ImageGlass Projesi Üzerinde QMOOD Metrik Analizi

Mücahit Zengin, 172803037
İsmet DAYAN, 192802079
Asem Al Salahi, 192803004
Muhammet Yusuf Bozkurt, 202802068
Yazılım Mühendisliği Bölümü
Celal Bayar Üniversitesi
Manisa, Türkiye
{172803037, 192802079, 192803004, 202802068}@ogr.cbu.edu.tr

***Özetçe* —Bu çalışmada, Windows tabanlı açık kaynaklı bir resim görüntüleyici olan ImageGlass'ın dört farklı sürümü incelenmiştir. Bu sürümler, projenin gelişim sürecindeki önemli aşamaları temsil etmektedir. Her bir sürüm için, yazılım tasarım kalitesini değerlendirmek amacıyla QMOOD (Quality Model for Object-Oriented Design) Modeli kullanılarak detaylı bir metrik analizi gerçekleştirilmiştir.**

Anahtar Kelimeler—Yazılım Kalitesi, Yazılım Metrikleri, QMOOD, Nesneye Dayalı Programlama

***Abstract*—In this work, four different versions of ImageGlass, a Windows-based open source image viewer, are analyzed. These versions represent important stages in the development process of the project. For each version, a detailed metric analysis was performed using the QMOOD (Quality Model for Object-Oriented Design) Model to evaluate the software design quality.**

Keywords—Software Quality, Software Metrics, QMOOD, Object Oriented Programming

I. GİRİŞ

Bu çalışma, popüler açık kaynaklı bir resim görüntüleyici olan ImageGlass'ın yazılım tasarım kalitesini değerlendirmeyi amaçlamaktadır. ImageGlass, Windows işletim sistemi için geliştirilmiş, kullanıcı dostu arayüzü ve geniş format desteği ile dikkat çeken bir uygulamadır. Bu projenin seçilmesinin temel nedeni, açık kaynaklı bir yazılım olarak, gelişim sürecinin ve sürüm geçmişinin net bir şekilde incelenebilir olmasıdır.

Çalışmanın odak noktası, ImageGlass'ın dört farklı sürümünün QMOOD (Quality Model for Object-Oriented Design) Modeli kullanılarak detaylı bir şekilde analiz edilmesidir. Bu analiz, yazılımın tekrar kullanılabilirlik, esneklik, anlaşılabilirlik, işlevsellik, genişletilebilirlik ve etkinlik gibi çeşitli kalite niteliklerini kapsamaktadır. Bu niteliklerin her bir sürümde nasıl değiştiğini ve bu değişikliklerin yazılımın genel kalitesi üzerindeki etkilerini incelemeyi hedeflemekteyiz.

Bu çalışma, yazılım mühendisliği alanında kalite değerlendirmesi yapmanın önemini vurgulamakta ve açık kaynaklı projelerin sürekli gelişimine katkıda bulunmayı amaçlamaktadır. Ayrıca, yazılım geliştirme sürecinde alınan tasarım kararlarının uzun vadeli etkilerini gözlemleyerek, benzer projeler için değerli içgörüler ve yönlendirmeler sunmayı hedeflemektedir.

II. İLGİLİ ÇALIŞMALAR

Yazılım kalitesinin ölçülmesi ve değerlendirilmesi, yazılım mühendisliği alanında sürekli gelişen ve önem kazanan bir konudur. Özellikle, nesneye dayalı programlama paradigmalarının yaygınlaşmasıyla birlikte, bu alandaki kalite değerlendirme modelleri ve metrikleri üzerine birçok çalışma yapılmıştır. Bu çalışmalar, yazılımın sürdürülebilirliği, esnekliği ve genel performansını etkileyen önemli faktörleri belirlemeyi amaçlamaktadır.

QMOOD (Quality Model for Object-Oriented Design) modeli, nesneye dayalı yazılım tasarım kalitesini değerlendirmek için geliştirilmiş bir modeldir ve bu alanda yapılan çalışmaların merkezinde yer alır. Örneğin, Bansiya ve Davis [1], QMOOD modelini kullanarak nesneye dayalı tasarım kalitesini değerlendiren bir hiyerarşik model geliştirmişlerdir. Bu model, yazılımın çeşitli tasarım özelliklerini ve bu özelliklerin kalite üzerindeki etkilerini inceler.

Yazılım kalitesi ölçüm tekniklerine geniş bir bakış sunan ve bu alandaki mevcut yaklaşımları ve zorlukları tartışan bir çalışma [2], yazılım kalitesinin çeşitli boyutlarını ve bu boyutların ölçümünde kullanılan teknikleri derinlemesine incelemektedir.

Ayrıca, yazılım kalitesi ölçümünde kullanılan diğer yaklaşımlar da literatürde yer almaktadır. McCall ve diğerleri [3] tarafından geliştirilen yazılım kalite modeli, yazılım paydaşları arasında iletişimi güçlendirmeyi ve yazılımın farklı kalite faktörlerini belirlemeyi amaçlar. Victor Basili'nin Hedef/Soru/Metrik (GQM) yaklaşımı [4], yazılım projelerindeki hataları belirlemek ve yazılım kalitesini geliştirmek için kullanılan bir yöntemdir.

ISO/IEC 9126 standardı [5], yazılım kalitesi ile ilgili tanımlamaları ve ölçütleri standartlaştırmayı amaçlar. Bu standart, yazılım kalitesinin farklı boyutlarını tanımlar ve yazılım mühendisleri arasında ortak bir dil oluşturmayı hedefler.

Bu çalışmaların ışığında, QMOOD modeli ve diğer kalite değerlendirme araçlarının, nesneye dayalı yazılım projelerinin kalitesini ölçmede etkili olduğu görülmektedir. Bu çalışma, ImageGlass projesinin farklı sürümlerini bu modeller ve metrikler kullanarak değerlendirip, yazılım tasarım kalitesinin zaman içindeki değişimini analiz etmeyi amaçlamaktadır.

III. KULLANILAN YAKLAŞIM VE ARAÇLAR

A. Kullanılan Yaklaşım

Bu çalışmada, yazılım tasarım kalitesini değerlendirmek için QMOOD modeli kullanılmıştır. QMOOD, nesneye dayalı yazılım tasarımının çeşitli yönlerini değerlendiren bir dizi metrikten oluşur. Bu model, yazılımın sürdürülebilirliği, esnekliği ve genel performansını etkileyen faktörleri belirlemeyi amaçlar. Modelin uygulanmasında, aşağıdaki ana metrikler dikkate alınmıştır:

- **Design Size in Classes (DSC):** Projedeki toplam sınıf sayısı olarak değerlendirilmiştir.
- **Number of Hierarchies (NOH):** Projedeki sınıf hiyerarşisi sayısı olarak ele alınmıştır.
- **Average Number of Ancestors (ANA):** Projedeki kalıtım ağacı boyunun ortalaması olarak hesaplanmıştır.
- **Data Access Metric (DAM):** Projedeki sınıflarda yer alan private ve protected niteliklerin toplamının, projede yer alan tüm niteliklere oranı olarak hesaplanmıştır.
- **Direct Class Coupling (DCC):** Projedeki sınıfların birbirleriyle bağımlılık oluşturduğu sınıf sayısı olarak ele alınmıştır.
- **Cohesion Among Methods in Class (CAM):** Metrik toplama aracından alınan LCOM (Lack of Cohesion) metriği kullanılarak 1-LCOM olarak hesaplanmıştır.
- **Measure of Aggregation (MOA):** Projede yer alan kullanıcı tanımlı sınıf nesnesi sayısı olarak ele alınmıştır.
- **Measure of Functional Abstraction (MFA):** Projede yer alan bir sınıfın atasından türettiği metod sayısının toplam metod sayısına oranı olarak ele alınmıştır.
- **Number of Polymorphic Methods (NOP):** Projede yer alan interface sayısı olarak ele alınmıştır.
- **Class Interface Size (CIS):** Projedeki public metod sayısı olarak değerlendirilmiştir.
- **Number of Methods (NOM):** Projedeki toplam metod sayısı olarak ele alınmıştır.

Bu metrikler ve karşılık gelen tasarım özellikleri Tablo I'de gösterilmiştir.

Tasarım Özelliği	**İlgili Tasarım Metriği**
Design Size	DSC (Design Size in Classes)
Hierarchies	NOH (Number of Hierarchies)
Abstraction	ANA (Average Number of Ancestors)
Encapsulation	DAM (Data Access Metric)
Coupling	DCC (Direct Class Coupling)
Cohesion	CAM (Cohesion Among Methods in Class)
Composition	MOA (Measure of Aggregation)
Inheritance	MFA (Measure of Functional Abstraction)
Polymorphism	NOP (Number of Polymorphic Methods)
Messaging	CIS (Class Interface Size)
Complexity	NOM (Number of Methods)

Tablo I: Tasarım Metrikleri ile Tasarım Özellikleri Arasındaki İlişki

Kullanılan tasarım özellikleri ve bu özelliklere karşılık gelen kalite metrikleri, belirli bir yazılım tasarım metodolojisinin çeşitli yönlerini değerlendirmek için seçilmiştir. Bu özellikler ve metrikler, yazılımın genel kalitesini etkileyen önemli faktörler olarak tanımlanmıştır. Tasarım özellikleri ve bunlara karşılık gelen kalite metrikleri, Tablo II'de gösterilmiştir.

Tasarım Özellikleri	T. Kul.	Esnek.	Anlaş.	İşlev.	Geniş.	Etkin.
Design Size	+0.5	-0.33	+0.22		+0.25	+0.2
Hierarchies			-0.33	+0.22		+0.2
Abstraction		-0.33		+0.5		+0.2
Encapsulation	+0.25		-0.33			+0.2
Coupling	-0.25	-0.25	-0.33		-0.5	
Cohesion	+0.25		+0.33		+0.12	+0.2
Composition		+0.5		+0.5		+0.2
Inheritance			-0.33	+0.22		+0.2
Polymorphism	+0.5		-0.33	+0.22		+0.2
Messaging	+0.5			+0.22		+0.2
Complexity		-0.33				

Tablo II: Kalite Özellikleri İle Kalite Metrikleri Hesaplama

Yazılımın tekrar kullanılabilirliği, esnekliği, anlaşılabilirliği, işlevselliği, genişletilebilirliği ve etkinliği gibi altı temel kalite niteliğinin hesaplanmasında, 1, 2, 3, 4, 5 ve 6 denklemleri kullanılmıştır.

$$\textit{Tekrar Kullanılabilirlik} = \begin{cases} +0.5 \cdot (\text{Design Size}) \\ -0.25 \cdot (\text{Coupling}) \\ +0.25 \cdot (\text{Cohesion}) \\ +0.5 \cdot (\text{Messaging}) \end{cases} \tag{1}$$

$$\textit{Esneklik} = \begin{cases} +0.25 \cdot (\text{Encapsulation}) \\ -0.25 \cdot (\text{Coupling}) \\ +0.5 \cdot (\text{Composition}) \\ +0.5 \cdot (\text{Polymorphism}) \end{cases} \tag{2}$$

$$\textit{Anlaşılabilirlik} = \begin{cases} -0.33 \cdot (\text{Design Size}) \\ -0.33 \cdot (\text{Abstraction}) \\ +0.33 \cdot (\text{Encapsulation}) \\ -0.33 \cdot (\text{Coupling}) \\ +0.33 \cdot (\text{Cohesion}) \\ -0.33 \cdot (\text{Polymorphism}) \\ -0.33 \cdot (\text{Complexity}) \end{cases} \tag{3}$$

$$\textit{İşlevsellik} = \begin{cases} +0.22 \cdot (\text{Design Size}) \\ +0.22 \cdot (\text{Hierarchies}) \\ +0.12 \cdot (\text{Cohesion}) \\ +0.22 \cdot (\text{Polymorphism}) \\ +0.22 \cdot (\text{Messaging}) \end{cases} \tag{4}$$

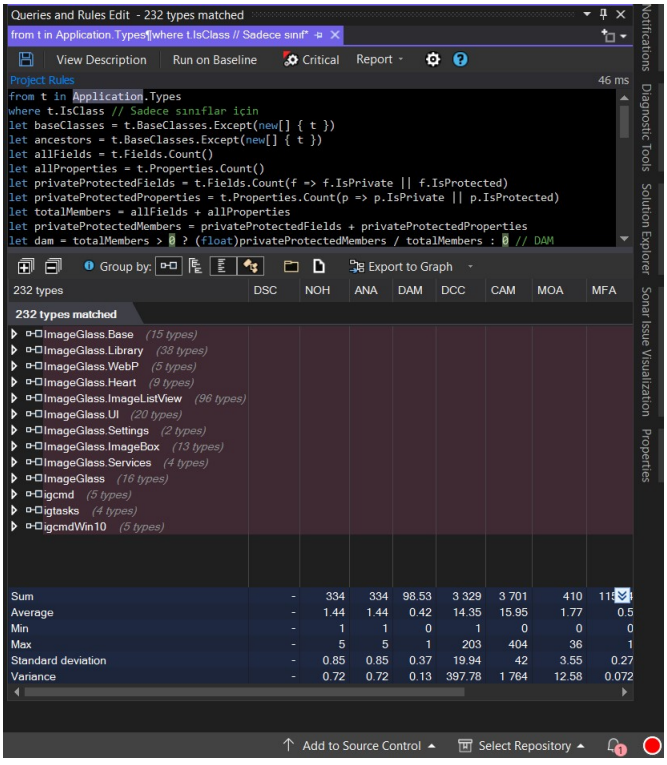
$$\textit{Genişletilebilirlik} = \begin{cases} +0.5 \cdot (\text{Abstraction}) \\ -0.5 \cdot (\text{Coupling}) \\ +0.5 \cdot (\text{Inheritance}) \\ +0.5 \cdot (\text{Polymorphism}) \end{cases} \tag{5}$$

$$Etkinlik = \begin{cases} +0.2 \cdot (\text{Abstraction}) \\ +0.2 \cdot (\text{Encapsulation}) \\ +0.2 \cdot (\text{Composition}) \\ +0.2 \cdot (\text{Inheritance}) \\ +0.2 \cdot (\text{Polymorphism}) \end{cases} \quad (6)$$

B. Kullanılan Araçlar

Bu çalışmada python dili için yeterli metrik toplama araçlarının bulunamaması ve java dili için CodeMR aracına erişim sağlanamaması nedeniyle, C# programlama dili, Visual Studio 2022 IDE'si ve NDepend aracı tercih edilmiştir. Bu süreçte NDepend aracının, kütüphane gibi yapılar üzerinde analiz yapamadığı gözlemlenmiştir. Bu nedenle, çalıştırılabilir bir proje olarak ImageGlass seçilmiştir. NDepend eklentisi, CQLinq (Code Query LINQ) özelliği sayesinde, .NET kodunu LINQ sorguları aracılığıyla sorgulama imkanı sunmuştur. Bu süreç ve sonuçlar Şekil 1'de gösterilmiştir. CQLinq, NDepend'in NDepend.CodeModel isim alanındaki tipler üzerine kurulu özel bir LINQ sorgu dilidir ve bu sayede, Visual Studio içerisinden doğrudan kod üzerinde karmaşık sorgular yapılmasını mümkün kılmıştır. Bu özellik, gerekli metriklerin toplanmasında etkili olmuştur. Elde edilen metrikler, Microsoft Excel kullanılarak tablo ve grafiklere dönüştürülmüştür.

Her ne kadar bazı araçlar beklenen sonuçları vermemiş olsa da, bu deneyimler araştırma metodolojisinin geliştirilmesine katkıda bulunmuştur.



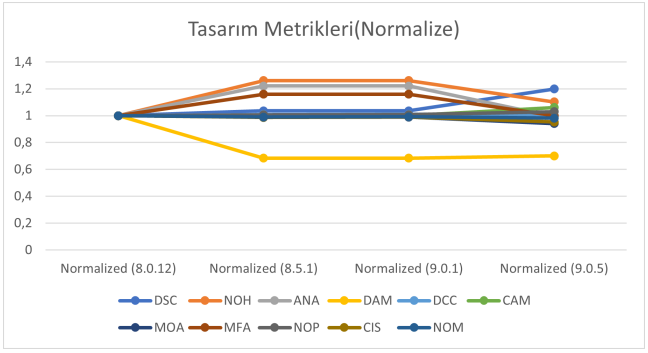
Şekil 1: NDepend aracı kullanılarak gerçekleştirilen CQLinq sorguları ve bu sorguların sonucunda elde edilen QMOOD metrikleri

IV. DEĞERLENDİRME

Bu bölüm, QMOOD modeli kullanılarak yapılan analizle elde edilen sonuçları sunar. Elde edilen metrikler, farklı sürümler arasında karşılaştırma yapabilmek için normalizasyon işlemine tabi tutulmuştur. Böylece, yazılımın gelişiminin ve kalitesinin zaman içindeki evrimi ortaya konulmuştur.

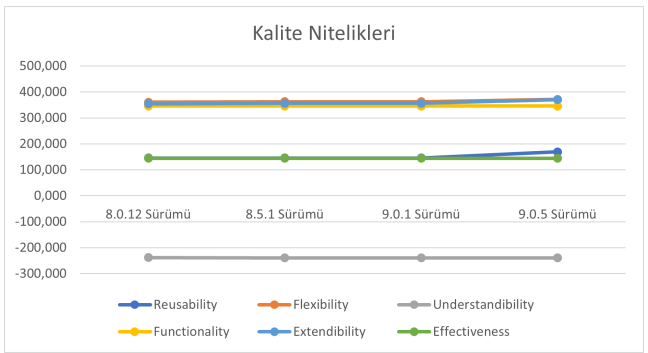
Değerlendirme sürecinde özellikle, DAM (Data Access Metric) değeri dikkate değer bir düşüş göstermiştir (Şekil 2). DAM, private ve protected özelliklerin sayısının tüm özelliklere oranı olarak tanımlanır ve [0,1] aralığında değerler alır. Yüksek bir DAM değeri tercih edildiği için, bu düşüşün analizi ve nedenleri, yazılımın gizlilik ve kapsülleme yönünden değerlendirilmesi açısından önemlidir.

DSC (Design Size in Classes) metriğinde bir artış gözlemlenmiştir. DSC, tasarımdaki toplam sınıf sayısını ifade eder ve bu artış, projenin karmaşıklığının ve genişliğinin bir göstergesi olarak yorumlanabilir. DSC'deki bu artış, projenin kapsam ve yapısındaki genişlemeyi temsil eder ve yazılımın genel yapısının daha karmaşık hale geldiğini gösterir.



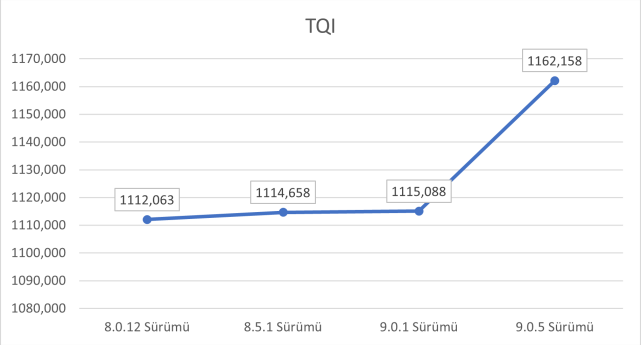
Şekil 2: Tasarım Metrikleri

Sürümler arası değerlendirme, Reusability ve Flexibility'de artış olduğunu, Functionality, Understandability ve Effectiveness'in sabit kaldığını ve Extendibility'nin önemli ölçüde geliştiğini göstermiştir (Şekil 3). Understandability'nin negatif değerleri, bu özelliğin düşük olmasının tercih edildiğini belirtir, bu da yazılımın anlaşılmasının ve öğrenilmesinin kolay olduğunu ifade eder. Bu sabit negatif değerler, tasarımın anlaşılabilirliğinin zaman içinde tutarlı bir şekilde korunduğunu gösterir.



Şekil 3: Kalite Nitelikleri

8 sürümlerinde 8.0'dan 8.5'e kadar olan dönemde TQI değerlerinde gözlenen minimal artış, bu sürümler arasındaki iyileştirmelerin kalite üzerinde yalnızca küçük etkiler yarattığını göstermektedir (Şekil 4). Buna karşın, 9 sürümlerinde, henüz 9.1 sürümü yayınlanmadan önce TQI'de yaşanan önemli artış, büyük olasılıkla, yazılımın temel yapısında ve kullanıcıya sunulan özelliklerde gerçekleştirilen önemli iyileştirmelerin ve yeniliklerin bir sonucudur. Bu durum, 9 sürümlerinin, kullanıcı ihtiyaçlarına ve pazar taleplerine hızlı bir şekilde yanıt verme kapasitesinin güçlendiğini ve yazılım kalitesinin belirgin bir şekilde iyileştirildiğini düşündürmektedir.



Şekil 4: TQI

Bu değerlendirmeler, yazılımın genel kalitesini, sürdürülebilirliğini ve performansını kapsamlı bir şekilde ele almakta ve bu alanlarda iyileştirme için potansiyel alanları ortaya koymaktadır.

V. SONUÇ

Araştırmanın temel amacı, çeşitli tasarım özelliklerinin, yazılım kalitesi üzerindeki etkilerini nicel olarak değerlendirmekti. Bu amaç doğrultusunda, Design Size, Coupling, Cohesion gibi özellikler üzerinde yoğunlaşılmış ve bu özelliklerin yazılımın tekrar kullanılabilirliği, esnekliği, anlaşılabilirliği, işlevselliği, genişletilebilirliği ve etkinliği üzerindeki etkileri analiz edilmiştir.

Çalışmanın bulguları, yazılım tasarımının kalitesinin, belirli metrikler üzerinden ölçülebileceğini ve bu metriklerin yazılımın genel kalitesi üzerinde doğrudan etkileri olduğunu göstermektedir. Örneğin, Design Size ve Coupling metriklerinin, yazılımın tekrar kullanılabilirliği ve esnekliği üzerinde önemli etkileri tespit edilmiştir. Bu sonuçlar, yazılım tasarım süreçlerinin iyileştirilmesi ve daha kaliteli yazılım ürünlerinin geliştirilmesi konusunda önemli ipuçları sunmaktadır.

Bununla birlikte, çalışmanın kapsamındaki sınırlamalar da göz önünde bulundurulmalıdır. Araştırma, belirli bir yazılım türü ve ölçek üzerinde yapılmış olup, bulguların diğer yazılım türleri ve ölçekler üzerinde tekrar değerlendirilmesi gerekebilir. Ayrıca, tasarım metriklerinin farklı yazılım geliştirme ortamlarında nasıl değişkenlik gösterebileceği konusu, gelecekteki araştırmalar için önemli bir alan olarak belirlenmiştir.

Sonuç olarak, bu çalışma, yazılım mühendisliği alanında, tasarım kalitesini değerlendirmek ve geliştirmek için kullanılabilecek metrikler ve ölçütler sunmaktadır. Elde edilen bulguların, yazılım geliştirme süreçlerinin daha etkin ve verimli hale getirilmesine katkıda bulunacağına inanılmaktadır. Gelecekteki çalışmaların, bu araştırmanın temelleri üzerine inşa edilerek, yazılım tasarım ve kalite değerlendirme metodolojilerini daha da ileriye taşıyacağı umulmaktadır.

KAYNAKÇA

[1] J. Bansiya, C.G. Davis, "A hierarchical model for object-oriented design quality assessment", IEEE Transactions on Software Engineering, 28(1), 4-17, 2002.

[2] Y. Özçevik, "Simplified QMOOD Model Proposal Based on Correlation Analysis in Different Client Applications", 2021 29th Signal Processing and Communications Applications Conference (SIU), 1-4, 2021.

[3] McCall, J. A., Richards, P. K. and Walters, G. F., "Factors in Software Quality", Nat'l Tech.Information Service, no. Vol. 1, 2 and 3, (1977).

[4] Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach, "The Goal Question Metric Approach", Encyclopedia of Software Engineering, Wiley1994.

[5] ISO "ISO,IEC 9126",http://www.iso.org