Erciyes Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
Cilt 37, Sayı 3, 2021

Erciyes University
Journal of Institue Of Science and Technology
Volume 37, Issue 3, 2021

Yeniden Kullanılabilirlik İçin Yazılım Tasarım Kalıplarının Kullanımı

Ali Aydilek*[1], Hacer Karacan[2], Mustafa Dursun[1],

[1] Aselsan A.Ş., ANKARA
[2] Gazi Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü, ANKARA

(Alınış / Received: 14.07.2021, Kabul / Accepted: 03.12.2021, Online Yayınlanma / Published Online: 30.12.2021)

Anahtar Kelimeler
Yazılım tasarım kalıpları, yeniden kullanılabilirlik, sorumluluk zinciri, ilgilerin ayrılığı

Öz: Günümüzde neredeyse her alanda kullanılan yazılımların geliştirilmesi sürecinde karşımıza çıkan takvim sıkışıklığı problemi ve kalite beklentisi nedeniyle yeniden kullanılabilirlik kavramı ön plana çıkmaktadır. Yazılımlarda yeniden kullanılabilirlik arttırıldığında hızlı, düşük maliyetli yazılımlar geliştirmek mümkün olacaktır. Yazılımların yeniden kullanılabilirliği üzerine literatürde mevcut bulunan çalışmalar incelendiğinde en sık kullanılan yöntemin yazılım tasarım kalıpları kullanımı olduğu ve yeniden kullanılabilirliği ciddi bir oranda artırdığı görülmektedir. Bu çalışmada, elde edilen bu bulgulara dayanarak daha önce geliştirilmiş olan bir algoritma yönetimine yönelik yeniden kullanılabilirliğini sağlamak adına yazılım tasarım kalıplarının kullanımına dayalı bir geliştirme yöntemi önerilmiştir. Önerilen yöntem Aselsan bünyesinde mevcut bir proje uygulanmış ve başarılı sonuçlar elde edilmiştir.

Using Software Design Patterns for Reusability

Keywords
Software design patterns, reusability, chain of responsibility, separation of concerns

Abstract: Today, the concept of reusability comes to the fore due to the calendar congestion problem and quality expectation that we encounter during the development process of software used in almost every field. When software reusability is increased, it will be possible to develop fast, low-cost software. When the studies available in the literature on the reusability of software are examined, it is seen that the most frequently used method is the use of software design patterns and it increases reusability significantly. In this study, based on these findings, a development method based on the use of software design patterns is proposed to ensure the reusability of a previously developed algorithm management. The proposed method was implemented in an existing project within Aselsan and successful results were obtained.

***İlgili Yazar, e-mail**: ali.aydilek1@gmail.com

1. Giriş

Günümüzde neredeyse her alanda yazılım kavramı karşımıza çıkmaktadır. Yazılımların geliştirilmesi sürecinde yaşanan takvim sıkışıklığı problemi ve kaliteli yazılım beklentisi nedeniyle yeniden kullanılabilirlik kavramı ön plana çıkmıştır. Yeniden kullanılabilirlik var olan yazılımın aynı şekilde veya ufak değişikliklerle yeni yazılımda kullanılması olarak tanımlanabilir. Yazılımlarda yeniden kullanılabilirlik artırıldığında düşük iş gücüyle daha büyük işler yapmak ve dolayısıyla hızlı geliştirilen, düşük maliyetli yazılımlar elde etmek mümkün olacaktır.

Yazılımda yeniden kullanılabilirlik aktif araştırma ve uygulama alanlarından biridir. Yazılımın yeniden kullanımı yalnızca üretkenliği artırmakla kalmaz, aynı zamanda yazılım ürünlerinin kalitesi ve sürdürülebilirliği üzerinde de

olumlu bir etkiye sahiptir [1]. Yazılımlarda yeniden kullanılabilirliği sağlamak için; yazılım tasarım kalıplarının kullanımı [2-13], yazılım framework kullanımı [14-17] , yazılım ürün hattı yaklaşımı [18-22], servis tabanlı sistemler [23-28], konfigüre edilebilir sistem tasarımı [29-33], eski sistemin revize edilmesi[34-37], cephe yönelimli programlama (Aspect-oriented programming) [38-40] gibi çeşitli yöntemler kullanılır. Yazılımın yeniden kullanılabilirliği, bir yazılım bileşeninin beklenen yeniden kullanım potansiyelini ifade eden bir özelliktir. Yeniden kullanılabilirliği arttırmak adına yapılması gerekenlerden biri de bileşenleri mümkün olduğunca birbirinden bağımsız şekilde tasarlamak ve kodlamaktır. Bir başka deyişle ilgilerin ayrılması (Seperation of Concern-SoC) ilkesine uygun kodlama ve tasarım yapmaktır. [41-47]

Hürsch ve Lopes [44] ilgilerin ayrılması (SoC) yaklaşımı üzerine bir araştırma yaparak mevcut uygulamaları ve faydalarını değerlendirmişlerdir. Çalışmalarının sonucunda ilgilerin ayrılması yaklaşımının faydalarını, azaltılmış karmaşıklık, daha iyi anlaşılabilirlik, artan esneklik ve yeniden kullanılabilirliğin sağlanması olarak ifade etmişlerdir.

Panunzio ve Vardanega [43] gömülü gerçek zamanlı yazılım geliştirmede ilgilerin ayrılması yaklaşımı ile bileşen tabanlı bir yazılım geliştirme metodu sunmuşlardır. Bileşen modeli, özellikle işlevsel ve işlevsel olmayan konular arasında kaygıların ayrılmasını desteklemek için tasarlanmıştır. Yaklaşımın yazılım varlıklarına farklı ilgilerin dikkatli bir şekilde tahsis edilmesiyle ilgilerin ayrılması sağlanır. Oluşturulan bileşenler eşzamanlılık boyutunda görev, senkronizasyon, zamanlama gibi işlevsel olmayan ilgilerden ayrılmıştır. Bu da farklı işlevsel olmayan gereksinimler altında farklı bağlamlarda doğrudan yeniden kullanılabilmelerini sağlamaktadır.

Zelinskyy ve arkadaşları [48] MEMS Program Modeli Oluşturma işlemi için dekoratör tasarım kalıbını önermişlerdir. MEMS, karmaşık bir yapıdır ve insanlığın daha küçük mikrosistemler oluşturmasına izin veren yeni teknolojiler, belirli boyutsal gereksinimleri karşılayan daha karmaşık sistemler oluşturmaya da izin vermektedir. MEMS sistemlerinin önde gelen tiplerinden biri N-katmanlı yapıdır. Bu nedenle N-katmanlı yapıya dayalı olarak oluşturulan program modelinin esnek olması ve MEMS ile ilgili farklı yazılım uygulamaları geliştirirken kullanılması mümkün olmalıdır. Bunu sağlayabilmek adına dekoratör yazılım tasarım kalıbının uygulanmasını öne sürmüşler ve uygulamışlardır. Yaptıkları çalışma sonucunda ortaya koydukları yapı esnek bir yapı olmuştur. Çalışma zamanında özellikler eklenebilir ve kaldırılabilir. Ayrıca geliştirmiş oldukları sisteme yeni bir dekoratör eklemenin de kolay olduğunu belirtmişlerdir.

Tasarım kalıpları yeniden kullanılabilirlik için yaygın olarak kullanılan bir yöntemdir. Ampatzoglou [65] ve arkadaşları yaptıkları çalışmada tasarım modellerinin, sınıflarının ve yazılım paketlerinin yeniden kullanılabilirliğini deneysel olarak araştırmışlardır. Bu araştırma için 100 açık kaynaklı proje üzerinde bir çalışma yapmışlardır. İnceledikleri yöntemler arasında yazılım tasarım kalıbı kullanımı önemli bir başarım göstermiştir.

Anguswamy ve Frakes [49] karmaşıklık ve yeniden kullanım tasarım ilkeleri ile kod bileşenlerinin yeniden kullanılabilirliği arasındaki ilişki üzerine bir çalışma yapmışlardır. Bir bileşenin yeniden kullanılabilirliğini, bileşeni yeniden kullanan kişiler tarafından algılanan yeniden kullanım kolaylığı olarak ölçülebileceğini ifade etmişler ve yapmış oldukları çalışmaya otuz dört denek katılmış ve her denek 5 bileşeni tekrar kullanarak 170 yeniden kullanım örneği oluşturulmuştur. Çalışmalarında bir bileşenin karmaşıklığı ve yeniden kullanım kolaylığı arasındaki ilişkiyi bir regresyon analizi ile analiz etmişlerdir. Çalışmalarında karmaşıklık arttıkça yeniden kullanım kolaylığının azaldığı görülmüştür, ancak korelasyon anlamlı değildir. Bir diğer sonuç ise yeniden kullanım tasarım ilkelerinden olan iyi tanımlanmış arayüzün, netliğin ve anlaşılabilirliğin, yeniden kullanım kolaylığını önemli ölçüde artırdığını görmüşlerdir.

Bir başka çalışmada Ahmaro ve arkadaşları [2] yeniden kullanılabilirliğin önemli bir kavram olduğunu ve Malezya'da bu konu hakkında az araştırma olduğu için bu konuda bir çalışma yaptıklarını belirtmişlerdir. Yapmış oldukları çalışmada Malezya'da benimsenen yazılım yeniden kullanılabilirlik yaklaşımlarını belirlemeye çalışmışlardır. Bunun için bir anket hazırlayıp Malezya'daki 183 yazılım mühendisine göndermişler ve cevaplarını toplayıp analiz etmişlerdir. Araştırma sonucunda kullanılan yöntemlerin yazılım tasarım kalıpları, model tabanlı geliştirme, uygulama çerçeveleri, servis tabanlı geliştirme olduğu gözlemlenmiştir. En çok kullanılan 2 yöntem ise sırasıyla yazılım tasarım kalıpları ve bileşen tabanlı geliştirmedir.

Chen ve arkadaşları [3] yeniden kullanımı sağlayabilmek adına gemi istifleme sisteminde adaptör tasarım kalıbını uygulamışlardır. Bu sayede, sisteme uygun bir şekilde yeni işlevler eklemenin kolaylaşacağını ve böylece kodun yeniden kullanılabilirliğinin ve sürdürülebilirliğinin geliştirileceğini öne sürmüşlerdir.

Kerji [4] yapmış olduğu çalışmada farklı uygulamalara sahip belirli yazılım gereksinimlerinin uygulanmasını basitleştirmek için Web uygulamasına Dekoratör Tasarım Modeli önermiş ve uygulamıştır. Standart web sayfasına ek olarak oturum açan kullanıcı türüne bağlı olarak, gerekli XHTML kodu oluşturulacak ve tarayıcıya işlenecektir.

Çalışmada XML web uygulamasının değişen gereksinimlere uyarlanabilirliğini artırmak için kullanılır. XML kullanımı, uygulamayı gereksinimdeki değişikliklere karşı esnek hale getirmiş ve bu da kod bakımını kolaylaştırmıştır. Ortaya çıkan uygulama, performansta gelişme, yeniden kullanılabilirliği ve bakım kolaylığını artırmıştır.

Sorumluluk zinciri (Chain of Responsibility) yazılım tasarım kalıbı önemli tasarım kalıplarından biridir. Bu tasarım kalıplarında zincir halkaları sabittir. Shirazi ve arkadaşları [5] ise dinamik zincir halkalarına sahip olmayı mümkün kılan yeni bir tür Sorumluluk Zinciri tasarım kalıbı önermişlerdir. Bu dinamik zincirin halkaları, önceliklerine göre değiştirilebilen ve öncelik sürecinde kullanılabilen halka pozisyonunu ifade eder. Böylelikle daha esnek bir yapı elde edilmektedir.

Sorumluluk zinciri (Chain of Responsibility) tasarım kalıbı bir zincire benzer ve sorumluluk zincirinin bir işleyicisi, talebi ele alır veya talep Sorumluluk zincirinin bir sonraki işleyicisine iletir. Yueping ve arkadaşları [6] yaptıkları çalışmada, gözlemci örüntüsünün bir tür bileşik örüntü haline gelmesi için sorumluluk zinciri örüntüsüne gömülmesini önermişlerdir. Böylelikle sorumluluk zinciri örüntüsünün yeteneğinin güçleneceğini ve daha esnek olabileceğini belirtmişlerdir.

Aygıt ağaçları, düğüm özelliklerine sahip gömülü bir sistem içindeki aygıtların ve çevre birimlerini tanımlayan yapılardır. Geliştiriciler aygıt ağaçlarının yapısı gereği aygıt ağacı uygulamaları oluştururken çoğunlukla zorluklarla karşılaşmaktadırlar. Bununla beraber farklı mikroişlemci mimarilerine ait aygıt ağacı geliştirilmesi zaman alıcı bir süreç olabilmektedir. Arslan ve Kardas [50] bu zorlukları aşabilmek için aygıt ağacı yazılımının model güdümlü olarak geliştirilmesini sağlayan DSML4DT isimli alana özgü bir modelleme dili önermişlerdir. Önerdikleri sistemi akıllı ulaşım sistemleri geliştiren bir firmada değerlendirmişlerdir. Sonuç olarak aygıt ağacı yapılarının %76'sının önerdikleri model yoluyla otomatik olarak geliştirilebileceğini göstermişlerdir. Mevcut sistemle kendi önerdikleri yöntemi karşılaştırdıklarında, önerdikleri yöntemin geliştirme sürecini yarıya indirdiğini belirtmişlerdir. Geliştiricilerden aldıkları geri bildirimler sonucunda önerdikleri yöntemin yeniden kullanılabilirliği arttırdığı sonucuna varmışlardır.

Günümüzde mikro servis mimarisi sıklıkla kullanılmaktadır. Büyük uygulamalar mikro servis mimarisiyle birlikte küçük servislere bölünerek yönetilebilmektedir. Mikro servislere olan ilginin artması mikro servis mimarisi ile geliştirme yapılırken farklı yaklaşımların ortaya çıkmasına neden olmuştur. Bu yaklaşımlardan biri de model güdümlü geliştirmedir. İçöz ve Kalıpsız [51] yaptıkları çalışmada mikro servis mimarisi kullanılarak restful web servislerinin geliştirilmesi için model güdümlü bir yöntem önermişlerdir. Yaptıkları değerlendirmeler sonucunda önerdikleri yöntemin yazılımın verimliliğini arttırdığını, geliştirme sürecini hızlandırdığını ifade etmişlerdir.

Bu çalışmada yeniden kullanılabilirlik için yazılım tasarım kalıplarının kullanılması üzerine bir araştırma yapılmıştır. Bu bulgulara dayanarak daha önce geliştirilmiş bir algoritma yönetiminin yeniden kullanılabilirliğini sağlamak amacıyla yazılım tasarım kalıplarının kullanılmasına dayalı bir yöntem önerilmiştir.

2. Materyal ve Metot

2.1. Yeniden Kullanılabilirlik

Yazılımın yeniden kullanımı, yeni bir sistem oluşturmak için mühendislik bilgisinin veya mevcut yazılım bileşenlerinden elde edilen eserlerin kullanılmasıdır [52-55]. Bir başka deyişle yeniden kullanılabilirlik geçmişte gerçekleştirilmiş olan yazılım geliştirme etkinliklerinde elde edilen bilginin yeni sistemler geliştirmede kullanılması olarak tanımlanmaktadır.

2.2. Yazılım Tasarım Kalıpları

Tasarım kalıplarının varlığı ilk olarak bir mimar olan Christopher Alexander [56] tarafından ortaya konulmuştur. Christopher Alexander yaptığı araştırmalar sonunda benzer problemleri çözmek için oluşturulan ve beğenilen (kaliteli) mimari yapılarda ortak özellikler (benzerlikler) olduğunu belirlemiştir. Bu benzerliklere kalıplar (patterns) adını vermiştir. Her kalıp gerçek dünyada defalarca karşılaşılan bir problemi ve o problemin çözümünde izlenmesi gereken temel yolu tarif etmektedir. Bir problemle karşılaşan tasarımcı eğer daha önce benzer problemle karşılaşan tasarımcının uyguladığı başarılı çözümü biliyorsa (kalıp) her şeyi yeniden keşfetmek yerine aynı çözümü tekrar uygulayabilir. Tasarım kalıpları gereksinimlerden tasarıma ve uygulamaya kadar uzanan ve her aşamada değişkenlik gösteren karmaşık soyutlamalardır. Tasarım kalıpları denenmiş ve test edilmiş, kanıtlanmış programlama ve kod yapılandırma yolları olarak da tanımlanabilir [57-59].

Toplam 23 adet olan bu tasarım kalıpları temel olarak Yaratımsal, Yapısal ve Davranışsal olmak üzere 3 başlığa ayrılır.

2.2.1. Yaratımsal Tasarım Kalıpları

Yaratımsal tasarım kalıpları, tek önerisi nesneleri ve sınıfları yaratma, başlatma ve yapılandırma işini kolaylaştırmak olan kalıplardır. Yazılım sistemindeki nesnelerin yaratılışı hakkında yol gösterirler. Bu tür desenler, nesnelerin örneklerini oluşturmamız, bu nesneleri depolamamız, nesnelerin kopyalarını oluşturmamız gerektiğinde kullanışlıdır [58-60].

2.2.2. Yapısal Tasarım Kalıpları

Yapısal tasarım modelleri, nesneleri daha büyük, daha yapılandırılmış bir yapıda birleştirmenize yardımcı olur. Yapısal sınıf kalıpları, genel olarak arayüzleri veya uygulamaları oluşturmak için kalıtımı kullanılır [59,62].

2.2.3. Davranışsal Tasarım Kalıpları

Davranışsal tasarım modelleri, bir kod tabanındaki birden çok nesne arasındaki iletişime yardımcı olmaya odaklanır. Davranış kalıpları, algoritmalar ve nesneler arasındaki sorumlulukların atanması ile ilgilidir. Davranışsal tasarım kalıpları sadece nesnelerin veya sınıfların kalıplarını değil, aynı zamanda bunlar arasındaki iletişim kalıplarını da tanımlar. Bu modeller, çalışma zamanında takip edilmesi zor olan karmaşık kontrol akışını karakterize eder [59,62].

Tiplerine göre tasarım kalıpları Şekil 1'de verilmiştir.

Yaratımsal Tasarım Kalıpları	Yapısal Tasarım Kalıpları	Davranışsal Tasarım Kalıpları
Factory	Adapter	Chain of Responsibility
Abstract Factory	Composite	Command
Builder	Decorator	Iterator
Prototype	Facade	Observer
Singleton	Flyweight	Mediator
	Mixin	Memento
	Module	Promises
	Proxy	Strategy

Şekil 1. Tiplerine göre tasarım kalıpları

2.3. Uygulanan Yöntem

Önerilen yöntemin uygulandığı Aselsan bünyesinde geliştirilen projelerde çok sayıda algoritma kullanılmaktadır. Bu algoritmalar ilgili algoritma birimi tarafından geliştirilmekte ve Yazılım Konfigürasyon Birimi tarafından kullanılmaktadır. Bu algoritmaların yeniden kullanımı ile ilişkili bazı problemler şu şekildedir:

1. Projeden projeye bu algoritmaların içerikleri değişebilir.
2. Girdi ve çıktıları değişebilir.
3. Bir projede kullanılan algoritma diğer projede kullanılmayabilir.
4. Aynı proje içerisinde farklı durumlara göre bu algoritmaların kullanılma sırası çalışma zamanında değişebilmektedir. Örneğin projede kullanılacak A1, A2, A3 olmak üzere 3 tane algoritma olduğunu varsayalım;
    - Durum 1 Algoritma çalışma sırası: A1-A2
    - Durum 2 Algoritma çalışma sırası: A1-A3
    - … vb olabilir.

Daha önce geliştirilmiş olan sistemlerde bu problemlerin çözümü kodda büyük değişiklikler yapılarak çözümlenmiş durumdadır. Anlaşılabilirlik son derece düşüktür. Ayrıca bir algoritma çıkarılmak veya eklenmek istendiğinde birçok yeri etkilemekte ve etkilediği yerlerde değişiklikler gerektirmektedir. İlgilerin ayrılığı ilkesine bağlı bir geliştirme yapılmamıştır. Bu nedenle de değişiklik yapma maliyeti yüksektir ve yeniden kullanıma uygun bir kod yapısı yoktur.

Yapılan literatür taraması sonucunda yazılım tasarım kalıplarının kullanımının yazılımda yeniden kullanılabilirliği artırdığı görülmüştür ve yeniden kullanılabilirliği sağlamak adına en çok tercih edilen yöntemlerden biridir.

Önerilen yöntemde algoritmaların ayrı bir modül içerisinde ele alınması kararlaştırılmıştır. Algoritmaların bir sıra içerisinde çalışacağı göz önüne alındığında ve literatüre de bakıldığında bunun için en doğru tasarım kalıbının Sorumluluk Zinciri (Chain of Responsibility) tasarım kalıbı olduğu görülmüştür.

Sorumluluk zinciri tasarım kalıbının seçilmesinin nedenleri şu şekilde sıralanabilir;

- Zincirin düğümleri çalışma sırasında dinamik olarak eklenebilir veya çıkarılabilir.
    - Geliştirmekte olduğumuz algoritma modülünde algoritmaların işleyiş sıraları durumlara göre değişiklik gösterebildiği için çalışma sırasında oluşan farklı durumlarda farklı zincirlerin kullanılabilmesini sağlayacaktır.

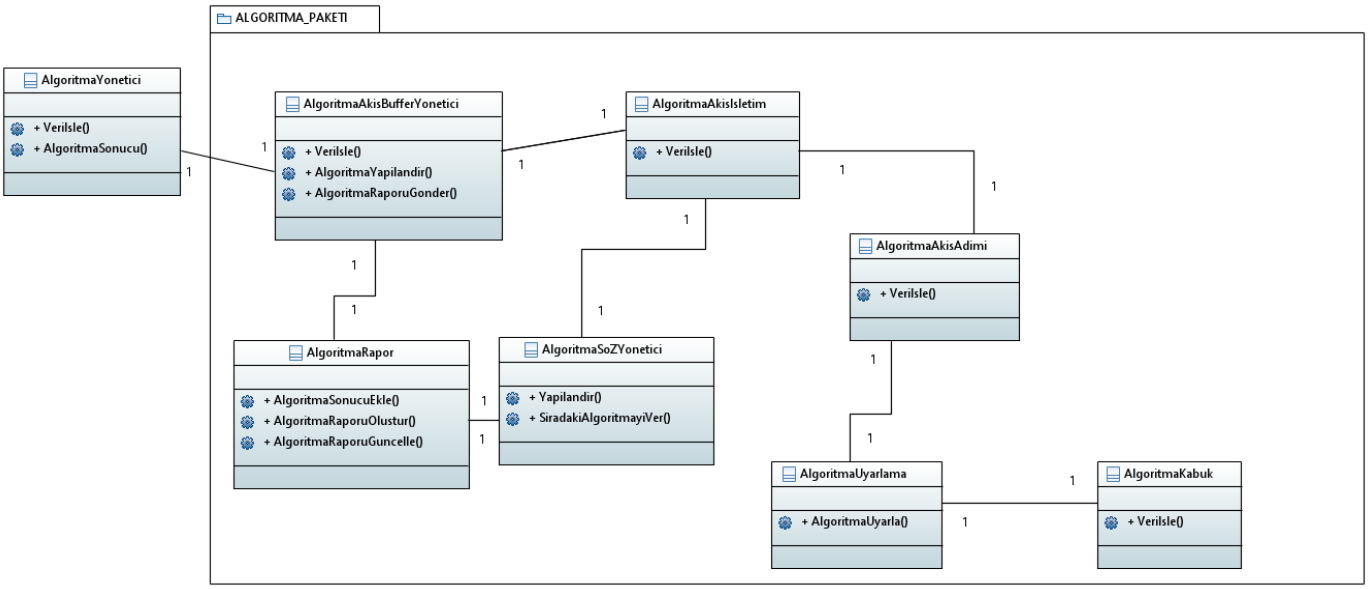
- Bir düğümün, zincirdeki bir sonraki düğümün isteği nasıl ele aldığını bilmesi gerekmez.
    - Yeni bir algoritma ekleneceği zaman diğer algoritmaların işleyişine bakılması gerekmeyecektir. Bağımsız olarak eklenebilecektir.

- Bir zincir düğümü çıkarıldığında diğer düğümler bu durumdan etkilenmez.
    - Önerdiğimiz yöntemde her algoritma bir düğümü ifade etmektedir. Bir algoritma bir projede kullanılırken başka bir projede kullanılmayabilir. Kullanılmayan algoritmayı zincirden çıkardığımızda diğer düğümler bu durumdan etkilenmeyeceği için başka bir değişik yapmaya gerek kalmadan ilgili algoritmayı zincirden çıkarmak yeterli olacaktır.

Algoritma modülünde kullanılan sınıflar ve veri yapıları şu şekildedir:

1. Algoritma Akış Buffer Yönetici Sınıfı: Algoritma işleyişlerinin kontrol edecek olan sınıftır. Ayrıca algoritmalara ait veriler burada tutulacaktır. Modülün dışarıyla olan arayüzünü sağlayacaktır.
2. Algoritma Akış İşletim Sınıfı: Akışları kontrol edecek olan sınıftır.
3. Algoritma Akış Adımı Sınıfı: Algoritma akış adımlarını kontrol edecek olan sınıftır.
4. Algoritma Uyarlama Sınıfı: Mevcut verilerden algoritmaya gerekli olan girdileri sağlayacak olan sınıftır.
5. Algoritma Kabuk Sınıfı: Algoritma kütüphanesi ile olan arayüzü sağlayacak olan sınıftır.
6. Algoritma SoZ (Sorumluluk Zinciri) Yonetici Sınıfı: Algoritma akışlarınının yapılandırıldığı ve ilgili akışa ait algoritma sırasını yönetecek olan sınıftır. Sorumluluk zinciri tasarım kalıbının uygulandığı sınıftır.
7. Algoritma Rapor Sınıfı: Algoritma sonuçlarında bir rapor oluşturacak olan sınıftır.
8. AkisBufferlari: İçerisinde algoritma kullanıcı tarafından gönderilen veriyi, algoritma girdi çıktı bilgilerini ve dışarıdan algoritmaların kullanacağı bilgileri barındıran veri yapısıdır.

Önerilen yönteme ait sınıf diyagramı Şekil 2'de verilmiştir. Ticari gizlilik nedeniyle çözümün anlaşılması için yeterli olan belli başlı fonksiyon isimleri verilmiştir. AlgoritmaUyarla ve AlgoritmaKabuk sınıfları temel sınıflar olmak üzere yeni eklenecek olan uyarlama ve kabuk sınıfları bu sınıflardan türetilmektedir.



Şekil 2. Önerilen yönteme ait sınıf diyagramı

Önerilen sistemde her durum bir akışı ifade etmektedir. İlgili algoritmalar ilgili sırayla akışlara eklenecektir. Sistemde hangi algoritmaların hangi sıralamada kullanılacağının belirlenebilmesi için Algoritma SoZ Yonetici sınıfındaki Yapilandır() metodu kullanılacaktır.

Önerilen yöntem iki aşamada değerlendirilebilir;

1. Algoritma Kullanıcı, Algoritma Yönetici ve Algoritma Modülü arasındaki işlemler
2. Algoritma Modülü içerisindeki işlemler

2.3.1.Algoritma kullanıcı, algoritma yönetici ve algoritma modülü arasındaki işlemler

Algoritma Kullanıcı, Algoritma Yönetici ve Algoritma Modülü arasındaki işlemlere ait sıra diyagramı Şekil 3'te verilmiştir. Bu işlemler aşağıda ifade edilmiştir;

1. Algoritma Yapılandırılması

Algoritma modülünün kullanılabilmesi için öncelikle yapılandırılması gerekmektedir. Bu işlem için Algoritma Kullanıcı tarafından gerekli parametrelerle Algoritma Yonetici sınıfına ait AlgoritmaYapilandir() metodu çağrılır. Algoritma Yönetici sınıfı bu isteği aldığında AlgoritmaAkisBufferYonetici sınıfına ait AlgoritmaYapilandir() metodunu çağırır.

2. Verinin İşlenmesi

Eldeki verinin işlenebilmesi için Algoritma Kullanıcı AlgoritmaYonetici sınıfının VeriIsle() metodunu çağırır. AlgoritmaYonetici bu isteği aldığında gelen parametrelere göre verinin sistemde var olan hangi akışın kullanılarak işleneceğini belirler ve bu akışı çalıştırmak üzere AkışBufferYonetici sınıfına ait VeriIsle() metodunu çağırır.

3. Algoritma Raporunun Alınması

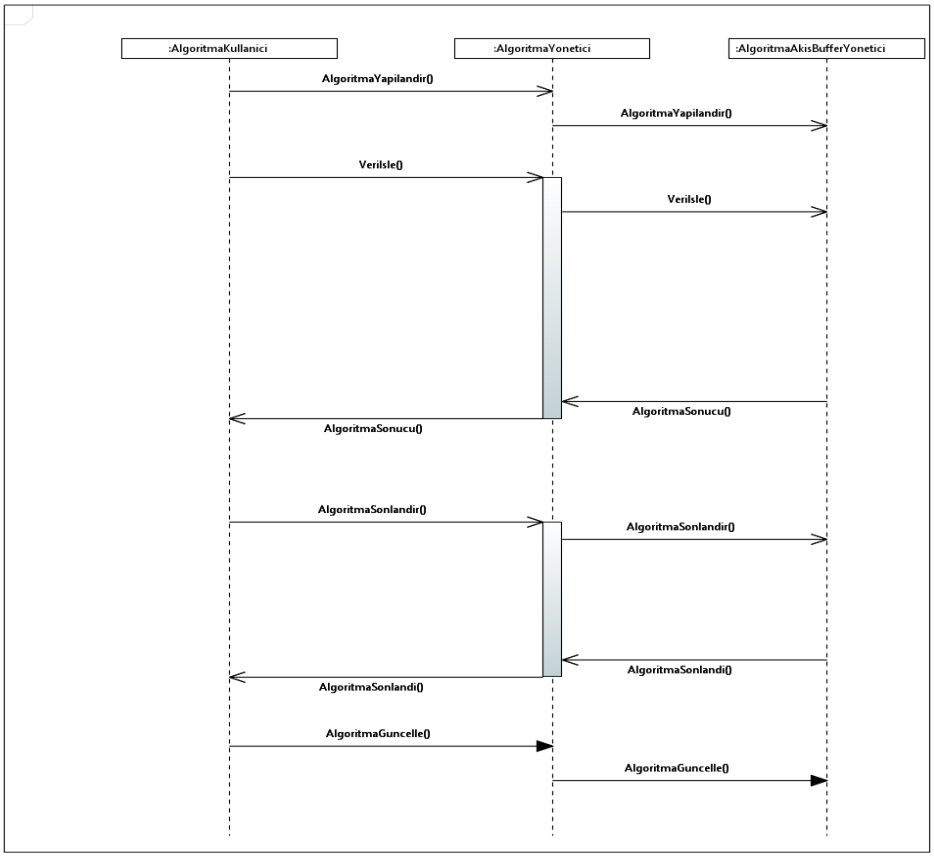
Algoritma Modülü veri işleme aşamasını tamamlayıp sonuçları raporladığında AlgoritmaYonetici sınıfına ait AlgoritmaSonucu() metodunu çağırır. AlgoritmaYonetici bu isteği aldığında gerekli çevrimleri yaparak Algoritma Kullanıcıya algoritma sonucunu iletir.

4. Algoritma Sonlandırılması

Algoritma Kullanıcı algoritmaların sonlandırılmasını istediğinde Algoritma Yönetici sınıfınına ait Algoritma Sonlandir metodunu çağırır. Algoritma Yönetici bu isteği Algoritma Akış Buffer Yönetici sınıfına aktarır.

5. Algoritma Parametrelerinin Güncellenmesi

Sistem çalışırken bazı değişimler sonucu Algoritmaların ayar ve ilklendirme parametrelerinin güncellenmesi gerekebilir. Böyle bir durum olduğunda işleyişi bozmadan güncelleme yapılabilmesi gerekir. Bu işlem için Algoritma Kullanıcı Algoritma Yönetici sınıfına ait AlgoritmaGuncelle() metonu çağırır. Algoritma Yönetici sınıfı bu isteği Algoritma Akış Buffer Yönetici sınıfına iletir.



Şekil 3. Algoritma Kullanıcı, Algoritma Yönetici ve Algoritma Modülü arasındaki işlemlere ait sıra diyagramı

2.3.2. Algoritma modülü içerisindeki işlemler

Algoritma Modülü içerisindeki işlemlere ait sıra diyagramı Şekil 4'te verilmiştir. Bu işlemler aşağıda ifade edilmiştir;

Verinin İşlenmesi ve Raporlanması Aşamaları;

1. AlgoritmaAkışBufferYonetici sınıfına ait VeriIsle() metodu çağrıldığında ilgili akış ait rAkisBufferlari oluşturulur.

2. AkışIsletim sınıfına ait VeriIsle() metodu çağrılır.

3. AkışIsletim sınıfı Algoritma SoZ Yonetici sınıfına ait olan SiradakiAlgoritmayiVer() fonksiyonunu çağırarak hangi algoritmayı işleteceğine karar verir.

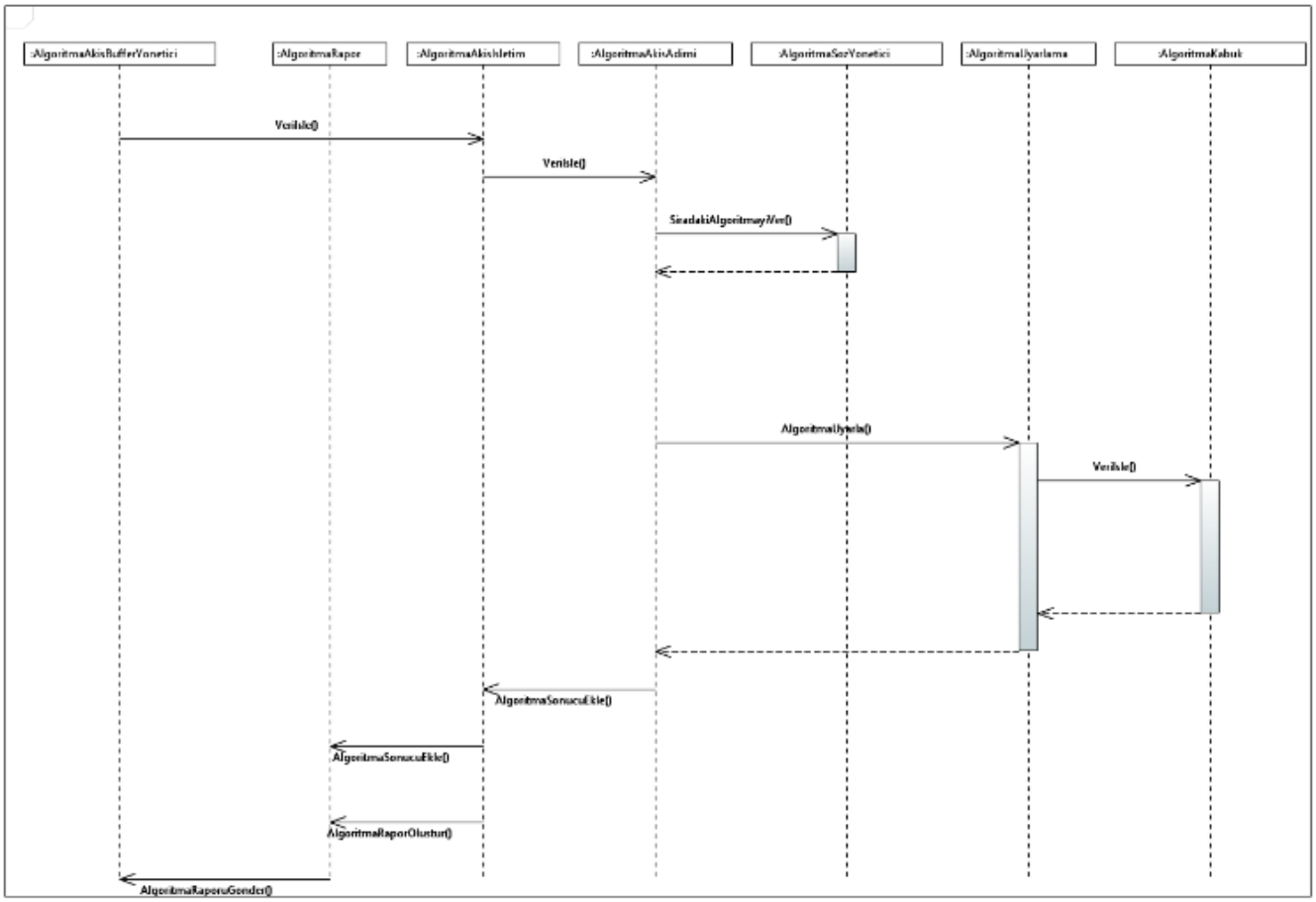
4. İlgili algoritmanın uyarlama sınıfı yardımıyla eldeki veriler ve eğer gerekiyorsa önceki algoritmaların sonuçları kullanılarak algoritma girdisi sağlanır.

5. Algoritma girdisi ayarlandıktan sonra Algoritma Kabuk yardımıyla algoritma çağrılır ve sonucu alınır.

6. Akış içerisindeki algoritmalar sırayla çalıştırılır.

7. Akışa ait tüm algoritmaların çalışması tamamlandığında Algoritma Rapor sınıfı yardımıyla sonuç raporu oluşturulur

8. Sonuç raporu Algoritma Yonetici sınıfına gönderilir.



Şekil 4. Algoritma Modülü içerisindeki işlemlere ait sıra diyagramı

Bu çalışmada Sorumluluk zinciri (Chain of Responsibility) tasarım kalıbı dışında Adaptör tasarım kalıbı da kullanılmıştır. Algoritma modülünün kullanıcısıyla arasında adaptör görevi yapacak olan Algoritma Yönetim sınıfı tasarlanmıştır. Bu sınıf algoritma kullanıcı ile algoritma modülü arasında adaptör görevi görecektir. Böylelikle algoritma modülü kullanıcıdan bağımsız olarak çalışacaktır. Bu şekilde proje bağımlılığının azaltılması amaçlanmaktadır. Proje farklılıklarında algoritma modülünün aynı şekilde kullanılarak Algoritma Yönetim sınıfında değişiklik yapılması amaçlanmaktadır.

3. Bulgular

Günümüzde neredeyse her alanda kullanılan yazılımların geliştirilmesi sürecinde karşımıza çıkan takvim sıkışıklığı problemi ve kalite beklentisi nedeniyle yeniden kullanılabilirlik kavramı ön plana çıkmaktadır. Yeniden kullanılabilirlik yazılım güvenilirliğini artırır, süreç riskini azaltır, geliştirme sürecini hızlandırır ve iş gücünün optimum seviyede kullanılmasını sağlar. [54,63-64] Bu nedenlerden dolayı bu makalede yeniden kullanılabilirlik kavramı ele alınmış ve yeniden kullanılabilirliği sağlamak adına bir çalışma yapılmıştır.

Daha önce yapılan çalışmalara bakıldığında yazılım yeniden kullanılabilirliğini sağlamak ve arttırmak için en çok uygulanan yöntemlerden birisi yazılım tasarım kalıplarının kullanılmasıdır [2.49,65]. Daha önceki çalışmalarda yazılım tasarım kalıbı uygulamalarından başarılı sonuçlar elde edilmiştir. [2-8,13]. Geçmiş çalışmalar referans alınarak bu çalışmada da yeniden kullanılabilirliği sağlamak adına yazılım tasarım kalıbı kullanılmış ve başarılı bir sonuç elde edilmiştir.

Uygulanan projede verinin belli algoritmalar kullanılarak işletilip raporlanması gerekmektedir. Algoritmalar art arda kullanıldığı ve bir algoritmanın sonucu diğer algoritma tarafından kullanılabildiği için ve literatürde de yeniden kullanılabilirlik için başarılı sonuçlar ortaya koyan Sorumluluk Zinciri tasarım kalıbının uygulanması doğru bulunmuştur. [5-6,59]

Sorumluluk zinciri tasarım kalıbının yanı sıra Adaptör tasarım kalıbı da uygulanmıştır. Algoritma Yönetici sınıfı Algoritma Kullanıcı ile Algoritma Modülü arasında adaptör görevi görecektir. Böylece Algoritma Kullanıcı, Algoritma Modülünün arayüzleri ve içeriğinden bağımsız olacaktır. Projeden projeye oluşabilecek bazı farklılıklarda Algoritma Yönetici sınıfında düzenlemeler yapılarak Algoritma Modülü değiştirilmeden kullanılabilir.

Yeniden kullanılabilirliği sağlamak için yapılması gerekenlerden birisi de ilgilerin ayrılığı ilkesine uygun tasarım ve geliştirme yapmaktır [41,42]. Önerilen Algoritma Modülü projeden bağımsız olacak şekilde tasarlanmış ve uygulanmıştır. Algoritma Kullanıcı ve Algoritma Modülü arasında iletişimi sağlamak için Algoritma Yönetici sınıfı tasarlanmıştır. Algoritma Kullanıcı Algoritma Modülünün içeriğinden haberdar değildir ve içeriği Algoritma Kullanıcıyı etkilemez. Algoritma Modülünün genel olarak amacı veriyi işlemek ve bu veriden bir rapor oluşturmaktır. Bu sayede, Algoritma Modülünün içinde yapılacak olan değişiklik Algoritma Kullanıcısını etkilemeyecektir.

Projeden projeye Algoritma Modülü'nden beklenen sonuç raporu formatı değişebilir. Bu durumda yapılması gereken yalnızca Algoritma Modülü içerisindeki Algoritma Rapor sınıfında değişiklik yapmak olacaktır.

Yazılımda var olmayan bir algoritma ekleneceği zaman bu algoritmanı uyarlama ve kabuk sınıfları yazılıma eklenecek ve gerekli kodlama yapılacaktır. Bunun ardından bu algoritma adımı ilgili akışa Algoritma SoZ Yönetici sınıfı içerisindeki yapilandir() metodu kullanılarak eklenecektir. Projede var olan algoritma başka projede kullanılmayacağı zaman sadece algoritma akışlarından çıkarılması yeterli olacaktır.

Algoritma modülünün yeniden kullanılabilirliğini artırmak için işlenmek üzere algoritmaya gelen veri char* şeklinde ele alınmıştır. Veri ilgili metodun içinde projeye özgü tipe dönüştürülmektedir. Böylelikle farklı projelerde girdi tipi değiştiğinde metotların argümanlarının değişimine gerek duyulmayacaktır. Arayüzler aynı şekilde kullanılabilecektir.

Yazılım kalitesinin ve yeniden kullanılabilirliğinin değerlendirilmesi için kullanılan yöntemlerden biri nesne yönelimli tasarım metriklerinin kullanılmasıdır. [66-73] Önerilen yöntemin başarısını ölçmek için Chidamber ve Kemerer'in geliştirdiği metrik kümesinden yararlanılmıştır [75]. İlgili metrikler ve tanımları [66-73] Şekil5'te verilmiştir.

Metrik	**Tanımı**
LCOM (Lack of Cohesion in Methods/Metotlardaki uyum eksikliği)	Ayrık yöntem çiftlerinin sayısı ile kullanılan benzer yöntem çiftlerinin sayısını çıkarmak için kullanılır.
DIT (Depth of Inheritance Tree / Kalıtım ağacının derinliği)	Bir sınıfın, o sınıftan kalıtım ağacının köküne kadar olan maksimum yol uzunluğunu ifade eder.
CBO (Coupling between object classes/Nesne sınıfları arasındaki bağımlılık)	Belirli bir sınıfın bağımlı olduğu sınıfların sayısını ifade eder.
NOC (Number of Children/Alt sınıf sayısı)	Bir sınıftan direkt olarak türetilmiş sınıfların sayısını ifade eder.
RFC (Response For a Class/Sınıfın tetiklediği metot sayısı)	Bir sınıfın nesnesi tarafından alınan bir mesaja yanıt olarak potansiyel olarak yürütülebilecek yöntemlerin sayısıdır.
WMC (Weighted Methods Per Class / Sınıfın ağırlıklı metot sayısı)	Bir sınıftaki yöntemlerin ağırlıklı toplamını ifade eder.

Şekil 5. Kullanılan metrikler ve tanımları [75]

Metriklerin ölçümü için "Understand" adlı uygulama kullanılmıştır. Şekil 6'da eski projeye ait nesne yönelimli tasarım metrikleri, Şekil 7'de ise önerilen yöntemin uygulandığı projenin nesne yönelimli tasarım metrikleri verilmiştir.

SINIF	LCOM	DIT	CBO	NOC	RFC	WMC
Algoritma	87	1	7	1	41	40
AlgoritmaKullanicisi	0	0	1	1	7	7
AlgoritmaParametreleri	72	0	0	1	10	10
AlgoritmaTakipcisi	65	1	8	0	23	16
AlgoritmaUretici	0	1	3	0	3	3
XY_Algoritmasi	95	2	24	0	189	148
XY_AlgParametreleri	92	1	0	0	50	40

Şekil 6. Eski projeye ait nesne yönelimli tasarım metrikleri

SINIF	LCOM	DIT	CBO	NOC	RFC	WMC
AlgoritmaAkisAdimi	75	1	6	0	28	28
AlgoritmaAkisBufferYonetici	94	1	17	0	71	61
AlgoritmaAkisIsletim	83	1	10	0	37	37
AlgoritmaXKabuk	71	1	12	0	24	7
AlgoritmaXUyarlama	0	1	1	0	11	3
AlgoritmaYKabuk	91	1	17	0	32	15
AlgoritmaYUyarlama	0	1	6	0	14	6
AlgoritmaKabuk	100	0	4	2	17	17
AlgoritmaUyarlama	100	0	4	2	17	17
AlgoritmaRapor	89	0	21	0	38	38

Şekil 7. Önerilen yöntemin uygulandığı projeye ait nesne yönelimli tasarım metrikleri

WMC değeri bir sınıftaki fonksiyonların sayısını ifade eder. WMC değeri büyüdükçe sınıf projeye özgü olma yolunda ilerler. Bu nedenden ötürü de yeniden kullanılabilirlik düşer. WMC değeri büyüdükçe aynı zamanda bakım maliyeti de artar. [66-73] Şekil 5 ve Şekil 6'ya bakıldığında önerilen yönteme ait sınıflardan en büyük WMC değerine sahip sınıfın WMC değeri 61 iken daha önce geliştirilen yönteme ait sınıflar arasında en büyük WMC değerine sahip sınıfın WMC değeri ise 148'dir. Bu metriğe bakıldığında önerilen yöntemin yeniden kullanılabilirliğinin eski yönteme göre daha yüksek olduğu çıkarımı yapılabilir. Ayrıca eski yöntemdeki bu en yüksek WMC değerine sahip sınıf (XY_Algoritmasi sınıfı) projeden projeye farklılık göstermesi ve değiştirilmesi beklenen sınıftır. Bu nedenden ötürü de eski projenin yeniden kullanılabilirliği zordur. Önerilen yöntemde projeden projeye farklılık göstermesi beklenen sınıfların WMC değerleri en büyük olanının (AlgoritmaXKabuk sınıfı) WMC değeri ise 7'dir. Değiştirilmesi ve bakım maliyeti düşüktür.

NOC değeri bir sınıftan direkt olarak türetilmiş sınıfların sayısını ifade eder. [66-73] Bilindiği üzere kalıtım yeniden kullanılabilirliği sağlamanın ve artırmanın yöntemlerinden biridir. [70,74] Bu nedenle NOC değeri büyüdükçe yeniden kullanılabilirliğin de artacağı sonucuna varılabilir. Şekil 5 ve Şekil 6'ya bakıldığında NOC değeri en büyük sınıfların AlgoritmaUyarlama ve AlgoritmaKabuk sınıflarının olduğu görülmektedir. Projeden projeye farklı olması beklenen yerlerden en önemlileri algoritma girdi ve çıktılarının değişmesidir. Bu durumda da AlgoritmaUyarlama ve AlgoritmaKabuk sınıflarında değişiklik yapılması gerekecektir. Bu sınıfların yeniden kullanılabilirliği artıracağı NOC değerlerine bakarak görülebilir.

CBO değeri bir sınıfın bağımlı olduğu sınıfların sayısını ifade eder. Nesne sınıfları arasında aşırı bağlantı, modüler tasarıma zarar verir ve yeniden kullanımı önler. [66-73]. CBO değerlerine bakıldığında önerilen yöntemin uygulandığı projenin eski projeye göre daha düşük CBO değerlerine sahip olduğu görülmektedir. Bu da önerilen yöntemim daha modüler ve yeniden kullanılabilir olduğunu göstermektedir.

4. Tartışma ve Sonuç

Bu çalışmada yeniden kullanılabilirliği sağlamak için yapılan çalışmalar incelenmiştir. Yapılan literatür taramasında yeniden kullanılabilirliğin yazılım geliştirme için oldukça önemli bir kavram olduğu ve yeniden kullanılabilirliği sağlamak ve artırmak için en çok uygulanan yöntemlerden birinin yazılım tasarım kalıbı kullanımının olduğu görülmüştür. Bu bulgulara dayanarak daha önce geliştirilmiş bir algoritma yönetim modülünün yeniden kullanılabilirliğini sağlamak amacıyla yazılım tasarım kalıplarının kullanılmasına dayalı bir yöntem önerilmiş ve Aselsan bünyesindeki geliştirme süreci devam eden mevcut bir projeye uygulanmıştır. Bununla birlikte ilgilerin ayrılığı ilkesine uygun olarak yazılım geliştirmenin de yeniden kullanılabilirliği artırdığı literatürde gözlemlendiği için ilgili modül ilgilerin ayrılığı ilkesine uygun olarak geliştirilmiştir.

Literatürdeki çalışmalara bakıldığında [3-6] yeniden kullanılabilirliği sağlamak için tasarım kalıplarının kullanıldığı çalışmalar ile önerdiğimiz yöntem benzer sonuçlar ortaya koymuştur. Tasarım kalıbının kullanımı yeniden kullanılabilirliği arttırmıştır. Yeniden kullanılabilirliği sağlamak için tasarım kalıpları önemli bir yöntemdir. İncelenen çalışmalarda adaptör, dekoratör ve sorumluluk zinciri tasarım kalıpları kullanılmıştır. Önerdiğimiz yöntemde ise sorumluluk zinciri tasarım kalıbı kullanılmıştır. Önerdiğimiz yöntemi etkin kılan noktalardan biri mevcut bir projeye uygulanması ve daha sonra gerçekleştirilecek olan projelerde kullanılacak olmasıdır. Önerdiğimiz yöntemin başarısı makale kapsamında incelenen çalışmaların [3-6] aksine yalnızca nitel olarak değil nicel olarak da ortaya koyulmuştur. Böylelikle önerilen yöntemin başarısı daha net bir şekilde ifade edilmiştir.

Önerilen yöntemin başarımını daha net ortaya koyabilmek için önerilen tasarımın başka bir projeye daha uygulanarak sonuçların değerlendirilmesi daha uygun olacaktır. Ancak proje büyüklüğü, insan kaynağı ve zaman kısıtları nedeniyle bu karşılaştırma yöntemi uygulanamamıştır. Gelecek dönemde böyle bir çalışma yapılması planlanmaktadır. Bununla birlikte başarımın ölçülmesi için kullanılan yöntemlerden biri olan uzman görüşü alma yöntemi de sonraki aşamalarda uygulanacaktır.

Teşekkür

Bu çalışma Aselsan Akademi Lisansüstü Tez projesi kapsamında desteklenmiştir.

Kaynakça

[1] Gill, N. S. (2006). Importance of software component characterization for better software reusability. ACM SIGSOFT Software Engineering Notes, 31(1), 1-3.

[2] Ahmaro, I. Y., bin Mohd Yusoff, M. Z., & Abualkishik, A. M. (2014, September). The current practices of software reusability approaches in Malaysia. In 2014 8th. Malaysian Software Engineering Conference (MySEC) (pp. 172-176). IEEE.

[3] Chen, X., Chen, J., Zhang, S., & Sui, L. (2010, July). Application of adapter pattern in container ship stowage system. In 2010 2nd International Conference on Industrial and Information Systems (Vol. 1, pp. 120-123). IEEE.

[4] Kerji, V. K. (2011, April). Decorator Pattern with XML in web application. In 2011 3rd International Conference on Electronics Computer Technology (Vol. 5, pp. 304-308). IEEE.

[5] Shirazi, M. N., Hejazi, M., & Dolatabadi, H. (2012). Applied Dynamic Chain of Responsibility in Web Application Security. International Journal of Computer Theory and Engineering, 4(6), 917.

[6] Yueping, Z., Yuefan, L., & Kesheng, X. (2009, December). The Compound Pattern on the Chain of Responsibility and Observer. In 2009 International Forum on Computer Science-Technology and Applications (Vol. 3, pp. 420-422). IEEE.

[7] Gill, N. S. (2006). Importance of software component characterization for better software reusability. ACM SIGSOFT Software Engineering Notes, 31(1), 1-3.

[8] Masoumi, S., & Mahjur, A. (2019). Reusable and interactive classes: a new way of object composition. Turkish Journal of Electrical Engineering & Computer Sciences, 27(5), 3615-3632.

[9] Patil, S., Drozdov, D., & Vyatkin, V. (2018, July). Adapting software design patterns to develop reusable IEC 61499 function block applications. In 2018 IEEE 16th International Conference on Industrial Informatics (INDIN) (pp. 725-732). IEEE.

[10] He, G. L., Wu, S., & Yao, J. P. (2013, April). Application of design pattern in the JDBC programming. In 2013 8th International Conference on Computer Science & Education (pp. 1037-1040). IEEE.

[11] Harrer, A., Pinkwart, N., McLaren, B. M., & Scheuer, O. (2008). The Scalable Adapter design pattern: Enabling interoperability between educational software tools. IEEE Transactions on Learning Technologies, 1(2), 131-143.

[12] He, Y., Wang, H., & Wang, L. (2015, December). Design and implementation of radar signal processing system based on design patterns. In 2015 8th International Symposium on Computational Intelligence and Design (ISCID) (Vol. 1, pp. 85-88). IEEE.

[13] Dai, W., & Vyatkin, V. (2013, November). A component-based design pattern for improving reusability of automation programs. In IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society (pp. 4328-4333). IEEE.

[14] Roselló, E. G., Lado, M. J., Méndez, A. J., Dacosta, J. G., & Cota, M. P. (2007). A component framework for reusing a proprietary computer-aided engineering environment. Advances in Engineering Software, 38(4), 256-266.

[15] Al Dallal, J., & Sorenson, P. (2005). Reusing class-based test cases for testing object-oriented framework interface classes. Journal of Software Maintenance and Evolution: Research and Practice, 17(3), 169-196.

[16] Chan, S. M., & Lammers, T. L. (1998, June). Reusing a distributed object domain framework. In Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203) (pp. 216-223). IEEE.

[17] Xin, T., & Yang, L. (2017, June). A framework of software reusing engineering management. In 2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA) (pp. 277-282). IEEE.

[18] Pelechano, V. (2011, September). Systematic reuse of web services through software product line engineering. In 2011 IEEE Ninth European Conference on Web Services (pp. 192-199). IEEE.

[19] Dikel, D., Kane, D., Ornburn, S., Loftus, W., & Wilson, J. (1997). Applying software product-line architecture. Computer, 30(8), 49-55.

[20] Griss, M. L. (2000, June). Implementing product-line features with component reuse. In International Conference on Software Reuse (pp. 137-152). Springer, Berlin, Heidelberg.

[21] Krueger, C. W. (2006, August). New methods in software product line development. In 10th International Software Product Line Conference (SPLC'06) (pp. 95-99). IEEE.

[22] Ramler, R., & Putschögl, W. (2013, March). Reusing automated regression tests for multiple variants of a software product line. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops (pp. 122-123). IEEE.

[23] Lee, J., Muthig, D., & Naab, M. (2010). A feature-oriented approach for developing reusable product line assets of service-based systems. Journal of Systems and Software, 83(7), 1123-1136.

[24] Istoan, P., Nain, G., Perrouin, G., & Jézéquel, J. M. (2009, October). Dynamic software product lines for service-based systems. In 2009 Ninth IEEE International Conference on Computer and Information Technology (Vol. 2, pp. 193-198). IEEE.

[25] Zhu, H. (2005, August). Building reusable components with service-oriented architectures. In IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005. (pp. 96-101). IEEE.

[26] Shatnawi, A., Seriai, A., Sahraoui, H., Ziadi, T., & Seriai, A. (2020). ReSIde: Reusable service identification from software families. Journal of Systems and Software, 170, 110748.

[27] Granell, C., Díaz, L., & Gould, M. (2010). Service-oriented applications for environmental models: Reusable geospatial services. Environmental Modelling & Software, 25(2), 182-198.

[28] Wang, J., Yu, J., & Han, Y. (2005, October). A service modeling approach with business-level reusability and extensibility. In IEEE International Workshop on Service-Oriented System Engineering (SOSE'05) (pp. 23-28). IEEE.

[29] Baillarguet, C., & Piumarta, I. (1999). An highly-configurable, modular system architecture for mobility, interoperability, specialisation and reuse. In Proceedings of the ECOOP'99 Workshop on Object-Orientation and Operating Systems.

[30] Drechsler, R., Drechsler, N., Mackensen, E., Schubert, T., & Becker, B. (2000, September). Design reuse by modularity: A scalable dynamical (re) configurable multiprocessor system. In Proceedings of the 26th Euromicro Conference. EUROMICRO 2000. Informatics: Inventing the Future (Vol. 1, pp. 425-431). IEEE.

[31] Cannon, D. M. (1997). U.S. Patent No. 5,644,698. Washington, DC: U.S. Patent and Trademark Office.

[32] DeBusk, B. C., Cofer, M. C., Shanks, M. W., & Lukens, W. F. (1999). U.S. Patent No. 5,995,937. Washington, DC: U.S. Patent and Trademark Office.

[33] Wang, J., Beu, J., Yalamanchili, S., & Conte, T. (2012, November). Designing configurable, modifiable and reusable components for simulation of multicore systems. In 2012 SC Companion: High Performance Computing, Networking Storage and Analysis (pp. 472-476). IEEE.

[34] Sneed, H. M. (2006, February). Wrapping legacy software for reuse in a SOA. In Multikonferenz Wirtschaftsinformatik (Vol. 2, pp. 345-360).

[35] Mishra, S. K., Kushwaha, D. S., & Misra, A. K. (2009). Creating Reusable Software Component from Object-Oriented Legacy System through Reverse Engineering. J. Object Technol., 8(5), 133-152.

[36] Thiran, P., & Hainaut, J. L. (2001, October). Wrapper development for legacy data reuse. In Proceedings Eighth Working Conference on Reverse Engineering (pp. 198-207). IEEE.

[37] Sneed, H. M. (2001, October). Wrapping legacy COBOL programs behind an XML-interface. In Proceedings Eighth Working Conference on Reverse Engineering (pp. 189-197). IEEE.

[38] Masoumi, S., & Mahjur, A. (2019). Reusable and interactive classes: a new way of object composition. Turkish Journal of Electrical Engineering & Computer Sciences, 27(5), 3615-3632.

[39] Kaur, P. J., Kaushal, S., Sangaiah, A. K., & Piccialli, F. (2018). A framework for assessing reusability using package cohesion measure in aspect oriented systems. International Journal of Parallel Programming, 46(3), 543-564.

[40] AlSobeh, A. M., AlShattnawi, S., Jarrah, A., & Hammad, M. M. (2020). Weavesim: a Scalable and Reusable Cloud Simulation Framework Leveraging Aspect-Oriented Programming. Jordanian Journal of Computers and Information Technology (JJCIT), 6(02).

[41] Akşit, M., Tekinerdoğan, B., & Bergmans, L. (2001). The six concerns for separation of concerns. In Proceedings of ECOOP.

[42] Juhár, J., & Vokorokos, L. (2015, November). Separation of concerns and concern granularity in source code. In 2015 IEEE 13th international scientific conference on informatics (pp. 139-144). IEEE.

[43] Panunzio, M., & Vardanega, T. (2014). A component-based process with separation of concerns for the development of embedded real-time software systems. Journal of Systems and Software, 96, 105-121.

[44] Hürsch, W. L., & Lopes, C. V. (1995). Separation of concerns.

[45] Kim, Y. S., & Lee, S. H. (2008, October). Separation of Concerns Security Model for Extension of Component Reuse. In 2008 International Symposium on Ubiquitous Multimedia Computing (pp. 23-28). IEEE.

[46] Ober, I., & Ober, I. (2017). On Patterns of Multi-domain Interaction for Scientific Software Development focused on Separation of Concerns. Procedia computer science, 108, 2298-2302.

[47] Kulkarni, V., & Reddy, S. (2003). Separation of concerns in model-driven development. IEEE software, 20(5), 64-69.

[48] Zelinskyy, A., Tesliuk, V., & Maloid, T. (2016, April). Applying decorator design pattern for MEMS program model creation. In 2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 19-21). IEEE.

[49] Anguswamy, R., & Frakes, W. B. (2012, September). A study of reusability, complexity, and reuse design principles. In Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 161-164). IEEE.

[50] Arslan, S., & Kardas, G. (2020). DSML4DT: A domain-specific modeling language for device tree software. Computers in Industry, 115, 103179.

[51] İçöz, B. & Kalıpsız, O. (2020). MODEL GÜDÜMLÜ YAZILIM GELİŞTİRME YAKLAŞIMI KULLANILARAK MİKRO SERVİS GELİŞTİRİLMESİ . Mühendislik Bilimleri ve Tasarım Dergisi , Special Issue: International Conference on Artificial Intelligence and Applied Mathematics in Engineering (ICAIAME 2020) , 142-148

[52] Frakes, W. B., & Fox, C. J. (1996). Quality improvement using a software reuse failure modes model. IEEE Transactions on Software Engineering, 22(4), 274-279.

[53] Prieto-Diaz, R. (1993). Status report: Software reusability. IEEE software, 10(3), 61-66.

[54] Sandhu, P. S., Kakkar, P., & Sharma, S. (2010, September). A survey on Software Reusability. In 2010 International Conference on Mechanical and Electrical Technology (pp. 769-773). IEEE.

[55] Kasikci, B. C., & Bilgen, S. (2009). Etkin Yeniden Kullanım: Yazılım Ürün Hatlarında Değişkenliğin Modellenmesi. Ulusal Yazılım Mühendisliği Sempozyumu.

[56] Alexander, C., Ishikawa, S., Silverstein, M. (1979). The Timeless Way of Building, Oxford University Press.

[57] Kuchana, P. (2004). Software architecture design patterns in Java. CRC Press.

[58] Radonjic, V. D., Bashardoust, S., Corriveau, J. P., & Arnold, D. (2011). Design Patterns-A Modeling Challenge. In Proceedings of the International Conference on Software Engineering Research and Practice (SERP) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[59] Odell, D. (2014). Design Patterns: Creational. In Pro JavaScript Development (pp. 119-135). Apress, Berkeley, CA.

[60] Lasater, C. G. (2006). Design patterns. Jones & Bartlett Publishers.

[61] Sarcar, V., & Metsker, S. J. (2004). Design Patterns in C#. Addison-Wesley Professional.

[62] Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). Elements of Reusable Object-Oriented Software. Design Patterns. massachusetts: Addison-Wesley Publishing Company.

[63] Ahmaro, I. Y., Abualkishik, A. M., & Yusoff, M. Z. M. (2014). Taxonomy, definition, approaches, benefits, reusability levels, factors and adaption of software reusability: a review of the research literature. Journal of Applied Sciences, 14(20), 2396.

[64] Poulin, J. S. (1994, November). Measuring software reusability. In Proceedings of 1994 3rd International Conference on Software Reuse (pp. 126-138). IEEE.

[65] Ampatzoglou, A., Kritikos, A., Kakarontzas, G., & Stamelos, I. (2011). An empirical investigation on the reusability of design patterns and software packages. Journal of Systems and Software, 84(12), 2265-2283.

[66] Calp, M. H., & ARICI, N. (2011). Nesne yönelimli tasarım metrikleri ve kalite özellikleriyle ilişkisi. Politeknik Dergisi, 14(1), 9-14.

[67] Jamali, S. M. (2006). Object oriented metrics. A survey approach Technical report, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

[68] Cheikhi, L., Al-Qutaish, R. E., Idri, A., & Sellami, A. (2014). Chidamber and kemerer object-oriented measures: Analysis of their design from the metrology perspective. International Journal of Software Engineering and Its Applications, 8(2), 359-374.

[69] Thwin, M. M. T., & Quah, T. S. (2005). Application of neural networks for software quality prediction using object-oriented metrics. Journal of systems and software, 76(2), 147-156.

[70] Gill, N. S., & Sikka, S. (2011). Inheritance hierarchy based reuse & reusability metrics in oosd. International Journal on Computer Science and Engineering, 3(6), 2300-2309.

[71] Neelamegam, C., & Punithavalli, M. (2009). A survey-object oriented quality metrics. Global Journal of Computer Science and Technology, 9(4), 183-186.

[72]Aggarwal, M., Verma, V. K., & Mishra, H. V. (2013). An Analytical Study of Object-Oriented Metrics (A Survey). International Journal of Engineering Trends and Technology (IJETT), 6(2), 76-83.

[73] Dadhania S., Galathiya S. , (2014). Survey on Object Oriented Matrices. International Journal of Engineering Research & Technology (IJERT) , 3(3), 1432-1437.

[74] Senthil, V. S. (2019, April). Quantitative Assessment of Inheritance Hierarchies for Aspect Oriented Software Development using a proposed Aspect Inheritance Reusability Model. In 2019 International Conference on Automation, Computational and Technology Management (ICACTM) (pp. 573-576). IEEE.

[75] Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6), 476-493.