Nesne Yönelimli Tasarım Metrikleri ve Kalite Özellikleriyle İlişkisi

M.Hanefi CALP[1], Nursal ARICI*[2]

Bilişim Enstitüsü, Gazi Üniversitesi, Ankara, Türkiye[1]

Elektronik ve Bilgisayar Eğitimi Anabilim Dalı, Gazi Üniversitesi, Ankara, Türkiye[2]

Geliş/Received : 07.02.2011, Kabul/Accepted : 20.07.2011

ÖZET

Nesne yönelimli yazılım kalitesini ölçmek için başvurulan yöntemler geleneksel yöntemlerden farklıdır. Dolayısıyla, sözkonusu yazılımların kalitesini ölçmede kullanılan metrikler ve kalite özellikleri de farklı olacaktır. Nesne yönelimli tasarım metrikleri; geliştirilen yazılımların; etkililik/verimlilik, karmaşıklık, anlaşılırlık, yeniden kullanılabilirlik, test edilebilirlik ve dayanıklılık gibi kalite özellik düzeylerinin ölçülmesinde kullanılır. Bu makalede, literatürde önemli ölçüde kabul görmüş ve nesne yönelimli tasarım kalitesini ölçmede kullanılan Chidamber& Kemerer'in metrik kümesi kısaca açıklanmış ve bu metriklerin yazılım kalite özellikleri ile ilişkisini göstermek üzere deneysel bir çalışma yapılmıştır.

Anahtar Kelimeler: Nesne yönelimli tasarım, metrikler, kalite özellikleri, yazılım kalitesi

Object Oriented Design Metrics and the Relationship With Quality Attribute

ABSTRACT

Referenced methods for measuring the quality of object-oriented software differ from traditional methods. Therefore, the metrics and quality attributes that used to measure the quality of such software will be also different. Object-oriented design metrics are used for measuring the level qualification of developed software such as the efficiency, complexity, understandability, reusability, testability and mantainability. In this article, the metric suite of Chidamber&Kemerer that accepted significantly in the literature and used for measuring object-oriented design quality was explained briefly, and an experimental study was carried out to show the relationship between this metrics and software quality attributes

Keywords: Object-oriented design, metrics, quality attributes, software quality

1. GİRİŞ (INTRODUCTION)

Yazılım endüstrisindeki gelişmeler, yazılım geliştirme sürecini etkileyerek bu süreci proje yönetimi çerçevesinde geliştirilen faaliyetler haline getirmiştir. Günümüzde artık küçük ölçekli yazılımlar bile sadece bir kişinin değil, iyi yönetilen bir yazılım ekibinin birlikte gerçekleştirdiği, binlerce kod satırından meydana gelen projelerdir. Yazılım projelerinin gittikçe büyümesi, karmaşıklaşması ve boyutlarının sürekli artması yazılımın kalitesini de etkilemekte, bakım maliyetlerinin zaman ve çaba olarak artması problemlerini de beraberinde getirmektedir. Nesne yönelimli programlama (object oriented programming) bu problemlere çözüm olarak ortaya çıkmış bir programlama yaklaşımıdır. Bu yaklaşımı yukarıda söz edilen problemlere karşı çözüm haline getiren en belirgin özelliği yazılımda birimselliği (modularity) benimsemesidir. Birimsellik; bilgi gizleme, veri soyutlama, kalıtım ve çok biçimlilik gibi nesne yönelimli programlama yaklaşımı ile ortaya çıkan tekniklerin etkin bir şekilde uygulanması sayesinde gerçeklenir. Bu sayede, yazılımın bakımının ve aynı yazılım üzerinde birden fazla kişinin çalışmasının kolaylaşması gibi yazılım kalitesini olumlu yönde etkileyen avantajlar elde edilmiş olur (1).

Bu avantajlardan dolayı, nesne yönelimli tasarım ve geliştirme, günümüz yazılım endüstrisinde önemli bir yere sahiptir. Dolayısıyla, nesne yönelimli programlama yaklaşımının kullanılması ile birlikte sözkonusu yazılımların kalitesini ölçmek için başvurulan yöntemler geleneksel yöntemlerden farklı olup yeni teknikler ve yazılım kalite metrikleri geliştirilmektedir (2).

Son yıllarda, yazılım kalite metrikleri çok önem kazanmaktadır. Metrikler, yazılımların birçok yönden değerlendirilmesini sağlar. Bu değerlendirmeler, kaliteli bir yazılım geliştirilmesinde yardımcı olur (3).

Yazılım kalitesi kavramı, kişiye göre farklı anlamlar yüklenen bir kavramdır (3,4). Geliştiricilerin bakış açısıyla kalite; maliyet ve zaman konusunda doğru tahmin etme, yazılımı daha kolay test etme ve daha iyi bakım yapma anlamını taşır (2,3). Kullanıcı bakış açısı ile de; tasarım düzeni, kullanım kolaylığı ve anlaşılabilirlik gibi kavramlarla ifade edilebilmektedir. Kaliteli bir yazılım, hem müşterinin sürekliliğini sağlar hem de oluşabilecek olumsuz durumlara karşı insan hayatını korur (3).

* *Sorumlu Yazar (Corresponding Author)*
e-posta: nursal@gazi.edu.tr

Digital Object Identifier (DOI) : 10.2339/2011.14.1, 9-14

Nesne yönelimli tasarım kalitesini değerlendirmek amacıyla hiyerarşik bir model tanımlanır. Bu modelde sınıfların, nesnelerin ve bunlar arasındaki ilişkilerin yapısal ve davranışsal tasarım özellikleri nesne yönelimli tasarım metrikleri kullanılarak değerlendirilmektedir (3,5).

Bu çalışmada, 1. Bölüm'de "Giriş", 2. bölümde ise Chidamber ve Kemerer'in geliştirdiği metrik kümesinin kısaca tanımları ve kalite özellikleri, 3. Bölüm'de de "Metriklerin Yorumlanması" ile ilgili bilgiler verilmiştir. 4. Bölüm'de seçilen bir yazılım projesinin dört ayrı sürümünün metrik değerleri ölçülerek elde edilen bulguların kalite özellikleri ile ilişkisi değerlendirilmiştir. Son olarak, Bölüm 5'te, tüm bu çalışmalardan çıkarılan "Sonuç" bölümü yer almaktadır.

2. NESNE YÖNELİMLİ METRİKLER VE KALİTE ÖZELLİKLERİ (OBJECT ORIENTED METRICS AND QUALITY ATTRIBUTES)

Nesne yönelimli metrikler, yazılım test sürecinin etkililiğinin önemli bir göstergesidir. Goodman yazılım metriklerini (6);

"Ölçüm temelli tekniklerin yazılım geliştirme sürecine sürekli uygulanması, süreç ve (onun) ürünlerinin gelişmesi için bu tekniklerin kullanılmasıyla birlikte, anlamlı ve zamanında bilgi yönetimini sağlamak" olarak tanımlar.

Schulmeyer ise metriği (7);

"Bir sistemin, bileşenin veya sürecin verilen özelliğe sahip olma derecesinin nicel ölçümü" olarak tanımlar.

Geleneksel tasarımda kullanılan metrikler, nesne yönelimli yazılımlarda doğrudan uygulanamayabilir. Bu bağlamda; nesne yönelimli yazılım ile ilgilenen geliştiriciler, araştırmacılar ve kalite kontrolcülerin ihtiyaçlarını karşılamak amacıyla yeni metrik kümeleri geliştirilmektedir (8). Literatürde; Chidamber&Kemerer, Brito e Abreu (Metrics for Object Oriented Design-MOOD) ve Bansiya&Davis QMOOD (Quality Model for Object-Oriented Design) gibi kabul görmüş nesne yönelimli birçok yazılım metrik kümesi bulunmaktadır (9). Bu çalışmada, Chidamber ve Kemerer'in geliştirdiği metrik kümesinden yararlanılacaktır. Bu metrikler ve özellikleri aşağıda kısaca açıklanmıştır.

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

Nesne yönelimli tasarım için kullanılan bu metrik grubu, bir sistemin bütün olarak değerlendirilmesinden ziyade sistemdeki sınıfları geniş bir şekilde değerlendirmeyi amaçlar. Chidamber & Kemerer, nesne yönelimli tasarım için altı adet metrik tanımlar.

2.1.1. Sınıfın ağırlıklı metot sayısı (Weighted Methods per Class - WMC)

Sınıfın ağırlıklı metot sayısı (WMC); bir sınıftaki metotların karmaşıklık derecesi veya sayısıdır. Bir sınıfın metotlarının karmaşıklığı ve sayısı, sınıfın geliştirilmesine ve bakımına harcanacak zaman-çaba hakkında fikir verir (9). Bu metrik, bir sistemdeki sınıfların ortalaması hesaplanarak ölçülür (10). WMC; nesne yönelimli bir yazılımın anlaşılırlığını, yeniden kullanılabilirliğini ve dayanıklılığını/bakılabilirliğini ölçmede kullanılır (11).

2.1.2. Kalıtım ağacının derinliği (Depth of Inheritance Tree - DIT)

Sınıfın kalıtım ağacının köküne uzaklığıdır (9). Bu değerin çok yüksek olması test edilebilirliğin çok düşük olduğunu, aksi halde nesne yönelim ilkelerinin fazla kullanılmadığını gösterir (12). Bu metrik; verimliliği, yeniden kullanımı, anlaşılırlığı ve test edilebilirliği ölçer (11).

2.1.3. Alt sınıf sayısı (Number of Children - NOC)

Bir sınıftan direk türetilmiş alt sınıfların sayısıdır. NOC, kalıtımlı ifadeler dikkate alınarak hesaplanır. Bir sınıf hiyerarşisinin genişliğini ölçer. Alt sınıf sayısının fazla olması; yeniden kullanımın yüksek olduğunu, daha çok hatanın oluşabileceğini (13), test esnasında harcanacak zamanı-çabayı ve kalıtımın yanlış kullanıldığını gösterir (9). NOC; verimlilik, yeniden kullanılabilirlik ve test edilebilirlik düzeyini ölçer (11).

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

Bir sınıf içindeki özellik (attribute) ya da metotların (method) diğer sınıfta kullanılması ve sınıflar arasında kalıtımın olmaması durumunda iki sınıf arasında bağımlılıktan bahsedilebilir (13). CBO; verimliliği ve yeniden kullanılabilirliği ölçmede kullanılır (11).

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

Bir sınıftan bir nesnenin metotları çağrılması durumunda, bu nesnenin tetikleyebileceği tüm metotların sayısı RFC değerini verir. Yani, bir sınıfta yazılan ve çağırılan toplam metot sayısıdır (9). Bu metrik; sınıf seviye tasarım metriklerinden olup (14); anlaşılabilirliği, dayanıklılığı, karmaşıklığı ve test edilebilirliği ölçmede kullanılır (11).

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

LCOM, n adet kümenin kesişiminden oluşan kümelerdeki uyumsuzlukların sayısıdır ve metotlardaki benzerlik derecesini ölçer. Metotlardaki uyum eksikliği; bir sınıfın, iki veya daha fazla alt sınıfa ayrıldığını gösterir ve karmaşıklığı arttırır. (9,15). Yapılan bir çalışmada, LCOM ölçütünün uyum özelliğini çok da iyi ayırt edemediği ispatlanmıştır (3). Literatürde LCOM2, LCOM3 (13) ve LCOM4 (9,16) adlarıyla yer alan farklı LCOM metrik tanımları da mevcuttur. LCOM metriği test ediciye; verimlilik ve yeniden kullanılabilirlik derecesi hakkında bilgi verir (11).

2.2. Kalite Özellikleri (Quality Attributes)

Yazılım kalitesinin, günlük yaşamı çeşitli bakımlardan etkilemesi sebebiyle önemli olması, bir yazılım ürünü için birçok kalite model yaklaşımının geliştirilmesine sebep olmuştur (17). Bu modeller; ürün kali-

tesini ya da kod kalitesini değerlendirmek üzere geliştirilmiştir.

Örneğin, literatürde McCall (17,18), Boehm (17,19), ISO/IEC 9126 (17,20), Dromey (17,21), Bansiya (17,5) ve NASA'nın Yazılım Güvence Teknoloji Merkezi (The Software Assurance Technology Center's-SATC) (22) tarafından geliştirilen yazılım kalite modelleri mevcuttur. Bu makalede kullanılan deneyin yorumlanmasında, literatürde yaygın bir şekilde kabul gören ve yazılımların kod kalitesini değerlendirme imkânı tanıyan SATC'ın geliştirdiği kalite modelinden yararlanılacaktır.

SATC, kodlama ve tasarım evresi için; verimlilik (efficiency), karmaşıklık (complexity), anlaşılırlık (understandability), yeniden kullanılırlık (reusability) ve testedilebilirlik/dayanıklılık (testability/maintainability) olmak üzere beş adet kalite özelliği önermiştir (22). SATC tarafından önerilen kalite özellikleri kısaca şu şekilde açıklanabilir;

2.2.1. Verimlilik (Efficiency):Yazılımın zaman ve kaynak kullanımı gibi konularda yeterli performansa sahip olma derecesidir (9). Nesne yönelimli tasarım özelliklerini kullanarak gerekli işlevsellik ve davranışı sağlamadaki tasarım yeterliliğidir (14). Yani, "Yapılar etkili bir şekilde tasarlanmış mı?" sorusuna cevap aramaktadır (11).

2.2.2. Karmaşıklık (Complexity): Nesne yönelimli yapıların, yazılım karmaşıklığını oluşturup oluşturmadığını gösterir (11). Karmaşıklık, geliştirilen yazılımların anlaşılırlık düzeyini olumsuz yönde etkilemektedir (23).

Whitmire'e göre karmaşıklık (8,24);

"Nesne yönelimli tasarım sınıflarının bir diğeriyle nasıl ilişkili olduğunun sınanmasıdır. Bir tasarım biriminin elemanları arasındaki ilişkinin derecesini ölçer. Bir nesnenin karmaşıklığı bileşiminin çokluğudur. Buna göre karmaşık bir nesne çok özelliğe sahip olur."

2.2.3. Anlaşılırlık (Understandability): Bir modelin anlaşılması için gerekli olan çaba miktarıdır. Anlaşılırlık, modelin değişikliğinin ve yeniden kullanımının kolaylaşmasını sağlar (23). Kolayca öğrenmeye ve anlamaya izin veren tasarım özelliğidir (14). Tasarımın, fiziksel karmaşıklığı ne yönde etkilediği sınanır (11).

2.2.4. Yeniden kullanılırlık (Reusability): Bir modelin belirli ölçüde diğer modeller tarafından yeniden kullanılabilmesidir (23). Bir problemi yeniden uygulamak için tasarıma izin veren nesne yönelimli tasarım karakteristiklerinin varlığını belirtme anlamındadır (14). Tasarım kalitesinin mümkün olduğu kadar yeniden kullanımı desteklemesi gerekir (11).

2.2.5. Testedilebilirlik / dayanıklılık (Testability / maintainability) : Yazılımın değişiklik veya düzeltme isteklerine uyum kabiliyeti olarak tanımlanmaktadır. Değiştirilebilirlik, test edilebilirlik, analiz edilebilirlik ve bağışıklık konuları ile yakından ilişkilidir (9). Bir programın hatalarını ortaya çıkarma ve özelliklerini toplama faaliyetlerindeki test etme kolaylığıdır (17). Nesne yönelimli yapının, test kolaylığını ve değişiklikleri destekleyip desteklemediği incelenir (11).

3. METRİKLERİN YORUMLANMASI (METRICS INTERPRETATION)

Nesne yönelimli yazılımlar belirli karakteristiklere sahiptir. Örneğin, Bansiya'nın geliştirdiği QMOOD modelinde, nesne yönelimli tasarımlar için soyutlama (abstraction), kapsülleme (encapsulation), bağımlılık (coupling), uyum (cohesion), karmaşıklık (complexity) ve tasarım hacmi (design size) gibi karakteristikler ele alınmaktadır (5).

Rosenberg ise (22,25), nesne yönelimli modeli SATC (NASA)' da; kapsülleme (encapsulation), çokbiçimlilik (polymorphism) ve kalıtım (inheritance) konularını ele almaktadır. Kalıtım; işlem ve operatörlerin sayısını azaltarak karmaşıklığı azaltır fakat, kapsülleme işlemi, bakım ve tasarım zorluğuna sebep olabilir. Çok biçimlilik; karmaşıklığı azaltmada ve yeniden kullanılırlık özelliğini sağlamada programcılara yardımcı olur. Bu yüzden, genellikle nesne yönelimli bir yaklaşım için gerekli olan bu üç temel özellik metriklerin geliştirilme sürecinde ele alınmaktadır. Çizelge 1'de, C&K metriklerinin SATC tarafından önerilen tasarım karakteristikleriyle ilişkisi gösterilmektedir.

Çizelge 1: Tasarım karakteristikleri ve metrikler

Nesne Yönelimli Tasarım Karakteristikleri	Metrik
Kapsülleme	RFC, WMC
Kalıtım	DIT, NOC
Bağımlılık	CBO
Uyum	LCOM

Metriklerin nasıl yorumlanacağı konusu hakkında birçok kural varken, bunları doğrulayacak deneysel çalışmalar yetersizdir. Bu yüzden SATC, bir kodun diğer modüllerinden neden farklı olduğunu kararlaştırmak için çıktılara bakarak elde edilen bu değerlerin karşılaştırılmasına dayanan yorum kuralları önerir. Çizelge 2'de, SATC tarafından önerilen metrik yorumlama kuralları gösterilmektedir (25).

Çizelge 2: Yorumlama kuralları (25)

METRİK	HEDEF
Sınıfın ağırlıklı metot sayısı (WMC)	Düşük
Sınıfın tetiklediği metot sayısı (RFC)	Düşük
Metotlardaki uyum eksikliği (LCOM) Metotların Uyumu (COM)	Düşük Yüksek
Nesne sınıfları arasındaki bağımlılık (CBO)	Düşük
Kalıtım ağacının derinliği (DIT)	Düşük
Alt sınıf sayısı (NOC)	Düşük

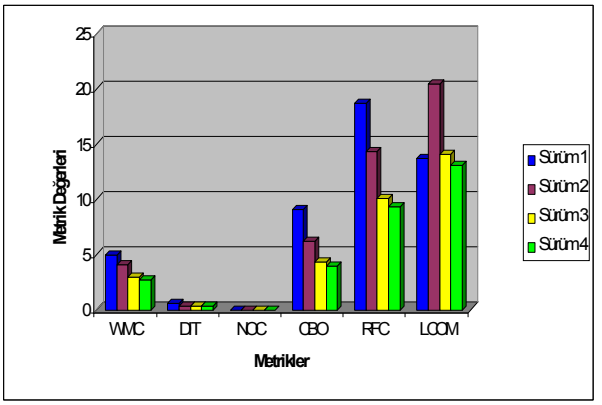
Çizelge 2'den de görülebildiği gibi, sadece "Metotların Uyumu (Cohesion of Methods - COM)" metrik değerinin yüksek, diğer tüm metriklerin değerlerinin ise düşük olması arzu edilen durumdur. Çizelge 3'te, önceki kısımda açıklanan metriklerin hangi kalite özelliğini ölçtüğü ile ilgili özet Çizelge yer almaktadır.

4. DENEYSEL ÇALIŞMA (EXPERIMENTAL STUDY)

Geliştirilen yazılımlarda maliyetlerin azaltılması için, bu yazılımların daha kaliteli üretilmesi veya mevcut yazılımların kalitelerinin arttırılması gerekir. Yazılım kalitesinin arttırılabilmesi, mevcut kalitenin doğru biçimde ölçülebilmesine bağlıdır (9). Bunun için ise iyi planlanmış bir test süreci ve bu sürecin doğru test yazılım araçlarıyla desteklenmesi gerekir (26). Literatürde; Radar, QuickBugs, Bugtrack, ZeroDefect, Roundup ve Abuky (27) gibi ücretli veya ücretsiz birçok geleneksel yazılım test aracı mevcut olmakla birlikte Findbugs (28), Metrics (29), PMD (30) ve Coverity (31) gibi nesne yönelimli yazılım test araçları da geliştirilmiştir.

Aşağıdaki uygulamada C&K metrik değerlerini gösteren "Metrics" (32) adlı ölçüm programı ile MoreUnit projesinin sürümleri değerlendirilmiştir. MoreUnit, bir Eclipse eklentisi olup test durumlarına sahip sınıfları belirler ve test altında bulunan metotları gösterir. MoreUnit'in Java'da geliştirilmiş olması, kalite açısından aralarında karşılaştırma yapılabilecek farklı sürümlerinin bulunması ve nesne tabanlı olması bu programın seçilmesinde etkili olmuştur.

MoreUnit programının her bir sürümü için elde edilen sonuçların aritmetik ortalaması alınmış olup metrik değerleri şu şekildedir.



Grafik 1: Metrik değerlerine göre sürümleri gösteren grafik

len sürümler içerisinde en karışık ve kod kalitesi en düşük olanı Sürüm1, en yüksek olanı ise Sürüm4'tür. Dolayısıyla, geliştirilmesi ve bakım-onarımı için en çok Sürüm1'e, en az ise Sürüm4'e zaman ve iş gücü harcanacaktır. Sürüm1'den Sürüm4'e doğru ağırlıklı metot sayısı (WMC) azalmış olup bu durum yazılım kalitesinin olumlu yönde arttığını gösterir. Bütün bunlara ilaveten, Sürüm1'de anlaşılırlık, dayanıklılık ve yeniden kullanılabilirlik düşük; karmaşıklık ise yüksektir. Buna karşın, Sürüm4'te; bu durum olumlu yönde gelişerek tam tersi bir durum oluşmuştur. Yani; Sürüm 4'te anlaşılırlık, dayanıklılık ve yeniden kullanılabilirlik yüksek, karmaşıklık ise düşüktür. Dolayısıyla, sözkonusu kalite özelliklerinin iyi yönde oluşabilmesi için WMC değerinin her zaman düşük olması tercih edilir.

Çizelge 3: C&K metrik kümesinin SATC tarafından önerilen kalite özellikleri ile ilişkisi

SATC KALİTE ÖZELLİKLERİ	C&K METRİK KÜMESİ					
	WMC	DIT	NOC	CBO	RFC	LCOM
Verimlilik (Efficieny)		✓	✓	✓		✓
Karmaşıklık (Complexity)	✓	✓			✓	✓
Anlaşılırlık (Understandability)	✓	✓			✓	
Yeniden kullanılırlık (Reusability)	✓	✓	✓	✓		✓
Testedilebilirlik (Testability)		✓	✓		✓	
Dayanıklılık (Maintainability)	✓				✓	

Çizelge 4: Sürümlerin metrik değerleri

SÜRÜMLER	METRİKLER					
	WMC	DIT	NOC	CBO	RFC	LCOM
Sürüm1	5	0,547170	0	9,094340	18,64151	13,64151
Sürüm2	4,063918	0,331876	0	6,167888	14,28320	20,45735
Sürüm3	2,960455	0,288186	0	4,378928	10,07153	14,11362
Sürüm4	2,769043	0,281433	0	3,978882	9,335815	13,03326

Çizelge 4'deki bulgular WMC metriği açısından incelendiğinde, Sürüm3 ile Sürüm4 birbirine çok yakın olmakla birlikte Sürüm4'ün **WMC** değeri en düşük, Sürüm1'nin WMC değeri ise en yüksektir. Yani, geliştiri-

Ağaçların derinliği, daha fazla metot ve sınıf içereceği için, tasarımı daha fazla karmaşık hale getirir. Çizelge4'teki bulgular **DIT** metriğine göre incelendiğinde; hiyerarşisi en derin olan Sürüm1'dir. Dolayısıyla, Sürüm1'in davranışlarını tahmin etmek diğer sürümlere göre çok daha zordur. Aynı zamanda kalıtım metotlarının tekrar kullanılma potansiyeli yüksektir. Bu da, bir yazılım ürününde istenmeyen bir durumdur. Sürüm arttıkça bu durumun olumlu yönde bir eğilim gösterdiği görülmektedir. Geliştirilen yazılımlarda DIT metrik değerinin düşük olması yeğlenir. Çizelge4'teki bulgulara bakılarak; Sürüm4'ün; verimlilik, yeniden kullanılabilirlik, anlaşılabilirlik ve test edilebilirlik özellikleri bakımından kendinden önceki sürümlere göre daha iyi; karmaşıklık bakımından ise daha az karmaşık olduğu görülmektedir.

NOC metriğinin bulgularına bakacak olursak, tüm sürümlerde bu değer 0 (sıfır)'dır. Dolayısıyla, herhangi bir alt sınıfa sahip sürüm olmadığı için bu metrik itibariyle hepsi aynı seviyededir. Yani, herhangi bir karmaşıklık veya bakım-onarım-test durumundan bahsetmek mümkün değildir.

CBO metriğinin bulgularına göre, Sürüm3 ve Sürüm4 birbirine çok yakın ve daha az bağımlı olmakla birlikte, Sürüm1, en bağımlı sürümdür. Bu da, bu sürümün modüler tasarımı iyi olmadığı ve tekrar kullanılma ihtimalinin az olduğu anlamına gelir. Dolayısıyla, Sürüm1'in bakımı diğerlerinden daha zordur ve test maliyeti daha fazladır. Yine, değerlere göre en az bağımlı olan Sürüm4; hem en kaliteli hem de bakım-onarımı kolay ve test maliyeti en düşüktür. Aynı zamanda, bu sürümün verimlilik ve yeniden kullanılabilirlik düzeyi yüksektir.

RFC değerlerine göre; sınıfın testi ve hata ayıklaması en karışık olan yine Sürüm1'dir. Test ve hata ayıklama işlemi bakımından en kolay olan dolayısıyla da bakım-onarım-test için en az zaman harcanacak olan sürüm, son sürümdür (Sürüm4). RFC değerinin düşük olması istenmekle birlikte, çizelgeye göre Sürüm4; en anlaşılabilir, dayanıklı ve test edilebilir durumdadır.

LCOM değerlerine bakılırsa; diğer metrik değerlerine göre ciddi farklılıkların bulunması, bu ölçütün yazılımlardaki uyumu ölçme konusunda çok yeterli olmadığını desteklemektedir. Çizelge 4'te belirtilen metrik değerlerinde olumsuz olarak değerlendirilen ilk sürüm (Sürüm1), son sürüm ile yaklaşık aynı değere sahiptir. Yani; uyumluluk düzeyi, yeniden kullanılabilirlik derecesi ve verimliliği yüksek olmakla birlikte sözkonusu sürümlerin karmaşıklığı daha azdır. Sürüm2 ise, uyumluluk bakımından en düşük olanıdır. Yani, geliştirme süreci esnasında hata olma ihtimali en yüksek ve karmaşık olan sürümdür. LCOM değerinin, geliştirilen yazılımlar için düşük olması beklenir.

Sonuç olarak; MoreUnit programının sürümlerinden elde edilen tüm metrik değerlerine bakıldığında sürümler arttıkça kalitenin de arttığını söylemek mümkündür.

5. SONUÇ (CONCLUSION)

Yazılım testleri, geliştirilen projenin/yazılımın hatalardan arındırılması, güvenirliliğinin sağlanması ve kalitesinin arttırılması amacıyla uygulanır. Bu faaliyetler belirli metrikler kullanılarak gerçekleştirilir. Bu metrikler, yazılımların kalitesinin tespit edilip geliştirilmesini sağlar. Dolayısıyla, metrikler aracılığıyla test edilen yazılımın ne kadar bağımlı, karmaşık, etkili ve kaliteli olduğu anlaşılır. Ancak, tüm bu çalışmaların başarılı bir şekilde sonuçlanması için büyük ölçüde doğru metriklerin tanımlanması, bu metriklerin doğru biçimde ölçülmesi ve doğru yorumlanması gerekir.

Bu çalışma kapsamında nesne yönelimli yazılım metrikleri ve kalite özellikleri kısaca açıklanmış olup söz konusu konulara uygun bir projenin sürümleri üzerinde deneysel bir çalışma ile yazılım kalitesi açıklanmaya çalışılmıştır. Bu projenin C&K metrik değerlerinin, SATC tarafından önerilen verimlilik, karmaşıklık, anlaşılırlık, yeniden kullanılırlık ve testedilebilirlik /dayanıklılık gibi yazılım kalite özellikleri ile ilişkisi ortaya konmuştur. Sonuç olarak, bu yorumlar göz önüne alındığında, deney için kullanılan projenin sürümleri arttıkça kalitenin de arttığı söylenebilir. Yani, başka bir ifadeyle, bu projenin metrik değerleri; verimlilik, anlaşılırlık, yeniden kullanılırlık ve test edilebilirlik /dayanıklılık derecelerinin yüksek, karmaşıklık derecesinin de düşük olduğunu göstermektedir.

6. KAYNAKLAR (REFERENCES)

1) İnternet: Vikipedi Özgür Ansikopedi, "Sınıf (programlama)", http://tr.wikipedia.org/wiki/S%C4%B1n%C4%B1f_(programlama)#Bilgi_gizleme_ve_kaps.C3.BClleme" (2011).
2) Baldassari, B., Robach, C. and du Bosquet, L., "*Early Metrics for Object Oriented Designs*", IEEE Testability Assessment, IWoTA 2004. Proceedings. First International Workshop, pp.62-69, (2004).
3) Ertemel, H.Ö., "Nesneye dayalı yazılım geliştirmede kalite ölçütlerinin incelenmesi", Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, İstanbul, (2009).
4) Kitchenham, B. and Pfleeger, S.L., "*Software Quality: the Elusive Target*", IEEE Software, vol. 13, no. 1, pp.12-21, January 1996.
5) Bansiya, J. and Davis, C. G., "*A Hierarchical Model for Object-Oriented Design Quality Assessment*", IEEE Transactions on Software Engineering, vol. 28, no. 1, pp.4-17, (2002).
6) Goodman, P., "The Practical Implementation of Software Metrics", McGraw-Hill, New York, USA, (1993).
7) Schulmeyer, G. G. and McManus I., J., "The Handbook of Software Quality Assurance, 3rd Edition", Prentice Hall PTR, USA, (1998)
8) Pressman, R. S., "Software Engineering: A Practitioner's Approach", 6th Ed., Mc Graw Hill, Singapore, (2005).
9) U. Erdemir, U. Tekin, F. Buzluca, "*Nesne Yönelimli Yazılım Metrikleri ve Yazılım Kalitesi*", 10/2008, s. 249-258, Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (YKGS08), İstanbul, (2008).

10) Gustafson, D. A., "Theory and Problems of Software Engineering", Mc Graw Hill, USA, (2002).

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

12) İnternet: McCabe Software "*Using Code Quality Metrics in Management of Outsourced Development and Maintenance*", http://www.mccabe.com/pdf/McCabeCodeQualityMetrics-OutsourcedDev.pdf (2009).

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

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

15) Chidamber, S.R, and Kemerer, C.F., "*Towards A Metric Suite For Object-Oriented Design*", Proceedings : OOPSLA '91, Phoenix, AZ, pp.197-211, (1991).

16) Hitz, M., and Montazeri, B., "*Chidamber and Kemerer's Metric Suite: A Measurement Theory Perspective*", IEEE Transactions on Software Engineering, Vol. 4, pp.267-271, (1996).

17) Jetter, A., "Assessing Software Quality Attributes With Source Code Metrics", Diploma Thesis, University of Zurich Department of Informatics, Zurih, October (2006).

18) 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).

19) 19.Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M., "Characteristics of software quality", North Holland, (1978).

20) İnternet: ISO "ISO/İEC 9126", http://www.iso.org/ (2006).

21) Dromey, R. G., "*A Model For Software Product Quality*", Software Engineering, IEEE Transactions on, 21(2):146-162 (1995).

22) Mustafa, K. ve Khan, R.A., "Software Testing: Concept And Practices", İndia, Lucknow, (2007).

23) Amstel, M.,v., Brand, M., v.,d., and Lange, C., "*Metrics For Analyzing The Quality Of Model Transformations*", In Falcone, G., Gu´eh´eneuc, Y., Lange, C., Porkol´ab, Z., Sahraoui, H., eds.: Proceedings of the 12th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Paphos, Cyprus, pp.41–51, (2008)

24) Whitmire, S., "Object Oriented Design Measurement", John Wiley and Sons Ltd, Newyork, pp. 413 (1997).

25) Rosenberg, L.,H., "Applying And Interpreting Object Oriented Metrics", Proc. Software Technology Conference. Utah, (1998)

26) Gürbüz, A., "Yazılım Test Mühendisliği", Papatya Yayıncılık Eğitim, İstanbul, (2010).

27) İnternet: Testingfaqs.org "Defect Tracking Tools" http://www.testingfaqs.org/t-track.html (2010)

28) İnternet: SourceForge.net, "FindBugs™-Find Bugs in Java

29) Programs" http://findbugs.sourceforge.net/index.html

30) (2008).

31) İnternet: SourceForge.net, "Metrics 1.3.6" http://metrics.sourceforge.net/ (2009).

32) İnternet: SourceForge.net, "PMD" http://pmd.sourceforge.net/ (2002).

33) İnternet: Coverity, "Coverity Analysis" http://www.coverity.com/ (2009).

34) İnternet: Yıldız Teknik Üniversitesi, "Metrics - Ölçüm programı ve kullanımı" http://www.ce.yildiz.edu.tr/myindex.php?id=63 (2010).