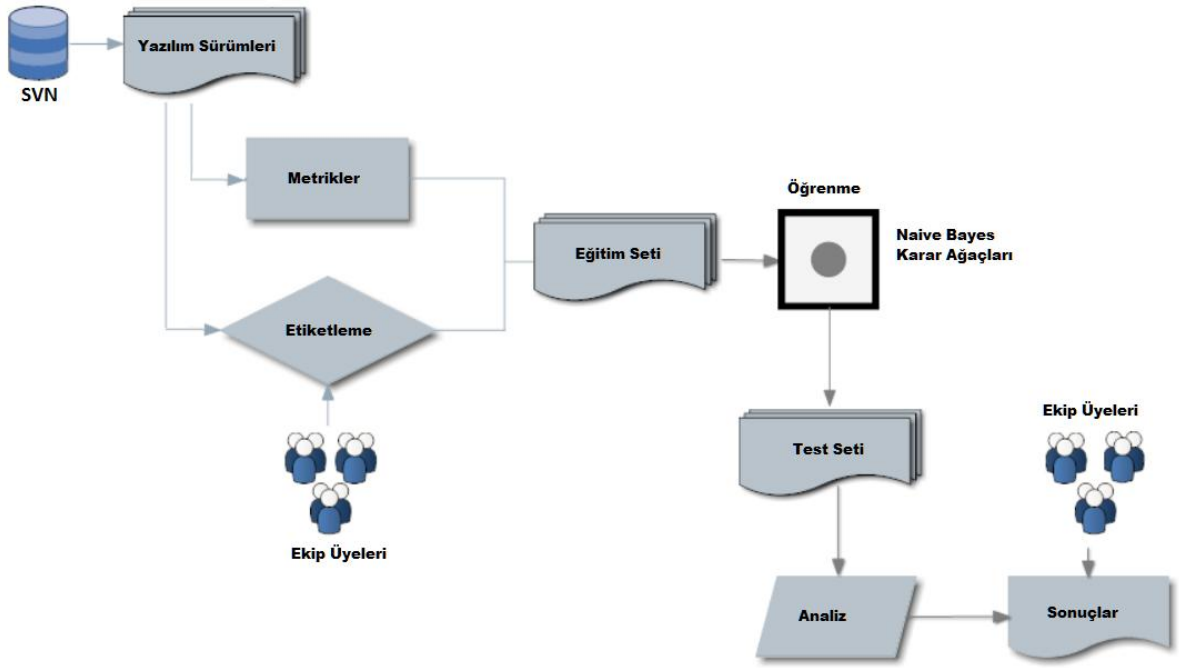
Çalışmada, yazılımda tanı eksikliğine bağlı olarakdüzenlenecek bölümlerin belirlenmesinde, düzeltilmesinde ve test edilmesinde gereken çabanınbulunması amaçlanmıştır. Bunun için bunların karşılığı olan ISO/IEC 25010:2011[28] kalite modelinde belirlenen bakım kolaylığı baz alınmıştır. Modelde genelden özele bir yaklaşım belirlenmiştir. Çalışmanın ilk adımında kalite modelini temsil eden kalite özellikleri belirlenmiştir, ardından kalite özelliklerini etkileyen iç özellikler belirlenmiştir. Son olarak iç özellikler üzerinde etkileri olan metriklere karar verilmiştir.

3.2.1 Öğrenme Tabanlı Ölçme Yöntemin Aşamaları

Model mimarisinde konfigürasyon yönetim aracından yazılıma ait sürümler alınmıştır. Sürümler üzerinde Understand [25] metrik ölçme aracı ile durağan kod analizi yapılarak yazılımın nesneye dayalı metrikleri elde edilmiştir. Yapılan deneysel çalışmalar sonucunda modelde seçilen metrik kümelerine ek olarak, proje yönetim portalından sürümler arasında çıkan hata sayıları ve hataların çözülmesi için gereken süreler de metrik kümelerine dâhil edilmiştir. Çalışanlar durumsal liderlik modelinegöreyöneticiler tarafından değerlendirilerek belirlenen metrik kümelerine dâhil edilmiştir.

Yazılımcılara verilen eğitimin ardından, her sürümde kendilerine ait yazılım paketlerindeki sınıflarına iyi ve kötü olarak etiket vermeleri istenmiştir. Her sürümde yazılımcılara ait paketler değiştiği için yazılımcıların hep aynı sınıfı etiketlememesi sağlanmıştır. Veri kümesinin oluşturulmasının ardından Rapid Miner [44] programı ile veri kümesi Naive Bayes ve Karar ağacı yöntemlerine göre eğitilmiştir.

Doğrulama için iki yöntem belirlenmiştir. Birincisi eğitim setinin %70'inin öğrenme için kullanılıp kalan %30'luk kısmının doğrulama için kullanılmasıdır. İkincisi ise tüm veri kümesinin eğitim için kullanılıp gelecek sürümün verisinin doğrulama verisi olarak kullanılmasıdır. Sonuçlar yazılım ekibi ve yöneticileri ile birlikte değerlendirilmiştir. Oluşturulan modelin mimarisi Şekil 3.1' de verilmiştir.



Şekil 3.1 : Model mimarisi

3.2.2 Hipotez ve Ölçüm Yaklaşımı

Tasarlanan modelde, bakım kolaylığının ISO/IEC 25010:2011[28] kalite modelinde bulunan Analiz edilebilirlik, Yeniden kullanılabilirlik, Test edilebilirlik ve Değiştirilebilirlik alt kriterleri seçilmiştir. Seçilen kalite özelliklerinin tanımları aşağıda verilmiştir.

- **Analiz edilebilirlik:** Tanı eksikliği ve çökme sebepleri veya düzenlenecek parçaların belirlenmesinde tanımlama için gereken çaba ile ilgilenen yazılım becerisidir.

- **Yeniden kullanılabilirlik:** Yazılımın başka bir yazılım ortamında kullanılabilmesi için fırsat sağlama ve harcanması gereken çaba ile ilgili yazılım becerisidir.
- **Test edilebilirlik:** Uyarlanan yazılımın onaylanması için gereken çaba ile ilgilenen yazılım becerisidir.
- **Değiştirilebilirlik:** Hata giderme, uyarlama veya çevresel değişiklik için gereken çaba ile ilgilenen yazılım becerisidir.

Kalite özelliklerinin alt özelliklerinin belirlenmesinin ardından belirlenen kalite özelliği ile ilişkili olan yazılım iç özellikleri belirlenmiştir. Belirlenen yazılım iç kalite özellikleri Çizelge 3.1'de verilmektedir.

Çizelge 3.1 : Yazılım Kalite Değişkenleri

	Kalite Değişkenleri			
	Değiştirilebilirlik	Analiz edilebilirlik	Test edilebilirlik	Yeniden kullanılabilirlik
Bağımlılık	*			*
Uyumluluk	*	*		*
Kalıtım			*	*
Karmaşıklık	*	*	*	*

Çalışmanın diğer aşamasında yazılımın iç özelliklerini etkileyen metrik değerleri belirlenmiştir. Belirlenen iç özelliklerin anlamları ve özellikleri etkileyen metrikler aşağıda verilmiştir. Bunlar;

Bağımlılık (Coupling); iki nesneden en az birinin diğerine etki etmesi olarak tanımlanır [26]. Bir sınıfın bağımlılığı; kendi sınıfları için diğer sınıfları ne kadar kullandığına ve başka sınıflar hakkında sahip olduğu bilgiye bağlıdır.

Bağımlılık için seçilen metrik:

- Bağımlı Sınıf Sayısı (CBO)

Uyum (Cohesion); uyumluluk bir sınıftaki metot ve değişkenlerin bir biri ile ne kadar ilişkili olduğuna bağlıdır. Eğer sınıf uyumluluğu düşükse sınıfın yeniden kullanılabilirliği ve bakımı zordur. Bu bağlamda müşteriden gelen değişik isteklerinde bu sınıfların değiştirilmesi hataya yol açabilir.

Uyum için seçilen metrikler:

- Uyum Eksikliği (LCOM)

- Yerel Metot Sayısı (CountDeclMethod)

Kalıtım; bir sınıfın özelliklerini taşıyıp yeni ek özellikler içeren sınıflar türetilmesidir. Nesneye dayalı yazılım geliştirmede önemli yer taşımasına karşılık dikkatli kullanılması gerekmektedir. Kalıtım ağacının derin olması durumda yazılımın karmaşıklığı ve bakım zorluğu, test edilecek durum sayısı artmaktadır. Ayrıca derin kalıtıma sahip sınıflarda esneklik azdır.

Kalıtım için seçilen metrikler:

- Ana Sınıf Sayısı (IFANIN)
- Alt sınıfların sayısı(NOC),
- Kalıtım Ağacının Derinliği (DIT)

Karmaşıklık; bir sınıfın anlaşılabilirliğinin zor olmasıdır. Anlaşılabilirliğin zor olması sınıfın ihtiva ettiği özelliklerin çokluğundan kaynaklanmaktadır. Karmaşıklığı çok olan bir sınıfın testi zordur. Beraberinde yeniden kullanılabilirliği azaltmıştır.

Karmaşıklık için seçilen metrikler:

- OrtalamaKarmaşıklık (AvgCyclomatic),
- Ortalama Değişen Karmaşıklık (AvgCyclomaticModified),
- Sınıfın ağırlıklı metot sayısı(WMC)
- Alt sınıfların sayısı(NOC)
- Kalıtım Ağacının Derinliği (DIT)
- Sınıfın tetiklediği metot sayısı (RFC)

3.2.3 Kullanılan Nesneye Dayalı Metrikler

Çalışmanın hazırlık aşamasında kullanılacak nesneye dayalı metriklerin belirlenmesi için deneysel çalışmalar yapılmıştır. C++ programlama dili ile yazılmış olan ve Qt grafiksel arayüz kütüphanesinin kullanıldığı gerçek zamanlı bir proje incelenerek projenin sürümler arasındaki metrik değişimleri değerlendirilmiştir.

Değişen metriklerin, değişme sebepleri incelenerek ve yazılımcılar ile görüşülerek bu metrik kümelerinden çalışmayadâhil edilecek olan metrikler belirlenmiştir.

Sınıfların nesneye dayalı metriklerinin sürümler boyunca nasıl değiştiğini görmek için sınıf metrik değişim grafikleri çıkarılmıştır. Burada her sınıfın, her sürüm için metrik değerinin nasıl değiştiğine farklı açılardan bakılmıştır.

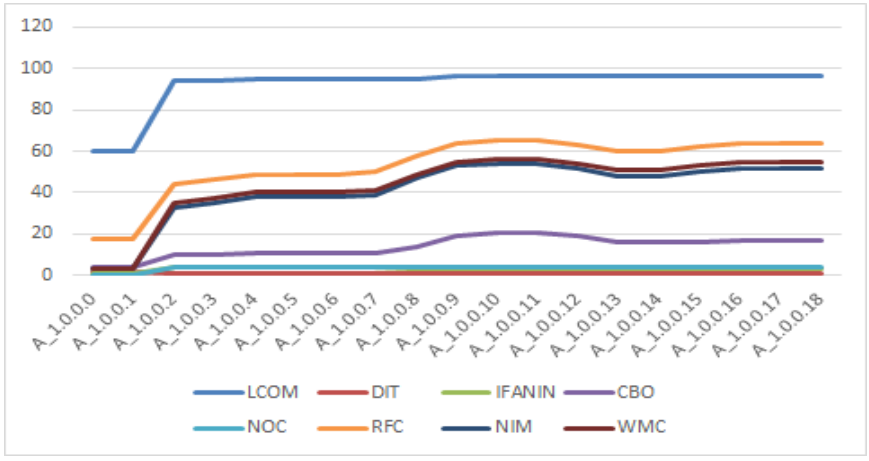
Konfigürasyon yönetim aracından sınıfların sürümleri alınarak, sürümler arasındaki metrik değişimleri incelenmiştir. Projede bulunan 82 sınıf içerisinden örnek olarak seçilen 9 sınıfa ait metriklerin değişim grafikleri Şekil 3.2, Şekil 3.3, Şekil 3.4, Şekil 3.5, Şekil 3.6, Şekil 3.7, Şekil 3.8, Şekil 3.9, Şekil 3.10'da verilmiştir.

Grafiklerde X ekseni sınıf sürümlerini temsil etmektedir. Y ekseni ise metrik değerlerinin aralıklarını belirtmektedir.

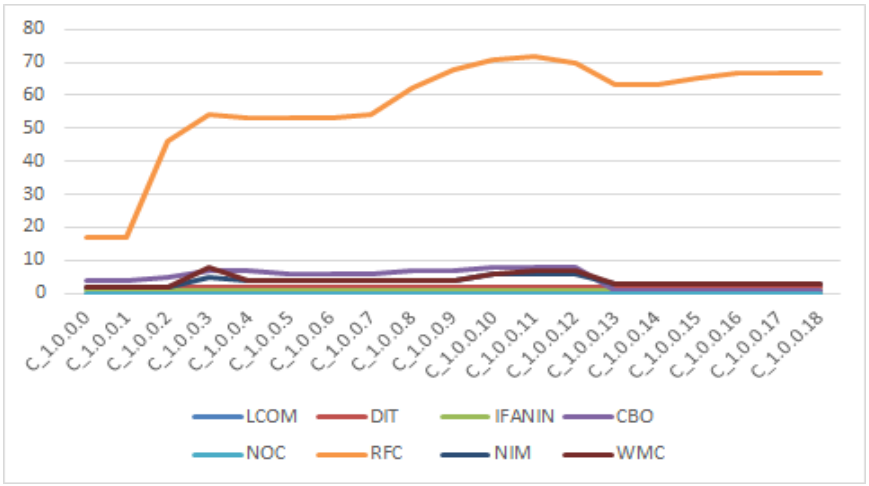
Sürümler arası metrik değişimleri incelip yazılım geliştiricilerle görüşülmüştür. İnceleme sonuçlarından bazıları şunlardır;

- Şekil 3.9'da K sınıfında 7.sürümde grafikte gözlenen metrik değerlerin (WMC) azalmasının nedeni incelendiğinde sınıfın bulunduğu paketin kodunun gözden geçirilme kararı verilip ve iyileştirmeler yapıldığı gözlenmiştir. Bir sonraki sürümde yeni bir özellik eklendiği için değerlerin yine değiştiği gözlenmiştir.
- Şekil 3.7' de benzer şekilde metrik değerlerinin belirgin şekilde değiştiği 9. sürümde sınıfın bulunduğu pakette çok hata çıkmasından dolayı iyileştirme yapıldığı ve değerlerin iyileştiği gözlenmiştir.
- Şekil 3.3 ve Şekil 3.4'te Sınıfın tetiklediği metot sayısı (RFC) değerinin diğer sınıflardan yüksek çıktığı gözlenmiştir. Bu sınıfların diğer sınıflara oranla anlaşılabilirliğinin daha az olduğu gözlenmiştir.
- Şekil 3.8' de metrik değerlerinin 14.sürüme kadar değişmediği gözlenmiştir. 14. sürümde metrik değerlerinin arttığı gözlenmiştir. Yapılan görüşmelerde tasarım aşamasında karar verilmeyen yeni özelliklerin sınıfın bulunduğu pakete eklendiği belirtilmiştir.

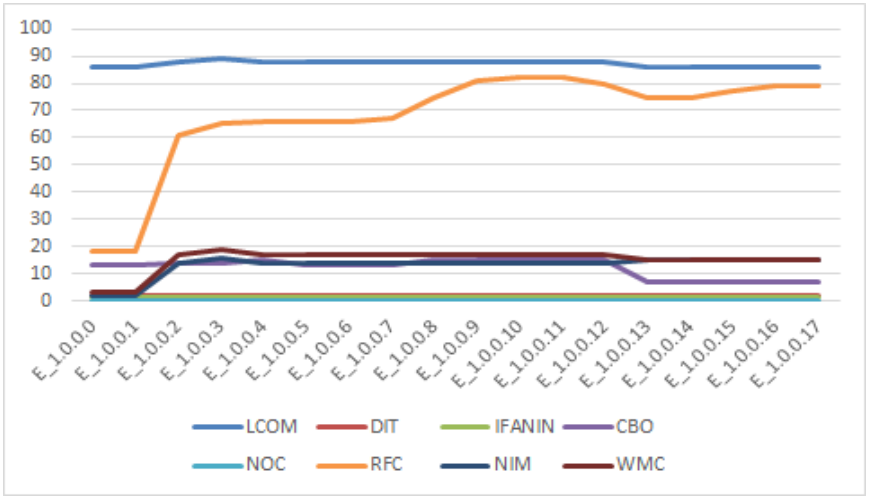
NIM (Count of Instance Method) metriğinin sürümler arasında değişimden etkilenmediği için metrik kümesine dahil edilmemiştir.



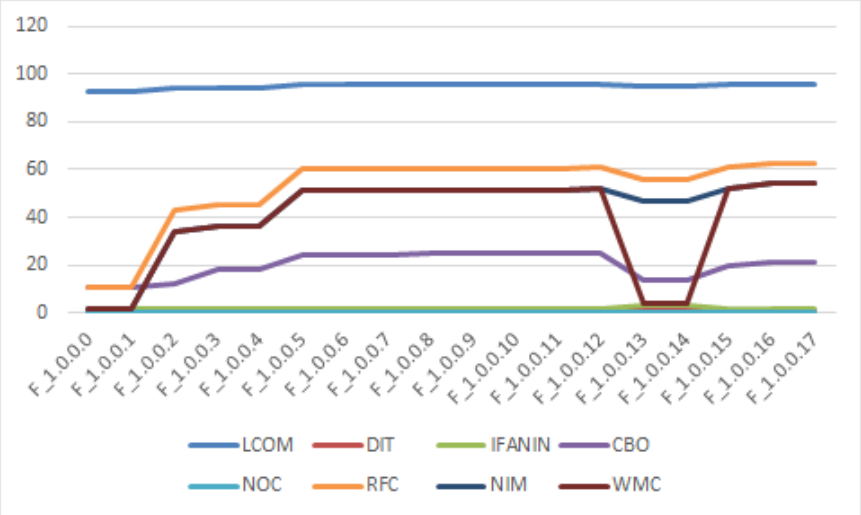
Şekil 3.2 :A sınıfına ait sürüm-metrik değişimi



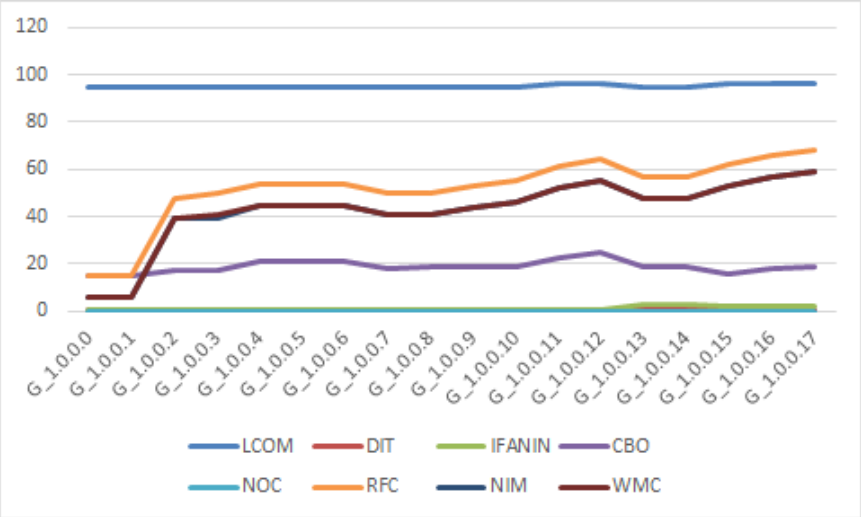
Şekil 3.3: C sınıfına ait sürüm-metrik değişimi



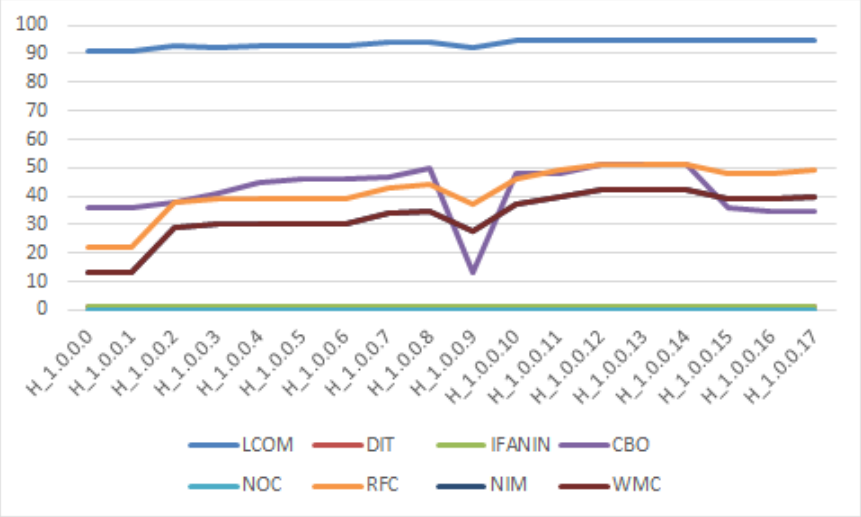
Şekil 3.4 : E sınıfına ait sürüm-metrik değişimi



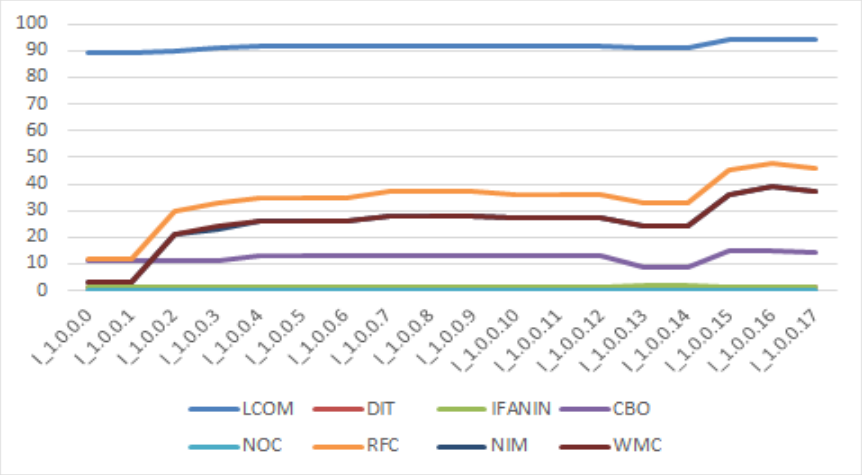
Şekil 3.5 : F sınıfına ait sürüm-metrik değişimi



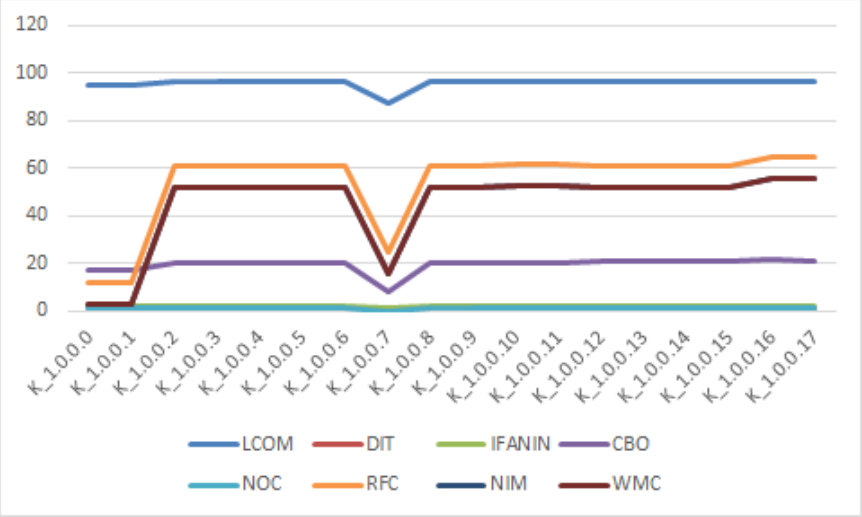
Şekil 3.6 : G sınıfına aitsürüm-metrik değişimi



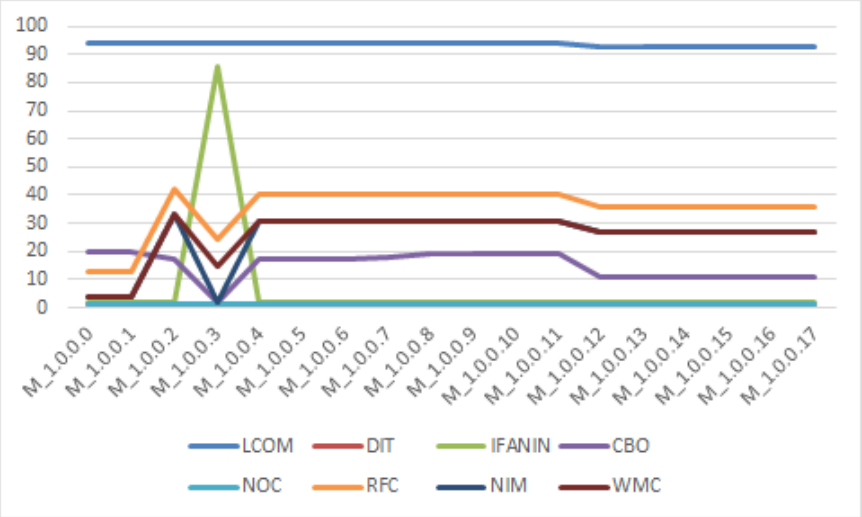
Şekil 3.7 : H sınıfına ait sürüm-metrik değişimi



Şekil 3.8 : I sınıfına ait sürüm-metrik değişimi



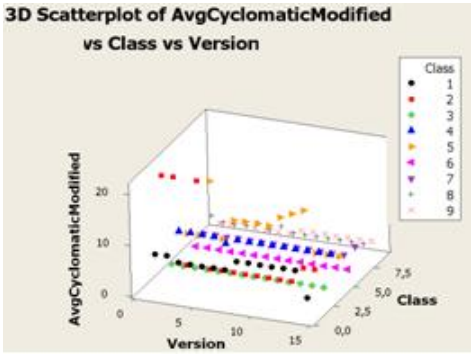
Şekil 3.9 : K sınıfına ait sürüm-metrik değişimi



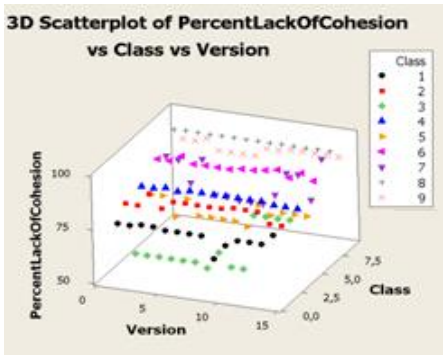
Şekil 3.10 : **M sınıfına ait sürüm-metrik değişimi**

Sınıf ve sürüm arasında metrik değerinin nasıl değiştiğinin analizi için geliştirilenaraçla metrik değerleri her sürüm için sınıf değerlerine ait metriklerin olduğu tablolar haline getirilmiş bu veriler veri madenciliği ve istatistiksel analiz aracı olan MiniTAB [39] kullanılarak 3D serpme çizim (Scattter plot) matematiksel analizi ile dağılımı hesaplanmıştır.

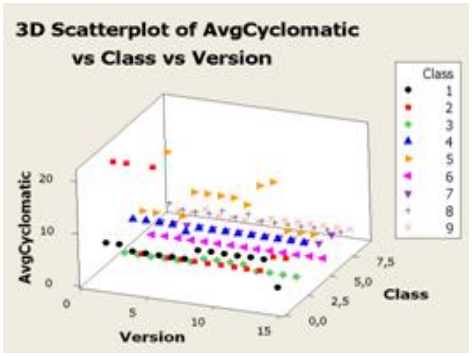
Bazı metriklere ait sınıf ve sürüm arasındaki korelasyona bağlı değişimlerden bazılarıŞekil 3.11, Şekil 3.12, Şekil 3.13, Şekil 3.14, Şekil 3.15, Şekil 3.16, Şekil 3.17, Şekil 3.18, Şekil 3.19 ve Şekil 3.20' de gösterilmektedir.



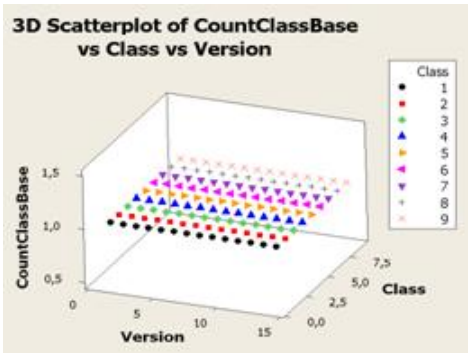
Şekil 3.11 : Ortalama Değişen Karmaşıklık-Sınıf-Sürüm



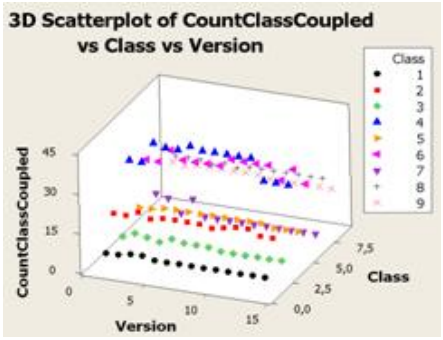
Şekil 3.12 : Uyum Eksikliği-Sınıf-Sürüm



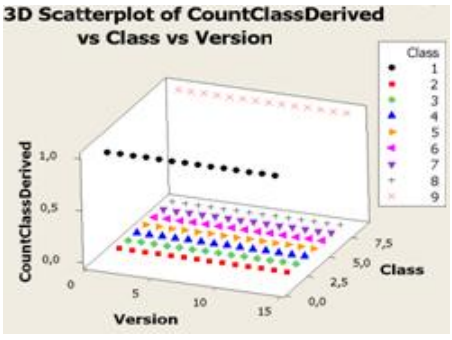
Şekil 3.13 : Ortalama Karmaşıklık-Sınıf-Sürüm



Şekil 3.14 : Ana Sınıf Sayısı-Sınıf-Sürüm

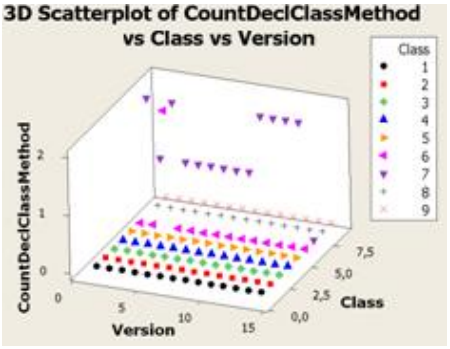


Şekil 3.15 : Bağımlı Sınıf Sayısı-Sınıf-



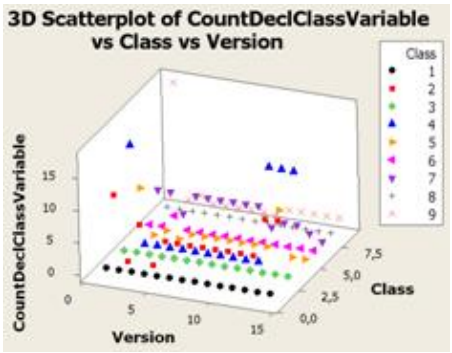
Şekil 3.16 : Türemiş Sınıf Sayısı-Sınıf-

Sürüm

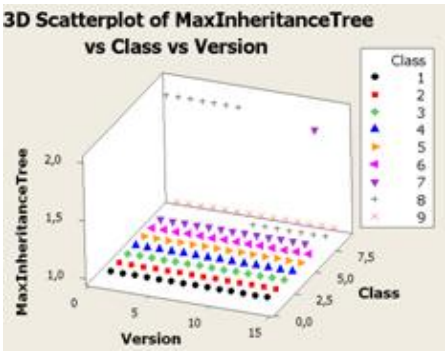


Şekil 3.17 : Sınıf Metot Sayısı-Sınıf-Sürüm

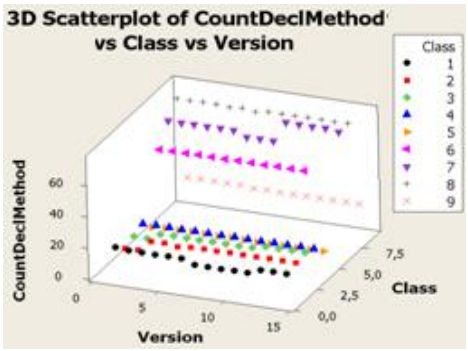
Sürüm



Şekil 3.18 : Sınıf Değişken Sayısı-Sınıf-Sürüm



Şekil 3.19 : Kalıtım Ağacının Derinliği-Sınıf-Sürüm



Şekil 3.20 : Yerel Metot Sayısı-Sınıf-Sürüm

Şekil 3.15'te bağımlı sınıf sayısının yüksek olduğu sınıflar ele alındığında bu sınıfların yeniden kullanılabilirliğinin az olduğu gözlemlenmiştir.

Şekil 3.13'te karmaşıklığı yüksek olan sınıflara bakıldığında bu sınıflarda daha çok hata çıktığı belirlenmiştir.

Yapılan deneysel çalışmalar ve araştırmalar sonucunda kullanılmaya karar verilen metrikler ve tanımları Çizelge 3.2'de verilmiştir.

Çizelge 3.2 : Tanımlı Nesneye Dayalı Metrikler

Metrik	Açıklaması
Ortalama Karmaşıklık (AvgCyclomatic)	Tüm metotlar için karmaşıklık sayısı
Ortalama Değişen Karmaşıklık (AvgCyclomaticModified)	Tüm iç içe metotlar için karmaşıklık sayısı
LCOM (Percent Lack of Cohesion)	Sınıflardaki uyum eksikliğinin yüzdesi
Yerel Method Sayısı (CountDeclMethod)	Yerel metotsayısı

DIT (Max Inheritance Tree):	Sınıfın kalıtım ağacının köküne uzaklığı
IFANIN (Count of Base Classes)	Ana sınıf sayısı
CBO (Count of Coupled Classes)	Bağımlı sınıfların sayısı
NOC (Count of Derived Classes)	Alt sınıfların sayısı
RFC (Count of All Methods)	Sınıfın tetiklediği metot sayısı
WMC (Count of Methods):	Sınıfın ağırlıklı metot sayısı

3.2.4 Kullanılan Dış Metrikler

Kalite özelliklerini oluşturan iç özellikleri etkileyen metrikler seçilirken Çizelge 3.1'de verilen nesneye dayalı yazılım metriklerine ek olarak, yazılımcılara ve programın davranışına ait metriklerde göz önünde bulundurulmuştur.Nesneye dayalı metrikler dışında belirlenen metrikler, geliştiren kişilerin tecrübeleri, pakette bulunan hata sayıları ve bu hataların yazılımcılar tarafından giderilmesi için geçen süredir.

Yazılımı geliştiren ekibin değerlendirilmesinde durumsal liderlik modeli benimsenmiştir. Durumsal liderlik, liderin görev ve ilişki davranışı tarzları ile herhangi bir faaliyette izleyicilerin ortaya koydukları olgunluk düzeyi arasındaki ilişkilere bağlı olarak geliştirilen bir süreçtir.[45] Bu model farklı dört olgunluk düzeyi ile dört temel liderlik tarzı üzerine kurulmuştur. Durumsal liderlik modelinde,Şekil 3.21'de gösterilen hazır olma seviyesi skalası baz alınarak, projedeki yazılımcılar bu skala doğrultusunda değerlendirilmişlerdir.

Durumsal liderlik modeli ile yazılımcıların 3 konudaki olgunluk düzeyleri metrik kümesine dahil edilmiştir.

Bunlar;

- Programlama diline hâkimiyetleri
- Yazılım tecrübeleri
- Yazılımın uygulama alanındaki tecrübeleri

8 yazılımcı bu 3 konuda olgunluk düzeylerine göre çalıştıkları bölümün yöneticileri tarafından gruplandırılmıştır. Yazılım yaşam döngüsü boyunca değişen sürümlerde yetkinlik düzeylerinin artıp artmadığı göz önünde bulundurulmuştur.

Şekil 3.21 : Durumsal Liderlik Modelinde Çalışan Değerlendirmesi

Kullanılan diğer metrikler ise, sınıfların bulunduğu pakette çıkan hata sayıları ve bu hataların çözülmesi için harcanan süredir. Harcanan sürenin metrik kümesine konması ile amaçlanan, hata bulunduğunda bunun tespit edilebilmesi için gerekli eforun tahmin edilmesine katkı sağlanmasıdır.

3.2.5 Metriklerin Gruplanması

Çalışmada, nesneye dayalı metrikler, durumsal liderlik modelini baz alan yazılımcının değerlendirilmesine yönelik metrikler ve ürünün ilerlemesi sırasında ortaya çıkan ürün bazlı metrikler kullanılmıştır. Kullanılan metrikler ve açıklamaları Çizelge 3.3'te verilmiştir.

Çizelge 3.3: Modelde kullanılan metrik kümesi

Metrik	Açıklaması
Ortalama Karmaşıklık (OK) (AvgCyclomatic)	Tüm metotlar için karmaşıklık sayısı
Ortalama Değişen Karmaşıklık(ODK) (AvgCyclomaticModified)	Tüm iç içe metotlar için karmaşıklık sayısı
LCOM (Percent Lack of Cohesion)	Sınıflardaki uyum eksikliğinin yüzdesi
Yerel Method Sayısı (YMS)(CountDeclMethod)	Yerel metotsayısı
DIT (Max Inheritance Tree):	Sınıfın kalıtım ağacının köküne uzaklığı
IFANIN (Count of Base Classes)	Ana sınıf sayısı

CBO (Count of Coupled Classes)	Bağımlı sınıfların sayısı
NOC (Count of Derived Classes)	Alt sınıfların sayısı
RFC (Count of All Methods)	Sınıfın tetiklediği metot sayısı
WMC (Count of Methods):	Sınıfın ağırlıklı metot sayısı
Programlama diline hakimiyetleri	Durumsal liderlik modeline göre S1,S2,S3,S4
Yazılım tecrübeleri	Durumsal liderlik modeline göre S1,S2,S3,S4
Yazılımın uygulama alanındaki tecrübeleri	Durumsal liderlik modeline göre S1,S2,S3,S4
Hata Sayısı	Sınıfın ait olduğu pakette bulunan hata sayısı
Hata düzeltme süresi	Hatanın giderilmesi için gerekli efor

Burada genelden özele ilerleyen bir yaklaşım belirlenerek model oluşturulmuştur. Çalışmanın ilk adımında kalite modelini temsil eden kalite özellikleri belirlenmiş, ardından kalite özelliklerini etkileyen iç özelliklere karar verilmiştir.

Son olarak iç özellikler üzerinde etkileri olan metriklere karar verilerek kalite modeli hazırlanmıştır.

Programın genel analizinin yapılmasının ardından makine öğrenmesi yöntemlerinden biri olan Karar Ağacı yöntemi ile programın gelecek sürümlerde belirlenen kalite özelliklerine (Yeniden Kullanılabilirlik, Analiz Edilebilirlik, Test Edilebilirlik, Değiştirilebilirik) uygun olup olmayacağı araştırıldı.

Biz çalışmamızda her sınıfa belirlediğimiz kalite özelliklerinin (Yeniden Kullanılabilirlik, Analiz Edilebilirlik, Test Edilebilirlik, Değiştirilebilirik) sınıflandırılması için 2 kriter belirledik.

Bu kriterlere uygun metrik kümeleri Çizelge 3.4'te tanımlanmaktadır.

Tabloda Yeniden kullanılabilirlik YK, Analiz edilebilirlik AE, Test edilebilirlik TE ve Değiştirilebilirlik D ile temsil edilmiştir.

Nesneye dayalı metriklerin kısaltmaları Çizelge 3.3'te verilmiştir.

Çizelge 3.4 : Kalite özelliklerine uygun metrik kümeleri

	METRİK KÜMESİ														
	OK	**ODK**	**LCOM**	**YMS**	**DIT**	**IFANIN**	**CBO**	**NOC**	**RFC**	**WMC**	**ÇK1**	**ÇK2**	**ÇK3**	**HS**	**HDS**
YK			*	*	*	*	*	*	*	*	*	*	*		
AE	*	*	*	*			*				*	*	*	*	*
TE	*	*			*	*	*	*	*	*	*	*	*	*	*
D	*	*	*		*	*	*	*	*	*	*	*	*		

Belirlenen sınıflandırma kriterleri aşağıda verilmiştir;

- Kötü
- İyi

Yazılımcılara her kalite özelliğinin kalite modelinde ne anlama geldiği anlatılarak, kalite modelinde verilen tanıma göre geliştirdikleri sınıfa iyi veya kötü olarak etiket vermeleri istendi.

Yazılımcılardan alınan bilgiler doğrultusunda elde edilen veri kümesi her bir kalite özelliğine (Yeniden Kullanılabilirlik, Analiz Edilebilirlik, Test Edilebilirlik, Değiştirilebilirik) göre oluşturuldu.

Yeniden kullanılabilirlik için hazırlanan veri örneği Şekil 3.22' de verilmiştir.

	LCOM	YMS	DIT	IFANIN	CBO	NOC	RFC	WMC	ÇK1	ÇK2	ÇK3	Label
A_1.0.3.0	60	10	2	1	4	0	18	3	S3	S3	S3	İyi
B_1.0.1.0	60	11	2	1	4	0	18	3	S2	S2	S2	Kötü
C_1.0.2.0	74	11	1	4	10	4	44	33	S2	S2	S2	İyi
A_1.0.1.0	74	13	1	4	10	4	46	35	S3	S3	S3	Kötü
B_1.0.2.0	75	13	1	4	11	4	49	38	S2	S2	S2	İyi

Şekil 3.22 : Yeniden Kullanılabilirlik için örnek veri kümesi

3.3 Çalışmada Kullanılan Projeler

Çalışma için farklı iki firmanın geliştirdiği yazılım projeleri kullanılmıştır.

Bunlardan ilki C# programlama dilinde yazılmış olan iş uygulama yazılımıdır. Proje yaşam döngüsü modeli olarak, çevik(agile) metodolojilerden scrum metodolojisinine uygun olarak geliştirilmiştir. Şuan proje sonuna gelinmiş olup 36 sprint geride bırakılmıştır. Her sprint bir sürüm olarak kabul edilerek projenin nasıl ilerlediğini takip edebilmek için bu sürümler kullanılmıştır.

Diğer yazılım projesi, C++ programa dili ile yazılmış olan ve Qt grafiksel arayüz kütüphanesinin kullanıldığı gerçek zamanlı bir uygulamadır.

Projenin tüm test faaliyetleri tamamlanmış ve müşteriye teslim edilmiştir. Yaşam döngüsünün bakım safhasında olan projenin, çalışmaya katkısı, projenin tüm yaşam döngüsü sürümlerine ulaşılabilmesidir.

Analiz ve tasarım evresi dâhil 30 ayda bitmiş olan projenin son 15 ayı kodlama için ayrılmıştır. Şelale modelinin kurum içi uyarlanmış halini benimseyen ekibin test faaliyetleri tüm yaşam döngüsünde devam etmiştir. Projenin kodlarının incelenmesi aşamasında yanıltıcı olmaması için ilk 5 aylık kod değişimi analiz edilmemiş yazılımın belirli bir aşamaya gelmesi amaçlanmıştır. Son 10 ay içirisinde sınıfların ne şekilde değiştiğinin incelenmesi üzerine çalışılmıştır.

Çalışmanın deneysel boyutunda proje ekibinden geliştiricilerle yapılan görüşmeler sonucunda en kritik dokuz sınıf belirlenip bunların aylık kod değişimlerine ait metrik ölçümleri çıkarılmıştır ve bunun sonucu olarak sınıflara ait metrik değerlerinin aylık olarak nasıl değiştiği bilgisine ulaşılmıştır.

Burada amaçlanan sınıfların değişimlerinin anlaşılabilir şekilde sunulmasıdır. Detayları, Bölüm 3.2.3 ‘ de verilmiştir.

Belirlenen nesneye dayalı metrik kümeleri Çizelge 3.2’ de verilmiştir. Bu metriklerin değerleri SCI Understand aracı [25] ile her ayki sınıf değişimleri için ayrı ayrı hesaplanmıştır. Yazılımcıların durumsal liderlik modelindeki olgunluk seviyelerine göre değerlendirilmesi proje yöneticisi ile birlikte yapılmıştır. Bu seviyelendirme yapılırken projelerin ilerlemesine bakarak yazılımcıların seviyelerinin değişip değişmediği göz önünde bulunmuştur.

Yazılımcıların,

- Programlama diline hâkimiyetleri
- Yazılım tecrübeleri
- Yazılımın uygulama alanındaki tecrübeleri

değerlendirilerek durumsal liderlik modeline uygun şekilde S1,S2,S3,S4 olarak gruplandırılmışlardır.

Bunlara ek olarak çalışmada kullanılan sınıfın bulunduğu pakette bulunan hata sayısı ve hatanın giderilmesi için gereken süre metrikleri proje yönetim portalı olarak kullanılan Jira aracından alınmıştır. İncelenen projelerde, her sürümde hatalar yazılımcılara Jira üzerinden gönderilmekte ve yazılımcılar hataları giderdikten sonra Jira maddelerini test uzmanına göndermeden önce bu hatayı düzeltmek için ne kadar çalıştıklarını (log work) girmektedirler.

Programın genel analizinin yapılmasının ardından ölçme için kullanılacak veri seti çıkarıldı. Burada, Bölüm 3.2.3 'de detaylandırılan ölçüm sonuçları kullanıldı. Örneğin yeniden kullanılabilirlik için WMC değerinin önemli olduğu yapılan deneysel sonuçlarda görüldü. Bununla birlikte çalışanların tecrübelerinin ve alan bilgisinin de yeniden kullanılabilirliği etkilediği gözlenmiştir. Hata sayılarının çok önemli olmamasından dolayı metrik kümesine dahil edilmemiştir. Yeniden kullanılabilirlik için hazırlanan örnek veri kümesiŞekil 3.23'de verilmiştir.

	LCOM	YMS	DIT	IFANIN	CBO	NOC	RFC	WMC	ÇK1	ÇK2	ÇK3	Label
A_1.0.3.0	60	10	2	1	4	0	18	3	S3	S3	S3	İyi
B_1.0.1.0	60	11	2	1	4	0	18	3	S2	S2	S2	Kötü
C_1.0.2.0	74	11	1	4	10	4	44	33	S2	S2	S2	İyi
A_1.0.1.0	74	13	1	4	10	4	46	35	S3	S3	S3	Kötü
B_1.0.2.0	75	13	1	4	11	4	49	38	S2	S2	S2	İyi

Şekil 3.23 : Yeniden kullanılabilirlik için örnek veri kümesi

Analizler sonucunda, deneysel gözlemlerden ve ekip ile görüşmelerden elde edilen verilerle birlikte veri setleri bakım kolaylığını etkileyen alt kalite faktörlerinden analiz edilebilirlik, test edilebilirlik ve değiştirilebilirlik için de hazırlanmıştır.

Test edilebilirlik sonuçlarında paketlerde çıkan hataların çözülme sürelerinin etkili olduğu gözlenmiştir. Versiyonlar ilerlerken buna dikkat edildiğinde sonraki sürümlerde hataların daha kısa sürede tespit edildiği gözlenmiştir. Projenin ortalarında alt yapı değişikliği yapılarak proje için Model View Presenter tasarım kalıbına uygun şekilde tasarım yeniden yapılmıştır. Sonuçlar incelendiğinde bu tasarım değişikliği sırasında metriklerin değiştiği, tasarımın uygulanmasının ardından hataların çözülme sürelerinin azaldığı ve test edilebilirliğin arttığı gözlenmiştir. Test edilebilirlik için hazırlanan örnek veri kümesi Şekil 3.24'te verilmiştir.

	OK	ODK	DIT	IFANIN	CBO	NOC	RFC	WMC	ÇK1	ÇK2	ÇK3	HS	HDS	Label
A_1.0.3.0	30	35	2	1	4	0	18	3	S3	S3	S3	3	60	İyi
B_1.0.1.0	25	30	2	1	4	0	18	3	S2	S2	S2	2	10	İyi
C_1.0.2.0	28	31	1	4	10	4	44	33	S2	S2	S2	0	0	Kötü
A_1.0.1.0	41	23	1	4	10	4	46	35	S3	S3	S3	2	30	Kötü
B_1.0.2.0	21	12	1	4	11	4	49	38	S2	S2	S2	1	10	Kötü

Şekil 3.24 : Test edilebilirlik için örnek veri kümesi

Analiz Edilebilirlik sonuçlarında uyum için seçilen metriklerin sonuçlara etki ettiği gözlenmiştir. Benzer şekilde daha yetkin çalışanların, alan bilgisi daha iyi olan kişilerin geliştirdiği sınıfların daha anlaşılır olduğu gözlenmiştir. Analiz edilebilirlik için hazırlanan örnek veri seti Şekil 3.25'te verilmiştir.

	OK	ODK	LCOM	YMS	CBO	ÇK1	ÇK2	ÇK3	HS	HDS	Label
A_1.0.3.0	30	35	60	10	4	S3	S3	S3	3	60	İyi
B_1.0.1.0	25	30	60	11	4	S2	S2	S2	2	10	Kötü
C_1.0.2.0	28	31	74	11	10	S2	S2	S2	0	0	İyi
A_1.0.1.0	41	23	74	13	10	S3	S3	S3	2	30	Kötü
B_1.0.2.0	21	12	75	13	11	S2	S2	S2	1	10	İyi

Şekil 3.25 : Analiz Edilebilirlik için hazırlanan örnek veri seti

Değiştirilebilirlik için de geliştiricilerin alan bilgilerinin ve kalıtımın etkili olduğu gözlenmiştir. Değiştirilebilirlik için hazırlanan örnek veri seti Şekil 3.26'te verilmiştir.

	OK	ODK	LCOM	DIT	IFANIN	CBO	NOC	RFC	WMC	ÇK1	ÇK2	ÇK3	Label
A_1.0.3.0	30	35	60	2	1	4	0	18	3	S3	S3	S3	İyi
B_1.0.1.0	25	30	60	2	1	4	0	18	3	S2	S2	S2	İyi
C_1.0.2.0	28	31	74	1	4	10	4	44	33	S2	S2	S2	Kötü
A_1.0.1.0	41	23	74	1	4	10	4	46	35	S3	S3	S3	Kötü
B_1.0.2.0	21	12	75	1	4	11	4	49	38	S2	S2	S2	İyi

Şekil 3.26: Değiştirilebilirlik için hazırlanan örnek veri kümesi

Hazırlanan veri setleri makine öğrenmesi yöntemlerinden biri olan Karar Ağacı ve Naive Bayes yöntemi ile programın gelecek sürümlerde belirlenen kalite özelliklerine (Yeniden Kullanılabilirlik, Analiz Edilebilirlik, Test Edilebilirlik, Değiştirilebilirik) uygun olup olmayacağı araştırıldı.

Elde edilen 8000 satırlık verinin %70'i eğitim için kullanılmıştır. Kalan %30'luk bölümü ise test verisi olarak ayrılmıştır. Sınama kümesi 2400 adet veriden oluşmaktadır. Amerika'da bulunan YALE üniversitesi bilim adamları tarafından Java dili kullanılarak geliştirilmiş olan RapidMiner programı kullanılarak karar ağacı algoritması ve naive bayes yöntemleri veri setine uygulanmıştır.

Model başarımını değerlendirirken kullanılan temel kavramlar hata oranı, kesinlik, duyarlılık ve F-ölçütüdür. Modelin başarısı, doğru sınıfa atanan örnek sayısı ve yanlış sınıfa atılan örnek sayısı nicelikleriyle alakalıdır.

Test sonucunda ulaşılan sonuçların başarım bilgileri karışıklık matrisi ile ifade edilebilir. Karışıklık matrisinde satırlar test kümesindeki örneklere ait gerçek sayıları, kolonlar ise modelin tahminlemesini ifade eder. Detayları Şekil 2.10'da verilmiştir. Kesinlik ve duyarlılık ölçütleri tek başına anlamlı bir karşılaştırma sonucu çıkarmamıza yeterli değildir.

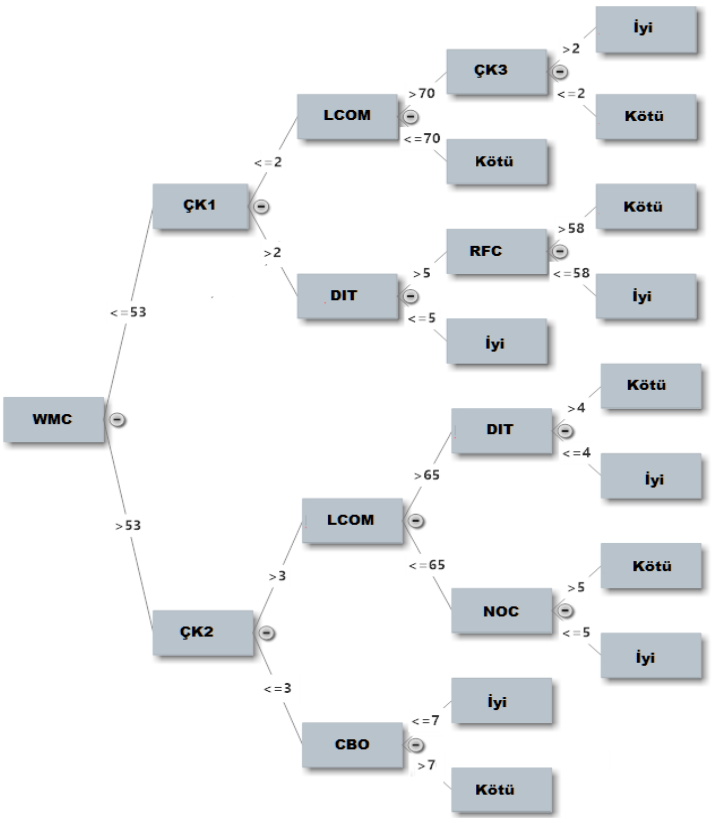
Her iki ölçütü beraber değerlendirmek daha doğru sonuçlar verir. Bunun için f-ölçütü tanımlanmıştır. F-ölçütü, kesinlik ve duyarlılığın harmonik ortalamasıdır.

3.4 Modelin Değerlendirilmesi

3.4.1 Karar Ağacı Yöntemi ile Yeniden Kullanılabilirliğin Değerlendirmesi

Yeniden Kullanılabilirlik için oluşturulan Karar Ağacı

Şekil **3.27**' de verilmiştir. Karar ağacı incelendiğinde Yeniden kullanılabilirlik için belirlenen metrik kümesi içinden YMS(CountDeclMethod), IFANIN(Count of Base Class) değerlerinin elendiği gözlenmiştir. Nesneye dayalı metrik kümeleri ile çalışan kriterlerinin birlikte değerlendirildiğinde çalışanların programlama dili hakkındaki bilgi birikimlerininve yazılım tecrübelerinin önemli olduğu sonucuna varılmıştır. WMC değeri 53'ten küçük olduğu durumda, çalışan tecrübesi durumsal liderlik modeline göre S2'den büyükse ve DIT değeri 5'ten küçük olduğu durumda sınıf yeniden kullanılabilirlik açısından iyi olarak etiketlenmiştir.



Şekil 3.27: Yeniden Kullanılabilirlik için Karar Ağacı

Yeniden kullanılabilirlik için yapılan sınama sonuçları Çizelge 3.5' te verilmiştir.İyi ve Kötü sütunları sınama setindeki yeniden kullanılabilirlik için iyi ve kötü olarak etiketlenmiş sınıfları temsil etmektedir.

İyi ve kötü kümelerindeki doğru pozitif (TP – true positive), yanlış pozitif (FP – false positive) değerleri gösterilmiştir. Çizelgede Kesinlik, Duyarlılık ve F-Ölçütü verilmektedir.

Kötü olarak etiketlenen sınıflar yazılım ekibi ile incelendiğinde sınıfların yeni özellik eklenirken kurallara daha az uyulduğu ve WMC değerlerinin yüksek olduğu gözlemlenmiştir. Ayrıca kötü olarak etiketlenen sınıflarda yazılımcı tecrübelerinin ve programlama diline hâkimiyetlerinin az olduğunu belirlenmiştir.

Yeniden kullanılabilirlik için doğruluk değeri %86.36, Kesinlik değeri %85.57, Duyarlılık değeri %96.53 ve F-Ölçütü %90.72 olarak belirlenmiştir.

Çizelge 3.5 : Yeniden Kullanılabilirlik Başarım Ölçütü

TP	FP	Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
0.895	0.148	%86.36	%85.57	%96.53	%90.72

3.4.2 Karar Ağacı Yöntemi ile Analiz EdilebilirliğinDeğerlendirmesi

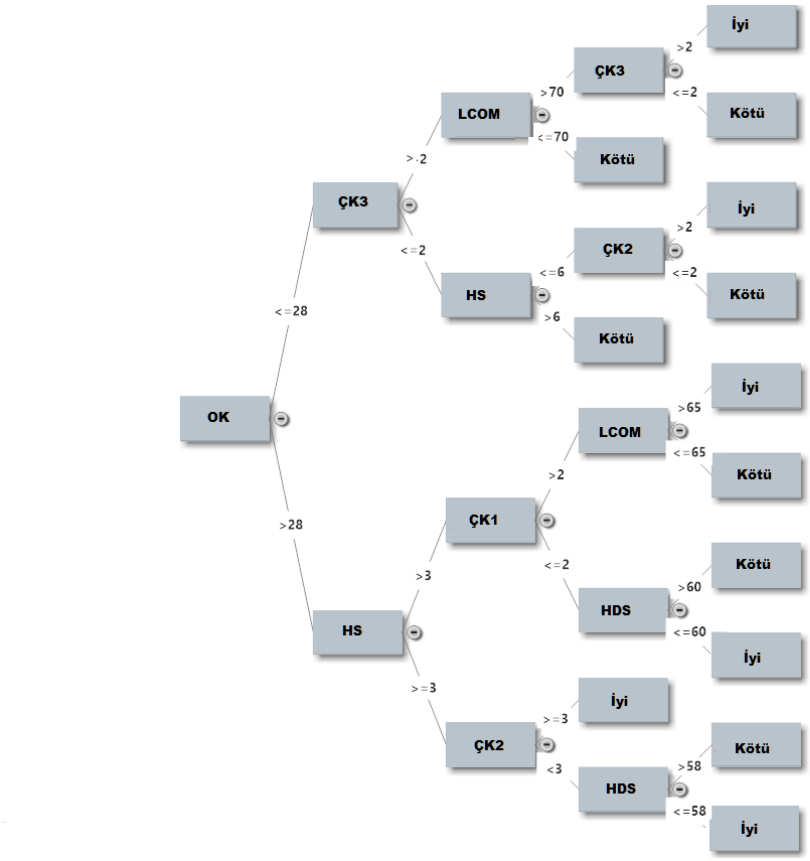
Analiz edilebilirlik için oluşturulan Karar Ağacı Şekil 3.28' de verilmiştir.

Karar ağacı incelendiğinde Analiz edilebilirlik için belirlenen metrik kümesi içinden YMS(CountDeclMethod), ODK (Tüm iç içe metotlar için karmaşıklık sayısı) değerlerinin elendiği gözlenmiştir. OK (Tüm metotlar için karmaşıklık sayısı) alındığında ODK değerine analiz edilebilirlikiçin ihtiyaç olmadığı belirlenmiştir.

Nesneye dayalı metrik kümeleri ile çalışan kriterlerinin birlikte değerlendirildiğinde, hata sayılarının, hata düzeltme sürelerinin, çalışanların programlama dili hakkındaki bilgi birikimlerinin ve yazılımcıların uygulama alanındaki tecrübelerinin önemli olduğu gözlenmiştir.

OK değeri 28'den küçük olduğu durumda, çalışanın uygulama alanındaki tecrübesi durumsal liderlik modeline göre S2'den küçükse ve hata sayısı 6'dan büyük olduğu durumda sınıf analiz edilebilirlik açısından kötü olarak etiketlenmiştir.

.



Şekil 3.28 : Analiz Edilebilirlikiçin Karar Ağacı

Analiz edilebilirlik için yapılan sınama sonuçları Çizelge 3.6' da verilmiştir.İyi ve Kötü sütunları sınama setindeki yeniden kullanılabilirlik için iyi ve kötü olarak etiketlenmiş sınıfları temsil etmektedir.İyi ve kötü kümelerindeki doğru pozitif (TP – true positive), yanlış pozitif (FP – false positive) değerleri gösterilmiştir. Çizelgede Kesinlik, Duyarlılık ve F-Ölçütü verilmektedir. Kötü olarak etiketlenen sınıflar yazılım ekibi ile incelendiğinde sınıfların hata çözülme sürelerinin uzun olduğu gözlenmiştir. Yazılımcıların programlama dili alanındaki tecrübelerinin ve uygulama alanındaki bilgi birikimlerinin analiz edilebilirlik için önem taşıdığı belirlenmiştir.

Çizelge 3.6 : Analiz Edilebilirlik için Başarım Ölçütü

TP	FP	Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
0.852	0.161	%85.21	%85.80	%94.15	%89.78

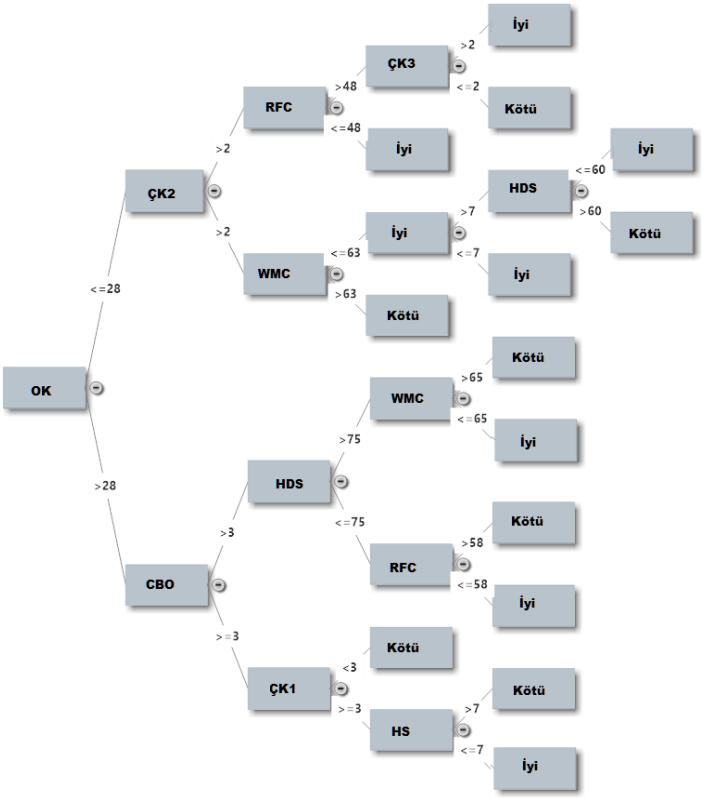
Analiz edilebilirlik için Doğruluk ölçütü %85.21, Kesinlik %85.80 ve Duyarlılık %94.15 olarak bulunmuştur. F-Ölçütü ise %89.78 olarak tespit edilmiştir.

3.4.3 Karar Ağacı Yöntemi ile Test Edilebilirliğin Değerlendirmesi

Test edilebilirlik için oluşturulan Karar Ağacı

Şekil **3.29**' da verilmiştir. Karar ağacı incelendiğinde Test edilebilirlik için belirlenen metrik kümesi içinden YMS(CountDeclMethod), IFANIN(Count of Base Class), ODK (Tüm iç içe metotlar için karmaşıklık sayısı) değerlerinin elendiği gözlenmiştir. OK (Tüm metotlar için karmaşıklık sayısı) alındığında ODK değerine test edilebilirlik için ihtiyaç olmadığı belirlenmiştir.

Nesneye dayalı metrik kümeleri ile çalışan kriterlerinin birlikte değerlendirildiği, nesneye dayalı metriklerden RFC, WMC ve CBO değerlerinin karar ağacında önemli yer teşkil ettiği gözlenmiştir. Hata sayılarının, hata düzeltme sürelerinin, çalışanların programlama dili hakkındaki bilgi birikimlerinin ve yazılım tecrübelerinin önemli olduğu gözlenmiştir. OK değeri 28'den, CBO'nun 3'ten büyük olduğu durumda, yazılımcının programlam dili hâkimiyeti durumsal liderlik modeline göre 3'ten küçükse sınıf test edilebilirlik açısından kötü olarak etiketlenmiştir.



Şekil 3.29 : Test Edilebilirlik için Karar Ağacı

Test edilebilirlik için yapılan sınama sonuçları Çizelge 3.6' da verilmiştir.İyi ve Kötü sütunları sınama setindeki yeniden kullanılabilirlik için iyi ve kötü olarak etiketlenmiş sınıfları temsil etmektedir.İyi ve kötü kümelerindeki doğru pozitif (TP – true positive), yanlış pozitif (FP – false positive) değerleri gösterilmiştir. Çizelgede Kesinlik, Duyarlılık ve F-Ölçütü verilmektedir.

Kötü olarak etiketlenen sınıflaryazılımcılarla incelendiğinde sınıfların hata çözülme sürelerinin uzun olduğu gözlenmiştir.

Yazılımcıların programlama dili alanındaki tecrübelerinin ve uygulama alanındaki bilgi birikimlerinin analiz edilebilirlik için önem taşıdığı kötü olarak seçilen sınıfların ortak özelliği olarak gözlenmiştir.

Çizelge 3.7: Test Edilebilirlik için Başarım Ölçütü

TP	FP	Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
0.863	0.134	%85.44	%83.70	%98.00	%90.28

Test edilebilirlik için Doğruluk ölçütü %85.44, Kesinlik %83.70 ve Duyarlılık %98.00 olarak bulunmuştur. F-Ölçütü ise %90.28 olarak tespit edilmiştir.

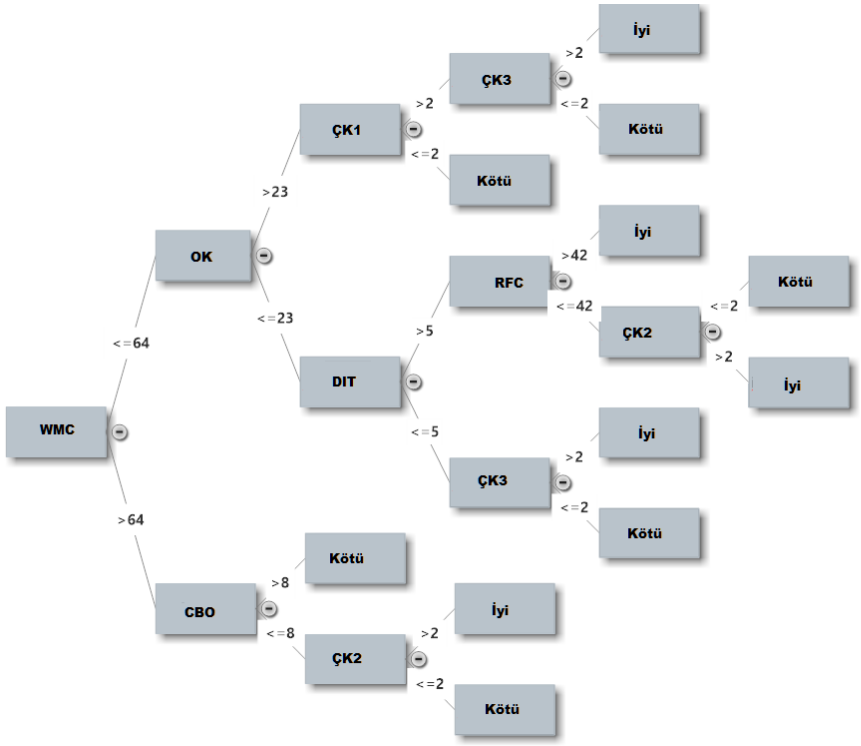
3.4.4 Karar Ağacı Yöntemi ile Değiştirilebilirliğin Değerlendirmesi

Değiştirilebilirlik için oluşturulan Karar Ağacı

Şekil **3.29**' da verilmiştir. Karar ağacı incelendiğinde değiştirilebilirlik için belirlenen metrik kümesi içinden NOC, IFANIN(Count of Base Class), ODK (Tüm iç içe metotlar için karmaşıklık sayısı) değerlerinin elendiği gözlenmiştir. OK (Tüm metotlar için karmaşıklık sayısı) alındığında ODK değerine anlaşılabilirlik için ihtiyaç olmadığı belirlenmiştir.

Nesneye dayalı metrik kümeleri ile çalışan kriterlerinin birlikte değerlendirildiğinde,çalışan kriterlerinin ağaçta önemli yer teşkil ettiği belirlenmiştir.

Nesneye dayalı metriklerden CBO ve WMC'nin önemli yer taşıdığı gözlenmiştir. Eğer WMC değeri 64'ten ve CBO değeri 8'den büyükse sınıf değiştirilebilirlik açısından kötü olarak nitelendirilmiştir.



Şekil 3.30 : Değiştirilebilirlik için Karar Ağacı

Değiştirilebilirlik için yapılan sınama sonuçları Çizelge 3.8' de verilmiştir. İyi ve Kötü sütunları sınama setindeki yeniden kullanılabilirlik için iyi ve kötü olarak etiketlenmiş sınıfları temsil etmektedir.

İyi ve kötü kümelerindeki doğru pozitif (TP – true positive), yanlış pozitif (FP – false positive) değerleri gösterilmiştir. Çizelgede Kesinlik, Duyarlılık ve F-Ölçütü verilmektedir.

Kötü olarak etiketlenen sınıflar incelendiğinde yazılımcıların programla dili hâkimiyetinin düşük olduğu, sınıflarınbirbirine çok bağımlı sınıflar olduğu ve sınıflardaki metod sayılarının çok olduğu gözleniyor.

Çizelge 3.8 : Değiştirilebilirik için Başarım Ölçütü

TP	FP	Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
0.878	0.121	%88.63	%86.58	%96.04	%91,06

Değiştirilebilirlik için Doğruluk ölçütü %88.63, Kesinlik %86.58 ve Duyarlılık %96.04 olarak bulunmuştur. F-Ölçütü ise %91,06 olarak tespit edilmiştir.

Karar ağacı yönteminde, belirlenen alt karakteristiklerin ilişkilerine bakıldığında, analiz edilebilirlik ve test edilebilirlik için oluşturulan ağaçlarda benzer metrikler kullanıldığı ve ağaç yapılarının bir birine yakın olduğu gözlenmiştir. Sınıflar değerlendirildiğinde anlaşılabilir olarak nitelendirilen bir sınıfın, test edilebilir olma olasılığı %81 olarak görülmüştür.

Benzer şekilde yeniden kullanılabilirlik ve değiştirilebilirlik için oluşturulan ağaç yapıları birbirine benzemektedir. Yeniden kullanılabilir olarak belirlenen bir sınıf %78 oranında değiştirilebilir olarak etiketlenmiştir.

3.4.5 Modelin Naive Bayes Yöntemi ile Değerlendirmesi

Oluşturulan modelin Naive Bayes yöntemine göre; yeniden kullanılabilirlik için başarım ölçütü Çizelge 3.9'da verilmiştir.

Çizelge 3.9 : Yeniden Kullanılabilirlik Başarım Ölçütü

Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
%87,21	%84,14	%92,42	%88,08

Yeniden kullanılabilirlik için doğruluk değeri %87.21, kesinlik değeri 84.14, duyarlılık 92.42 olarak bulunmuştur. F-ölçütü %88.08 olarak tespit edilmiştir.

Analiz edilebilirlik için başarım ölçütü Çizelge 3.10'da verilmiştir.

Çizelge 3.10: Analiz Edilebilirlik için Başarım Ölçütü

Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
%86,43	%86,16	%92,76	%89,33

Analiz edilebilirlik için doğruluk değeri %86.43, kesinlik değeri 86.16, duyarlılık 89.33 olarak bulunmuştur. F-ölçütü %89.16 olarak tespit edilmiştir.

Test Edilebilirlik için başarım ölçütü Çizelge 3.11'de verilmiştir.

Çizelge 3.11 : Test Edilebilirlik için Başarım Ölçütü

Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
%88,24	%84,87	%96,83	%90,45

Test edilebilirlik için doğruluk değeri %88.24, kesinlik değeri %84.87, duyarlılık %96.83 olarak bulunmuştur. F-ölçütü %90.45 olarak tespit edilmiştir.

Değiştirilebilirlik için başarım ölçütü Çizelge 3.12'de verilmiştir.

Çizelge 3.12: Değiştirilebilirik için Başarım Ölçütü

Doğruluk	Kesinlik	Duyarlılık	F-Ölçütü
%86.39	%89.20	%92.58	%90,85

Değiştirilebilirlik için doğruluk değeri % 86.39, kesinlik değeri %89.20, duyarlılık değeri %92.58 olarak bulunmuştur. F-ölçütü %90.85 olarak tespit edilmiştir.

Sonuçlar analiz edildiğinde, analiz edilebilirlik kalite kriterinin belirlenmesinde Naive Bayes yönteminin daha başarılı olduğu gözlenmiştir. Yeniden kullanılabilirlik, Test edilebilirlik ve değiştirilebilirlik kriterlerinin belirlenmesinde ise karar ağacı yönteminin daha başarılı sonuçlar verdiği gözlemlenmiştir.

Yapılan incelemelerde projede yeniden kullanılabilirliğin, anlaşılabilirliğin, test edilebilirliğin ve değiştirilebilirliğin yazılımcılardan alınan geribildirimlere uygun olduğu gözlemlenmiştir.

Çalışma hazırlanırken yapılan analizelerde çalışanların alan bilgisinin ve yazılım geliştirmedeki tecrübelerinin programın bakım kolaylığını etkilediği gözlenmiştir. Pakette bulunan hataların çözülme süresinin ise paketteki sınıfların anlaşılabilirliği üzerinde etkisi olduğu sonucuna varılmıştır. Yapılan değişiklikle birlikte hataların çözülme sürelerinin azaldığı, programın hatalarının anlaşılması için geçen eforun azaldığı, bunun da süre ve maliyete etki ettiği ekip ile yapılan görüşmelerde de teyit edilmiştir.

Çalışmada nesneye dayalı metrikler, durumsal liderlik modelini baz alarak çalışanların değerlendirilmesi, hata sayıları ve hataların giderilmesi için geçen sürenin değerlendirilmesiyle birlikte %80'lere varan başarı ile projelerin tamamlanmadan önce bakım kolaylığının tespit edilebileceği gösterilmiştir.

4. SONUÇ VE ÖNERİLER

Gerçek dünyada yazılım projesinin tek bir metrik ile değerlendirilmesinin doğru olmadığı görülmektedir. Bu çalışmada yazılımın ölçülmesinde ISO/IEC 25010:2011 kalite modeli baz alınarak, bakım kolaylığı kalite özelliğinin projenin yaşam döngüsü süresince nasıl değiştiğini ve iyileştirilebileceğini göstermek amaçlanmıştır.

Yapılan çalışmada veri madenciliği teknikleri kullanılarak projelerde belirlenecek olan kalite özelliklerinin ilerleyen sürümlerde tahmin edilebileceği belirlenmiş ve tahminlerin gerçek dünya ile örtüştüğü gözlenmiştir. Çalışma ile birlikte projede belirlenen kalite modellerinin projenin gelişimi aşamasında ele alınarak daha kaliteli ürünler ortaya çıkması hedeflenmiştir.

Çalışmada projeyi geliştiren ekiple birlikte çalışılıp ekibin yazılım kalitesini ölçmesi konusunda fikir sahibi olması planlanmıştır. Çalışmanın sonucunda yazılım firmalarının kendilerine veri ölçme havuzu oluşturmaları, yazılımda kalite anlayışını sürecin geneline yaymaları ve projenin başlangıcından itibaren ölçmeye önem vermeleri durumunda sürekli iyileştirme yaşanarak daha kolay test edilebilen, yeni özelliklerin daha kolay eklenebildiği ve değiştirilebildiği bir yazılım elde etmelerinin sağlanmış olacağı öngörülmektedir.

Her ne kadar yazılım kalitesi yapılan bir çok çalışma tarafından somutlaştırılıp, bunlara uygun yöntemler belirlense de önemli olan bir nokta yazılım geliştiricilerin bu konuda bilgilenmesi ve yazılım geliştirme süresince buna önem vermesi gerektiğidir. Çalışmada yazılım geliştiricilerden alınan geri bildirimler içerisinde, geliştirdikleri sınıfları belirlenen kalite özelliklerine göre değerlendirdiklerinde ve bunu tekrar eden sürümlerde yaptıklarında, sınıf tasarımlarına daha önem verdikleri ve kodlamada yapılabilecek bazı hataları daha kodlama aşamasında fark ettikleri vardır.

Çalışmanın devamında farklı veri madenciliği teknikleri kullanılarak otomatik olarak analiz edebilen bilgisayar destekli yazılım aracı geliştirilmesi ve proje için belirlenen kalite özellikleri için puan vermesi amaçlanmaktadır. Böylelikle proje yürütücülerinin projenin geneline tam olarak bakabilmesine, bununla birlikte müşterinin yazılımın ne durumda olduğunu takip edebilmesine olanak sağlanması planlanmaktadır.

KAYNAKLAR

[1] **Hakim Lounis,Tamer Fares Gayed**, (2011), Mounir Boukadoum, "Using Efficient Machine-Learning Models to Assess Two Important Quality Factors:Maintainabilitiy and Reusability,21st International Workshop on Software Measurement

[2] **Tibor Bakota, Péter Heged˝us, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy**,(2011) "A Probabilistic Software Quality Model", 27th International Conference on Software Maintenance(ICSM)

[3] **Liafna Li, Hareton Leung**, (2011), "Mining Static Code Metrics for a Robust Prediction of Software Defect-Proneness", Internal Symposium on Empirical Software Engineering and Measurement,

[4] **E.Arisholm,L.C.Briand and M.J.Fuglerud**,(2007) "Data mining techniques for building fault-proneness models in telecom java software, in simula Technical Report

[5] **V.R.Basili,L.C.Briand and W.L.Melo**,(1996), "A validation of object oriented design metrics as quality indicators", IEEE Transaction on Software Engineering, vol.22,no.10,p.751-761

[6] **J.Osbeck, Waverly**, (2011),"Investigaton of Automatic Prediction of Software Quality", Fuzzy Information Processing Society Annual Meeting of the North America,

[7] **J. Ross Quinlan**, (1993) C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA

[8] **M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten**, (2009) "The WEKA Data Mining Software: An Update", SIGKDD Explorations, vol. 11, no. 1, pp. 10-18.

[9] **J. Bansiya and C.G. Davis**,(2002), "A hierarchical model for objectoriented design quality assessment", IEEE Trans. Softw. Eng., vol. 28, no. 1, pp.4-17

[10] **Shaheen Khatoon, Azhar Mahmood**,(2011) "An Evaluation of Source Code Mining Techniques", Eight International Conference on Fuzzy System and Knowledge Discovery (FSKD)

[11] **Aljahdali, Sheta** , (2011)"Predicting the Reliability of Software System Using Fuzzy Logic", Information Technology: New Generations (ITNG), 2011 Eighth International Conference

[12] **Hakim Lounis, Tamer Fares Gayed, Mounir Boukadoum**, (2011), "Machine-Learning Models for Software Quality: a Compromise Between Performance and Intelligibility", 23rd IEE E International Conference on Tools with Artificial Intelligence

[13] **E.Ceylan. (2006)**, ,"Software Defect Identification Using Machine Learning Technique", Software Engineering and Advanced Applications, 2006. SEAA '06. 32nd EUROMICRO Conference

[14] **Ali R. Sharafat, Ladan Tahvildari**,(2007),"A Probabilistic Approach to Predict Changes in Object-Oriented Software System", , Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference,

[15] **Kayarvizhy**,(2011), "Analysis of Quality of Object Oriented System using Object Oriented Metrics" , Electronics Computer Technology (ICECT), 2011 3rd International Conference

[16] **Okumoto** ,(2011),"Software Defect Prediction Based on Stability Test Data", Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2011 International Conference on

[17] **Ko-Li Cheng,**(2011),"Software Fault Detection using Program Patterns", Software Engineering and Service Science (ICSESS), IEEE 2nd International Conference on,2011

[18] **Baldassari, B., Robach, C. and du Bosquet**, (2004),"Early metrics for Object Oriented Designs", Proc. 1st Int'l. Workshop on Testability Assessment (IWoTA), pp. 62-69.

[19] **Url-1**<http://denizkilinc.com/2013/07/30/yazilim-kalite-olcumu-uzerine>,12.04.2014

[20] **DeMarco, Tom.** Controlling Software Projects: Management, Measurement and Estimation. ISBN 0-13-171711-1.

[21] **T. W. Kwan, H. Leung**,**(**2010) "A risk management methodology for project risk dependencies", IEEE Transaction on Software Engineering.

[22] **IEEE**,(1998) "IEEE Std. 1061-1988, Standard for a Software Quality Metrics Methodology, revision." Piscataway, NJ,: IEEE Standards Dept.

[23] **DavidGarvin**, "Kalite Yönetimi", 1988, s:41-45

[24] **ISO**, (2007) "ISO/IEC 15939:2007, Systems and Software engineering – Measurement process." Geneva, ISO/IEC.

[25] **Url-2**<*http://www.scitools.com/features/metrics.php*>, 09.03.2015

[26] **Ural Erdemir, Umut Tekin, Feza Buzluca**, (2008), "Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi", Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu

[27] **McCall,J.A.,Richards,P.K and Walters, G.F**.,(1997), "Factors in Software Quality",Nat'1 Tech.Information Service,no. Vol. 1,2 and 3

[28] **Url-3**< ISO "ISO/IEC 25010:2011,http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733>, 12.04.2015

[29] **Kan S.H**.,(2005), Metrics and Models in Software Quality Engineering. 2. ed. Addison Wesley

[30] **Goodman, P.**,, (1993), "The Practical Implementation of Software Metrics", McGraw-Hill, New York, USA

[31] **Kaur, J., P., Verma A. and Thapar, S**., , (2007) ,"Software Quality Metrics for Object-Oriented Environments", Proceedings of National Conference on Challenges & Opportunities in Information Technology (COIT-2007), RIMT-IET, Mandi Gobindgarh, pp.13-16

[32] **McCabe**,, (2009)."Using Code Quality Metrics in Management of Outsourced Development and Maintenance"

[33] **Gustafson, D. A**., (2002), "Theory and Problems of Software Engineering", Mc Graw Hill, USA

[34] **Chidamber, S.R. and Kemerer**,(1994) C.F., "A Metrics Suite For Object-Oriented Design" IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp.482-491,

[35] **Thirugnanam, M. and Swathi J.N**., (2010)., "Quality Metrics Tool for Object Oriented Programming", International Journal of Computer Theory and Engineering, Vol. 2, No. 5, pp.1793-8201

[36] **SPSS**, (1999), "AnwerTree Algorithm Summary", SPSS White Paper, USA,

[37] **SUN, Jie ve Hui LI**,(2008), "Data Mining Method for Listed Companies, Financial Distress Prediction", Knowledge-Based Systems, 21, No. 1, 2008.

[38] **TÜRE, Mevlut, Füsun TOKATLI, İmran KURT**, (2008), "Using KaplanMeirer Analysis Together With Decision Tree Methods (C&RT, CHAID, QUEST, C4.5 and ID3) In Determining Recurrence-Free Survival of Breast Cancer Patients", Expert Systems With Applications, Article in Pres, 2008.

[39] **Url-4**<*http://www.minitab.com/en-US/default.asp*, 10.04.2015

[40] **Larose, D. T. 2005**. Discovering Knowledge in Data: An İntroduction in Data Mining, Wiley, USA.

[41] **Linoff, G. S.. Berry, M. J. A.**,(2011), Data Mining Techniques for Marketing, Sales and Customer Relationship Management, Wiley, Canada

[42] **Marakas, G. M.**,(2003). Decision Support Systems in The 21st Century, Prentice Hall, USA

[43] **Hudairy, H**, (2004), "Data Mining and Decision Making Support in The Governmental Sector," Master Thesis, Faculty of Graduate School of The University of Louisville, Kentucky

[44] **Url-5**<https://rapidminer.com/>, 15.03.2015

[45] **Blanchard K., Zigarmı P., Zigarmı D**, (1999), "Leadership and the One Minute Manager: İncreasing Effectiveness Through Sitüational Leadership".

ÖZGEÇMİŞ

Ad Soyad:	Nurdan CANBAZ
Doğum Yeri ve Tarihi:	İstanbul 28.10.1984
Adres:	Osmanyılmaz mah. Atatürk cad. Şerefhan Apt. D: 6 Gebze/KOCAELİ
E-Posta:	nurdancanbaz@gmail.com
Lisans:	Maltepe Üniversitesi Bilgisayar Mühendisliği

Yüksek Lisans(Varsa):

Mesleki Deneyim ve Ödüller:

Maltepe Üniversitesi Mühendislik Fakültesi Birinciği Ödülü(2009)
Maltepe Üniversitesi Mühendislik Fakültesi Başarı Bursu (2006-2007-2008)
Maltepe Üniversitesi Mühendislik Fakültesi Onur Öğrenciliği (2006-2007-2008)

Yayın ve Patent Listesi:

N.CANBAZ, F.BUZLUCA, Yazılım Kalitesi için Yinelemeli Ölçme Yöntemi, Ulusal Yazılım Mimarisi Konferansı, Ege Üniversitesi, 2012

N.CANBAZ, F.BUZLUCA, A Continuous Measurement Method for Design Quality of Object-Oriented Software Systems, International Conference on Computer and Information Technology International Conference on Computer and Information Technology, Ankara Turkey,April 28-29 2015

TEZDEN TÜRETİLEN YAYINLAR/SUNUMLAR

- N.CANBAZ, F.BUZLUCA, *Yazılım Kalitesi için Yinelemeli Ölçme Yöntemi,* Ulusal Yazılım Mimarisi Konferansı, Ege Üniversitesi, 2012
- N.CANBAZ, F.BUZLUCA, *A Continuous Measurement Method for Design Quality of Object-Oriented Software Systems*, International Conference on Computer and Information Technology, Ankara Turkey,April 28-29 2015