İletişim Katmanı Yazılım Mimarisinin Kalite Analizi

Ebru Doğan, Tanın Afacan, Özgür Başol, Emrah Demircan, İbrahim Karaaslan, Erman Zaim

Aselsan A.Ş., Ankara, Türkiye
{edogan, tafacan, obasol, edemircan, ikaraaslan, ezaim}@aselsan.com.tr

Özet. Bu makalede, nesneye dayalı bir iletişim katmanı yazılım mimarisi ve bu mimari kullanılarak tasarlanan TCP/IP taşıma katmanının kalite özelliklerine etkisi tartışılmıştır. Hem mimari model hem de örnek tasarım sürdürülebilirlik, test edilebilirlik, yeniden kullanılabilirlik, anlaşılabilirlik ve güvenilirlik kalite özellikleri kapsamında yorumlanmıştır.

Anahtar Kelimeler: Yazılım Mimarisi, Yazılım Kalite Özellikleri, Yazılım Kalite Ölçümleri, Yazılım Kalite Analizi, Mimari Model

1 GİRİŞ

Gitgide büyüyen ve karmaşıklaşan yazılımlar için yapılan mimari tasarım çalışmaları, artık algoritma ve veri yapılarının tasarımı çalışmalarından daha öncelikli hale gelmiştir. Karmaşık yazılım sistemlerinin kaliteli olarak tasarlanması zorunluluğu, yeni bir takım problemleri de beraberinde getirmiştir. Bu problemlerin çözümü sürecinde sistem ve yazılım mimarisi, kalite özellikleri, mimari kararlar ve yazılım şablonları gibi konular ön plana çıkmıştır.

Sistem mimarisi, karmaşık sistemlerin birbirleriyle ilişkili daha küçük parçalara bölünmesini ve bu parçalar arasındaki ilişkilerle daha kolayca ortaya çıkan ve daha belirgin bir biçimde görülebilen büyük resmin oluşturulmasını sağlar. Yazılım mimarisi ise, yazılım gereksinimleri ile gerçekleme arasında köprü görevini üstlenir. Yazılım sistem gereksinimlerini sağlamak ve söz konusu sistem üzerindeki riskleri azaltmak için yazılım geliştirme sürecinin ilk aşamalarından itibaren kalite ölçütlerinin göz önünde tutulması gerekmektedir [1].

Mimari kararlar, yazılım sisteminin bütününü ya da bir veya birden çok çekirdek parçasını ilgilendiren ve sistem kalitesini doğrudan etkileyen tasarım kararlarıdır [2, 3]. Yazılım geliştirme sürecinin erken aşamalarında alınan mimari kararlar, ilgili sistemin yazılım kalitesi gibi işlevsel olmayan gereksinimlerini dolaylı ya da dolaysız olarak etkilerler. Bundan dolayı mimari kararlar tasarımcılar tarafından muhtemel yan etkiler dikkate alınarak değerlendirilmelidirler. Mimari kararların işlevsel olmayan gereksinimler kapsamında etkilerinin değerlendirilebileceği tipik kalite özellikleri sürdürülebilirlik, test edilebilirlik, yeniden kullanılabilirlik, anlaşılabilirlik ve güvenilirliktir.

Son yıllarda yapılan yazılım mimarisi araştırmaları kapsamında, başta yazılım sistemlerinin genel yapısı olmak üzere, özellikle alt sistemler ile bileşenler arasındaki ilişkileri konu alan ilkesel çalışmalar yayınlanmıştır. Araştırmalar başlarda pratik yazılım çalışmaları olarak adlandırılırlarken, günümüze kadar olan süreçte karmaşık yazılım tasarımı ve geliştirmesi probleminin çözümünde somut bir yol gösterici görevini üstlenmişlerdir. Bu çalışmaların yazılım dünyasında yer bulmasıyla beraber yazılım sistemlerinin geliştirilmesinde alınan mimari kararlar bu çalışmalarla eşgüdümlü hale gelmiştir. Güncelliğini koruyan çalışmalardan biri olan Şablon Tabanlı Yazılım Mimarisi de günümüzde çokça kullanılmakta olup, yazılım sistemleri tasarımında önemli bir rol oynamaktadır [4].

Yazılım şablonu, yazılım mimarisinin anahtar kavramlarından biridir ve kısaca bir problemin çözümü olarak ifade edilebilir. Öyle ki bu şablonların yeniden kullanılması sayesinde genel bir ilkeye bağlı kalınarak problemlerin çözümü gerçekleştirilir. Böylece, şablonlar çeşitli sistem tasarımlarında benzeri görülebilecek tekrarlayan sorunlara rahatlıkla uygulanabilecek ortak bir çözüm sundukları için yazılım maliyetlerini düşürmektedirler. Fakat önerilen yazılım mimari tasarım ve şablonlarının her zaman beklenen katkıyı sağlamadıkları, daha da önemlisi her zaman belli başlı kalite özelliklerine olumlu etki yapmadıkları bilinmektedir [5]. Bununla birlikte, birçok büyük ölçekli projede, son ürün haline getirilmiş yazılımın son kalitesi hakkında yargıya varabilmek için, gerçekleme aşaması beklenmek zorundadır. Şüphesiz ki yazılım kalitesi hakkında -mümkünse tasarım aşamasında- gerçeğe yakın bir yargıya sahip olmak, çok büyük bir artı değer ortaya çıkaracaktır. Bunu sağlamanın yolu ise, tasarım kalite özelliklerini incelemek ve kullanılan yazılım mimarilerinin ve şablonlarının bu kalite özelliklerine olan etkilerini belirlemektir. Bu konu özelinde tartışmaların yapıldığı öncül çalışmalar da vardır [6], ancak bu çalışmaların büyük çoğunluğu, son ürün aşamasında kalite özelliklerine odaklanarak yazılımın kalitesini tartışmıştır.

Bu makalede, İletişim Katmanı Yazılım Mimarisi (İKYM) adı altında verilen [7], nesneye dayalı, iletişim katman ve protokolleri tasarımı için kullanılan bir yazılım mimari modelinin, tasarım kalite özelliklerine etkisi araştırılmıştır. Bu amaçla İKYM kullanılarak, içinde iki ana protokol (TCP ve UDP) barındıran, TCP/IP taşıma katmanı tasarlanmıştır. İKYM tabanlı bu tasarımın sürdürülebilirlik, test edilebilirlik, yeniden kullanılabilirlik, anlaşılabilirlik ve güvenilirlik kalite özellikleri ölçülmüş, hem İKYM hem de İKYM tabanlı tasarım hakkında yargıya varılmaya çalışılmıştır. Sonuç olarak, yazılım kalitesine, gerçekleme aşamasını beklemeden, güncellemenin ve tasarım değişikliğinin daha az maliyetli olduğu tasarım aşaması sırasında karar verilebileceği gösterilmiştir.

Bu makale şu şekilde organize edilmiştir. 2. bölümde öncül çalışmalar genel olarak anlatılmış ve 3. bölümde kalite ölçütleri detaylarıyla açıklanmıştır. 4. bölümde İKYM ile TCP/IP taşıma katmanı tasarımı ve kalite analizi yer almıştır. Son bölümde ise İKYM tabanlı örnek TCP/IP taşıma katmanının kalite ölçüm sonuçları ve İKYM modelin tasarıma katkıları tartışılmıştır.

2 İLGİLİ ÇALIŞMALAR

Bu bölümde, nesneye yönelik bir yazılım kapsamındaki hatalı sınıfların, ilgili yazılıma uygulanan kalite ölçümlerinin sonuçları üzerinden tespitine yönelik yapılmış olan çalışmalar incelenmiştir. Yapılan literatür çalışması, tasarım aşamasında kalite ölçümü yapan çalışmalarla birlikte, geliştirdikleri yaklaşımların tasarıma uygulanabilirliği gözlemlenen kaynak kod temelli çalışmaları da kapsamaktadır.

Tahvildari ve Kontogiannis [8] yazılımdaki olası tasarım hatalarının bulunması için nesneye yönelik tasarım özelliklerinden faydalanmışlardır. Bu amaçla kullandıkları ölçütleri, kaliteli bir nesneye yönelik yazılımın sahip olması gereken farklı özellikleri temsil eden karmaşıklık, bağımlılık ve uyumluluk ölçütlerine göre üç ana sınıfta toplamışlardır. Yazarlar araştırmalarında, karmaşıklık için CDE (*Class Definition Entropy*), RFC (*Response For a Class*) ve WMC (*Weighted Methods per Class*); bağımlılık için DAC (*Data Abstraction Coupling*) ve RFC (*Response For a Class*); uyumluluk için ise LCOM (*Lack of Cohesion in Methods*) ve TCC (*Tight Class Cohesion*) ölçütlerini kullanmışlardır.

Tahvildari ve Kontogiannis'in bir başka çalışması [9] ise nesneye yönelik yazılımlarda sürdürülebilirlik kalite özelliği üzerinedir. Çalışmada, bir yazılımın hem mimari tasarımı hem de kaynak kod seviyesinde incelemesi yapılmış, yazılım sürdürülebilirliğini olumsuz yönde etkileyen kısımların ortak özellikleri bulunup bazı tespitler yapılmıştır. Salehie ve arkadaşları da nesneye yönelik yazılımda sürdürülebilirlik kalite özelliği üzerine bir çalışma [10] yapmışlardır. Adı geçen çalışmada, uygulanan ve önerilen yöntem iki aşamalı olup, ilk aşamada bariz yazılım tasarım kusurları bulunmaya çalışılmış, ikinci aşamada ise o an kusur gibi görünmeyen ama ileride sorun olabilecek alanlar tespit edilmeye çalışılmıştır.

Marinescu [11] ise iyi bilinen iki tasarım hatasının (büyük sınıf, veri sınıfı) yazılımlarda belirlenmesine yönelik bir yöntem geliştirmiş ve bu yöntemi örnek bir endüstriyel yazılımda uygulamıştır. Çalışmada kullanılan ölçütlerin (WOC-*Weight of a Class*, NOPA-*Number of Public Attributes*, NOAM-*Number of Access Methods*) tasarım modeli üzerine uygulanabilmesinden dolayı, yazar tamamen dilden bağımsız bir kalite ölçüm yöntemi geliştirdiğini ileri sürmektedir. Marinescu'nun diğer bir çalışmasında [12] ise, yazılım tasarım hataları daha farklı ölçütlerle tespit edilmeye çalışılmıştır. Üç farklı ölçüt (WMC-*Weighted Method Count*, NOC-*Number of Children*, TCC-*Tight Class Cohesion*) üç farklı yazılıma uygulanmış ve sonuçlar yorumlanmıştır. Sahraoui ve arkadaşları yaptıkları çalışmada [13], C++ ile kodlanmış yazılımlardaki potansiyel tasarım hatalarını çeşitli ölçütlerle (CBO-*Coupling Between Object Classes*, DAC-*Data Abstraction Coupling, vb.*) tespit etmeyi amaçlamışlardır. Araştırmaları kapsamında hata tespit işlemini otomatik hale getiren bir araç geliştirmişler ve böylece kullanıcılara tasarım hatalarını bulma imkânı sağlamışlardır.

Son olarak, Jehad Al Dallal [24] tasarım aşamasında uygulanabilecek yeni bir uyumluluk ölçütü geliştirdiğini iddia etmektedir. Geliştirilen ölçüt doğrudan etkileşimde bulunan metot-metot, veri üyesi-metot ve veri üyesi-veri üyesi çiftleri üzerinden bir hesaplama yapmaktadır. Ayrıca, yapılan karşılaştırmalarla bu yeni ölçütün,

örnek olarak seçilen diğer uyumluluk ölçütlerinden daha duyarlı olduğu ileri sürülmektedir.

3 KALİTE ÖLÇÜMLERİ

Kalite özelliklerinin doğrudan ölçümü zor, hatta bazı durumlarda imkânsızdır [22, 23]. Değişik ölçütler bir araya getirilerek kalite özellikleri hakkında bazı sayısal değerlere ulaşılmaya çalışılmaktadır [11], fakat bu çalışmalar genelde öznel nitelik taşımaktadır. Ayrıca bu ölçümler genel olarak yazılımın gerçekleme aşaması sonrasında yapılmakta ve elde edilen sonuçlara göre tasarımın değiştirilmesine, hatta buna bağlı olarak gerçeklemenin değiştirilmesine ihtiyaç duyulmaktadır. Ne yazık ki bu tür güncellemeler gerek proje takvimi, gerekse doğacak maliyet açısından her zaman mümkün olmamaktadır. Bundan dolayı erken tasarım aşamasında yapılan kalite ölçümleri ve değerlendirmeler büyük önem kazanmaktadır.

Bu çalışma kapsamında nesneye yönelik tasarım için temel olan özellikler göz önünde bulundurulmuş, bu özellikleri doğrudan etkileyen ölçütler seçilmiş ve yapılan ölçümlere dayanarak İKYM tabanlı TCP/IP taşıma katmanı tasarımının kalite özellikleri hakkında bazı sonuçlar çıkarılmaya çalışılmıştır. Seçilen ölçütleri üç ana başlık altında toplamak mümkündür: Karmaşıklık ölçütleri (*complexity metrics*), bağımlılık ölçütleri (*coupling metrics*) ve uyumluluk ölçütleri (*cohesion metrics*).

3.1 Karmaşıklık Ölçütleri (Complexity Metrics):

Yazılımdaki karmaşıklık, sınıfların iç ve dış yapısını, sınıflar arası ilişkileri kavramadaki zorluğun derecesidir [16]. Yazılımın karmaşıklık seviyesi arttıkça yazılımı çözümlemek, test etmek, değiştirmek, yeniden kullanabilmek ve yazılımın sürdürülebilirliğini korumak zorlaşır.

Karmaşıklık ölçüm yöntemlerinin, genel geçer kıstaslara sahip olmamaları ve farklı çalışmalarda farklı tanımlara ve sınırlara sahip olmaları sebebiyle uygun ölçütü seçmek oldukça zordur. Bu çalışmada karmaşıklık ölçümü için Malik ve Chhillar [18]'ın çalışması referans alınmış ve aşağıda listelenen ölçütler kullanılmıştır:

- CMCM (*Class Member Complexity Measure*): Bir sınıfın ortak veya korumalı metot ve veri üyeleri toplam sayısıdır. CMCM değeri arttıkça sınıftaki öznitelik sayısı artacağından karmaşıklık artacak ve güvenilirlik azalacaktır. Ayrıca, CMCM değeri yüksek olan bir sınıfın arayüzü daha karmaşık hale geleceğinden sınıfın yeniden kullanılabilirlik değeri de düşük olacaktır [17, 27].

$$CMCM = N_d + N_f \qquad (1)$$

N_d = Ortak/Korumalı veri üye sayısı

N_f= Ortak/Korumalı metot sayısı

- CICM (*Class Inheritance Complexity Measure*): Kalıtım (*inheritance*) yoluyla aktarılan karmaşıklık ölçütüdür. CICM değerinin hesaplanma yöntemi aşağıdaki formül ile belirlenmiştir:

$$CICM = \begin{cases} 0, & i = 0 \\ n + \sum_{i=1}^{n} R_i, & i \geq 1 \end{cases} \quad (2)$$

n = bir sınıfa ait üst sınıf sayısı

$R_i = i$ nolu üst sınıfın CICM değeri

Bilindiği üzere kalıtım yeniden kullanılabilirliği tetikleyen bir unsurdur. Dolayısıyla CICM değerinin belirli bir aralıkta kalması şartı ile yeniden kullanılabilirlik kalite özelliği ile doğru orantılı olduğu söylenebilir. Ayrıca üst sınıftan kalıtımla alt sınıfa geçen üyeler alt sınıfın karmaşıklığını etkilediğinden, üst sınıfın CICM değeri arttıkça alt sınıfın karmaşıklığı da artacaktır. Bununla beraber, bir sınıfın kalıtım karmaşıklık seviyesi arttıkça anlaşılabilirlik ve sürdürülebilirlik değerleri azalacaktır [17, 28].

3.2 Bağımlılık Ölçütleri (Coupling Metrics)

Bağımlılık ölçümü genel olarak sınıf bazında yapılır ve iki sınıftan en az birinin diğerine etki etmesiyle oluşan bağımlılığın derecesi olarak tanımlanır. A sınıfının B sınıfına olan bağımlılığını belirleyen etmenler aşağıdaki şekilde sıralanabilir [16]:

- A sınıfının içinde B sınıfı cinsinden bir referans, işaretçi ya da nesne vardır.
- A sınıfının nesneleri B sınıfının nesnelerinin metotlarını çağırıyordur.
- A sınıfının bir metodu parametre olarak B sınıfı tipinden veriler alıyordur ya da geri döndürüyordur.
- A sınıfının bir metodu B tipinden bir yerel değişkene sahiptir.

Nesneye yönelik kaliteli bir yazılımda sınıflar arası bağımlılığın mümkün olduğunca düşük olması tercih edilir (*low coupling*). Bir sınıfın bağımlılık seviyesi yükseldikçe, bu sınıfı sistemin modüler bir parçası olarak görüp işlem yapmak zorlaşacaktır. Bağımlılığı yüksek olan bir sınıftaki değişim diğer sınıfları etkileyeceğinden genel olarak yazılımın sürdürülebilirliği de düşük olacaktır. Sistemde bağımlılığı yüksek sınıflar arttıkça sınıfları birbirinden ayrı anlamak zorlaşacak, dolayısıyla anlaşılabilirlik ve test edilebilirlik de düşecektir. Ayrıca, bağımlılığı yüksek olan sınıfları tekrar kullanmak zor olacağından, bu sınıfların yeniden kullanılabilirlik değeri de düşecektir [17, 28].

Bu çalışma kapsamında CALM (*Class Aggregation Level Measure*) [18] bağımlılık ölçütü kullanılmıştır. Bu ölçüt bir sınıftaki kullanıcı tanımlı öznitelik sayısının, o sınıftaki toplam öznitelik sayısına oranı olarak hesaplanır.

$$CALM = \frac{U_d}{N_d} \quad (3)$$

U_d = Kullanıcı tanımlı öznitelik sayısı

N_d = Toplam öznitelik sayısı

3.3 Uyumluluk Ölçütleri (Cohesion Metrics):

Uyumluluk ölçümü, sınıftaki metot ve veri üyelerinin kendi içindeki uyumluluğunu belirtir. Her sınıfın tek bir sorumluluğu olmalıdır. Eğer bir sınıf kendi içinde birbirinden farklı ve bağımsız işler yapıyorsa, birbiriyle ilgili olmayan veri üyeleri barındırıyorsa veya çok fazla iş yapıyorsa sınıfın uyumluluğu düşüktür. Kaliteli bir nesneye yönelik yazılımda sınıfların kendi içlerindeki uyumluluğunun mümkün olduğunca yüksek olması tercih edilir (*high cohesion*). Uyumluluk arttıkça sınıfın kendi içindeki kararlılığı artacak, bundan dolayı sınıfın sürdürülebilirlik ve güvenilirlik değerleri de artacaktır [17]. Ayrıca, uyumluluk arttıkça sınıf daha modüler bir yapıya kavuşacağından test edilebilirlik değeri de artacaktır.

Bu çalışmada, uyumluluk ölçütü olarak CCOM (*Class Cohesion Measure*) [18] kullanılmıştır. CCOM değeri aşağıda belirtilen formül ile hesaplanır.

$$CCOM = \frac{AA_{SC}}{MA_{SC}} \qquad (4)$$

AA_{SC} = Bir sınıfta bulunan her bir veri üyesinin o sınıfın kaç tane metodu tarafından erişildiği hesaplanır ve bu değerler toplanır

MA_{SC} = Bir sınıfta bulunan her metot her bir veri üyesine erişirse AA_{SC} 'nin göstereceği değerdir (toplam veri üye sayısı x (toplam metot sayısı - 1))

4 İKYM ile TCP/IP TAŞIMA KATMANI TASARIMI ve KALİTE ANALİZİ

TCP/IP taşıma katmanının temel görevi uçtan uca birimler arası oturumları yöneterek uygulamalar arası veri iletimini sağlamaktır. Taşıma katmanının üstünde uygulama katmanı, altında ise internet katmanı yer alır. TCP/IP taşıma katmanında TCP ve UDP olmak üzere iki ana protokol bulunur. UDP bağlantısız bir protokol olup veri güvenliği içermez ve genellikle ses, video v.b. aktarımı yapan gerçek zamanlı uygulamalar tarafından tercih edilir. TCP ise bağlantılı, akış kontrollü ve güvenilir bir protokol olup verinin hedefe ulaşıp ulaşmadığını onay mekanizmasıyla belirleyip, ulaşmadığı durumda veri tekrarını gerçekleştirir. TCP, dosya transferi, internet tarayıcı gibi kayıpsız veri transferi gerektiren uygulamalar tarafından tercih edilir.

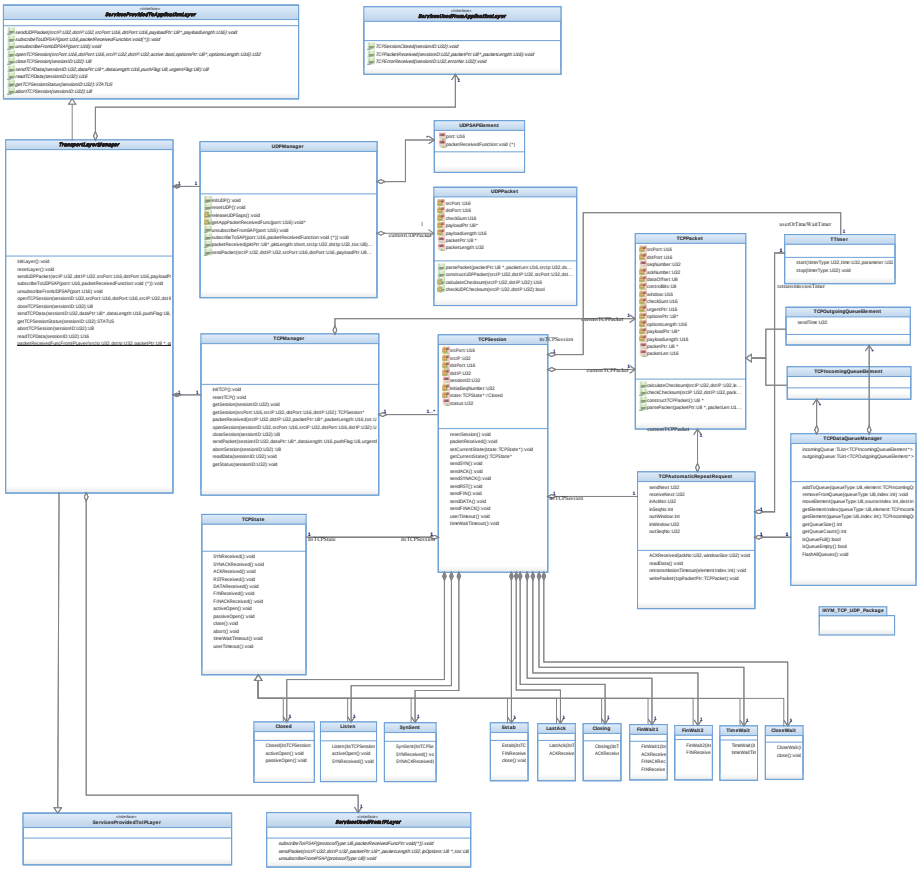
UDP ve TCP protokolleri sırasıyla RFC768 [19] ve RFC793'de [20] tanımlanmıştır. Bu bölümde İKYM (İletişim Katmanı Yazılım Mimarisi) [7] modeli kullanılarak TCP/IP taşıma katmanının tasarımı anlatılmış, kalite ölçümleri yapılarak ölçüm sonuçlarının analizi yapılmış ve bu sonuçlar yorumlanmıştır.

4.1 TCP/IP Taşıma Katmanı Tasarımı

Önceki çalışmamızda [7] TCP/IP taşıma katmanı sadece sınıf ve ilişkiler seviyesinde tasarlanmıştı. Bu çalışmamızda sınıfların içerdiği değişkenler, metotlar ve me-

totlarla değişkenler arasındaki ilişkiler detaylandırılarak detaylı bir tasarım elde edilmiştir. Uygun kalite ölçütleri seçilerek gerçekleme aşaması beklenmeden kalite ölçümleri yapmak ve ölçüm sonuçlarına göre yazılımın kalitesi hakkında değerlendirme yapmak mümkündür. Malik ve Chhillar [18]'ın da belirttiği gibi kodlamaya (gerçeklemeye) başlamadan alınan geribildirim zaman, para ve işgücü maliyetini düşürecek, bunun yanı sıra tasarımın doğru yönde ilerlemesine katkıda bulunacaktır.

İKYM modeli kullanılarak tasarlanan TCP/IP taşıma katmanı, toplam 26 adet sınıf ve bu sınıfların birbirleriyle olan ilişkilerinden oluşmaktadır. Bu çalışma kapsamında tasarlanan TCP/IP taşıma katmanına ait UML Sınıf Modeli Şekil 1'de gösterilmiştir.



Şekil 1. TCP/IP Taşıma Katmanı UML Sınıf Modeli

4.2 TCP/IP Taşıma Katmanı Tasarımının Kalite Ölçümleri

Bu çalışma kapsamında, seçilen kalite ölçütleri Malik ve Chhillar [18]'ın belirttiği yöntemle örnek TCP/IP taşıma katmanı için hesaplanmıştır. Her ölçüt için elde edilen ölçüm sonuçlarının en düşük, en yüksek ve ortalama değerleri ayrı tablolar halinde

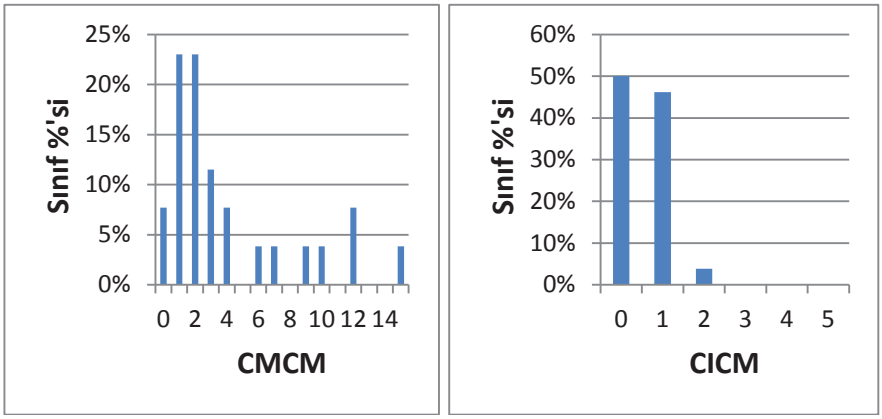
gösterilmiştir (bkz. Tablo 1-4). Buna ek olarak her ölçütün sınıfların yüzdesel dağılımına göre değer değişimi de ayrıca grafiksel olarak gösterilmiştir (bkz. Şekil 2 ve 3). Sonrasında ise elde edilen bu verilere dayanılarak hem örnek TCP/IP taşıma katmanı tasarımının hem de İKYM modelinin kalitesi hakkında bazı çıkarsamalar yapılmıştır.

CMCM Ölçüm Analizi.

Minimum	Maksimum	Ortalama
0	15	3,96

Tablo 1. CMCM Ölçüm Sonuçları

Tablo 1'de görüldüğü üzere örnek TCP/IP taşıma katmanı tasarımının CMCM değeri ortalaması 3,96 çıkmıştır. Bu değerin düşük çıkması ele alınan sınıfların karmaşıklık düzeyinin düşük, güvenilirliğinin yüksek olduğunu göstermektedir. Bunun yanı sıra CMCM değeri düşük olan bu sınıfların yeniden kullanılabilirlik değerinin yüksek olduğunu söylemek mümkündür. CMCM değerinin yüksek çıktığı sınıflar, beklendiği gibi, karmaşıklığı yüksek olan yönetici sınıflardır.



Şekil 2. CMCM ve CICM Ölçümlerinin Yüzdesel Dağılımları

CICM Ölçüm Analizi.

Minimum	Maksimum	Ortalama
0	2	0,54

Tablo 2. CICM Ölçüm Sonuçları

CICM ölçümlerinin istatistiksel değerleri Şekil 2'de gösterilmiştir. Görüldüğü üzere, en düşük CICM değeri 0, en yüksek CICM değeri 2 ve ortalama CICM değeri 0,54 çıkmıştır. Yapılan ölçümlerde kalıtım seviyesinin 2 ile sınırlı kalması tasarımın kont-

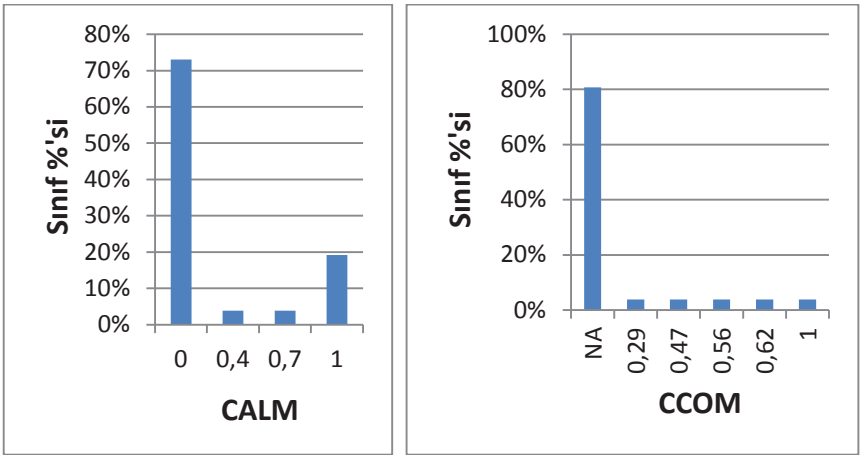
rol edilebilir düzeyde olduğunu göstermektedir [18]. Ortalama değer göz önüne alındığında ise sistemdeki sınıfların karmaşıklık değerlerinin düşük olduğu, anlaşılabilirlik, tekrar kullanılabilirlik ve sürdürülebilirlik değerlerinin uygun seviyede olduğu söylenebilir.

CALM Ölçüm Analizi.

Minimum	Maksimum	Ortalama
0	1	0,24

Tablo 3. CALM Ölçüm Sonuçları

Şekil 3'teki CALM ölçümünün yüzdesel dağılım grafiğine bakıldığında, sınıfların %73'ünün 0 değerine sahip olduğu görülmektedir. Buna bağlı olarak Tablo 3'te gösterilen ortalama CALM değeri oldukça düşük çıkmıştır. Önceki bölümlerde anlatıldığı üzere CALM değerinin düşük çıkması sınıflar arasındaki bağımlılığın az olduğunu, dolayısıyla anlaşılabilirlik, test edilebilirlik ve yeniden kullanılabilirlik kalite özelliklerinin yüksek olduğunu göstermektedir. CALM değerinin yüksek çıktığı sınıflar, CMCM ölçütünde olduğu gibi karmaşıklığı yüksek olan yönetici sınıflarıdır.



Şekil 3. CALM ve CCOM Ölçümlerinin Yüzdesel Dağılımları

CCOM Ölçüm Analizi.

Minimum	Maksimum	Ortalama
0,29	1	0,59

Tablo 4. CCOM Ölçüm Sonuçları

Sadece veri üyesi veya metoda sahip olan sınıfların CCOM değerleri tanımsızdır (bkz. Formül 4). Örnek TCP/IP taşıma katmanı tasarımında CCOM ölçümü yapılamayan sınıf oranı %81 gibi yüksek bir değerdir. Alt ve üst katmanlarla olan arayüz sınıfları ile Durum Tasarım Şablonuna [21] göre oluşturulan durum sınıfları herhangi bir veri üyesine sahip değildir ve bu yüzden CCOM değerleri tanımsızdır. Aynı şekilde, tasarımda kullanılan veri sınıflarının da metodu olmadığı için CCOM değeri ölçülememektedir.

Nesneye dayalı bir tasarımda beklenen CCOM değeri 1 civarındadır. Tablo 4'te görüldüğü üzere örnek tasarımın ortalama CCOM değeri 0,59 çıkmıştır. Henüz gerçekleme aşamasına gelinmeyen bu sistemdeki sınıfların CCOM değerlerine bakıldığında, bu tasarımın kararlılığının ideal seviyeye yakın [18] olduğu söylenebilir. Bundan dolayı ölçümü yapılan sınıfların sürdürülebilirlik, güvenilirlik ve test edilebilirlik kalite özelliklerinin ideal seviyeye yakın olduğu gözlemlenmiştir.

Bunun yanında, Jehad Al Dallal [24], mevcut uyumluluk ölçütlerinin, kalıtım gibi bazı anahtar nesneye dayalı dil özelliklerini dikkate almadığını, gerçekleme aşamasında belirlenecek olan gerçek etkileşimlere motive olduklarını, dolaylı etkileşimleri ölçemediklerini belirtmiştir. Taranjeet Kaur [25] ve Heung Seok Chae [26] ise mevcut ölçütlerin uyumluluk ölçümleri için yeterli olmadığını ve sorgulanır olduklarını belirtmiştir.

Sonuç olarak, her ne kadar literatürde uyumluluk ölçütlerine çeşitli eleştiriler varsa da; tasarım aşamasında elde ettiğimiz bu farkındalıkla, gerçekleme aşamasına geçmeden sınıflar tekrar bu gözle incelendi. TCP/IP taşıma katmanı özelinde de düşük CCOM değerine sahip olan *TCPAutomaticRepeatRequest* ve *TCPSession* sınıflarının birbirinden bağımsız işlevleri öznesinde toplayan sınıflar olduğu gözlendi ve bu sınıfların bölünmesine karar verildi.

5 Sonuç

Bu çalışma kapsamında, TCP/IP taşıma katmanı yazılımı nesneye yönelik bir iletişim katmanı yazılım mimarisi olan İKYM ile tasarlanmıştır. İletişim katmanı yazılım tasarımlarında İKYM kullanılarak, yazılım kalite özelliklerinin sağlanması ve yazılım kalitesinin arttırılması hedeflenmektedir. Ayrıca, tasarımda kullanılan İKYM'nin nesneye yönelik olması hem nesneye yönelik programlamanın faydalarından yararlanılmasına hem de nesneye yönelik tanımlanan yazılım kalite ölçütlerinin kullanılmasına olanak sağlamıştır.

Ayrıca bu çalışmada, TCP/IP taşıma katmanı tasarımı uygun ölçütlerle, sınıf bazında, gerçekleme yapmadan ölçülmüş ve sorunlu olabilecek sınıflar belirlenerek tasarımın erken aşamalarında ilgili iyileştirmelerin yapılabileceği gösterilmiştir. TCP/IP taşıma katmanının sınıf bazındaki kalite ölçüm sonuçlarının kabul edilebilir seviyenin üstünde çıkması ve yazılımın görece kısa sürede tasarlanabilmiş olması; İKYM'nin başarılı bir nesneye yönelik yazılım mimarisi olduğunu göstermiştir. Sonuçlar, İKYM'nin test edilebilir, sürdürülebilir, yeniden kullanılabilir, anlaşılabilir ve güvenilir olduğuna da işaret etmektedir. Bu özelliklere İKYM kullanılan diğer protokol tasarımlarının da sahip olacağı düşünülebilir. Son olarak, yazılım tasarımında

İKYM kullanımının yazılım geliştirme maliyetlerini büyük oranda düşürmesi beklenmektedir.

6 Kaynakça

1. David Garlan, "*Software Architecture: a Roadmap*", Conference on The Future of Software Engineering (ICSE '00), pp. 91-101, 2000
2. David Garlan and Mary Shaw, "*An Introduction to Software Architecture*", Technical Report, January 1994
3. International Standards Organization: Information Technology - Software Product Quality - Part 1: Quality Model, ISO/IEC FDIS 9126-1
4. Francisca Losavio and Ledis Chirinos, Nicole Lévy and Amar Ramdane-Cherif, France "*Quality Characteristics for Software Architecture*" in Journal of Object Technology, vol. 2, no. 2, March-April 2003, pp. 133-150.
5. Mawal Ali and Mahmoud O. Elish, "*A Comparative Literature Survey of Design Patterns Impact on Software Quality*", International Conference of Information Science and Applications (ICISA), June 2013
6. Brian Huston, "*The Effects of Design Pattern Application on Metric Scores*", The Journal of Systems and Software, 2001, pp. 261-269
7. İ. Karaaslan, T. Afacan, E. Demircan, Ö. Başol and E. Zaim, "*İletişim Katmanı Yazılım Mimarisi – Communication Layer Software Architecture*" 7. Ulusal Yazılım Mühendisliği Sempozyumu – National Software Engineering Symposium (UYMS), İzmir, September, 2013
8. Ladan Tahvildari and Kostas Kontogiannis, "*A Metric-Based Approach to Enhance Design Quality Through Meta-Pattern Transformations*", Conference On Software Maintenance And Reengineering (CSMR'03), March 2003, pp. 183-192
9. Ladan Tahvildari and Kostas Kontogiannis, "*A Software Transformation Framework for Quality-Driven Object-Oriented Re-Engineering*", International Conference on Software Maintenance (ICSM'02), 2002
10. M. Salehie, S. Li and L. Tahvildari, "*A Metric-Based Heuristic Framework to Detect Object-Oriented Design Flaws*", Conference on Program Comprehension (ICPC'06), 2006
11. Radu Marinescu, "*Detecting Design Flaws via Metrics in Object-Oriented Systems*", Conf. and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS'01), 2001
12. Radu Marinescu, "*Using Object-Oriented Metrics for Automatic Design Flaws Detection in Large Scale Systems*", Object-Oriented Technology Ecoop'98, 1998
13. Houari A. Sahraoui, Robert Godin and Thierry Miceli, "*Can Metrics Help to Bridge the Gap Between the Improvement of 00 Design Quality and Its Automation?*", International Conference on Software Maintenance, 2000, pp.154-162
14. ODTÜ Bilgisayar Topluluğu Elektronik Dergisi, http://e-bergi.com/2008/Eylul/Yeniden-Kullanim-Software-Reuse, Last date accessed: April 2014
15. Tuna Türk, "*The Effect of Software Design Patterns on Object-Oriented Software Quality and Maintainability*", A thesis submitted to the Graduate School of Natural and Applied Sciences of METU, September 2009
16. U. Erdemir, U. Tekin, F. Buzluca, "*Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi*", Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS08), 2008

17. Nurdan Canbaz ve Feza Buzluca, "*Yazılım Kalitesi İçin Yinelemeli Ölçme Yöntemi*", 4. Ulusal Yazılım Mimarisi Konferansı (UYMK'12), 27-29 Eylül, İzmir, 2012
18. Nisha Malik and Rajender Singh Chhillar, "*New Design Metrics for Complexity Estimation in Object Oriented Systems*", International Journal on Computer Science and Engineering (IJCSE), Vol. 3 No. 10, pp.3367-3382, October 2011
19. J. Postel, "*User Datagram Protocol"*, RFC768, 28 August 1980
20. Information Sciences Institute University of Southern California, "*Transmission Control Protocol"*, RFC 793, September 1981
21. E. Gamma, R. Helm, R. Johnson and J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addson Wesley, 1995
22. Barbara Kitchenham and Shari Lawrence Pfleeger, "*Software Quality: The Elusive Target*", *IEEE Software*, pages 12-21, 1996.
23. Linda H. Rosenberg and Lawrence E. Hyatt, "*A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality*", *8th Annual Software Technology Conference Utah*, April 1996
24. Jehad Al Dallal, "*A Design-Based Cohesion Metric for Object-Oriented Classes*", World Academy of Science, Engineering and Technology, Vol:1, No:10, 2007
25. Taranjeet Kaur and Rupinder Kaur, "*Comparison of Various Lacks of Cohesion Metrics*", International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-2, Issue-3, February 2013
26. Heung Seok Chae and Yong Rae Kwon, "*A Cohesion Measure for Classes in Object-Oriented Systems*", Software Practice and Experience, 2001
27. Ladan Tahvildari, "*Assessing the Impact of Using Design-pattern-based Systems*", A thesis submitted to the University of Waterloo, 1999
28. Ladan Tahvildari and Kostas Kontogiannis, "*Improving design quality using meta-pattern transformations: a metric-based approach*", Journal of Software Maintenance and Evolution, 16: 331-361, 2004