T.C.
MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

NESNEYE YÖNELİK YAZILIM PROJELERİNDE ÖNCELİKLİ OLARAK TEST EDİLECEK SINIFLARIN YAZILIM ÖLÇÜTLERİ YARDIMIYLA BELİRLENMESİNE YÖNELİK BİR YÖNTEM

RAMAZAN MURAT DEMİRBAŞ

Yüksek Lisans Tezi

Tez Danışmanı
Yrd. Doç. Dr. Fatih Yücalar

İSTANBUL – 2014

T.C.
MALTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

NESNEYE YÖNELİK YAZILIM PROJELERİNDE ÖNCELİKLİ OLARAK TEST EDİLECEK SINIFLARIN YAZILIM ÖLÇÜTLERİ YARDIMIYLA BELİRLENMESİNE YÖNELİK BİR YÖNTEM

YÜKSEK LİSANS TEZİ

RAMAZAN MURAT DEMİRBAŞ

Tez Danışmanı
Yrd. Doç. Dr. Fatih Yücalar

İSTANBUL – 2014

ÖZET

Yüksek Lisans Tezi, Nesneye Yönelik Yazılım Projelerinde Öncelikli Olarak Test Edilecek Sınıfların Yazılım Ölçütleri Yardımıyla Belirlenmesine Yönelik Bir Yöntem, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Yazılım projelerinde ortaya çıkan hataların önceden tespit edilip düzeltilmesi öngörülen maliyeti ve proje zamanını aşma risklerini azaltır. Ortaya çıkması muhtemel hataları mümkün olduğu kadar erken tespit edebilmek için, verimli ve etkili bir test planının uygulanması gerekir. Yazılım ölçütlerinin etkili kullanılmasıyla daha erken aşamalarda kod analiz edilip, hata yatkınlığıyla ilgili fikir sahibi olunabilir, gerekirse önlem alınabilir.

Bu tez çalışmasında yazılım ölçütleri kullanılarak, öncelikli olarak test edilmesi gereken sınıfların belirlenmesine yönelik bir yaklaşım ele alınmıştır. Daha sonra bu yaklaşım ele alınan projelerin seçilen sürümleri için uygulanmış ve sonuçlar değerlendirilmiştir.

Toplam yedi bölümden oluşan tezin birinci bölümünde tez konusunun önemi ve seçilme nedeni açıklanmıştır. İkinci bölümde literatür taraması yapılmıştır. Üçüncü bölümde yazılımda ölçüm ve kalite kavramları irdelenmiş, yazılım ölçütleri ve yazılım kalitesinin iyileştirilmesi için kullanılan araçlar tanıtılmıştır. Dördüncü bölümde tez kapsamında ele alınan yaklaşımın adımları ve kullanılan ölçütler anlatılmıştır. Beşinci bölümde kullanılan projelerle ilgili bilgiler ve çalışmada elde edilen sonuçlar verilmiştir. Altıncı bölümde elde edilen sonuçlar irdelenmiştir. Yedinci bölümde sonraki çalışmalara değinilmiş, benzer konuda çalışma yapacaklara öneriler sunulmuştur.

Bu tez 2014 yılında tamamlanmıştır ve 68 sayfadan oluşmaktadır.

Anahtar Kelimeler: Yazılım Kalitesi, Yazılım Ölçütleri, Nesneye Yönelik Yazılım Testi.

ABSTRACT

Master Thesis, A Method Using Software Metrics to Determine Software Classes will be Firstly Tested in Object-Oriented Software Projects, T.C. Maltepe University, Graduate School of Science and Engineering, Department of Computer Engineering.

Early detection and correction of errors appearing in software projects reduces the estimated cost and risk of delaying the deadline. In order to detect potential errors as early as possible, an efficient and effective test plan should be implemented. By efficiently employing software criteria, codes can be analyzed at earlier phases, insight can be gained into susceptibility to errors, and measures can be taken if necessary.

In this thesis study, an approach for identifying the classes that should primarily be tested has been developed, applied to the selected versions of the software handled and results were assessed.

In the first part of the thesis composed of a total of 7 parts, the rationale underlying the selection of the thesis subject is described. The second part is dedicated to literature review. In part three, measuring and quality concepts in the software are discussed, and software measures and tools employed to improve the software quality are introduced. In part four, the steps of the approach taken and measures used throughout the thesis are described. In part five, project details and study findings are presented. In part six, the findings are analyzed. In part seven, subsequent studies are addressed accompanied by suggestions to fellows intending to study on similar concepts.

This thesis has been completed in 2014 and consists of 68 pages.

Keywords: Software Quality, Software Metrics, Object Oriented Software Testing.

TEŞEKKÜR

Tez konusunu seçmemde beni yönlendiren, tez süreci boyunca destek ve yardımlarını esirgemeyen, değerli bilgilerinden istifade ettiğim danışman hocam Yrd. Doç. Dr. Fatih YÜCALAR'a, tez sürecinde bana gösterdiği olağanüstü anlayış ve yardım için eşim Nurdan DEMİRBAŞ'a, manevi desteğini benden hiçbir zaman esirgemeyen çok değerli aileme ve çalışmalarım sırasında emeği geçen herkese teşekkürlerimi sunarım.

İÇİNDEKİLER

KISALTMALAR

Kısaltma	İngilizcesi	Türkçesi
ÖDL	Metric Values List	Ölçüt Değerleri Listesi
ÖTDL	Metric Total Values List	Ölçüt Toplam Değerleri Listesi
SVN	SubVersioN	-
QMOOD	Quality Model for Object Oriented Design	Nesne Tabanlı Tasarım Kalite Modeli
MOOD	Metrics for Object Oriented Design	Nesne Tabanlı Tasarım Ölçütleri
ISO	International Organization for Standardization	Uluslararası Standartlar Örgütü
SPICE	Software Process Improvement and Capability dEtermination	Yazılım Süreç Geliştirme ve Kapasite Belirleme
CMMI	Capability Maturity Model Integration	Tümleşik Yetenek Olgunluk Modeli
NASA	National Aeronautics and Space Administration	Ulusal Havacılık ve Uzay Dairesi
XML	eXtensible Markup Language	Genişletilebilir İşaretleme Dili
HTML	Hyper Text Markup Language	Zengin Metin İşaret Dili
SQL	Structured Query Language	Yapısal Sorgulama Dili
DIT	Depth of Inheritance Tree	Kalıtım Ağacının Derinliği
NOC	Number of Children	Alt Sınıf Sayısı
CBO	Coupling Between Object Classes	Nesne Sınıfları Arasındaki Bağımlılık
RFC	Response For a Class	Sınıfın Tetiklediği Metot Sayısı
LCOM	Lack of Cohesion in Methods	Metotların Uyumsuzluk Yüzdesi
MHF	Method Hiding Factor	Metot Gizleme Faktörü
AHF	Attribute Hiding Factor	Nitelik Gizleme Faktörü
MIF	Method Inheritance Factor	Metot Kalıtım Faktörü
AI	Attribute Inheritance Factor	Nitelik Kalıtım Faktörü
PF	Polymorphism Factor	Çok Biçimlilik Faktörü
CF	Coupling Factor	Bağımlılık Faktörü

DSC	Design Size in Classes	Sınıf Tasarımlarının Büyüklüğü
NOH	Number Of Hierarchies	Hiyerarşi Sayısı
ANA	Average Number of Ancestors	Ortalama Ata Sayısı
DAM	Data Access Metric	Veri Erişim Ölçütü
DCC	Direct Class Coupling	Doğrudan Sınıf Bağımlılığı
CIS	Class Interface Size	Sınıf Arayüz Boyutu
MOA	Measure Of Aggregation	Kümeleme Ölçüsü
CAM	Cohesion Among Methods	Metotlar Arası Uyumluluk
MFA	Measure of Functional Abstraction	İşlevsel Soyutlama Ölçüsü
NOP	Number Of Polymorphic methods	Polimorfik Metot Sayısı
NOM	Number Of Methods	Metot Sayısı
NIV	Number of Instance Variables	Nesne Değişkenlerinin Sayısı
WMC	Weighted Methods per Class	Ağırlıklı Metot Sayısı

ŞEKİLLER

DENKLEMLER

ÇİZELGELER

1 GİRİŞ

Günümüz yazılım dünyasında, yazılım test yaklaşımları, yazılım kalitesi ve nesneye dayalı yazılım ölçütleri en çok çalışılan konular haline gelmiştir. Bunun başlıca nedeni, yazılımların büyümesiyle birlikte yazılımlarda ortaya çıkan hataların ve beraberinde bu hataları düzeltme maliyetlerinin artmasıdır.

Yazılım projeleri gereksinimlerin belirlenmesi, analiz, tasarım, kodlama, entegrasyon ve test süreçlerinden oluşur. Bu yazılım geliştirme adımları ardışıl olarak birbirini izler, ancak yazılım sınamaları, yazılım projelerinin tüm geliştirme sürecine yayılmalıdır. V modeli, sınama sürecinin yazılımın her safhasında yer aldığı ve her safhaya karşılık bir sınama adımının olduğu bir yazılım geliştirme modelidir [1]. Bu model, yazılım sınama sürecini; projenin geliştirilmesinin başından itibaren, her safhada yapılacak sınamaların tasarımının ve hazırlıklarının yapılması ile uygular. Planlanmış olan bu sınamalarla, yazılımın kalitesi kontrol edilmiş olur. Kaynak kodda ya da tasarımda ortaya çıkabilecek olası hatalar mümkün olan en erken safhada belirlenir.

Yazılım projelerinin öngörülen sürede, öngörülen bütçeyle bitirilmesi ve teslim alındığı gibi kullanılması amaçlanır. Bu kıstaslarda yaşanan olumsuzluklar yazılım test yaklaşımları, yazılım kalitesi ve nesneye dayalı ölçütlerin günümüzde en çok çalışılan konular haline gelmesinde başlıca etmendir.

Yazılım projelerinde hata takibi için çeşitli hata takip yazılımları kullanılır. Tespit edilen hatalar, hata takip sistemi üzerinden ilgililere atanır. Atamanın ardından hatanın durumu

bu sistem üzerinden takip edilir. Hata, bulunduğu modül, tarih gibi bilgilerle de kayıt edilebilir.

Yazılım projelerinde ortaya çıkan hataların önceden tespit edilip düzeltilmesi öngörülen maliyeti ve zamanı aşma riskleri açısından çok önemlidir. Bu riskleri en aza indirmek için yazılım hatalarının incelenmesi son derece kritiktir.

Yazılımlardaki hataların tespit edilme zamanları, bunların giderilmesi için harcanan zamanı ve bakım maliyetlerini doğrudan etkilemektedir. Bir yazılım, ürün haline dönüştükten sonra bulunan hataların düzeltilme maliyeti oldukça yüksek iken, yazılımın geliştirilmesi süresince yapılan testler ile erken belirlenen hatalar bu maliyetleri oldukça azaltmaktadır.

Yazılım geliştirme maliyetinin yarısından fazlasını yazılım bakım maliyetleri oluşturmaktadır. Yazılım sınamaları, yazılımın kaynak kodundaki ya da tasarımındaki hataların yazılımın erken fazlarında bulunması ve bakım maliyetlerinin düşürülmesinde büyük öneme sahiptir. Ayrıca yazılımlar; yeni özellik ekleme, kalite iyileştirme ve hata düzeltme gibi farklı sebeplerle gelişim süreleri boyunca sürekli değişime uğramaktadır. Her bir değişimden sonra yazılımın bütünlüğünü ve kararlılığını korumak için yazılım testlerinin yapılması gereklidir. Ancak, yazılım testleri, yazılım yaşam döngüsünün en çok zamana ve kaynağa ihtiyaç duyan faaliyetidir [2], [3].

Zamanı ve kaynakları daha etkin kullanabilmek için yazılım geliştiricilerin, test uzmanlarının ve yöneticilerin yazılımlarda öncelikli olarak test edilmesi gereken kritik kısımları belirlemesi gereklidir. Bir yazılım projesindeki tüm birimlerin ve

işlevselliklerin sınanması çok maliyetlidir, hatta bazen de zaman ve maliyet açısından mümkün değildir.

Ayrıca yazılımda kalitesi yüksek ve istikrarlı kısımlarını uzun süre test edilmesi kaynakların verimsiz kullanılmasına neden olur; çünkü bu zaman ve kaynaklar kalitesi daha düşük kısımların daha fazla test edilmesi ve yeniden düzenlenmesi için kullanılabilir. Özellikle, kısıtlı zaman ve kaynak altında hangi kısımların öncelikli test edilmesi gerektiği yönünde yapılacak öneriler, kaynakların ve zamanın daha etkin kullanılmasına yardımcı olur.

2 LİTERATÜR ÖZETİ

Bu bölümde yazılım kalitesinin nesneye dayalı ölçütler ile ilişkilendirilerek incelendiği araştırmalara yer verilmiştir.

Wang, yaptığı çalışmada [4], bakım safhasında kullanılan ölçütlerin doğruluğunu kanıtlamak için bir ölçüt sistemi kullanılmıştır. Bakım safhasında sıklıkla kullanılan ölçütleri karşılaştırmışlardır. İş yükünü ölçmek için bir denklem kullanılmış ve bu denklemde pek çok değişken tanımlanmıştır. Ağırlıklandırılmış ölçütlerin ve ölçüt modellerinin kullanımı analiz edilmiş, ölçütlerin karmaşıklıkları detaylı olarak tartışılmıştır. Yazılım bakımı için bir ölçüt sistemi oluşturulmuştur. Bu sistemdeki ölçütler, yazılım bakım organizasyonlarına adapte edilmiş ve kalite yönetimi problemlerini çözmek için kullanılabileceği kanıtlanmıştır.

Kaur ve arkadaşları yaptıkları çalışmada [5], hataya eğilimli noktaları proje kodlanmaya başlamadan ortaya çıkararak ve bu noktaları deneyimli çalışanların sorumluluğuna vererek, daha kaliteli yazılımlar ortaya çıkarılabileceğini ve müşteri memnuniyetinin artırılabileceğini savunmuştur. Projelerdeki hataya eğilimli/hataya eğilimli olmayan modüllerin tahmin edilmesi için "Bulanık C Means Kümeleme Tekniği" kullanılmıştır.

Raymond ve arkadaşları yaptıkları çalışmada [6], kod okunabilirliğini incelemişler, yazılım kalitesi ile olan ilişkisini araştırmışlardır. Okunabilirlik konusunda, 120 kişiden veri toplayarak, yerel kod özelliklerinin basit bir kümesi ile insan fikirleri arasındaki ilişkiler ortaya konulmuştur. Bu özellikleri kullanarak otomatik bir okunabilirlik ölçüsü oluşturmuşlar ve bunu kullanarak yapılan ölçüm sonucunun ortalama bir insanın okunabilirlik konusundaki görüşüne göre %80 daha etkili olduğunu göstermişlerdir.

Ayrıca, bu ölçütün kod değişiklikleri, otomatik hata raporları, hata log mesajları olmak üzere üç yazılım kalite göstergesiyle yüksek oranda ilişkisi olduğunu göstermişlerdir. Bu ilişkileri seçilmiş projelerin bir çok sürümünü kapsayan 2.2 milyon satır kod ile ölçmüşlerdir. Son olarak, bu çalışmalarından programlama dili tasarımı ve mühendislik pratiğiyle ilgili yapılabilecek çıkarımları irdelemişlerdir. Bu çalışmada elde edilen çıkarımlara göre kodda yer alan boş satırların, yorum satırlarına göre okunabilirliği arttırdığını savunmuşlardır.

Ogasawara ve arkadaşları yaptıkları çalışmada [7], geçtiğimiz yıllarda yazılım ürünleri ve süreçlerini anlamak için pek çok yazılım kalite ölçütü önerildiğine değinmişlerdir. Gerçek projelerde kalite ölçütlerinin, yazılım kalitesini yönetmek için kullanıldığını savunmuşlardır. Bununla birlikte, yazılım ürünleri ve yazılım geliştirme süreçlerine etkili bir geribildirim sağlamakla ilgili problemlerin olduğunu belirtmişlerdir.

Kolayca ve otomatik olarak ölçülebilecek kalite ölçütlerini kullanarak bir yazılım kalite yönetim sistemi önermişlerdir. Yazılım kalitesi oluşturmak için düzenli olarak ölçen ve inceleyen bir yöntem bulabilmeyi amaçlamışlardır. Çalışmalarında kalite ölçütlerini kullanarak yazılım kalitesini ölçen bir yöntem ortaya koymuşlar, bu yöntemi gerçek projelere uygulamışlar ve sonuçlarına yer vermişlerdir. Sonuç olarak yazılım ürün ve süreçlerinin her safhası için kalite ölçütlerinin kullanılarak, problemleri bulmak ve ortadan kaldırmak için kullanabileceğini göstermişlerdir. Ayrıca, test safhasında önerdikleri modelle hızlı bir şekilde düzeltici önlem alınabileceğini göstermişlerdir.

Sistem tasarımının değişebilirliği ve değişimin büyümeye olan etkisi de araştırma konusu olmuştur. M. Chaumun ve diğerleri yaptıkları çalışmada [8] sistem tasarımının değişebilirliğe etkisini incelemek üzere bir değişim etkisi modeli ortaya koymuşlardır. Yazılımda yapılan değişimin, yazılımdaki hangi kısımları etkilediğini bulmak; yazılımın

değişimden sonra da kararlı ve doğru çalışmasını sağlamak için önemlidir. Bu sebeple; ana odakları, sistemin bir değişimi nasıl karşılayacağı olmuştur. Yazılım projeleri birbiri ile etkileşimleri olan yazılım bileşenlerinden (sınıf) oluşur ve tasarım açısından sınıflar arası ilişkiler, sınıf içi değişimlerden daha çok etkiye sahiptir. Geliştirdikleri model, dilden bağımsız olarak tasarım üzerine dayandırılmıştır.

Değişimleri üç ana bölümde ele almışlardır: niteliklerin eklenmesi, silinmesi, tipinin ve etki alanının değişmesi; metotların eklenmesi, silinmesi, gerçeklemesinin, imzasının ve etki alanının değişmesi; sınıfların eklenmesi, silinmesi ve yapısının değişmesi. Sınıflar arası ilişkileri; bir sınıfın diğer bir sınıfa ait referansa sahip olması (association), bir sınıfın tanımının diğer bir sınıfın nesnesinin olmasına bağlılığı (aggregation), türeme (inheritance) ve bir sınıfın diğerini tetiklemesi (invocation) olarak dört şekilde ve sınıf içi değişikliklerin etkisi tanımlamak için sanal bir bağlantı (local) tanımlamışlardır. Farklı tipteki değişimler farklı sınıf kümelerini etkiler. Örneğin, bir değişkenin üzerinde yapılan değişim ona bağlı olan sınıfları etkilerken, eğer bu değişiklik bir ata sınıfta yapılırsa en azından türeyen tüm sınıfları etkiler. Ayrıca değişim yapıldığı bağlantının tipine göre de bu etki değişmektedir. Örneğin bir sınıfın metodu public (+) iken protected (#) yapılırsa, bu değişiklikten bu metodu çağıran tüm sınıflar etkilenirken, türeyen sınıflar etkilenmez. Verilen bağlantılara göre değişimden etkilenecek olan sınıf kümeleri çıkarılmıştır.

Çalışmanın deneysel kısmında, C++ ile yazılmış ve 1044 sınıfa sahip endüstriyel bir uygulamada, sadece metodun parametre değişimine bağlı olan değişimin etkisini araştırmışlardır. Analiz kısmındaki performans problemlerinden dolayı uygulamanın tüm sınıfları yerine örneklenen kümeler üzerinde çalışmışlardır. Örneklemede üç grup oluşturulmuştur: 1, 2 metodu olan sınıflar, 3 ile 29 arasında metodu olan sınıflar ve 30'dan fazla metodu olan sınıflar. Deneysel çalışmada sınıfın Ağırlıklı Metot Sayısı

(Weighted Method Count - WMC) ölçütü, sınıfın metot sayısına eşit olacak şekilde kullanılmıştır ve metodun parametre değişimine bağlı olan değişim etkisi ile sınıfın WMC ölçütü arasındaki korelasyon araştırılmıştır. Elde edilen sonuçlara göre korelasyon en yüksek 0.55 olarak bulunmuştur ve düşük olduğu ifade edilmiştir.

Yazılımın gelişim sürecindeki ölçütlerin değişimlerini analiz eden ve bu analizlerden projelerin ilerleyen aşamalarındaki kalite değişimi hakkında bilgi çıkarmak üzerine yapılmış çalışmalar da vardır.

Young Lee ve diğerleri tarafından yapılan çalışmada [9], fan-in/out bağımlılık ve uyum ölçütleri ile yazılımın gelişim davranışlarını anlamak için açık kaynak kodlu JFreeChart yazılımı analiz edilmiştir. Yazılım ölçütlerini ve kalite özelliklerini çıkarmak için JamTool adında bir yazılım geliştirmişlerdir. Yaptıkları deneysel çalışmada incelenen yazılımdaki sınıf sayısı artısı ile bağımlılık ve uyum ölçütleri arasındaki ilişkiyi incelemişlerdir. Ayrıca eklenen ve silinen sınıfların yazılımın bağımlılık ve uyum ölçütleri üzerindeki değişimini araştırmışlardır. Silinen sınıfların düşük kalite özelliklerine sahip olduklarını bu nedenle bağımlılıklarının yüksek, uyumlarının düşük olmasını; eklenen sınıfların ise daha kaliteli olacaklarını bu nedenle bağımlılıklarının düşük, uyumlarının yüksek olmasını beklemektedirler. JFreeChart yazılımının ardışıl sürümlerindeki sınıf sayıları ve ortalama fan-in/out bağımlılık ölçütleri ve uyum ölçütlerini çıkararak incelemişlerdir. Elde edilen sonuçlar bağımlılık ölçütlerinin sınıf sayısı ile ilişkili olduğunu göstermiştir. Uyum ölçütü için ise negatif korelasyon bulunmuştur. Ayrıca yeni eklenen sınıfların, silinen sınıflara göre fan-in ölçütleri değerleri yüksek, fan-out ölçüt değerleri ise daha düşüktür. Bu da tekrar kullanılabilirlik için istenen bir durumdur. Yapılan çalışmada, ölçütlerin yazılım değişim döngüsündeki durumları incelenmiş, elde edilen bilgiler kullanılarak yazılımın geçtiği aşamalar hakkında fikir sahibi olunabildiği gösterilmiştir.

Sürüm denetim sistemleri incelenerek, ilerleyen sürümlerindeki hata sayısını tahmin etmek ve hatanın çıkabileceği modülleri belirlemek üzerine yapılan araştırmalar da vardır.

Kastro ve Bener, yazılımın yeni sürümündeki hata sayısını eski sürümlerine göre olan değişiklikleri dikkate alarak tahmin etmek için, yapay sinir ağlarını kullanan bir yöntem [10] önermişlerdir. Yazılıma eklenen değişiklikler: eklenen bir özellik, algoritma değişikliği, hata ayıklama olabilir. Bunların yanı sıra kodun boyutundaki hacimsel değişiklikleri de dikkate alarak, sürümde çıkabilecek hata sayısını doğru olarak tahmin etmeye çalışmışlardır. Yapay sinir ağlarında kullanılmak üzere iki farklı eğitim seti tanımlamışlardır. İlk set, bir sürümdeki düzeltilen hata sayısı, eklenen özellik sayısı, tamamlanan değişiklik sayısı ve boyut olarak önceki sürümle farkıdır. İkinci set ise; eklenen satır sayısı, silinen satır sayısı, değiştirilen satır sayısı ve boyut olarak önceki sürümle olan farktan oluşmaktadır. Bu veriler kullanılarak incelenen yazılımların önceki sürümlerinde yapay sinir ağları eğitilmiş ve incelenen sürümde çıkacak hata sayısı belirlenmeye çalışılmıştır.

Li ve Leung araştırmalarında, hataya yatkınlığın bulunması için denetimsiz bir öğrenme modeli geliştirmişlerdir [11]. Çalışmalarındaki ana düşünce, aynı ölçüt kümelerinde bulunan bileşenlerin benzer hata yatkınlıklarına sahip olmasıdır. Kullandıkları veri kümesi NASA'dan sağlanan 12 farklı projeye [12] ait hata kayıtlarını ve kaynak kodları içermektedir. Elde edilen veriler üzerinde ön işlem yapılarak veriler normalleştirilmiş ve ölçüt değerleri hesaplanmıştır. "En Yakın Komşu" algoritması kullanılarak oluşturdukları model ile hata yatkınlıklarının bulunmasını amaçlamışlardır.

Veri madenciliği teknikleri kullanılarak yapılan bir diğer çalışma ise yazılım kalitesinin tahmin edilebilmesi amacını taşımaktadır [13]. Bu çalışmayı yapan araştırmacılar, daha

önce C4.5 [14] algoritmasını kullanarak yaptıkları çalışmayı J48 [15] algoritmasını kullanarak yapmışlardır. Araştırmacılar, proje kalitesinin tahmin edilmesi için Nesne Tabanlı Tasarım için Kalite Modeli (Quality Model for Object Oriented Design – QMOOD) [16] kalite modelini tercih etmişlerdir. QMOOD modelinin yapısında bulunan "Çok Biçimlilik, Karmaşıklık, Kalıtım" vb. için "İyi, Kötü, Çok İyi" şeklinde tanımlayıcılar atamış ve Weka [17] açık kaynak kodlu veri madenciliği aracını kullanarak tahminler yapmışlardır. Araştırmada yazılım ürününün sürümleri arasında bir kıyaslama yapmamışlar, tek bir sürüm üzerinde inceleme yapmışlardır.

Literatürde ölçütlerin gruplamasına yönelik çalışmalar da bulunmaktadır. Ölçütlerin gruplanması ve farklı projelerin sonuçlarının incelenmesine yönelik çalışmalardan biri C++' da nesneye dayalı ölçütlerin incelenmesine yöneliktir. Bu çalışmada iki proje nesneye dayalı ölçütler aracılığı ile karşılaştırılmıştır. Karşılaştırılan projelerden biri bir kolejin 9 sınıftan oluşan kütüphane yönetim sistemi, diğeri 8 sınıftan oluşan grafik editör yazılımıdır [18].

Hata bulmaya yönelik bir çalışma da 2 milyon satırlık kodun test verileri ile yapılmıştır [19]. Bu çalışmada, müşteriye teslim edilmeden önce yapılan sağlamlık ve kararlılık testlerinde bulunan hatalı veriler kullanılıp, ileride oluşacak hataların tahmin edilmesi amaçlanmıştır. Araştırmacılar, uzun süren testlerden önce hatayı kısa sürede tahmin eden bir model geliştirmek için "poison" dağılımından faydalanmışlardır.

3 YAZILIM KALİTESİ

3.1 Yazılımda Ölçüm Kavramı

Toplam kalite yönetiminin üretim bantlarında kullanılmasıyla, endüstriyel ürünlerde kalitenin arttığı görülmüştür. Toplam kalite yönetimi, süreçlerin iyileştirilmesini amaçlamaktadır. Süreç, girdi alıp çıktı veren işlemler bütünüdür. Toplam kalite yönetiminin temel prensibi, iyi tanımlanmış süreçlerden kaliteli ürünler çıktığıdır. Mevcut süreçlerin de sürekli iyileştirilmesi sağlanarak daha kaliteli ürünler elde edilmesi amaçlanmaktadır. Toplam kalite prensiplerini temel alan SPICE [20], CMMI [21], ISO 12207 [22] gibi standartlar yazılım şirketleri tarafından kullanılmaktadır.

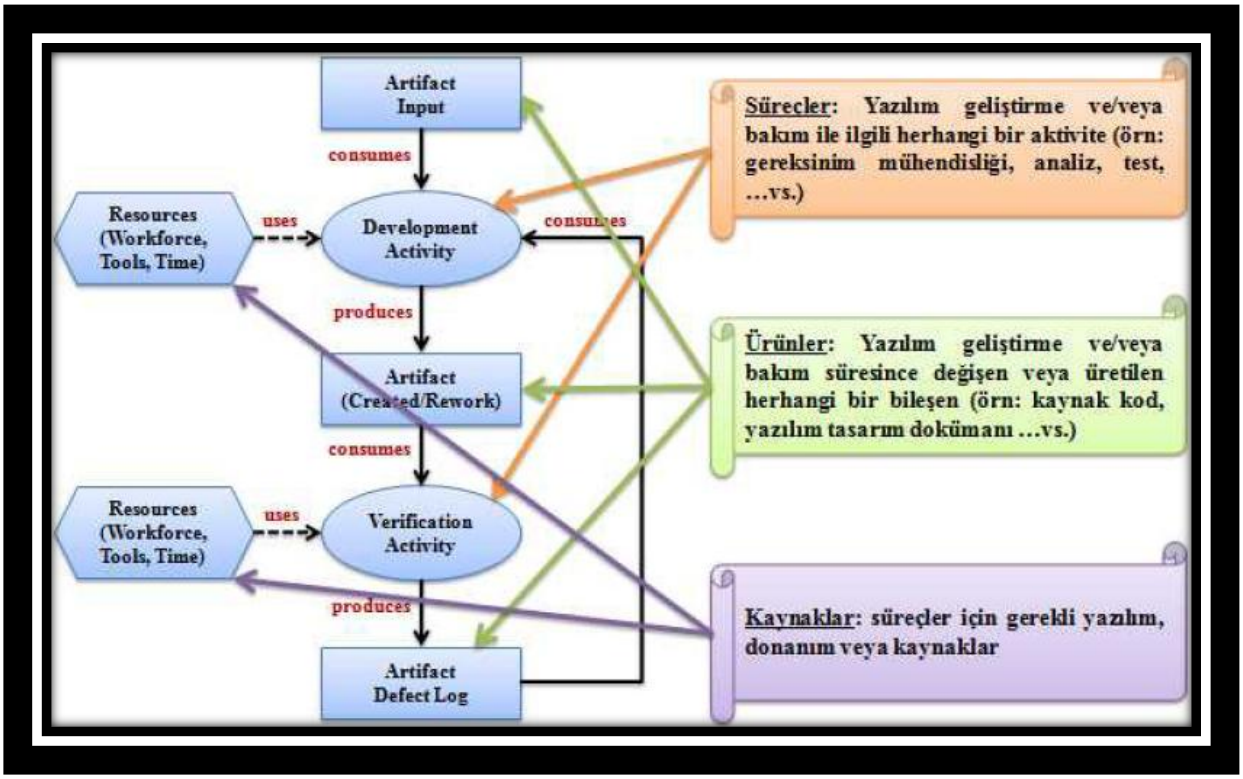
Sürekli iyileştirme ilkesinin temelinde sürecin planlanması, denetlenmesi, çıktılarının ve performansının ölçülmesi ve sürecin değerlendirilmesi vardır. Bir süreç içerisinde birçok ara ürün de oluşturulabilir. Yazılım geliştirme sürecini düşünürsek müşteri gereksinimleri, analiz dokümanı, tasarım dokümanı, yazılım kodu, test sonuçları çıktı olarak düşünülebilir.

Sürecin değerlendirilmesi ve iyileştirilmesi için süreç sırasında elde edilen ölçümler kullanılır.

Tanımlanmış kurallara göre ölçme (measurement), varlıkların (entities - objects) belli bir niteliğe (attribute) sahip olup olmadığını, sahip ise; sahip oluş derecesini gözleyip, gözlem sonuçlarını daha çok sayısal sembollerle ifade etmeye denir [23].

Ölçme adımında, ölçülecek nitelik diğerlerinden ayırt edilir ve ölçülecek niteliğe uygun bir ölçme birimi veya aracı bulunur.

Yazılım ölçümünde bir varlığın (entity) neler olabileceği Şekil 3.1'de verilmektedir.



Şekil 3.1 Yazılım Ölçümünde Varlıklar [23]

Nitelik (attributes), varlığın bir özelliğidir. İki genel nitelik vardır;

- İç nitelikler, sadece varlığın kendisi temel alınarak ölçülebilmesidir [23]. Örnek; varlık: kod – iç nitelik: büyüklük, modülerlik, modüller arası bağlantılar.
- Dış nitelikler, varlığın kendi çevresi ile nasıl ilişki kurması gerektiğine göre ölçülebilmesidir. Örnek; varlık: kod – dış nitelik: güvenilirlik, bakım kolaylığı.

Yazılım ölçümü, yazılım süreçlerinin anlaşılması, kontrol edilmesi ve yönetilmesi için gerekli bir süreçtir. Yazılım ölçümü yazılım hataların analiz edilmesini, karmaşıklığının azaltılmasını, süreçlerin izlenmesini ve değerlendirilmesini sağlar. 5 temel yazılım ölçütü vardır [24]:

- Büyüklük (size),
- Emek (effort),
- Süre (duration),
- Maliyet (cost),
- Kalite (quality).

Her yazılım projesinin temel hedefi, müşterinin ihtiyaçlarını karşılayan, öngörülmüş bütçe ile zamanında teslim edilen hatasız bir yazılım geliştirmektir [24]. 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.

Yazılımın ölçülebilmesi, harcanılan zaman, emek, proje büyüklüğü ve kalite gibi faktörlerin belirlenmesine olanak sağlamaktadır.

Yazılım projelerinde ölçütler yardımıyla, yazılımla ilgili sayısal veriler elde edilebilir. Bu veriler, proje yöneticilerine yazılımın gelişimiyle ilgili kritik bilgiler sunar. Proje yöneticileri, önlem alınması gereken riskleri tespit edip, bunlarla ilgili çalışma yapabilir. Ayrıca proje planlarını bu verileri kullanarak güncelleyebilir. Yazılım bakımı ve sınamalarında ortaya çıkacak sonuç tablosunu bilgi edinir. Gerekiyorsa iyileştirme planları/çalışmaları yapabilir. Yazılım projelerinde ölçüm sonuçları, proje yöneticilerine, daha etkin ve daha az riskli bir proje yönetimi için yardımcı olur.

3.2 Yazılımda Kalite Kavramı

Crosby, yazılımda kaliteyi gereksinimlere uygunluk olarak tanımlar [25]. Ancak kaliteli bir yazılımın öngörülen bütçe ve öngörülen sürede tamamlanması gerekmektedir. Müşteri gereksinimlerini karşılayan ancak bütçesini kabul edilemeyecek oranda aşan ya da kabul edilemeyecek kadar daha uzun sürede tamamlanan yazılım kaliteli olarak kabul görmemektedir. Yazılım açısından kalitenin ölçülebilmesi için bazı teknikler geliştirilmiştir. Bu teknikler yazılımın bazı özellikleri sayısallaştırarak ölçmek ve ölçüm sonuçlarını yorumlamak/değerlendirmek suretiyle yazılım kalitesi hakkında fikir elde etmek prensibine dayanır.

Yazılım kalitesi çeşitli sınıflar halinde kategorize edilebilir. ISO 9126'ya [26] göre kalite sınıfları şunlardır:

İşlevsellik: Yazılımın, tanımlanan ihtiyaçları karşılamak üzere bir araya gelmiş olan özelliklerdir. Uygunluk, doğruluk, birlikte çalışabilirlik, güvenlik konuları işlevsellik kategorisinde incelenir.

Güvenilirlik: Yazılımın, düzgün çalışma halini koruyabilmesi olarak tanımlanır. Olgunluk, hata toleransı ve geri kurtarma konuları güvenilirlik kategorisinde incelenir.

Kullanılabilirlik: Yazılımın, kullanım kolaylığı sağlamak için kolay öğrenebilme, anlaşılabilme özelliklerini ifade eder. Öğrenebilme, anlaşılabilirlik, işletilebilirlik ve kullanıcı etkileşimi konuları kullanılabilirlik kategorisinde incelenir.

Verimlilik: Yazılımın, yeterli performansla çalışabilme becerisi olarak tanımlanır. Zaman ve kaynak kullanımı konuları verimlilik kategorisinde incelenir.

Bakım Yapılabilirlik: Yazılımın, yeni isteklere uyum sağlayabilmesi ve değişiklik/düzeltme yapmaya cevap verebilme yeteneği olarak tanımlanır.

Değiştirilebilirlik, sınanabilirlik, analiz edilebilirlik konuları bakım yapılabilirlik kategorisinde incelenir.

Taşınabilirlik: Yazılımın, farklı çalışma ortamlarına uyum sağlayabilme yeteneği olarak tanımlanır. Adaptasyon yeteneği, yüklenebilirlik özellikleri, ortam değiştirme imkânı ve diğer yazılımlarla uyum konuları taşınabilirlik kategorisinde incelenir.

ISO 9126 standardı kalite kavramını ürün tabanlı olarak müşteri gözünden ele alır. Yazılımı geliştirenler açısından yazılımın iç özelliklerinin kalitesi ön plandadır. Bu kapsamda yazılım ister toplama aşamasından, tasarım ve geliştirme aşaması boyunca yazılım kalitesinin ölçülmesi gerekir. Günümüzde yazılım geliştirmede çoğunlukla nesneye dayalı yaklaşım kullanılır. Nesneye dayalı yazımlarda en temel bileşen sınıftır. Sınıflar arası bağımlılıklar, sınıfların ve metotlarının uyumu, sınıfların karmaşıklığı, sınıfların büyüklüğü en temel kalite göstergeleridir. Bu göstergeleri ölçmek için ölçütler tanımlanmıştır. Ölçütler, gerekli ölçümler yapıldıktan sonra yorumlanarak karar vermek için kullanılır.

3.3 Yazılım Ölçütleri

Yazılım ölçütlerini, bilgilerin toplanma zamanına, ölçmede kullanılan bilgilere ve ürettikleri verinin tipi ve aralığına göre sınıflandırmak mümkündür.

- **Bilgilerin toplanma zamanına göre yazılım ölçütleri;**

Statik ve dinamik olarak iki temel sınıfa ayrılır. Statik ölçütler yazılım çalıştırılmadan elde edilen bilgileri kullanır. Dinamik ölçütler ise yazılımın çalışması esnasında elde edilen bilgileri kullanır.

- **Ölçmede kullanılan bilgilere göre yazılım ölçütleri;**

Bazı ölçütler sadece parametre erişimine bakarken, bazıları metotların tüm veri erişimini dikkate alır.

- **Ürettikleri verinin tipi ve aralığına göre yazılım ölçütleri;**

Ölçütler [0,+∞] aralığında, sonlu aralıkta gerçel ya da tam sayı değerler, sınırlı aralıkta gerçel sayılar olarak değer üretebilirler.

Nesneye dayalı yazılım ölçütleri olarak yaygın olarak kullanılan üç ölçüt kümesi vardır. Bunlar Chidamber & Kemerer, MOOD ve QMOOD ölçüt kümeleridir [27].

3.3.1 Chidamber & Kemerer (CK) ölçüt kümesi

Chidamber & Kemerer ölçüt kümesinde 6 temel ölçüt bulunmaktadır ve tanımları aşağıda verilmiştir [28].

- **Sınıfın Ağırlıklı Metot Sayısı – Weighted Methods per Class (WMC):** Bir sınıftaki tüm metotların karmaşıklıklarının toplamıdır. Sınıfın geliştirilmesine ve bakımına ne kadar zaman harcanacağının kestirilmesinde yardımcı olur.

- **Kalıtım Ağacının Derinliği – Depth of Inheritance Tree (DIT):** Bir sınıfın, kalıtım ağacının köküne uzaklığı olarak tanımlanır. Türetilmemiş sınıflar için bu ölçütün değeri 0'dır. Çoklu kalıtım söz konusuysa ölçütün değeri en uzak köke olan uzaklıktır.

- **Alt Sınıf Sayısı – Number of Children (NOC):** Bir sınıftan doğrudan türetilmiş alt sınıfların sayısı olarak tanımlanır. Çok alt sınıfı olan sınıfların metotlarını daha çok test etmek gerekir. Bu ölçüt ilgili sınıfı test etmek için harcanacak bütçeye karar verilmesi açısından kullanılabilir.

- **Nesne Sınıfları Arasındaki Bağımlılık – Coupling Between Object Classes (CBO):** Bir sınıfın bağımlı olduğu sınıf sayısıdır. Bir sınıf içindeki metotlar ya da nitelikler başka bir sınıfta da kullanılıyorsa ve sınıflar arası katılım yoksa bu

iki sınıf arasında bağımlılık vardır. Yüksek bağımlılık bakım zorluğu anlamına gelir. Ayrıca tekrar kullanılabilirliği de azaltır. Yüksek bağımlılık daha çok test edilmeyi gerektireceğinden test maliyetlerini de arttırır.

- **Sınıfın Tetiklediği Metot Sayısı – Response For a Class (RFC):** Sınıftaki bir nesnenin metotları çağrıldığında tetiklenebilecek tüm metotların sayısıdır. Bir mesajın çok sayıda metodu çağırması test maliyetinin artması, hata ayıklamanın zorlaşması demektir.
- **Metotların Uyumsuzluk Yüzdesi – Lack of Cohesion in Methods (LCOM):** P hiçbir ortak nitelik değişkeni paylaşmayan metot çiftlerinin kümesi, Q en az bir ortak nitelik değişkeni paylaşan çiftlerin kümesi olursa, {|P|>|Q| ise |P||Q| ; aksi halde 0} değerini alan ölçüttür. Metotların uyumluluğu düşükse sınıfın alt bileşenlere ayrılması gerekir. Düşük uyumluluk karmaşıklığı artıracağından geliştirme aşamasında hata yapma riski artar. Sınıfların tasarımdaki hatalar da bu ölçütle öngörülebilir.

3.3.2 MOOD ölçüt kümesi

MOOD (Metrics for Object Oriented Design – Nesne Tabanlı Tasarım Ölçütleri) ölçüt kümesi, nesneye dayalı yöntemin kapsülleme, kalıtım, çok biçimlilik, mesaj aktarımı gibi mekanizmalarını ele alır [29].

- **Metot Gizleme Faktörü – Method Hiding Factor (MHF):** Kalıtımla gelen metotlar kullanılmaksızın; tüm sınıflardaki çağrılabilir metotların, tüm metotlara oranı olarak tanımlanır. Bu ölçütle sınıfın görünürlüğü ölçülür.

- **Nitelik Gizleme Faktörü – Attribute Hiding Factor (AHF):** Kalıtımla gelen nitelikler kullanılmaksızın; tüm sınıflardaki erişilebilir niteliklerin, tüm niteliklere oranı olarak tanımlanır. Bu ölçütle de sınıfın görünürlüğü ölçülür.

- **Metot Kalıtım Faktörü – Method Inheritance Factor (MIF):** Tüm sınıflardaki kalıtım ile gelen metot sayısının, tüm metotların sayısına oranı olarak tanımlanır.

- **Nitelik Kalıtım Faktörü – Attribute Inheritance Factor (AI):** Tüm sınıflardaki kalıtım ile gelen niteliklerin, tüm niteliklere oranı olarak tanımlanır.

- **Çok Biçimlilik Faktörü – Polymorphism Factor (PF):** Bir C sınıfı farklı çok şekilli durumların, en fazla olası çok şekilli durumlara oranı olarak tanımlanır.

- **Bağımlılık Faktörü – Coupling Factor (CF):** Kalıtımla gelen bağımlılıklar kullanılmaksızın; sınıflar arasındaki bağımlılık sayısının, oluşabilecek en fazla bağımlılık sayısına oranıdır.

3.3.3 QMOOD ölçüt kümesi

QMOOD (Quality Model for Object Oriented Design – Nesne Tabanlı Tasarım Kalite Modeli) ölçütleri, yazılımın toplam kalite endeksini hesaplamak için tanımlanmıştır [30]. Dört aşamadan oluşan hiyerarşik bir modeli vardır:

- **Yazılım Kalite Nitelikleri (Attribute)**

  QMOOD yazılım kalite nitelikleri işlevsellik, verimlilik, anlaşılabilirlik, genişleyebilirlik, yeniden kullanılabilirlik ve esnekliktir. Bu kalite niteliklerinin hesaplanma formülü Çizelge 3.1 de verilmiştir [31].

Çizelge 3.1 QMOOD Kalite Nitelikleri Hesaplama Formülleri

Kalite Nitelikleri	İndeks Hesaplama Denklemi
Yeniden Kullanılabilirlik	-0.25 * Bağımlılık + 0.25 * Uyumluluk + 0.5 * Mesajlaşma + 0.5 * Tasarım Büyüklüğü
Esneklik	0.25 * Kapsülleme - 0.25 * Bağımlılık + 0.5 * Birleştirme + 0.5 * Polimorfizm
Anlaşılabilirlik	-0.33 * Soyutlama + 0.33 * Kapsülleme - 0.33 * Bağımlılık + 0.33 * Uyumluluk - 0.33 * Polimorfizm - 0.33 * Karmaşıklık - 0.33 * Tasarım Büyüklüğü
İşlevsellik	0.12 * Uyumluluk + 0.22 * Polimorfizm + 0.22 * Mesajlaşma + 0.22 * Tasarım Büyüklüğü + 0.22 * Hiyerarşi
Genişletilebilirlik	0.5 * Soyutlama - 0.5 * Bağımlılık + 0.5 * Miras + 0.5 * Polimorfizm
Verimlilik	0.2 * Soyutlama + 0.2 * Kapsülleme + 0.2 * Birleştirme + 0.2 * Kalıtım + 0.2 * Polimorfizm

- **Nesneye Dayalı Yazılım Özellikleri (Property)**

Bu aşamadaki ölçütler katılım, kapsülleme, çok biçimlilik, soyutlama, bağımlılık, mesajlaşma, hiyerarşi, yazılım büyüklüğü ve karmaşıklıkla ilgili fikir verir.

- **Nesneye Dayalı Yazılım Ölçütleri**

  QMOOD ölçüt kümesinde on bir ölçüt yer almaktadır. Bu ölçütler aşağıda açıklanmıştır:

  - **Sınıf Tasarımlarının Büyüklüğü – Design Size in Classes (DSC):** Sınıfların toplam sayısı olarak tanımlanmıştır.

- **Hiyerarşi Sayısı – Number Of Hierarchies (NOH):** Sınıf hiyerarşilerinin toplam sayısı olarak tanımlanmıştır.
- **Ortalama Ata Sayısı – Average Number of Ancestors (ANA):** Tüm sınıfların DIT (Kalıtım Ağacının Derinliği) değerlerinin ortalaması olarak tanımlanıştır. Yazılımda soyutlamanın kullanımını gösterir.
- **Veri Erişim Ölçütü – Data Access Metric (DAM):** Sınıfın "private" ve "protected" niteliklerinin tüm niteliklere oranı olarak tanımlanmıştır. Yazılımın kapsülleme özelliği hakkında fikir verir.
- **Doğrudan Sınıf Bağımlılığı – Direct Class Coupling (DCC):** Bir sınıfı parametre olarak kullanan sınıfların sayısı ile bu sınıfı nitelik değişkeni olarak bulunduran sınıfların sayısının toplamı olarak tanımlanmıştır. Sınıfın bağımlılığı hakkında fikir verir.
- **Sınıf Arayüz Boyutu – Class Interface Size (CIS):** Sınıfın "public" metotlarının sayısı olarak tanımlanmıştır. Yazılımda mesajlaşma özelliğiyle ilgili fikir verir.
- **Kümeleme Ölçüsü – Measure Of Aggregation (MOA):** Kullanıcı tarafından tanımlanan sınıf bildirimlerinin, int, char gibi tanımlı sistem veri tiplerine oranı olarak tanımlanmıştır.
- **Metotlar Arası Uyumluluk – Cohesion Among Methods (CAM):** Metotların imzaları arasındaki benzerliğin ölçüsü olarak tanımlanmıştır.
- **İşlevsel Soyutlama Ölçüsü – Measure of Functional Abstraction (MFA):** Tüm türetilmiş metot sayısının, tüm metot sayısına oranı olarak tanımlanmıştır. Kalıtım özelliği hakkında fikir verir.
- **Polimorfik Metot Sayısı – Number Of Polymorphic Methods (NOP):** Polimorfik metotların toplam sayısıdır.
- **Metot Sayısı – Number Of Methods (NOM):** Sınıfta tanımlanmış tüm metotların sayısı olarak tanımlanmıştır.

QMOOD yazılım ölçütlerinin hangi yazılım özelliği ile ilgili ölçüm yaptığı Çizelge 3.2'de verilmiştir [31].

Çizelge 3.2 QMOOD Yazılım Ölçütü ve Ölçtüğü Yazılım Özelliği Tablosu

Tasarım Özelliği	Türetilmiş Tasarım Ölçütü
Tasarım Büyüklüğü	DSC
Hiyerarşi	NOH
Soyutlama	ANA
Kapsülleme	DAM
Bağımlılık	DCC
Uyumluluk	CAM
Birleştirme	MOA
Kalıtım	MFA
Polimorfizm	NOP
Mesajlaşma	CIS
Karmaşıklık	NOM

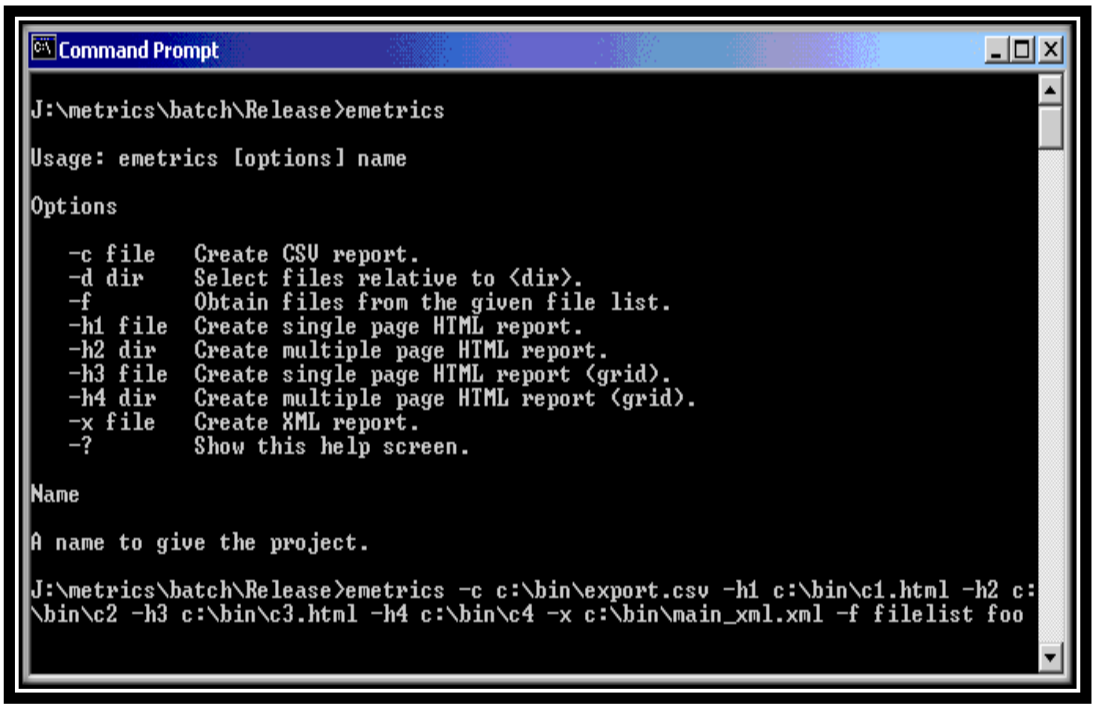
- **Nesneye Dayalı Yazılım Bileşenleri (Components)**

  Nesneye dayalı yazılım bileşenleri nitelik, metot, sınıf, ilişkiler ve sınıf hiyerarşisini içerir.

3.4 Yardımcı Araçlar

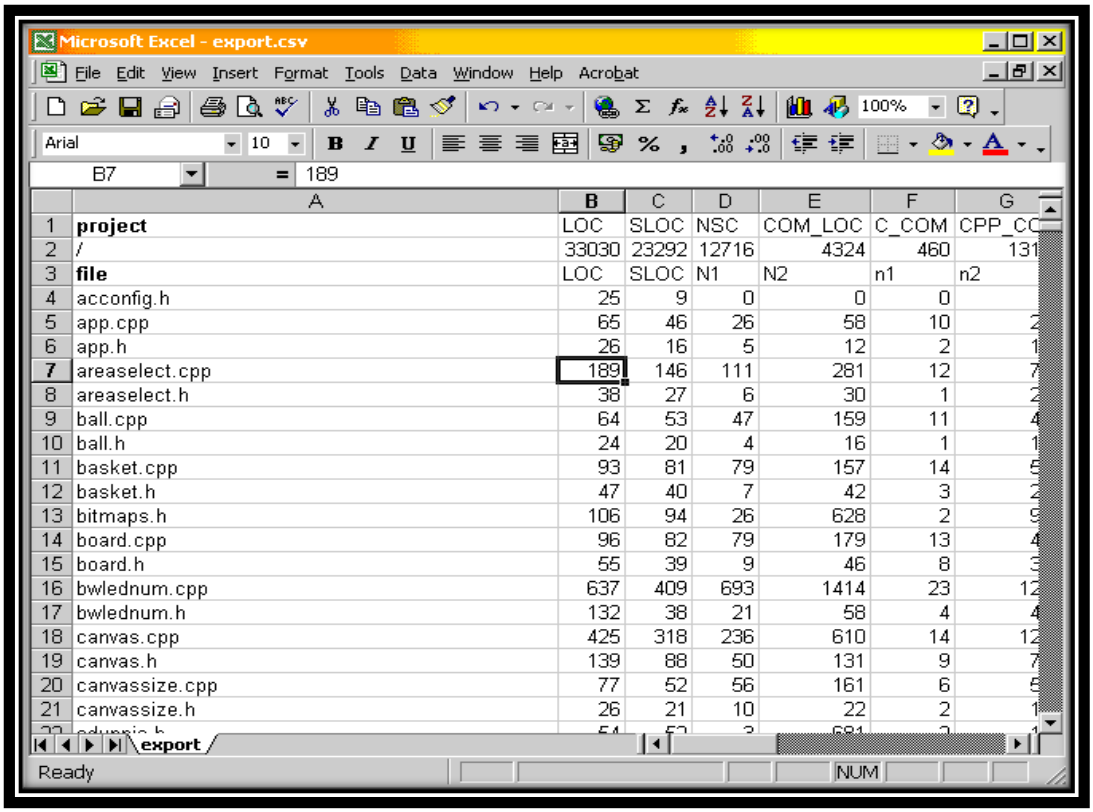
Bu bölümde nesneye dayalı yazılımların kalite çalışmalarında kullanılabilecek bazı açık kaynak kodlu ve ticari araçlar tanıtılmıştır. Bu araçlar yazılım kalitesinin iyileştirilmesi için kullanılacak ölçümleri otomatik ve hızlı bir şekilde yapabilirler.

Bu bağlamda tanıtılacak ilk program olan Essential Metrics [32], C/C++ ve Java dillerinde kullanılabilen bir komut satırı ölçüt ölçüm aracıdır. Bu aracın ekran görüntüsü Şekil 3.2'de görülmektedir.



Şekil 3.2 Essential Metrics Aracı Kullanıcı Arayüzü

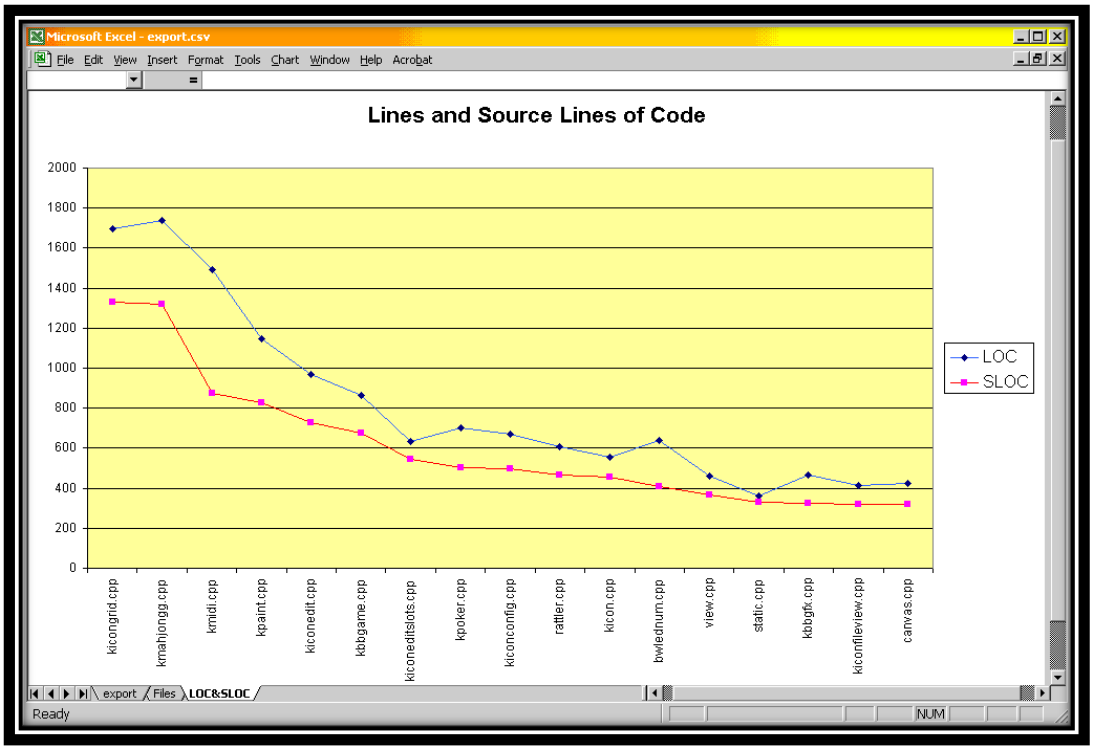
Bu araç, 5 kullanıcıya kadar ücretsiz olarak kullanılabilen ticari bir araç olup, kullanıcıya XML, Excel ve HTML formatlarında sonuçları Şekil 3.3'te görüldüğü gibi verebilmektedir.



	A	B	C	D	E	F	G
1	**project**	LOC	SLOC	NSC	COM_LOC	C_COM	CPP_CC
2	/	33030	23292	12716	4324	460	131
3	**file**	LOC	SLOC	N1	N2	n1	n2
4	acconfig.h	25	9	0	0	0	
5	app.cpp	65	46	26	58	10	2
6	app.h	26	16	5	12	2	1
7	areaselect.cpp	189	146	111	281	12	7
8	areaselect.h	38	27	6	30	1	2
9	ball.cpp	64	53	47	159	11	4
10	ball.h	24	20	4	16	1	1
11	basket.cpp	93	81	79	157	14	5
12	basket.h	47	40	7	42	3	2
13	bitmaps.h	106	94	26	628	2	9
14	board.cpp	96	82	79	179	13	4
15	board.h	55	39	9	46	8	3
16	bwlednum.cpp	637	409	693	1414	23	12
17	bwlednum.h	132	38	21	58	4	4
18	canvas.cpp	425	318	236	610	14	12
19	canvas.h	139	88	50	131	9	7
20	canvassize.cpp	77	52	56	161	6	5
21	canvassize.h	26	21	10	22	2	1

Şekil 3.3 Essential Metrics Aracının Ürettiği Bir Rapor Örneği

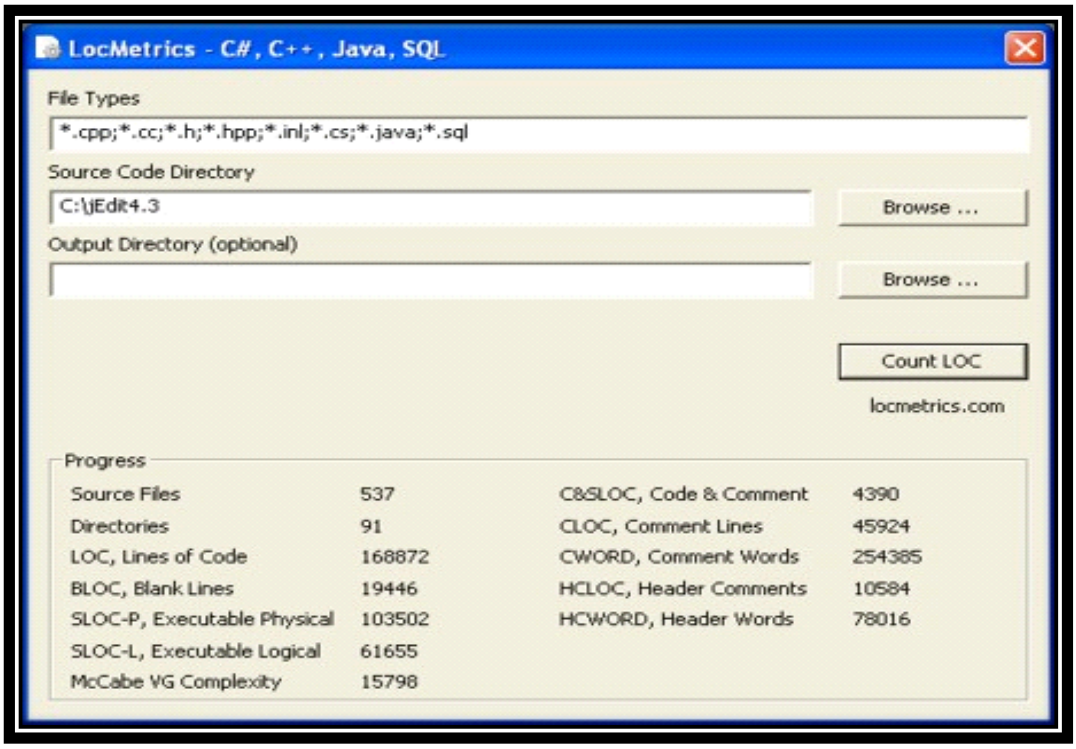
Ayrıca ölçüm sonuçlarını Şekil 3.4'te görüldüğü gibi çeşitli grafiksel yöntemlerle de sunabilmektedir.



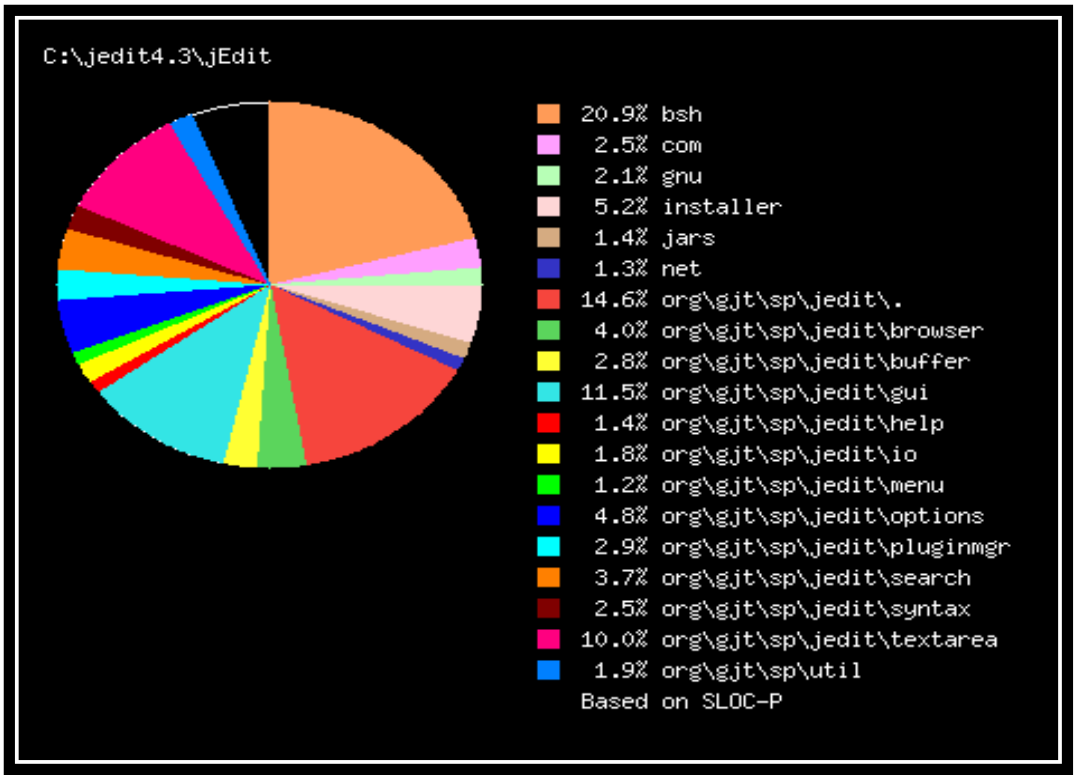
Şekil 3.4 Essential Metrics Aracının Ürettiği Grafiksel Bir Rapor Örneği

Essential Metrics AVPATHS, C_COM, COM_LOC, COM_RAT, EOL_COM gibi proje (project) ölçütlerini; C_COM, COM_LOC, COM_RAT, EOL_COM gibi dosya (file) ölçütlerini; LOC, SLOC, TLOC gibi sınıf (class) ölçütlerini; ANION, BRANCH, CDENS, CONTROL, EXEC, LLOC, LOC, NION, NSTAT, RLOC, SLOC gibi metot (method) ölçütlerini ölçebilmektedir.

Şekil 3.5'te görülen LocMetrics [33] LOC, BLOC, CLOC, C&SLOC, SLOC-L, MVG, CWORDS gibi ölçütleri ölçebilen ve ücretsiz olarak kullanılabilecek bir ölçüt ölçüm aracıdır. C#, C++, Java, SQL dillerinde yazılmış programlarla uyumlu olarak çalışabilmektedir. Şekil 3.6'da görüldüğü üzere hesaplanan ölçütün yüzdesel olarak dağılımını da grafiksel olarak sunabilmektedir.

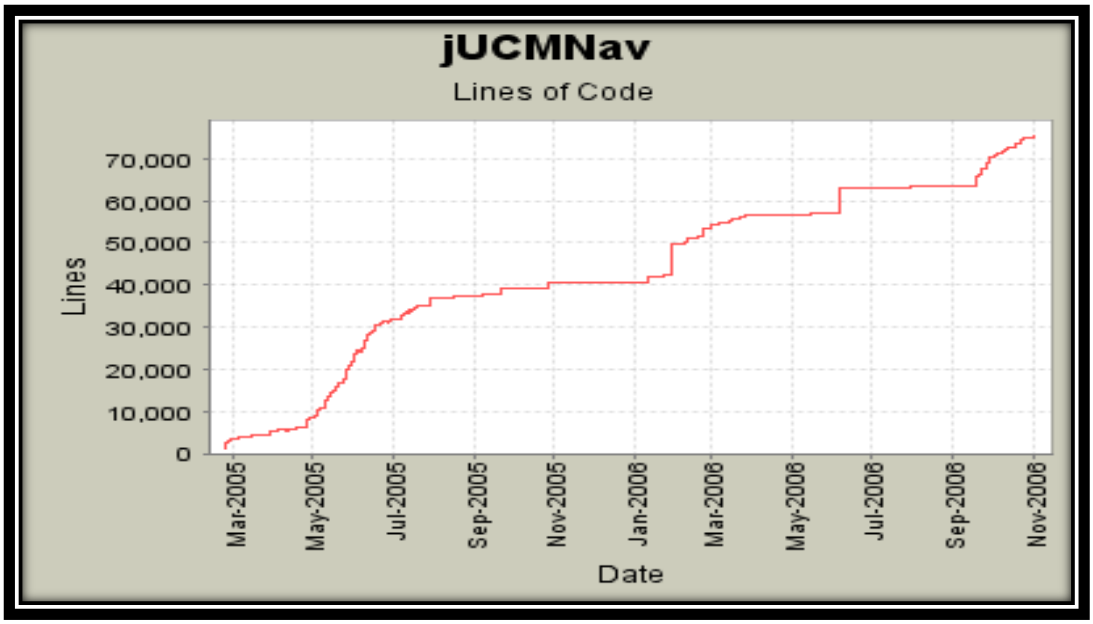


Şekil 3.5 LocMetrics Aracı Kullanıcı Arayüzü



Şekil 3.6 LocMetrics Aracının Ürettiği Bir Rapor Örneği

StatSVN [34] bir konfigürasyon yönetim aracı olan Subversion'un deposundan verileri çekip, projenin gelişimiyle ilgili çeşitli tablo ve grafikler üreten bir araçtır. Bu ekran görüntüsü Şekil 3.7'de verilmektedir. StatSVN açık kaynak kodlu bir araç olup, en son versiyonu HMTL ve XDOC formatında rapor üretebilmektedir.

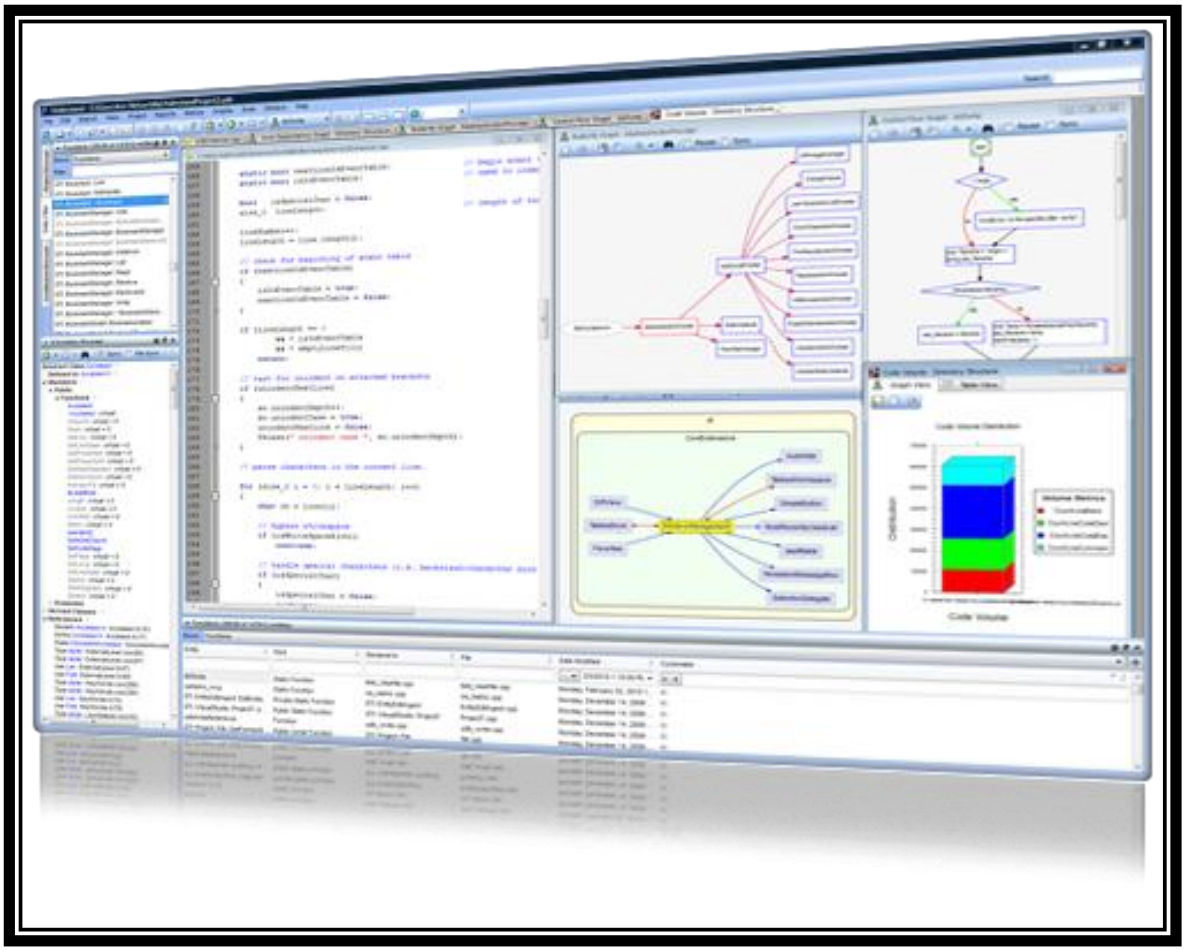


Şekil 3.7 StatSVN Aracının Ürettiği Bir Rapor Örneği

Coverity [35], ticari ve oldukça kapsamlı bir kod analiz aracıdır. Java, C++ ve C dilleriyle uyumlu olarak çalışabilmektedir. Analiz sonuçlarının yanında bazı otomatik düzeltmeler de yapabilmektedir. Yazılım dünyasında maliyeti yüksek ve kapsamlı projelerde kullanılmaktadır. Bu araç, çalışma sırasında dinamik analizleri de yapabilmektedir.

SDMetrics [36], kod üzerinde değil UML tasarım dokümanları üzerinde çeşitli görsel ve sayısal analizler yapabilen ticari bir araçtır. Bu araç, geliştirme başlamadan yazılım tasarımını analiz ederek bağımlılık ve karmaşıklığa dair birçok uygunsuzluğu ortaya koyabilmektedir. Bu noktada yaptığı erken tespitlerle maliyet açısından büyük kazanımlar sağlanmaktadır.

Şekil 3.8'de görülen Understand [37], kodun kritik kısımlarının veya tamamının bakımı, ölçümü veya analizi için kullanılan statik bir analiz aracıdır. C, C++, C#, Objective C, Ada, Java, Pascal, Cobol, JOVIAL, VHDL, FORTRAN, PL/M, Python, PHP, HTML, CSS, JavaScript, XML dilleriyle uyumlu çalışabilen bu araç, 20'den fazla da grafik sunmaktadır.



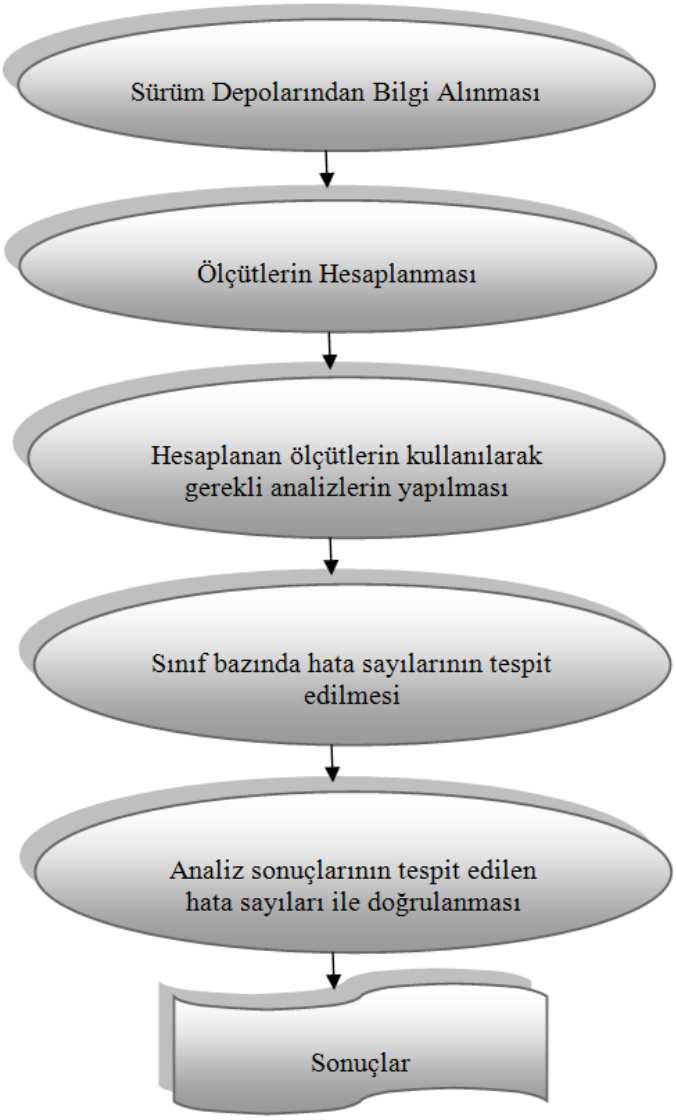
Şekil 3.8 Understand Aracı Kullanıcı Arayüzü

4 YAKLAŞIM

Bu bölümde tez kapsamında yapılan çalışmanın adımları ve kullanılan ölçütler, bu ölçütler ile sınıfların hatalılık ilişkisi anlatılmaktadır. Ayrıca hataların sınıflandırılması için benimsenen yaklaşım ele alınmıştır.

4.1 Çalışmanın Adımları

Tez kapsamında yapılan çalışmanın adımları Şekil 4.1 görüldüğü biçimde genel olarak, sürüm depolarından bilgilerin alınması, ölçütlerin hesaplanması, hesaplanan ölçütlerin kullanılarak gerekli analizlerin yapılması, sınıf bazında hata sayılarının tespit edilmesi, analiz sonuçlarının tespit edilen hata sayıları ile doğrulanması adımlarından oluşmaktadır.



Şekil 4.1 Çalışmanın Adımları

4.1.1 Sürüm depolarından bilgi alınması

Bu adımda SVN deposundan ilgili projeye ait yeteri kadar olgunlaşmış bir sürüm alınır.

4.1.2 Ölçütlerin hesaplanması

Bu adımda projeye ait sınıflar için belirlenen ölçütler Understand aracı kullanılarak ölçülür.

4.1.3 Hesaplanan ölçütlerin kullanılarak gerekli analizlerin yapılması

Bu adımda sınıfların ölçüt değerleri ve geliştirilen formül kullanılarak, her sınıf için ölçüt toplamı değeri hesaplanır. Hesaplanan ölçüt toplamı değerleri en yüksek değerden başlayarak sıralanır. Sıralama sonucu en yüksek ölçüt toplamı değerine sahip sınıflar (toplam sınıf sayısının %10'u olacak şekilde) öncelikli olarak test edilmesi gereken sınıflar olarak işaretlenir.

4.1.4 Sınıf bazında hata sayılarının tespit edilmesi

Bu adımda proje hata takip sisteminde konfigürasyon öğesi bazında tanımlanmış olan hataların hangi sınıfa ait olduğu analiz edilip, her bir sınıfa ait toplam hata sayısı değeri bulunur. Bu adımda yapılan çalışmada hata takip sisteminde tanımlı olan hatalardan donanım, üçüncü parti gibi herhangi bir yazılım sınıfı ile ilgili olmayan hatalar dikkate alınmayarak, doğrulama sonuçlarının daha sağlıklı olması amaçlanmıştır. Ayrıca hata takip sisteminde yalnızca müşteri ve proje ekibinin birlikte gerçekleştirdiği testler sonucu oluşturulmuş hatalar değerlendirmeye alınarak yine doğrulama sonuçlarının daha sağlıklı olması amaçlanmıştır.

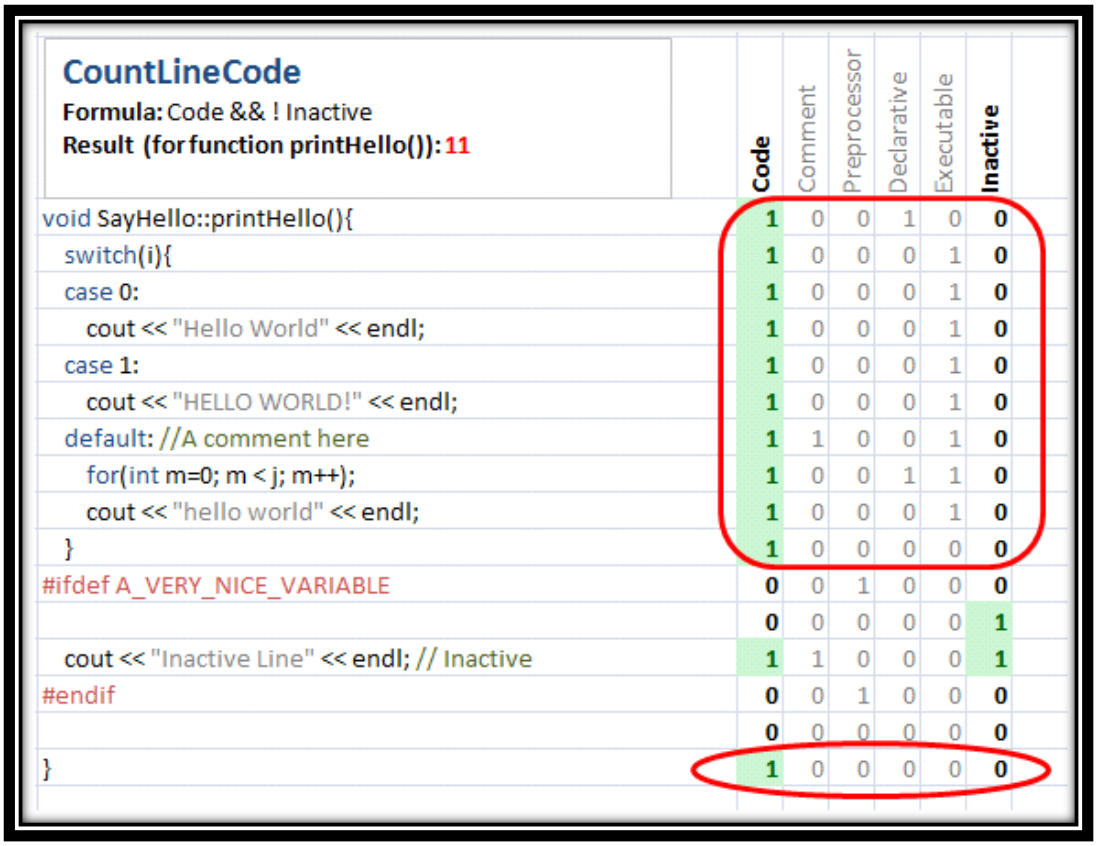
4.1.5 Analiz sonuçlarının tespit edilen hata sayıları ile doğrulanması

Bu adımda öncelikli olarak test edilmesi gerektiği belirlenen sınıflar ile en çok hatası olan sınıfların örtüşme oranı hesaplanır.

4.2 Kullanılan Ölçütler

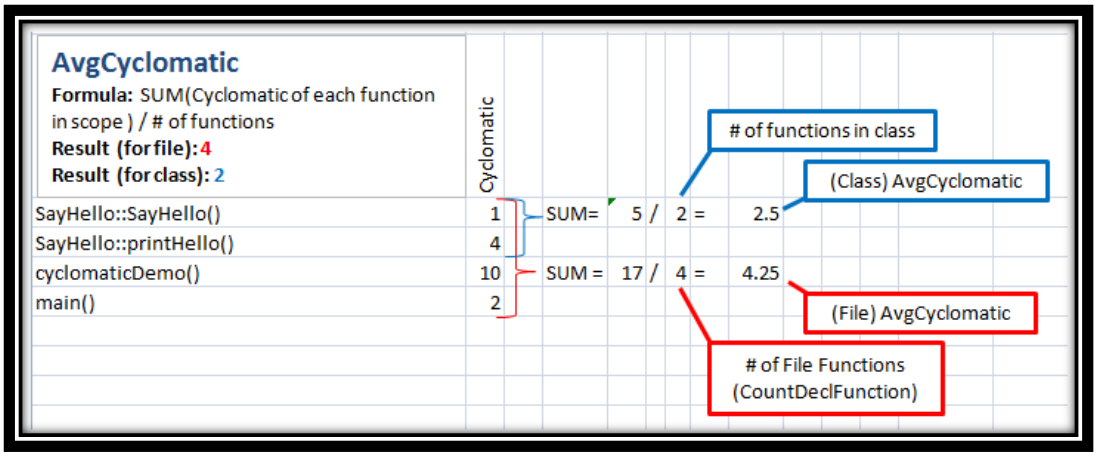
Öncelikli olarak test edilecek sınıfların belirlenmesi için toplam yedi adet ölçüt seçilmiştir. Bu ölçütlerin sınıfların hata yatkınlıkları açısından seçilme nedeni ve nasıl hesaplandığı aşağıda açıklanmıştır:

Lines of Code (Kod Satır Sayısı): Bir sınıftaki yorum olmayan ve boş olmayan satırların toplam sayısıdır. Bu değerin yüksek olması sınıfın karmaşık olma ve hataya açık olma ihtimalini artırmaktadır. Lines of Code ölçütü Şekil 4.2'de görüldüğü gibi hesaplanmaktadır:



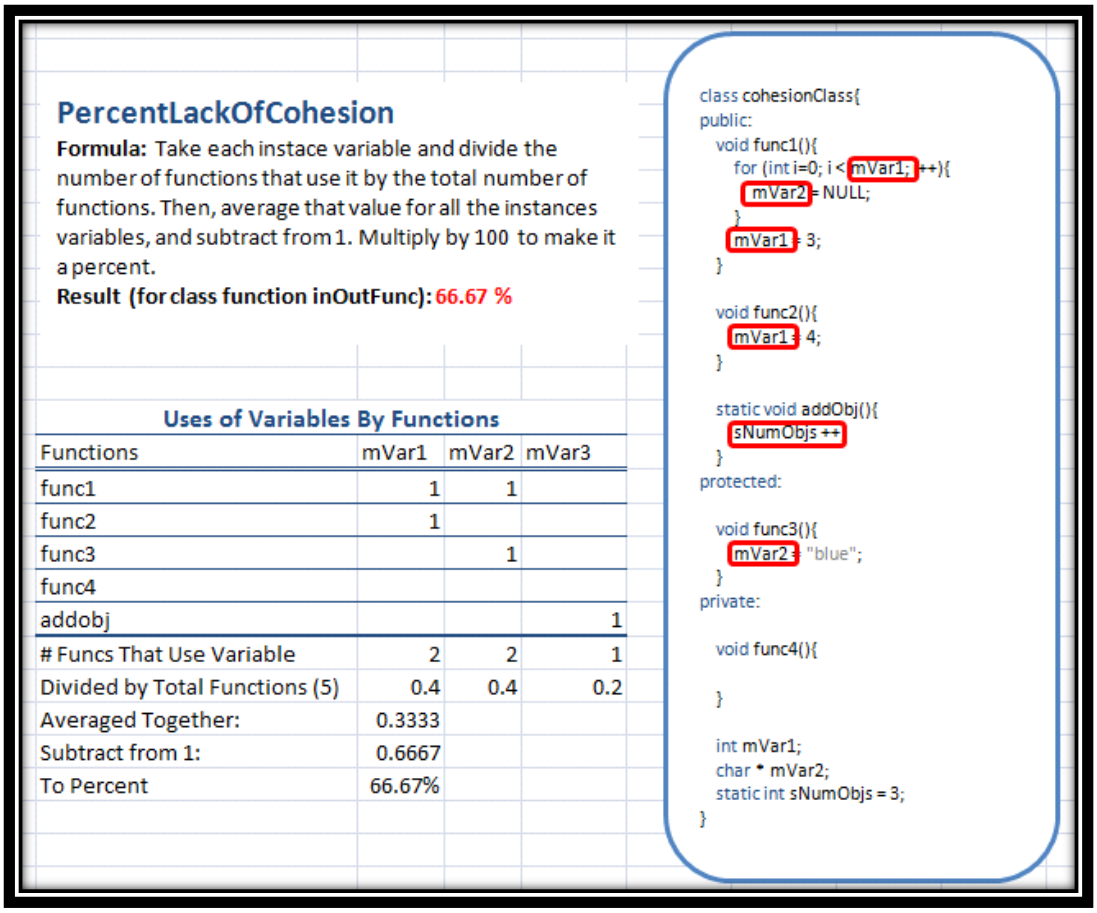
Şekil 4.2 Lines of Code Ölçütünün Hesaplanması

Average Complexity (Ortalama Karmaşıklık): Bir sınıfın tüm metotlarının ortalama karmaşıklık değeridir. Bu değerin yüksek olması sınıfın daha karmaşık ve hataya daha açık olması anlamına gelir. Average Complexity ölçütü Şekil 4.3'te görüldüğü gibi hesaplanmaktadır:



Şekil 4.3 Average Complexity Ölçütünün Hesaplanması

LCOM (Lack of Cohesion in Methods – Metotların Uyumsuzluk Yüzdesi): Bir sınıftaki metotların birbirleriyle olan uyumsuzluğunun yüzdesidir. Yüksek uyumsuzluk sınıfın birbirinden bağımsız işler yaptığını ve bu nedenle hataya daha açık olduğunu gösterir. LCOM ölçütü Şekil 4.4'te görüldüğü gibi hesaplanmaktadır:



PercentLackOfCohesion

Formula: Take each instace variable and divide the number of functions that use it by the total number of functions. Then, average that value for all the instances variables, and subtract from 1. Multiply by 100 to make it a percent.

Result (for class function inOutFunc): 66.67 %

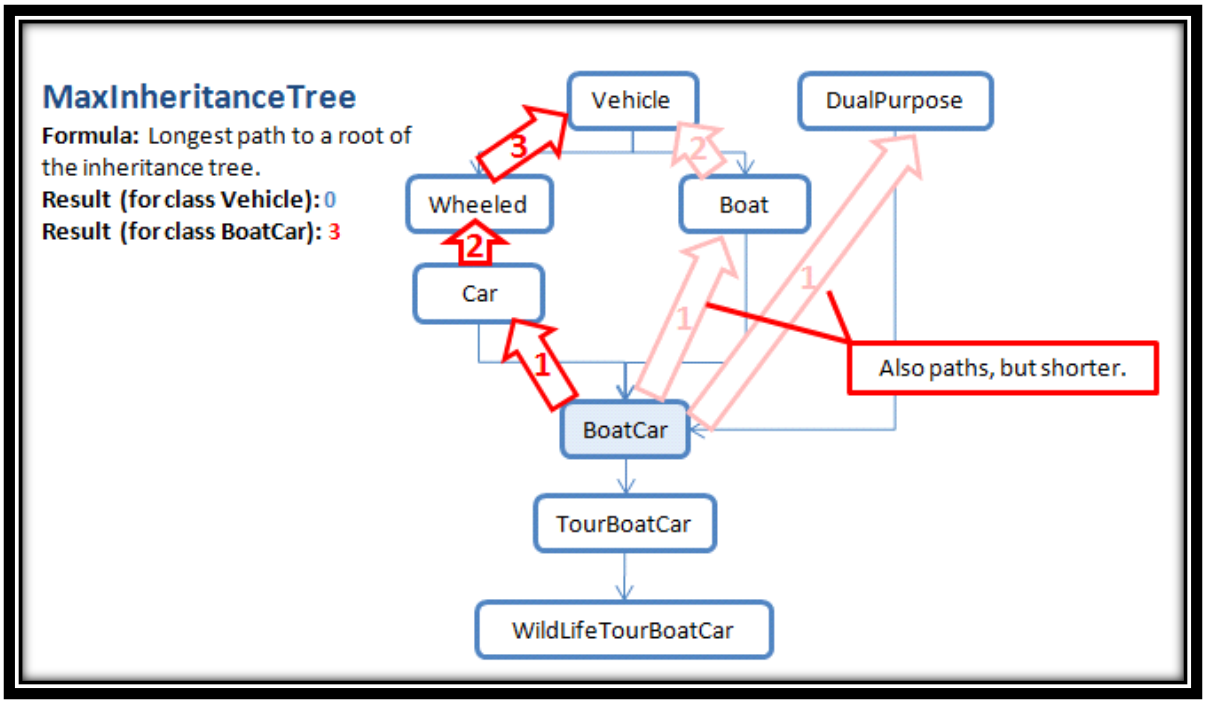
Uses of Variables By Functions

Functions	mVar1	mVar2	mVar3
func1	1	1	
func2	1		
func3		1	
func4			
addobj			1
# Funcs That Use Variable	2	2	1
Divided by Total Functions (5)	0.4	0.4	0.2
Averaged Together:	0.3333		
Subtract from 1:	0.6667		
To Percent	66.67%		

```
class cohesionClass{
public:
  void func1(){
    for (int i=0; i < mVar1; ++){
      mVar2 = NULL;
    }
    mVar1 = 3;
  }

  void func2(){
    mVar1 = 4;
  }

  static void addObj(){
    sNumObjs ++
  }
protected:

  void func3(){
    mVar2 = "blue";
  }
private:

  void func4(){

  }

  int mVar1;
  char * mVar2;
  static int sNumObjs = 3;
}
```

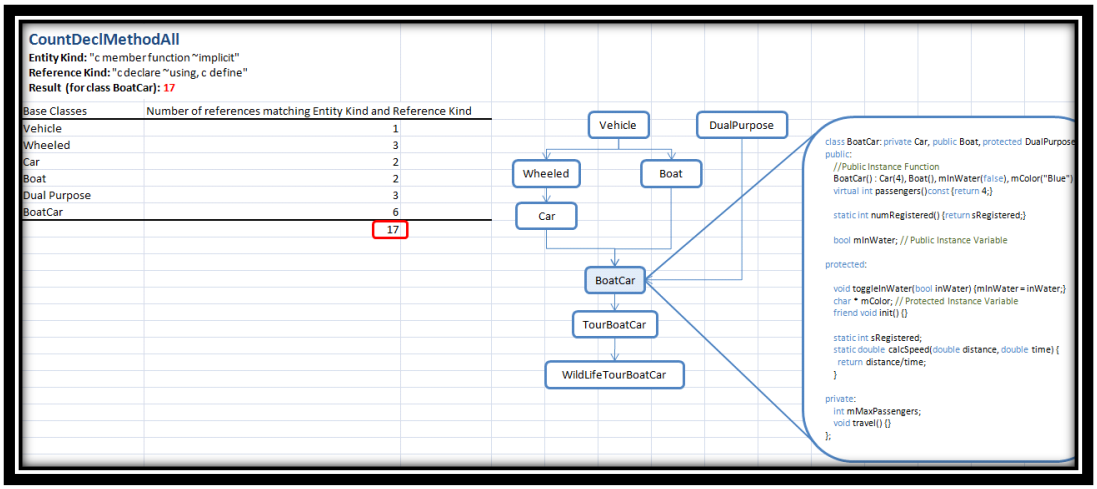
Şekil 4.4 LCOM Ölçütünün Hesaplanması

DIT (Depth of Inheritance Tree): Bir sınıfın, kalıtım ağacının köküne uzaklığı olarak tanımlanır. Bu değerin yüksek olması karmaşık hiyerarşi ve hataya yatkınlık anlamına gelir. DIT ölçütü Şekil 4.5'te görüldüğü gibi hesaplanmaktadır:



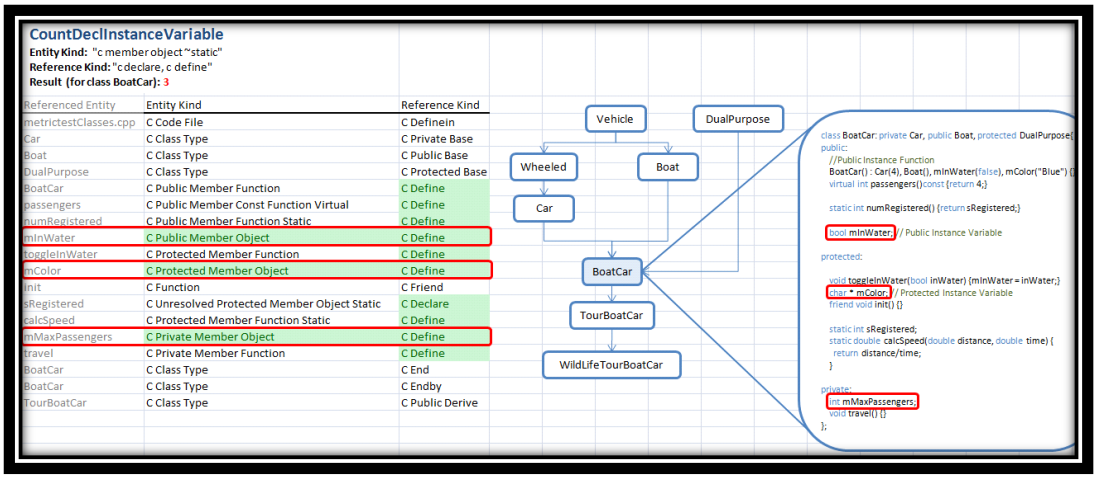
Şekil 4.5 DIT Ölçütünün Hesaplanması

RFC (Response For a Class): Kalıtımla türetilmişler dâhil olmak üzere sınıfın tüm metotlarının toplam sayısıdır. Bu değerin yüksek olması sınıfın daha büyük olması, dolayısıyla hataya daha açık olması anlamına gelir. RFC ölçütü Şekil 4.6'da görüldüğü gibi hesaplanmaktadır:



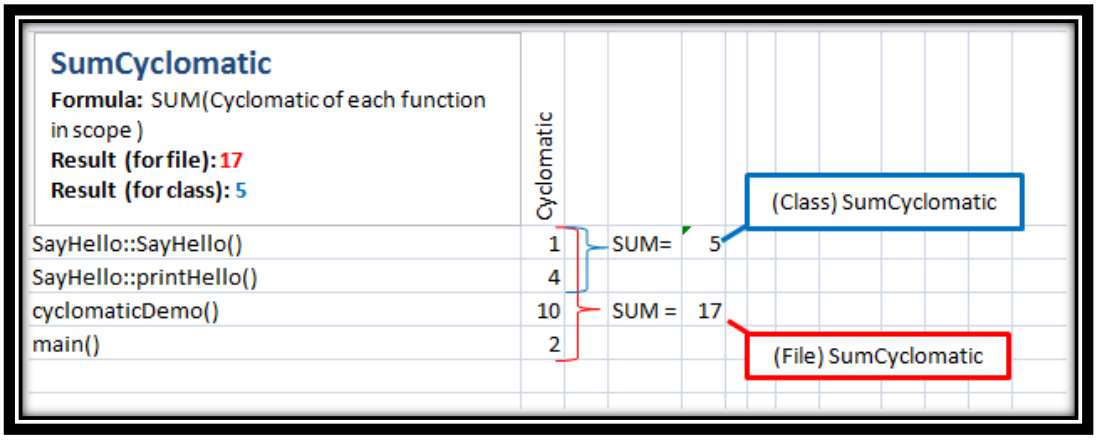
Şekil 4.6 RFC Ölçütünün Hesaplanması

NIV (Number of Instance Variables): Bir sınıftaki nesne değişkenlerinin sayısıdır. Değişken sayısının fazla olması sınıfın daha karmaşık bir yapıda olması dolayısıyla hataya daha açık olması anlamına gelmektedir. NIV ölçütü Şekil 4.7'de görüldüğü gibi hesaplanmaktadır:



Şekil 4.7 NIV Ölçütünün Hesaplanması

WMC (Weighted Methods per Class): Tüm metotların karmaşıklıklarının toplamıdır. WMC ölçütü Şekil 4.8'de görüldüğü gibi hesaplanmaktadır:



Şekil 4.8 WMC Ölçütünün Hesaplanması

5 DENEYSEL SONUÇLAR

Bu bölümde önerilen yöntemin doğrulanması için kullanılan, gerçek zamanlı bir simülasyon projesi ve gerçek zamanlı bir sinyalizasyon projesi tanıtılmıştır.

5.1 İncelenen Projeler

5.1.1 Gerçek zamanlı simülasyon projesi (D Projesi)

D projesi gerçek zamanlı bir denizaltı taktik simülatörü projesidir. Yaklaşık 60 kişilik bir ekiple gerçekleştirilmektedir. Proje 2005 yılında başlamış olup, devam etmektedir. Yaklaşık 1.500.000 satır koda ulaşmış bulunmaktadır. Bu çalışmada müşteriye 2011'de teslimi yapılan versiyona ait hata sayılarıyla doğrulama yapmak için 1.0.0 sürümü, 2014'te teslimi yapılan versiyona ait hata sayılarıyla doğrulama yapmak için 2.0.0 sürümü kullanılmıştır.

Çizelge 5.1 D Projesi Sürüm Bilgileri

D Projesi Sürümü	Sürüm Tarihi	Sürüm Konfigürasyon Öğesi Sayısı	Sürüm Sınıf Sayısı
1.0.0	01.04.2007	70	1720
2.0.0	04.05.2012	95	1908

5.1.2 Gerçek zamanlı sinyalizasyon projesi (U Projesi)

U projesi gerçek zamanlı bir sinyalizasyon projesidir. Yaklaşık 10 kişilik bir ekiple gerçekleştirilmiştir. Proje 2008 yılında başlamış olup, 2011 yılında tamamlanmıştır. Bu çalışmada müşteriye 2009'da teslimi yapılan versiyona ait hata sayılarıyla doğrulama yapmak için 1.0.0 sürümü, 2011'de teslimi yapılan versiyona ait hata sayılarıyla doğrulama yapmak için 2.0.0 sürümü kullanılmıştır.

Çizelge 5.2 U Projesi Sürüm Bilgileri

U Projesi Sürümü	Sürüm Tarihi	Sürüm Konfigürasyon Öğesi Sayısı	Sürüm Sınıf Sayısı
1.0.0	05.10.2008	32	652
2.0.0	05.11.2010	45	840

5.2 Öncelikli Olarak Test Edilecek Sınıfları Belirleme Yöntemleri

5.2.1 Rastgele işaretleme yöntemi

Bu yöntemde projelerin seçilen sürümlerindeki sınıfların %10'u rastgele öncelikli olarak test edilecek sınıf olarak işaretlenmiştir. Daha sonra en çok hata tespit edilen sınıf sıralamasında %10'a giren sınıflarla karşılaştırılmış, başarı oranı çıkarılmıştır. İşaretleme işlemi her projenin her sürümü için 100 kez yapılmış, en az, en fazla ve ortalama başarım değerleri çıkarılmış ve Çizelge 5.3'te sunulmuştur. Bu yöntem, tez kapsamında geliştirilen yöntem ile karşılaştırma yapmak, öncelikli olarak test edilecek sınıfları

belirleme kapsamında iyileştirme olup olmadığı hakkında fikir verebilmek için kullanılmıştır.

Çizelge 5.3 Rastgele İşaretleme Yöntemi Sonuçları

Proje	Sürüm	En Az	En Fazla	Ortalama
D	1.0.0	2	32	16
	2.0.0	2	45	21
U	1.0.0	0	15	6
	2.0.0	1	18	8

Elde edilen bulgulara göre rastgele seçim yöntemi ile öncelikli olarak test edilecek sınıfları doğru belirleme oranı, D projesi 1.0.0 sürümü için %9.3, D projesi 2.0.0 sürümü için %11.05, U projesi 1.0.0 sürümü için %9.23, U projesi 2.0.0 sürümü için ise %9.52 olmuştur.

5.2.2 Ölçüt toplamı yöntemi

Tez kapsamında geliştirilen ölçüt toplamı yönteminde önce sınıflara ait Ölçüt Değerleri Listesi (ÖDL) oluşturulmuştur. Bu ölçüt değerleri listeleri kullanılarak da yine sınıflara ait Ölçüt Toplamı Değerleri Listesi (ÖTDL) oluşturulmuştur.

Sınıflara ait ÖDL, projenin ilgili sürümü için ölçüt değerlerinin hesaplanması ve elde edilen değerlerin büyükten küçüğe sıralanmasıyla oluşturulur. Tüm ölçütler, aynı ağırlıkta olması için 0-100 arasında değer alacak şekilde normalizasyon işlemine tabi tutulmuştur. Normalizasyon işlemi için kullanılacak denklem aşağıdadır:

$$V_n = \left(\frac{V - Min}{Max - Min}\right) x(NewMax - NewMin) + NewMin$$

Denklem 5.1 Normalizasyon Denklemi

Denklemdeki Min ve Max değerleri, sınıfların hesaplanmış ilgili ölçüt değerinin en küçük ve en yükseğini ifade etmektedir. V değeri, ilgili ölçütün ilk değerini, V_n değeri ise ilgili ölçütün normalize edilmiş değerini belirtmektedir.

Ölçüt değerleri listesi oluşturulduktan sonra, her sınıfa ait yedi ölçüt değeri toplanarak ve yine elde edilen değerler büyükten küçüğe sıralanarak ÖTDL oluşturulur.

Elde edilen ÖDL'in en üstündeki %10'luk kısmı "öncelikli olarak test edilmesi gereken sınıf" olarak işaretlenir.

Ölçüt toplamı yönteminin işleyişi örnek verilerle oluşturulan çizelgelerle adım adım özetlenmiştir:

Çizelge 5.4 ÖDL Oluşturulması

X Ölçütü	
Sınıf	Değer
S4	8
S2	11
S1	1
S3	4

İlk olarak proje sınıflarının ölçüt değerleri hesaplanır ve çizelge haline getirilir. Tez kapsamında kullanılmak üzere yedi ölçüt seçilmiştir. Çizelge 5.4 herhangi bir ölçütün hesaplanan değerlerinin çizelge haline getirilmesini göstermektedir.

Çizelge 5.5 ÖDL Sıralanması

X Ölçütü	
Sınıf	Değer
S2	11
S4	8
S3	4
S1	1

Daha sonra hesaplanan ölçüt değerleri büyükten küçüğe olmak üzere sıralanır. Çizelge 5.5 herhangi bir ölçütün hesaplanan değerlerinin sıralanmasını göstermektedir.

Çizelge 5.6 ÖDL Normalleştirme

X Ölçütü	
Sınıf	Değer
S2	100
S4	70
S3	30
S1	0

Hesaplanan ölçüt değerleri sıralandıktan sonra Denklem 5.1 kullanılarak normalleştirme işlemi yapılır. Çizelge 5.6 sıralanmış ölçüt değerleri üzerinde normalleştirme işlemi yapıldığını göstermektedir.

Çizelge 5.7 ÖTDL Oluşturulması

Ölçüt Toplamları	
Sınıf	Değer
S3	**230**
S4	180
S2	175
S1	50

X Ölçütü			Y Ölçütü			Z Ölçütü	
Sınıf	Değer		Sınıf	Değer		Sınıf	Değer
S2	100		**S3**	**100**		**S3**	**100**
S4	70		S4	90		S2	75
S3	**30**		S1	50		S4	20
S1	0		S2	0		S1	0

Son olarak, projedeki her sınıf için, yedi ölçütün de normalleştirilmiş çizelgelerindeki değerleri toplanır. Böylece sınıfların ölçüt değerleri toplamı elde edilir. Bu değerler yine büyükten küçüğe sıralanarak yeni bir çizelgeye aktarılır ve ÖTDL elde edilmiş olur.

5.3 Deneysel Çalışmalar

Çalışılan projelerin seçilen sürümleri için ÖDL ve ÖTDL oluşturulmuş, öncelikli olarak test edilecek sınıflar belirlenmiş ve sınıf bazında raporlanan hata sayıları çıkarılmıştır.

Çizelge 5.1 ve Çizelge 5.2'de verildiği gibi,

- D projesi 1.0.0 sürümü sınıf sayısı 1720,
- D projesi 2.0.0 sürümü sınıf sayısı 1908,
- U projesi 1.0.0 sürümü sınıf sayısı 652,
- U projesi 2.0.0 sürümü sınıf sayısı 840'tır.

Buna göre %10 olarak işaretlenecek sınıf sayıları,

- D projesi 1.0.0 sürümü sınıf sayısı 172,
- D projesi 2.0.0 sürümü sınıf sayısı 190,
- U projesi 1.0.0 sürümü sınıf sayısı 65,
- U projesi 2.0.0 sürümü sınıf sayısı 84 olmalıdır.

Çizelge 5.8 ÖDL ve ÖTDL Yöntemlerine Göre Öncelikli Olarak Test Edilmesi Gereken Sınıf Sayıları

		D Projesi		U Projesi	
		1.0.0	**2.0.0**	**1.0.0**	**2.0.0**
ÖDL	LC	122	95	32	60
	AC	103	108	27	29
	LCOM	60	47	28	39
	DIT	43	72	21	24
	RFC	107	104	38	66
	NIV	89	122	25	31
	WMC	124	120	30	38
ÖTDL	TÜMÜ	117	110	36	52

ÖDL ve ÖTDL yöntemleri kullanılarak işaretlenmiş sınıflardan kaç tanesinin, gerçekten en çok hata tespit edilen %10'luk kısım içerisinde yer aldığı Çizelge 5.8'de verilmiştir. Çizelgede her projenin her sürümü için, tez kapsamında kullanılan yedi ölçütün ayrı ayrı kullanılmasıyla hesaplanman doğru tespit edilmiş sınıf sayıları ve yedi ölçütün tamamı kullanılarak hesaplanman doğru tespit edilmiş sınıf sayıları yer almaktadır.

Çizelge 5.9 ÖDL ve ÖTDL Yöntemlerinin Başarı Oranları

		D Projesi		U Projesi	
		1.0.0	**2.0.0**	**1.0.0**	**2.0.0**
ÖDL	LC	%70,93	%50,00	%49,23	%71,42
	AC	%59,88	%56,84	%41,53	%34,52
	LCOM	%34,88	%24,73	%43,07	%46,42
	DIT	%25,00	%37,89	%32,30	%28,57
	RFC	%62,20	%54,73	%58,46	%78,57
	NIV	%51,74	%64,21	%38,46	%36,90
	WMC	%72,09	%63,15	%46,15	%45,23
ÖTDL	TÜMÜ	%68,02	%57,89	%55,38	%61,90

ÖDL ve ÖTDL yöntemleri kullanılarak işaretlenmiş sınıflardan yüzde kaçının, gerçekten en çok hata tespit edilen %10'luk kısım içerisinde yer aldığı Çizelge 5.9'da verilmiştir. Çizelgede her projenin her sürümü için, tez kapsamında kullanılan yedi ölçütün ayrı ayrı kullanılmasıyla hesaplanan doğru tespit edilmiş sınıf yüzdeleri ve yedi ölçütün tamamı kullanılarak hesaplanan doğru tespit edilmiş sınıf yüzdeleri yer almaktadır.

Çizelge 5.9'daki yüzdelik değerler, Çizelge 5.8'deki hata sayılarının, Çizelge 5.1 ve Çizelge 5.2'de verilen proje sınıf sayılarının 1/10'una bölünmesiyle hesaplanmıştır.

Çizelge 5.9 incelendiğinde;

- D projesi 1.0.0 sürümü için tek olarak ölçüt toplamı yönteminden (ÖTDL) daha iyi tahmin yapan ölçütler sırasıyla WMC ve LC'dir.

- D projesi 2.0.0 sürümü için tek olarak ölçüt toplamı yönteminden (ÖTDL) daha iyi tahmin yapan ölçütler sırasıyla NIV ve WMC'dir.
- U projesi 1.0.0 sürümü için tek olarak ölçüt toplamı yönteminden (ÖTDL) daha iyi tahmin yapan ölçütler sırasıyla RFC'dir.
- U projesi 2.0.0 sürümü için tek olarak ölçüt toplamı yönteminden (ÖTDL) daha iyi tahmin yapan ölçütler sırasıyla RFC ve LC'dir.

Bu analizden, tek olarak ölçüt toplamı yönteminden daha iyi tahmin yapan ortak bir ölçütün olmadığı çıkarımı yapılabilir.

Çizelge 5.9 incelendiğinde;

- Tüm sürümler için ölçüt toplamı yönteminin, yedi ölçütün ayrı olarak yaptığı tahminlerin ortalamasından daha iyi sonuç verdiği görülmektedir.

Çizelge 5.9 incelendiğinde;

- Tüm sürümler için en iyi sonucu veren ortak bir ölçütün olmadığı,
- Tüm sürümler için en kötü sonucu veren ortak bir ölçütün olmadığı görülmektedir.

6 SONUÇ ve DEĞERLENDİRMELER

Elde edilen sonuçlara göre tez kapsamında geliştirilen ölçüt toplamı yönteminin öncelikli olarak test edilecek sınıfları doğru olarak belirleme oranının %55 ile %68 arasında olduğu görülmüştür.

Karşılaştırma yapılabilmesi amacıyla kullanılan rasgele seçim yönteminde ise test edilecek sınıfların doğru olarak belirlenme oranının %9,23 ile %11,05 aralığında bulunmuştur. Ölçüt toplamı yönteminde elde edilen sonuçlarda rasgele seçim yöntemine göre belirgin oranda iyileşme olduğu gözlenmektedir.

Ayrıca, her projenin seçilen her sürümü için, ölçütlerin ayrı ayrı öncelikli olarak test edilecek sınıfları doğru olarak belirleme oranlarının ortalamaları, ölçüt toplamı yöntemiyle bulunan orandan düşük çıkmıştır. Ölçütlerin ayrı ayrı başarı oranlarının ortalaması, D projesi 1.0.0 sürümü için %53,82, D projesi 2.0.0 sürümü için %50,22, U projesi 1.0.0 sürümü için %44,17 ve U projesi 2.0.0 sürümü için %48,80'dir. Ölçüt toplamı yönteminde ise bu değerler sırasıyla %68,02, %57,89, %55,38 ve %61,90'dır.

Sonuç tabloları incelendiğinde bazı ölçütlerin tek olarak, ölçüt toplamı yöntemine oranla daha doğru tahmin yapmış olduğu gözlenmektedir. Ancak bu durumun incelenen tüm sürümlerde geçerli olmadığı görülmüştür. Ayrıca, kullanılan dört sürüm göz önüne alındığında, tek olarak daha doğru tahmin yapan ortak bir ölçütün olmadığı görülmüştür. Dolayısıyla ölçüt toplamı yönteminin daha güvenilir olduğu anlaşılmaktadır.

7 ÖNERİLER

İlerleyen çalışmalarda seçilen ölçütlerin tamamının yanı sıra tüm olası ölçüt kombinasyonlarının da test edilecek sınıfları doğru olarak belirleyebilmedeki başarı oranları hesaplanarak daha kapsamlı bir çalışma yapılması planlanmaktadır.

Yazılım projelerinde kullanılan ölçüt araçlarının büyük çoğunluğu sadece ölçütleri ölçmekte, sonuçların yorumunu ilgili uzmana bırakmaktadır. Ölçüt ölçmenin yanında ölçütleri yorumlayıp kullanıcıya tez kapsamında geliştirilen yöntemin yaptığı gibi bilgiler sunan bir yazılım ürünü oluşturulması da planlanmaktadır.

Tez kapsamında geliştirilen ölçüt toplamı yönteminin, öncelikli olarak test edilecek sınıfları doğru olarak belirlemesindeki başarı oranı, en çok hata tespit edilen sınıflarla karşılaştırılarak bulunmuştur. Günümüzde pek çok yazılım projesinde sınıf bazlı hata raporları tutulmamaktadır. Dolayısıyla hataların sınıflarla eşleştirilmesi oldukça zaman almıştır. Bu yöntem yerine sınıfların belirlenen aralıklarda ne kadar değişim geçirdiği gibi farklı yöntemler üzerinde de araştırma yapılması önerilmektedir.

8 KAYNAKLAR

1) Tiftik N., Öztarak, H., Ercek, G., Özgün, S., Sistem/Yazılım Geliştirme Sürecinde Doğrulama Faaliyetleri, Mikrodalga ve Sistem Mühendisliği, 2000.
2) Song, O., Sheppard, M., Cartwright, M., and Mair, C., 2006: "Software Defect Association Mining and Defect Correction Effort Prediction", IEEE Transactions on Software Engineering, 2006
3) Fenton, N., and Ohlsson, N., "Quantitative Analysis of Faults and Failures in a Complex Software System", IEEE Transactions on Software Engineering, 2000.
4) Xiaowei, W., "The Metric System about Software Maintenance", 2011 International Conference of Information Technology, Computer Engineering and Management Sciences, ISBN: 978-0-7695-4522-6/11,Wuhan, 2011
5) Kaur, A., Sandhu, P. S., Brar, A. S., "An Empirical Approach for Software Fault Prediction", 2010 5th International Conference on Industrial and Information Systems, ISBN: 987-1-4244-6653-5/10,India, 2010
6) Raymond, P. L., Weimer, B., Weimer, W. R., "Learning a Metric for Code Readability", IEEE Transactions of Software Engineering, Cilt. 36, No. 4, ISBN: 0098-5589/10, 2010.
7) Ogasawara, H., Yamada, A., Kojo, M., "Experiences of Software Quality Management Using Metrics through the Life-Cycle", ISBN:0270-5257/96 , Japan, 1996
8) Chaumun, M., Kabaili, H., Keller, R., and Lustman, F., "Change Impact Model for Changeability Assessment in Object-Oriented Software Systems", Science of Computer Programming, 2002.
9) Lee, Y., Yang, J., and Chang, K. H., "Metrics and Evolution in Open Source Software", Seventh International Conference on Quality Software (QSIC 2007), 2007.

10) Kastro, Y., Bener, A. B., "A defect prediction method for software versioning", Springer Science.
11) Liafna, L., Hareton, L., "Mining Static Code Metrics for a Robust Prediction of Software Defect Proneness", Internal Symposium on Empirical Software Engineering and Measurement, 2011
12) NASA Datasets. http://promise.site.uottawa.ca/SERepository/datasets-page.html (22.08.2014)
13) J. Osbeck, Waverly, "Investigation of Automatic Prediction of Software Quality", Fuzzy Information Processing Society Annual Meeting of the North America, 2011
14) Quinlan, J., R., "C4.5: programs for machine learning", Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
15) Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I., "The WEKA Data Mining Software: An Update", SIGKDD Explorations, cilt. 11, no. 1, s. 10-18, 2009.
16) Bansiya, J., Davis, C. G., "A hierarchical model for object-oriented design quality assessment", IEEE Transactions of Software Engineering, Cilt. 28, No. 1, 2002.
17) WEKA. http://www.cs.waikato.ac.nz/ml/weka/ (22.06.2013)
18) Kayarvizhy, N. , "Analysis of Quality of Object Oriented System using Object Oriented Metrics" , Electronics Computer Technology (ICECT), 2011 3rd International Conference, 2011
19) Okumoto, K. ,"Software Defect Prediction Based on Stability Test Data", Quality, Reliability, Risk, Maintenance, and Safety Engineering (ICQR2MSE), 2011 International Conference on,2011
20) SPICE. http://en.wikipedia.org/wiki/ISO/IEC_15504 (22.05.2014)
21) CMMI. http://whatis.cmmiinstitute.com/ (22.05.2014)
22) ISO. http://www.iso.org/iso/home.html (22.05.2014)
23) http://www.yucalar.com/YZM417/YZM417-01.pdf (25.03.2014)
24) Schach, S., R. (2007). "Object-Oriented & Classical Software Engineering", 7th Ed., McGraw Hill.

25) Crosby, P. Quality Is Free: The Art of Making Quality Certain. McGraw- Hill, New York, 1979.
26) ISO/IEC 9126-1: Information Technology – Software Product Quality – Part 1: Quality Model. ISO/IEC JTC1/SC7/WG6 (1999)
27) http://web.deu.edu.tr/fmd/s40/s40-m5.pdf (25.08.2014)
28) Chidamber, S., Kemerer, C., "A Metrics Suite for Object-Oriented Design," IEEE Trans. Software Eng., vol.20, no. 6, pp. 476-493, June 1994
29) Brito e Abreu, F., Pereira, G., Soursa, P., "Coupling-Guided Cluster Analysis Approach to Reengineer the Modularity of Object-Oriented Systems," Proc. Euromicro Conf. Software Maintenance and Reeng., pp. 13-22, 2000.
30) J. Bansiya and C. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," IEEE Trans. Software Eng., vol 28, no. 1, pp. 4-17, Jan. 2002.
31) http://www.ijarcsse.com/docs/papers/Volume_3/3_March2013/V3I3-0261.pdf (22.08.2014)
32) Essential Metrics. http://www.powersoftware.com/em/ (13.06.2013)
33) LocMetrics. http://www.locmetrics.com/ (17.02.2014)
34) StatSvn. http://www.statsvn.org/ (13.03.2014)
35) Coverity. http://www.coverity.com/ (15.03.2014)
36) SDMetrics. http://www.sdmetrics.com/ (13.09.2013)
37) Understand. http://www.scitools.com/index.php (20.03.2014)

9 ÖZGEÇMİŞ

Ramazan Murat Demirbaş, 1986 yılında Isparta'da doğdu. Öğrenimini sırasıyla Şener Birsöz İlköğretim Okulu, Haydarpaşa Anadolu Lisesi'nde tamamladı. 2005 yılında Maltepe Üniversitesi Endüstri Mühendisliği bölümünü kazandı. 2006 yılında Maltepe Üniversitesi'nde Bilgisayar Mühendisliği bölümünde çift anadala başladı ve 2009 yılında bu iki bölümden mezun oldu. Şubat 2011'den bu yana TÜBİTAK'ta test mühendisi pozisyonunda çalışmaktadır. Eylül 2011'da Maltepe Üniversitesi'nde başladığı Bilgisayar Mühendisliği Yüksek Lisans programına tez aşamasında devam etmektedir.