Nesne Tabanlı Yazılımların Yapısal Özelliklerinin Hata Yatkınlığı Üzerine Etkilerinin İncelenmesi

Halit Gölcük

Gömülü ve Gerçek Zamanlı Yazılım Tasarım Müdürlüğü, SST Sektör Bşk.
ASELSAN A.Ş.

hgolcuk@aselsan.com.tr

Özet. Modelleme dillerinin Yazılım Mühendisliği alanında kullanımının artmasıyla sadece yazılım kaynak kodunun değil bir yazılım modelinin de kalitesini ölçmek ve kontrol etmek önemli hale gelmiştir. Ancak bir yazılım ürününün kalitesini ölçebilmek için öncelikle o yazılım ürününden kalite anlamında ne beklendiği ortaya konulmalıdır. Bu çalışma, Birleşik Modelleme Dili (UML) kullanılarak geliştirilen nesne tabanlı yazılımların model seviyesinde gözlenebilen bazı yapısal özelliklerinin yazılım kalitesi üzerine etkilerini incelemektedir. Yazılım kalitesi yazılımın hata yatkınlığı olarak tanımlanmıştır. Çalışma kapsamında Aselsan bünyesinde geliştirilen gömülü ve gerçek zamanlı yazılım bileşenleri analiz edilmiştir. Yazılım bileşenlerinin yapısal özelliklerini yansıtan ve UML modelleri üzerinden ölçülen yazılım tasarım metrikleri ve bu bileşenlere ait hata yatkınlık metrikleri arasındaki ilişki ortaya konulmuştur. Yazılım tasarım metrikleri ve hata yatkınlık metrikleri arasındaki ilişki istatistiksel olarak doğrulanmıştır.

Anahtar Kelimeler: Yazılımın yapısal özellikleri, UML metrikleri, Yazılım kalitesi, Hata yatkınlığı, Deneysel çalışma.

Investigation of the Effects of Structural Characteristics of Object-Oriented Software on Fault-Proneness

Abstract. With the increasing use of modeling languages in Software Engineering, it has become important to measure and control the quality of a software model, not just the software source code. However, in order to measure the quality of a software product, it is first necessary to determine the expectations from the software product in terms of quality. This study examines the effect of some structural features observed at the model level on software quality of object-oriented software developed using the Unified Modeling Language (UML). Software quality is defined as fault-proneness. Embedded and real-time software components developed in Aselsan are analyzed. The correlation between software design metrics that reflect structural characteristics of software com-

ponents and measured through UML models and fault-proneness metrics of these components are presented. The correlation between software design metrics and fault-proneness metrics is statistically verified.

Keywords: Structural software characteristics, UML metrics, Software quality, Fault-proneness, Empirical study.

1 Giriş

Bir yazılımın kalitesini ölçebilmek için o yazılımdan kalite anlamında ne beklendiği ortaya konulmalıdır. Hata yatkınlığı bir yazılımın kalitesine yönelik önemli bir göstergedir. Literatürde yazılımın hata dayanıklılığını ölçmek üzere, hata sayısı ve hata yoğunluğu gibi, farklı metrikler kullanılmıştır. Ayrıca [1] gibi yazılıma ait bir takım tasarım metrikleri ile yazılımın hataya yatkınlığını deneysel olarak ilişkilendirmeye çalışan çalışmalar mevcuttur. Chidamber ve Kemerer [2] tarafından ortaya konulan nesne tabanlı yazılımlara ait tasarım metrikleri hatalı yazılım bileşenlerini tespit etmek için sıklıkla kullanılmıştır.

Modelleme dillerinin yazılım mühendisliği alanında kullanımının artmasıyla bir yazılım modelinin kalitesini ölçmek ve kontrol etmek daha önemli hale gelmiştir. Nesne tabanlı yazılımların geliştirilmesi amacıyla en sık kullanılan modelleme aracı Birleşik Modelleme Dili (UML)'dir. UML tasarım metriklerinin yazılımın hataya yatkınlığını ölçmek ve kontrol etmek amacıyla kullanılmasına dair oldukça az çalışma bulunmaktadır [3].

Bu çalışma kapsamında, UML kullanılarak geliştirilen nesne tabanlı yazılımlara ait tasarım metrikleri ile yazılımın hataya yatkınlığı arasındaki ilişki incelenmiştir. Araştırma konusu olan yazılım bileşenleri Aselsan Savunma Sistem Teknolojileri Sektör Başkanlığı Gömülü ve Gerçek Zamanlı Yazılım Tasarım Müdürlüğü bünyesinde geliştirilen atış kontrol yazılımlarına aittir. Bütün metrikler incelenen yazılım bileşenlerinin UML modelleri üzerinden ölçülmüştür. Hata yatkınlığı metrikleri Aselsan'da kullanılan hata izleme aracından elde edilmiştir.

Bildirinin geri kalanı şu şekilde düzenlenmiştir: Bölüm 2'de yazılım kalitesi ve kaliteyi etkileyen faktörler ile ilgili literatür bilgileri özetlenmiştir. Bölüm 3'te yapılan çalışmanın yöntemi hakkında bilgi verilmiştir. Bölüm 4'te çalışma kapsamında elde edilen bulgular ortaya konulmuştur. Bölüm 5'te ise değerlendirme ve sonuçlara yer verilmiştir.

2 Literatür

2.1 Yazılım Kalite Ölçümü

Farklı özelliklerdeki yazılım ürünleri için farklı kalite hedefleri öne çıkmaktadır. Örneğin, bir istatiksel analiz sistemi için en önemli fonksiyonel olmayan gereksinim güvenilirlik iken bir banka sistemi için güvenlik daha ön plandadır [4]. Bu sebeple, bir

yazılım ürünü için doğru kalite hedefleri koyabilmek adına kalite gereksinimi çalışması yapılmalıdır.

Yüksek kalitede yazılım ürünleri üretebilmek için kalite gereksinimlerini karşılayan doğru kalite modelleri kullanmak çok önemlidir. Kullanılacak kalite modelini seçmede iki yöntem izlenebilir. Bunlardan biri ISO/IEC 9126 gibi genel kalite modelini kullanmaktır. Ancak, bu tür kalite modelleri soyut kalmakta ve uyarlamada sıkıntılar yaşanmaktadır. Diğer bir yöntem ise mevcut kalite modellerinden yararlanarak kendi kalite modelini belirlemektir. İkinci yöntem özelleşmiş kalite gereksinimlerini karşılayan doğru kalite modelini bulabilmek için daha uygundur [5,6].

Gömülü ve gerçek zamanlı yazılımlar yüksek güvenlik, güvenilirlik ve performans gereksinimleri ile birlikte geniş bir uygulama alanına sahiptir. Bu yüzden yazılım kalitesi gömülü ve gerçek zamanlı yazılım içeren sistemlerde oldukça önemlidir [7]. Gerçek zamanlı bir yazılım ürünü için hata dayanıklılığı bir kalite göstergesi olarak kullanılabilir.

2.2 Yazılım Kalite Metrikleri

Literatürde yazılım kaynak kodundan kalite ölçümü yapabilmek için önerilen birçok metrik bulunmaktadır. Chidamber ve Kemerer [2] nesne tabanlı yazılımların kalitesini değerlendirmek için metrikler önermişlerdir. Bu değerli çalışmada yazılım tasarım kalitesinin altı adet kalite metriği kullanılarak ölçülebileceği belirtilmiştir. Bir diğer önemli metrik seti ise MOOD metrikleri [8] olarak bilinmektedir. Bu metrikler kalıtım (inheritance), bilgi saklama (information hiding), çok biçimlilik (polymorphism) gibi nesne tabanlı tasarım yöntemlerini ölçmek için kullanılır. Bir başka çalışmada [9] Lorenz ve Kidd nesne tabanlı bir yazılımın durağan niteliklerini ölçebilmek için yazılım metrikleri ortaya koymuşlardır.

Model tabanlı yazılım geliştirmenin yaygınlaşması ile birlikte sadece yazılım kaynak kodunun değil aynı zamanda modellerin de kalitesinin ölçülmesi önemli hale gelmiştir. Bugün birçok kuruluş UML modellerini tasarım veya kaynak kod üretme gibi amaçlarla kullanmaktadır. Nugroho ve arkadaşları [3] UML metriklerinin hata yatkınlığı yüksek sınıfı bulmada önemli olduğunu deneysel olarak göstermişlerdir.

Aşağıdaki konu başlıklarında mevcut çalışma kapsamında kullanılacak yazılım metrikleri gözden geçirilmiştir.

CK Metrikleri [2]. Bu metrik grubunun nesne tabanlı yazılımlar için hatalı sınıfı bulmada önemli olduğu tespit edilmiştir [10] ve literatürde sıklıkla yazılım kalitesi ölçme amacıyla kullanılmaktadır [11]. Bu metrikler özgün olarak nesne tabanlı yazılımların kalitesini kaynak koddan ölçmek için önerilmiş olmalarına rağmen literatürde UML modellerine uygulanabilirliğini gösteren çalışmalar mevcuttur [12].

Metrik isimleri ve tanımları şu şekilde verilmiştir:

- Weighted Methods per Class (WMC): Sınıf içerisinde tanımlanmış metotların karmaşıklık değerinin toplamıdır.
- Depth of Inheritance Tree (DIT): Kalıtım hiyerarşisindeki kalıtım derinliğini ölçer.

- Number of Children (NOC): Herhangi bir temel sınıftan direkt olarak türetilmiş sınıf sayısını verir.
- Coupling between Objects (CBO): Sınıflar arasındaki bağımlılığın ölçümüdür.
- Response for a Class (RFC): Bir sınıfa gönderilen mesaja karşılık o sınıftan çağırılan metot sayısıdır.
- Lack of Cohesion in Methods (LCOM): Bir sınıf içindeki metotların bağımlılığına dair bir ölçümdür.

Sınıf Diyagram Metrikleri [13]. Genero ve arkadaşları [13] bu metrikleri UML sınıf diyagramları için deneysel olarak doğrulamışlardır.

- Number of Associations (NAssoc): Sınıf diyagramı içindeki ilişki sayısıdır.
- Number of Aggregation (NAgg): Sınıf diyagramı içindeki içerim sayısıdır.
- Number of Dependencies (NDep): Sınıf diyagramı içindeki bağımlılık sayısıdır.
- Number of Generalizations (NGen): Sınıf diyagramı içindeki genelleme sayısıdır.
- Number of Aggregations Hierarchies (NAggH): Sınıf diyagramı içindeki içerim hiyerarşisi sayısıdır.
- Number of Generalizations Hierarchies (NGenH): Sınıf diyagramı içindeki genelleme hiyerarşisi sayısıdır.
- Maximum DIT (MaxDIT): Sınıf diyagramı içindeki kalıtım hiyerarşisinin maximum derinliğidir.
- Maximum HAgg (MaxHAgg): Sınıf diyagramı içindeki içerim hiyerarşisinin maximum derinliğidir.
- Number of Classes (NC): Sınıf diyagramı içindeki sınıf sayısıdır.
- Number of Attributes (NA): Sınıf diyagramı içindeki nitelik sayısıdır.
- Number of Methods (NM): Sınıf diyagramı içindeki metot sayısıdır.

Durum Grafiği (Statechart) Metrikleri [14]. Durum grafikleri bir sınıfın dinamik davranışını temsil eder. Bu nedenle bir durum grafiğinin karmaşıklığı sınıfın karmaşıklığı ile yakından ilgilidir. Böylece, durum grafiği karmaşıklığı bir sınıfın hata yatkınlığını etkilemektedir [15].

- Number of Entry Actions (NEntryA): Durum grafiğindeki giriş eylemi sayısıdır.
- Number of Exit Actions (NExitA): Durum grafiğindeki çıkış eylemi sayısıdır.
- Number of Activities (NAc): Durum grafiğindeki aktivite sayısıdır.
- Number of Simple States (NSS): Durum grafiğindeki basit durum sayısıdır.
- Number of Composite States (NCS): Durum grafiğindeki karma durum sayısıdır.
- Number of Guards (NG): Durum grafiğindeki koruma koşulu sayısıdır.
- Number of Events (NE): Durum grafiğindeki olay sayısıdır.
- Number of Transitions (NT): Durum grafiğindeki geçiş saysıdır.
- Cyclomatic Complexity (CC): McCabe [16] tarafından önerilen karmaşıklık birimi |NT-NS+2| olarak uyarlanmıştır.

2.3 Hata Yatkınlığı Ölçüm Yöntemleri

Oyetoyan ve arkadaşları [17] farklı hata ölçümlerinin hataya yatkın yazılım bileşenlerini belirlemedeki rolünü tespit etmek için bir çalışma ortaya koymuşlardır. Bu çalışmada hata sayısı, hata yoğunluğu, hata önem derecesi ve hata düzeltme çabası olmak üzere dört farklı hata ölçümü karşılaştırılmıştır. Hata sayısı, bir yazılım bileşeninde tespit edilen toplam hata sayısıdır. Hata yoğunluğu, hata sayısının kod satır sayısına bölünmesi ile elde edilir. Hata önem derecesi, tespit edilen hatanın kritiklik seviyesidir. Hata düzeltme çabası ise yazılım bileşeninde tespit edilen hatanın düzeltilmesi için harcanan işçiliği ifade eder.

3 Yöntem

Aselsan Savunma Sistem Teknolojileri Sektör Başkanlığı Gömülü ve Gerçek Zamanlı Yazılım Tasarım Müdürlüğü (GGZYTM) bünyesinde geliştirilen yazılım bileşenleri organizasyonel olarak kabul edilmiş ve yayınlanmış yazılım referans mimarisine [18] göre geliştirilmektedir. Bu şekilde yazılım bileşenlerinin tasarım olgunluğu belli bir ölçüde garanti edilmektedir. Bununla birlikte, referans mimaride tanımlanan arayüzler dışında, bileşenler için sınırlı tasarım bilgisi bulunması nedeniyle yazılım bileşenlerinin kalitesinin geliştiriciye bağlı olduğu gözlenmiştir. Bu nedenle referans mimariyle uyumlu olarak geliştirilen yazılım bileşenleri için yapısal özelliklerin belirtilmesi yararlı olacaktır. Bu çalışmanın amacı, geliştirilecek olan yazılım bileşenlerinin kalite düzeyini artırmak ve hâlihazırda geliştirilmiş olan yazılım bileşenlerinin kalite düzeyini ortaya koymaktır. GGZYTM tarafından yazılım bileşenleri bir UML aracı kullanılarak geliştirildiği için yazılım metriklerini UML modellerinden toplamak önemlidir. Bu çalışmanın sonuçları daha yüksek kalitede yazılım bileşenleri geliştirmek için yazılım geliştiricilerine rehberlik etmek amacıyla kullanılacaktır.

Bu çalışmada, bir yazılım bileşeninin hata yatkınlığı kalitesinin önemli bir göstergesi olarak kabul edilmiştir. GGZYTM tarafından geliştirilen projelere ait bazı yazılım bileşenleri seçilmiş ve bu bileşenlere ait tasarım metrikleri toplanmıştır. UML kullanılarak tasarlanan gömülü ve gerçek zamanlı yazılım bileşenlerinin hatalarının değerlendirilmesinde hangi tasarım metriğinin önemli olabileceğini bulmak için yazılım bileşenlerinin metrikleri ve hataları analiz edilmiştir.

3.1 Yazılım Ekibi Tarafından Kullanılan Referans Yazılım Mimarisi

Bu çalışma boyunca GGZYTM bünyesinde faaliyet gösteren bir yazılım geliştirme ekibiyle çalışılmıştır. Bu ekip, otomatik kod üretme yeteneği bulunan bir UML aracı ile C++ kullanarak atış kontrol sistemleri için yazılım geliştirmektedir. Atış kontrol sistemleri birçok algılayıcıdan gelen verileri işleyerek, en doğru zamanda en doğru atışı yapabilmek için gerekli işlemleri yapan sistemlerdir. Yazılım geliştirme çalışmaları sırasında Silah Sistemleri Referans Mimarisi (SSRM) [18] olarak adlandırılan bir referans mimari kullanılmaktadır.

Referans mimarinin temel bileşenleri kısa açıklamalarıyla aşağıda verilmiştir.

- Görevler proje gereksinimlerine göre farklılık gösteren senaryo işleticileridir.
- Yetenekler belirli bir görevi yerine getirmeyi amaçlar, örneğin; hedef yönetimi, platform yönetimi. Yetenekler yeniden kullanılabilir bileşenler olarak geliştirilmiştir.
- Yazılım Yöneticisi yazılımın durumuna göre hangi yazılım bileşeninin aktif olacağına karar verir. Referans mimarinin bu bileşeni projeye özeldir.
- Dış Arayüz sistemde bulunan kullanıcı arayüzünü, komuta kontrol arayüzünü, vs. temsil eder. Yazılımda böyle bir bileşen tanımlanmasının amacı dış ortamdaki değişkenliği yazılımdan ayırabilmektir.
- Sistem Çevresi sistemde bulunan algılayıcı ve eğleyicilerin servislerini yazılıma aktarmaktan sorumludur.
- İşletim Çevresi yazılımın donanım ve işletim sisteminden bağımsızlığını sağlar.

3.2 Araştırma Konusu Yazılım Bileşenleri

Yazılım ekibi içerisinde en fazla bileşen Sistem Çevresi Katmanı için geliştirilmiş ve hata sayısının en fazla bu katmanda olduğu tespit edilmiştir. Bu nedenle çalışma kapsamında incelenen yazılım bileşenlerinin hepsi referans mimarinin Sistem Çevresi Katmanından seçilmiştir. Yazılım bileşenleri, tamamlanmış ve müşteriye teslim edilmiş en az iki projede kullanılan bileşenler arasından seçilmiştir. Böylece araştırma konusu olacak yazılım bileşenlerinin yeterince test edilmiş bileşenler olması amaçlanmıştır. Sistem Çevresi Katmanında birçok yazılım bileşeni olmasına karşın bunlardan ancak 10 tanesi belirlenen ölçüte uymuştur.

3.3 Seçilen Yazılım Tasarım Metrikleri

Bu çalışmada kullanılacak yazılım metriklerini seçerken GGZYTM'nin gereksinimleri ve öncelikleri dikkate alınmıştır. Hangi metrik veya metrik setinin kullanılacağını belirlemek amacıyla literatür taranmış ve geniş bir metrik seti ölçüm alınacak projelerin yazılım geliştiricileri ile birlikte değerlendirilmiştir.

Metrik seçimi gerekçeleriyle birlikte Tablo 1'de verilmiştir. "Metrik Seçimi" sütunu ilgili metriğin ölçülüp ölçülmeyeceğini belirtir, "+" işareti metriğin ölçüleceği, "-" işareti metriğin ölçülmeyeceği anlamına gelir.

Tablo 1. Metrik Seçimi

Metrik Seti	Metrik Seçimi	Metrik İsmi	Yorum
CK Metrikleri	+	WMC	Bu metrik seti bir sınıfın hataya yatkınlığını belirlemede iyi bir gösterge olarak kabul edilmiştir.
	+	DIT	
	+	NOC	
	+	CBO	
	+	RFC	
	+	LCOM	
Sınıf Diyagramı	-	NAssoc	GGZYTM bünyesinde geliştirilen yazı-

Metrikleri			lımlarda sınıflar arası ilişkiler UML portları üzerinden kurulmaktadır.
	-	NAgg	GGZYTM bünyesinde geliştirilen yazılımlarda içerim ilişkisi kurulmamaktadır.
	-	NAggH	
	-	MaxHAgg	
	-	NGen	GGZYTM bünyesinde geliştirilen yazılımlarda uygulama kalıtımından ziyade arayüz kalıtımı kullanılmaktadır.
	-	NGenH	
	-	MaxDIT	CK Metrik setinde DIT metriği bulunduğu için bu metriğin ölçülmesinin gerekli olmadığı değerlendirilmiştir.
	-	NDep	GGZYTM bünyesinde geliştirilen yazılımlar sadece referans mimariye bağımlıdır.
	+	NC	Bir yazılım bileşeninin boyutu ile hata yatkınlığı arasında bir ilişki olması beklenmektedir.
	+	NA	
	+	NM	
Durum Grafiği Metrikleri	+	NEntryA	Bir sınıfın durum grafiğinin yapısal karmaşıklığı ile hata yatkınlığı arasında bir ilişki olması beklenmektedir.
	+	NExitA	
	+	NAc	
	+	NSS	
	+	NCS	
	+	NG	
	+	NE	
	+	NT	
	+	CC	

Çalışma kapsamında ölçülecek metrikler seçildikten sonra bu metriklerin ölçüm yöntemleri değerlendirilmiştir. Metrik ölçümlerinin hazır bir araç ile yapılıp yapılamayacağını görmek için literatür gözden geçirilmiş ve bu amaçla kullanılabilecek SDMetrics [19] ön plana çıkmıştır. SDMetrics, UML modellerini XMI formatında dışa aktarım yeteneği bulunan UML araçları ile birlikte kullanılabilmektedir.

Yazılım geliştirme ekibi tarafından kullanılan UML aracına java eklentileri ile yeni özellikler katılabilmektedir. Bu sayede çalışma ihtiyaçlarını göz önünde bulundurarak metrik ölçümünün otomatikleştirilebileceği değerlendirilmiştir.

Bu bilgiler ışığında yazılım tasarım metriklerinin java eklentileri geliştirerek ölçülebileceği, bazı metrik ölçümleri için ise SDMetrics aracının kullanılabileceğine karar verilmiştir.

3.4 Yazılım Bileşenlerine ait Hata Verileri

Yazılım bileşenlerine ait hata verilerinin elde edilmesinde Aselsan bünyesinde kullanılan hata takip aracının verileri kullanılmıştır. Bununla birlikte, hata kayıtlarının birçoğu sadece ilgili yazılım bileşeninin içinde yer aldığı projeyi adreslemekteydi. Bu

noktada, ilgili yazılım bileşenine ait hata kayıtlarının ortaya çıkarılmasında projede çalışan yazılım geliştiricilerden destek alınmıştır.

4 Bulgular

4.1 Metrik Ölçüm Sonuçları

Metrik ölçümleri, çoğunlukla yazılım geliştirme ekibi tarafından kullanılan UML aracı için geliştirilen java eklentisi kullanılarak elde edilmiştir. Bazı metrik ölçümleri içinse hazır araçlar kullanılmıştır.

LCOM metriğini UML modelinden ölçen bir araç bulanamamıştır. Ayrıca java eklentisi ile de metrik ölçümü elde edilememiştir. Bu nedenle, LCOM metriği değerlendirmeden çıkarılmıştır.

Seçilen metrikler, Sınıf Diyagramı Metrikleri dışında, sınıf seviyesinde ölçülen metriklerdir. Bu nedenle, birden fazla sınıf içeren bileşenlerin metrik değerlerinin bir araya getirilmesi gerekmiştir. Bunun için [20]'de verilen yöntemden faydalanılmıştır. Yönteme göre, bileşen seviyesinde WMC, NOC ve RFC metrik değerleri toplanmalı, DIT için en yüksek değer kullanılmalı, CBO içinse bileşen içindeki sınıflara bağımlı harici sınıfların sayısı dikkate alınmalıdır. Durum grafiği metrikleri boyut ve karmaşıklık metrikleri olarak kabul edildiği için bireysel sınıfların metrik değerleri toplanmalıdır. Bunun sonucunda metrik ölçüm sonuçları Tablo 2'deki haliyle oluşmuştur.

Tablo 2. Metrik Ölçüm Sonuçları

	Bileşen_1	**Bileşen_2**	**Bileşen_3**	**Bileşen_4**	**Bileşen_5**	**Bileşen_6**	**Bileşen_7**	**Bileşen_8**	**Bileşen_9**	**Bileşen_10**
WMC	143	236	238	201	251	269	233	64	117	260
DIT	0	0	0	1	1	0	0	0	0	0
NOC	0	0	0	0	0	0	0	0	0	0
CBO	12	12	15	12	12	17	16	10	10	16
RFC	29	30	66	37	47	67	27	20	26	26
NC	1	1	5	3	3	5	1	4	1	1
NA	26	41	117	48	46	79	21	22	12	34
NM	35	26	61	68	50	76	23	25	13	20
NEntryA	16	24	88	38	35	22	4	5	6	5
NExitA	3	1	11	5	2	4	0	2	0	0
NAc	29	15	93	27	27	41	3	3	7	5
NSS	16	21	88	41	37	24	10	4	6	4
NCS	10	14	34	36	33	11	7	3	4	4
NG	11	8	84	29	31	13	1	2	1	0
NE	26	31	105	57	55	26	14	4	5	3

NT	37	48	193	98	91	47	21	7	8	6
CC	13	15	75	25	25	18	6	2	0	0

4.2 Hata Yatkınlık Ölçüm Sonuçları

Yazılım bileşenleri ile ilgili hatalar ölçümlerin alındığı projelerde çalışan yazılım mühendisleri yardımıyla elde edilmiştir. Bu kapsamda yaklaşık 2500 hata kaydı incelenmiş, bunlardan 184'ünün araştırma konusu yazılım bileşenlerine ait olduğu tespit edilmiştir. Her bir yazılım bileşeni için hata yatkınlık ölçüm sonuçları Tablo 3'te verilmiştir.

Tablo 3. Hata Yatkınlık Metrikleri Ölçüm Sonuçları

	Bileşen_1	**Bileşen_2**	**Bileşen_3**	**Bileşen_4**	**Bileşen_5**	**Bileşen_6**	**Bileşen_7**	**Bileşen_8**	**Bileşen_9**	**Bileşen_10**
Hata Sayısı	5	24	64	26	11	17	15	0	2	20
Hata Yoğunluğu	0,794	3,701	2,678	1,819	0,766	1,502	3,95	0	0,53	5,249
Hata Önem Derecesi	14	76	232	79	34	58	55	0	7	58
Hata Düzeltme Çabası	7	88	206	63	17	33	22	0	1	45

4.3 Yazılım Bileşenlerinin Yazılım Metrikleri ve Hata Yatkınlığı Arasındaki İlişki

Hesaplanan yazılım metrik değerlerini hata yatkınlığı ile ilişkilendirmeden önce Tablo 2'ye baktığımızda DIT ve NOC değerleri için anlamlı sonuçlar alınamadığını görüyoruz. Bunun nedeni GGZYTM bünyesinde geliştirilen yazılım bileşenlerinde uygulama kalıtımından ziyade arayüz kalıtımı kullanılmasıdır. Bu nedenle, DIT ve NOC metrikleri kapsam dışında bırakılarak analizden çıkarılmıştır. Ayrıca DIT ve LCOM metriklerinin nesne tabanlı yazılımlar için kalite metriği olarak kullanılmasını eleştiren ve NOC metriğinin hata yatkınlığını tespit etmek için kullanılmaması gerektiğini ifade eden çalışmalar vardır [21].

Yazılım metrikleri ve hata yatkınlığı arasındaki ilişkiyi doğrulamak için ilk olarak Pearson ilişki katsayısı kullanılmıştır. Pearson ilişki katsayısının kullanılabilmesi için veriler içerisinde aykırı değer olmamalı ve veriler normal dağılım göstermelidir. Bu nedenle ilk önce veriler üzerinde aykırılık ve normallik testleri uygulanmıştır.

Tablo 4. Aykırılık ve Normallik Testi Sonuçları

	Aykırı Veri	Normallik
Hata Sayısı	Bileşen_3	Normal
Hata Yoğunluğu	-	Normal
Hata Önem Derecesi	Bileşen_3	Normal Değil
Hata Düzeltme Çabası	Bileşen_3	Normal

Tablo 4'ten de görülebileceği gibi hata sayısı, hata önem derecesi ve hata düzeltme çabası verileri aykırı veri ve/veya normallik testlerinden geçememiş, sadece hata yoğunluğu verisi bu testlerden geçmiştir. Bu nedenle, istatistiksel anlamda daha anlamlı sonuçlar elde edebilmek için analize Spearman ilişki katsayısı ile devam edilmiştir.

Spearman ilişki katsayısı parametrik olmayan bir testtir ve dağılım normal olmadığında iki sürekli değişken arasındaki ilişkinin belirlenmesinde kullanılır. 0 ve +1 arasındaki değerler pozitif ilişkiyi, 0 ve -1 arasındaki değerler ise negatif ilişkiyi temsil eder. Bu test için sonuçlar Tablo 5'da verilmiştir.

Tablo 5. Yazılım Metrikleri ve Hata Yatkınlığı Arasındaki Spearman İlişki Katsayısı

	Hata Sayısı	Hata Yoğunluğu	Hata Önem Derecesi	Hata Düzeltme Çabası
WMC	0,552 p = 0,098	0,539 p = 0,108	0,559 p = 0,093	0,576 p = 0,082
CBO	0,529 p = 0,116	0,717 p = 0,019	0,540 p = 0,107	0,529 p = 0,116
RFC	0,596 p = 0,069	0,140 p = 0,700	0,643 p = 0,045	0,584 p = 0,077
NC	0,221 p = 0,539	-0,299 p = 0, 401	0,264 p = 0,460	0,176 p = 0,627
NA	0,745 p = 0,013	0,224 p = 0,533	0,722 p = 0,009	0,721 p = 0,019
NM	0,479 p = 0,162	-0,055 p = 0,881	0,529 p = 0,116	0,430 p = 0,214
NEntryA	0,614 p = 0,059	-0,049 p = 0,894	0,637 p = 0,048	0,590 p = 0,073
NExitA	0,431 p = 0,214	-0,178 p = 0,622	0,469 p = 0,171	0,369 p = 0,294
NAc	0,463 p = 0,177	-0,049 p = 0,894	0,502 p = 0,140	0,445 p = 0,197
NSS	0,650 p = 0,042	0,085 p = 0,815	0,686 p = 0,029	0,614 p = 0,059
NCS	0,729 p = 0,017	0,213 p = 0,555	0,753 p = 0,012	0,693 p = 0,026
NG	0,413 p = 0,235	-0,207 p = 0,567	0,451 p = 0,191	0,377 p = 0,283
NE	0,620 p = 0,056	0,061 p = 0,868	0,649 p = 0,042	0,596 p = 0,069
NT	0,636 p = 0,048	0,067 p = 0,855	0,669 p = 0,035	0,612 p = 0,060
CC	0,604 p = 0,065	0,049 p = 0,894	0,639 p = 0,047	0,573 p = 0,083

İki değişken arasındaki ilişki p değeri 0,05'in altındaysa ($p<0,05$) istatistiksel olarak anlamlıdır. İlişki katsayısının gücünün sınıflandırılması hakkında genel bir kural olmamasına rağmen, Cohen [22] ilişki katsayısının mutlak değeri 0,1 ile 0,3 arasındaysa ilişki gücü zayıf, ilişki katsayısının mutlak değeri 0,3 ile 0,5 arasındaysa ilişki gücü orta, ilişki katsayısının mutlak değeri 0,5'ten büyük ise ilişki gücü yüksek olarak tanımlamıştır. Bu bilgiler ışığında yazılım metriklerinin hata yatkınlığı ölçümleri ile ilişkisi Tablo 6'deki şekliye oluşmuştur.

Tablo 6. Yazılım Metrikleri ve Hata Yatkınlığı Arasındaki İstatiksel Olarak Anlamlı İlişki

	Yüksek	**Orta**	**Düşük**
Hata Sayısı	NA, NSS, NCS, NT	-	-
Hata Yoğunluğu	CBO	-	-
Hata Önem Derecesi	RFC, NA, NEntryA, NSS, NCS, NE, NT, CC	-	-
Hata Düzeltme Çabası	NA, NCS	-	-

Tablo 6'deki istatiksel olarak anlamlı ilişkilere bakıldığında hata sayısı ile NA, NSS, NCS ve NT metrikleri yüksek ilişki göstermiştir.

Hata yoğunluğu ile sadece CBO metriği arasında ilişki olduğu tespit edilmiştir. Ayrıca CBO metriği diğer hata yatkınlık ölçümleri ile ilişki göstermemiştir. Buradan hata yoğunluğunun fazla olmasına neden olabilecek metrikler için anlamlı bir sonuç alınamadığı gözlenmektedir. Bunun nedeni, bu çalışma kapsamında incelenen yazılım bileşenlerinin çok değişken oranda yazılım kaynak kodu içermesi olabilir. Örneğin, Bileşen_3 3771 satır, Bileşen_9 23893 satır kaynak kod içermektedir. Literatürde büyük yazılım bileşenlerinin düşük hata yoğunluğu gösterme eğilimi olduğunu gösteren çalışmalar mevcuttur [23].

Hata önem derecesi için NA, NSS, NCS, NT metriklerine ek olarak RFC, NEntryA, NE ve CC metrikleri yüksek ilişki göstermiştir. Araştırma kapsamında incelenen 15 yazılım metriğinden 8'i hata önem derecesi ile yakından ilgili olduğu tespit edilmiştir.

Hata düzeltme çabası için ise sadece NA ve NCS metrikleri yüksek ilişki göstermiştir. Ayrıca bu iki metrik hata sayısı ve hata önem derecesi için de hata yatkınlık ölçümleri ile yakından ilişki içindedir. NA ve NCS metriklerinin hata sayısı, hata önem derecesi ve hata düzeltme çabası için doğrulanmış olması sonuçların tutarlılığını göstermektedir ancak hata düzeltme çabası için sadece iki metrik ile ilişkinin tespit edilmiş olması hata takip aracındaki hata düzeltme çabası kayıtlarının sübjektif olduğunu düşündürtmüştür.

5 Değerlendirme ve Sonuç

Literatürde nesne tabanlı yazılımlara ait yapısal özelliklerin yazılım kalitesi üzerine etkilerini metriklerin kullanımıyla inceleyen çok sayıda çalışma bulunmaktadır. Bu çalışmalar yazılım tasarım metrikleri ile yazılım hata yatkınlığı arasında güçlü bir

ilişki olduğunu doğrulamıştır. Ancak bu çalışmaların önemli bir kısmı yazılım tasarım metriklerini yazılım kaynak kodundan ölçmektedir. UML modellerinden toplanan metrikler ile yazılım hata yatkınlığı arasındaki ilişkiyi inceleyen çalışmaların azlığı bu çalışmanın temel motivasyonunu oluşturmuştur.

Çalışma kapsamında Aselsan bünyesinde geliştirilen 10 yazılım bileşeni incelenmiştir. Literatür araştırması ile ortaya çıkarılan tasarım metrikleri yazılım bileşenlerinin UML modellerinden ölçülmüştür. Hata yatkınlık ölçümleri ise kurum içerisinde kullanılan hata takip aracından elde edilmiştir. Yazılım metrikleri ve hata yatkınlık ölçümleri arasında istatistiksel olarak anlamlı bir ilişki bulabilmek için öncelikle Pearson ilişki katsayısı kullanılmıştır. İncelenen bir takım verilerin ayrık veri içermesi ve normal dağılım göstermemesi sebebiyle analize Spearman ilişki katsayısı ile devam edilmiştir. Analiz sonucunda 15 yazılım metriğinin 9 tanesinin bir veya daha fazla hata yatkınlık ölçümü ile istatistiksel olarak anlamlı ilişki gösterdiği tespit edilmiştir.

Yapılan çalışma sonucunda hata yatkınlığı ile ilişkilendirilen yazılım tasarım metriklerinin gelecek çalışmalar için hatalı yazılım bileşenlerini tahmin etmede kullanılabileceği değerlendirilmektedir. Böylece hataya yatkın olduğu tahmin edilen yazılım bileşenleri için daha fazla kod gözden geçirme, statik kod analizi ve birim test işçiliği harcanarak karşılaşılabilecek hataların önüne geçilebilir.

Ayrıca tasarım metrikleri UML modellerinden ölçüldüğü için henüz kaynak kod üretilmemişken tasarlanan yazılımların kalitesi hakkında öngörü sahibi olunabilir. Ancak yazılım modellerinin soyutlama seviyesinin yüksek olduğu durumlar göz önünde bulundurulmalıdır.

Çalışma kapsamında kısıtlı sayıda yazılım bileşeninin incelenmiş olması, incelenen yazılım bileşenlerinin tek bir yazılım ekibi içerisinde geliştirilmiş olması ve tüm yazılım bileşenlerinin atış kontrol alanına ait olması çalışma sonuçlarının genellenebilirliğini kısıtlamaktadır.

Sonuç olarak, bu çalışmada UML kullanılarak geliştirilen nesne tabanlı yazılımlara ait tasarım metrikleri ile hata yatkınlığı arasındaki ilişki gömülü ve gerçek zamanlı yazılımlar bağlamında gerçek proje verileri kullanılarak analiz edilmiştir.

Kaynaklar

1. Kaur, A., Sandhu, P.S., Bra, A.S.: Early Software Fault Prediction Using Real Time Defect Data. In: 2009 Second International Conference on Machine Vision. IEEE (2009).
2. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering. 20, 476-493 (1994).
3. Nugroho, A., Chaudron, M.R.V., Arisholm, E.: Assessing UML design metrics for predicting fault-prone classes in a Java system. In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). IEEE (2010).
4. Hneif, M., Lee, S.P.: Using Guidelines to Improve Quality in Software Nonfunctional Attributes. IEEE Software. 28, 72–77 (2011).
5. Klas, M., Lampasona, C., Munch, J.: Adapting Software Quality Models: Practical Challenges, Approach, and First Empirical Results. In: 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications. (2011).

6. Nakai, H., Tsuda, N., Honda, K., Washizaki, H., Fukazawa, Y.: A SQuaRE-based software quality evaluation framework and its case study. In: 2016 IEEE Region 10 Conference (TENCON). IEEE (2016).
7. Zhang, B., Shen, X.: The effectiveness of real-time embedded software testing. In: The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety. IEEE (2011).
8. Brito e Abreu, F.: The MOOD Metrics Set. ECOOP'95 Workshop on Metrics (1995).
9. Lorenz, M., Kidd, J.: Object-oriented software metrics. Prentice-Hall, Englewood Cliffs, N.J. (1994).
10. Srivastava, S., Kumar, R.: Indirect method to measure software quality using CK-OO suite. In: 2013 International Conference on Intelligent Systems and Signal Processing (ISSP). IEEE (2013).
11. Binanto, I., Warnars, H.L.H.S., Gaol, F.L., Abdurachman, E., Soewito, B.: Measuring the quality of various version an object-oriented software utilizing CK metrics. In: 2018 International Conference on Information and Communications Technology (ICOIACT). IEEE (2018).
12. McQuillan, J.A., Power, J.F.: On the Application of Software Metrics to UML Models. In: Models in Software Engineering. pp. 217–226. Springer Berlin Heidelberg (2007).
13. Genero, M., Piattini, M., Calero, C.: Empirical validation of class diagram metrics. In: Proceedings International Symposium on Empirical Software Engineering. (2002).
14. Cruz-Lemus, J., Maes, A., Genero, M., Poels, G., Piattini, M.: The impact of structural complexity on the understandability of UML statechart diagrams. Information Sciences. 180, 2209-2220 (2010).
15. Wagner, S., Jürjens, J.: Model-Based Identification of Fault-Prone Components. In: Dependable Computing - EDCC 5. pp. 435–452. Springer Berlin Heidelberg (2005).
16. McCabe, T.J.: A Complexity Measure. IEEE Transactions on Software Engineering. SE-2, 308-320 (1976).
17. Oyetoyan, T.D., Conradi, R., Cruzes, D.S.: A Comparison of Different Defect Measures to Identify Defect-Prone Components. In: 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement. IEEE (2013).
18. Kahraman, E., İpek, T., İyidir, B., Bazlamaçcı, C.F., Bilgen, S.: Bileşen Tabanlı Yazılım Ürün Hattı Geliştirmeye Yönelik Alan Mühendisliği Çalışmaları. In: UYMS'09, pp. 283–287 (2009).
19. Wüst, J.: SDMetrics - the design quality metrics tool for UML models, https://www.sdmetrics.com. Accesed 2013/03/15.
20. Vernezza, T., Granatella, G., Succi, G., Benedicenti, L., Mintchev, M.: Defining metrics for software components. In: World Multi-Conference on Systematics, Cybernetics and Informatics. pp. 16-23 (2000).
21. Bansal, M., Agrawal, C.P.: Critical Analysis of Object Oriented Metrics in Software Development. In: 2014 Fourth International Conference on Advanced Computing & Communication Technologies. IEEE (2014).
22. Muller, K., Cohen, J.: Statistical Power Analysis for the Behavioral Sciences. Technometrics. 31, 499 (1989).
23. Ostrand, T.J., Weyuker, E.J.: The distribution of faults in a large industrial software system. ACM SIGSOFT Software Engineering Notes. 27, 55 (2002).