Tasarım Kalıplarını Öğretme Stratejileri: Seminer ya da Uygulama

Mehmet KAYA

Elektrik-Elektronik Mühendisliği Bölümü, Adıyaman Üniversitesi, Adıyaman, Türkiye
m.kaya@adiyaman.edu.tr
(Geliş/Received: 15.01.2015; Kabul/Accepted: 17.02.2016)
DOI: 10.17671/btd.34872

Özet—Bilgisayar bilimlerinin bazı temel alan derslerinin etkili bir şekilde öğretilmesi oldukça zordur. Bu derslere Programlamaya Giriş dersleri örnek gösterilebilir. Karşılaşılan bu zorluğun başta gelen ve en önemli sebebi ise bu derslerin içeriğinin soyut yapısı ve dersi alan öğrencilerin bu soyut düşünme yetilerinin henüz gelişmemiş olmasıdır. Genellikle yüksek lisans ve doktora seviyesinde verilen Tasarım Kalıpları dersi içeriğinde de aynı şeyi gözlemlemek kaçınılmazdır. Başta karmaşık gibi görünen ve temelde sınıflar ve nesneler arasındaki yapısal veya davranışsal ilişkilere dayanan bu Tasarım Kalıpları, özellikle yeteri seviyede nesne tabanlı tasarım altyapısına sahip olmayan öğrencileri oldukça zorlamaktadır. Dünyanın hemen her ülkesinde üniversite ve araştırma kuruluşlarında bu dersin anlatımında aynı yöntem kullanılmaktadır. Genellikle seminer şeklinde verilen bu derste, Gamma ve ekibi tarafından yazılan ve çok yaygın etki yaratan Tasarım Kalıpları kitabındaki içerik ve sunum şekli kullanılmaktadır. Bu dersi 2013 yılında ABD'nin New York eyaletindeki Syracuse Üniversitesinde ve 2015 yılında Fırat Üniversitesinin Uluslararası Yüksek Lisans ve Doktora Programında verdiğimiz dönemlerden elde ettiğimiz tecrübelerimize dayanarak, bu çalışmada tasarım kalıplarının uygulamalı proje tasarımı şeklinde nasıl daha etkili öğretilebileceği incelenmektedir. Buna yönelik Factory Method ve Singleton tasarım kalıpları detaylandırılmış ve uygulamalı proje desteği ile bu detayların öğretimine yönelik örnek bir proje taslağı sunulmuştur.

Anahtar Kelimeler—Tasarım kalıpları, Factory method, Singleton, bilgisayar bilimleri eğitimi

Teaching Strategies For Design Patterns: Seminar Or Projects

Abstract—It is very difficult to teach some fundamental concepts and topics effectively in computer science. Introductory level programming courses can be a good example for this. The most important reason behind this difficulty is the abstract structure of the content for these subjects and the fact that students taking these introductory courses may not have developed that abstract thinking ability yet. It is inevitable to experience the same difficulties with teaching Design Patterns which are usually taught as a graduate level software engineering course. These patterns rely on structural and behavioral interactions and dependencies between a number of classes and objects and may seem quite complicated at first glance for those who do not have sufficient object oriented design background and experience. The same teaching strategy has been used in many different countries to teach this particular subject. This course is usually offered in a seminar fashion and the content and presentation structure of the widely renowned text book in this field by Gamma et al. is used. In this paper, we investigate how design patterns can be taught more effectively through a mid-size application development, based on the experience we gathered from teaching this subject in two different universities: Syracuse University, NY, USA and Firat University, Elazig, Turkey in 2013 and 2015 respectively. For this, we provide a detailed discussion on Factory Method and Singleton design patterns together with an example application project design to support teaching these details.

Keywords—Design Patterns, Factory Method, Singleton, Computer Science Curriculum

1.GİRİŞ (INTRODUCTION)

Zaman zaman Tasarım Örüntüleri ve Tasarım Desenleri olarak da adlandırılan Tasarım Kalıpları, Yazılım Mühendisliğinde sıkça tekrar edebilecek problemlerinin çözümüne yönelik yazılım mimarisi tasarımları sunmaktadır. Tasarım Kalıplarını kullanmanın birkaç farklı avantajı vardır. Bunlardan bazıları aşağıdaki şekilde sıralanabilir:

- Tasarım kalıpları, problemlerin çözümünde kullanılacak ortak bir iletişim şekli oluşturur. Başka bir deyişle, problemlere ve bu problemlerin çözümlerine genel olarak kullanılacak isimler verir.
- Tasarım kalıpları tekrar etmesi beklenen durumlara yönelik çözümler sunduğu için, bu durumlarda problemin anlatılması ya da çözümün tartışılmasını kolaylaştırır. Genellikle Yazılım Mühendisliğinde karşılaşılan problemlerin diğer ilgili kişilere izahı zor olabilir. Tasarım kalıplarıyla bu iletişim kolaylaştırılmış olur.
- Aynı şekilde, yazılım dokümantasyonunda etkili bir dil sağlar ve tartışmaların daha etkili olmasını destekler. Böylece Yazılım Mühendisleri arasında tasarım ve çözümlerde kullanılacak ortak bir dil ortaya çıkmış olur.
- Yazılım Tasarımının en önemli kısımlarını kapsar ve bu kısımların mimarisini tanımlar.
- Bir problemin çözümüne yönelik değişik tasarımlar sunar.
- Yazılım Tasarımının anlaşılmasını kolaylaştırır ve böylece hataların tespiti ve yazılımın bakım(maintenance) aşaması da daha az maliyetli bir iş haline gelmiş olur.

Büyük çaplı profesyonel yazılım projelerinde, maintenance denen ve yazılım yaşam çevriminin genellikle son adımı olan kısım, yazılımı geliştirmenin genel maliyetini en çok etkileyen kısımdır [1, 2]. Yazılım içerisinde, yazılımın piyasaya sürümünden sonra tespit edilen hataların düzeltilmesi bu adımda gerçekleştirilen bir işlemdir. Dolaysıyla, anlaşılması güç ve karmaşık bir yazılım tasarımında bu tip hataları tespit edip onarmak daha çok zaman almakta ve yazılım geliştirme maliyetini oldukça etkilemektedir. Yazılım tasarımında, Tasarım Kalıplarının kullanılması tasarımın çoğunlukla daha anlaşılır olmasını sağlamakta ve böylece bu tür hataların tespitini ve onarımını kolaylaştırmaktadır [3]. Yani Tasarım Kalıpları, sadece yazılımın estetik görünüşünü geliştirme amacına hizmet eden bir olgu değil, aksine yazılım geliştirme maliyetini doğrudan etkileyen bir faktördür. Dolaysıyla Tasarım Kalıpları üzerinde belli bir seviyede bilgi ve beceriye sahip olmak, bir Yazılım Mühendisinin sahip olması gereken çok önemli bir konudur. Bunun yanında bilgisayar bilimleri müfredatının aynı kalmak yerine, modern ve yenilikçi bir yaklaşımla yeni dersler ve materyaller içermesinin gerekliliği bazı çalışmalarla ortaya konmuştur [4]. Öyle ki dünyanın önde gelen yazılım firmalarının iş görüşmelerinde veya mülakat sınavlarında adaylara sordukları soruların başında Tasarım Kalıpları ile ilgili sorular görmek mümkündür. Öte yandan tasarım kalıplarının gerekliliği ve yazılım mimarisinde kullanımını desteklemeye yönelik çalışmalar bulunmaktadır[5, 6, 7].Bu çalışmamızda, Yazılım Mühendisliğinde böylesine önemli olan bu konunun, özellikle çokta sağlam bir nesne tabanlı tasarım zemini olmayan öğrencilere öğretilmesinde, genel olarak kullanılan tasarım kalıpları dilinin nasıl uygulamalı projelerle desteklenebileceğini açıklamayı amaçlamaktayız. Bu amaçla, bu makalede Factory Method ve Singleton tasarım kalıplarının önemli detayları açıklanmış ve bunlara yönelik nasıl bir içerik hazırlanabileceği sunulmuştur.

2. İLGİLİ ÇALIŞMALAR(RELATED WORKS)

Bu bölümde tasarım kalıplarıyla ilgili diğer çalışmaların kısa bir özeti verilmiştir. Tasarım kalıpları literatürde her ne kadar değişik yönleriyle ele alınmış olsa da, bunların etkili aktarımına yönelik pek fazla çalışmaya rastlanmamıştır. Bazı çalışmalarla var olan yazılım projelerinde tasarım kalıpları bulmak ya da yazılımın anlaşılabilirliğini artırmak için tasarım kalıpları kullanılabilecek noktaları tespit etmek amaçlanmaktadır. Tsantalis ve ekibi tarafından,benzerlik skoruna (similarity scoring) dayanarak kaynak kodlar içerisinde tasarım kalıbı tespit edilmesine yönelik bir algoritma oluşturulmuştur [8]. İki farklı tasarım kalıbı birlikte kullanılabilir mi? Farklı tasarımlar benzer problem alanlarına hitap ediyor mu? Bu ve benzeri soruları açıklamak için bazı çalışmalarda tasarım kalıpları arasındaki ilişkiler benzerliklerine göre incelenmiştir. Bu ilişkilere göre tasarım kalıpları Zimmer tarafından yeniden gruplanmıştır [9]. Büyük yazılım projelerini anlamak için (software comprehension) tasarım kalıplarına dayalı reverse-engineering yaklaşımları da ortaya atılmıştır [9]. Burada ki amaç, bu tür projeleri anlamak için, yazılımın karmaşık ve anlaşılması güç bir hale gelmeden önce, uygulama esnasında kullanılan tasarım kalıplarının bulunması ve bunlara

Tablo 1. Tasarım Kalıbı Grupları(A Catalog of Design Patterns)

		AMAÇ		
		OLUŞUMSAL	***YAPISAL***	***DAVRANIŞSAL***
HEDEF	**SINIF**	• FactoryMethod	• Adapter	• Interpreter
	NESNE	• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Vistor

Dayanarak bir reverse-engineering yaklaşımı geliştirilmesidir [10, 3]. Buna benzer şekilde, UML diyagramları içerisinde otomatik olarak Tasarım Kalıbı tespit etmeye yönelik çalışmalarda bulunmaktadır [11, 12]. Makine Öğrenmesi de verilen bir sistem içerisinde Tasarım Kalıbı tespit etmek için kullanılmıştır [13].

Bu makalemizde ki çalışmaya benzer uygulamalı proje tabanlı bir yaklaşım, Astrachan ve ekibi tarafından Command, Proxy, Composite, and Mediator tasarım kalıplarını öğretmeye yönelik olarak verilmiştir [14]. Ayrıca bu çalışmada da tasarım kalıplarının bilgisayar bilimleri müfredatının önemli ve vazgeçilmez bir konusu olduğu ifade edilmekle birlikte yeterli tecrübe olmadan tasarım kalıplarının kullanılmasının zorluğu vurgulanmaktadır.

Bilgisayar Bilimlerinin bazı derslerinin ya da konularının etkili öğretiminde ifade ettiğimiz zorlukların üstesinden gelmek için, uygulamalı proje tabanlı yaklaşımlar daha öncede benimsenmiştir [15, 16]. Bu çalışmalarda da öğretilmesi soyut yapısından dolayı zor olan farklı konular için, bu çalışmamızdakine benzer bir şekilde, öğrencilerden verilen bir uygulamayı inceleyip istenen şekilde tamamlamaları istenmektedir.Bazı çalışmalarda ise, programlama gibi konuların öğretilmesinin zorluklarından bahsedilmiştir ve bir programlama kalıbı kullanarak öğrenme yoluna gidilmiştir [17].

Bunun yanında, Application Programming Interface (API) geliştirme gibi bazı özel durumlarda, bazı tasarım kalıplarının kullanımından kaçınılması gerektiğini vurgulayan çalışmalar yapılmıştır [18].

3. TASARIM KALIPLARI KATALOĞU VE DİLİ(LANGUAGE AND CATALOG OF DESIGN PATTERNS)

Desen Dili (Pattern Language) ilk olarak 1977 yılında Christopher Alexander tarafından herhangi bir alandaki iyi bir tasarımı etkili bir şekilde ifade etmek için kullanılmıştır [19, 20]. Daha sonra Gamma ve ekibinin "Design Patterns: Elements of Reusable Object-Oriented Software" adlı kitabı ile tasarım kalıpları Yazılım Mühendisliğinde yaygın olarak kullanılmaya başlanmıştır [21]. Bu kitapla Tasarım Kalıpları, Yazılım Mühendisliğinin önemli bir konusu haline gelmiş ve bu kalıpların izahına yönelik ortak bir dil ortaya çıkmıştır. Bunu takiben bu alan aktif bir araştırma konusu haline gelmiş ve çeşitli konferanslar düzenlenmeye başlanmıştır [22]. Bu konferansların başında her yıl yenisi düzenlene "Pattern Languages of Programs (PLoP) Conferences" adlı konferans gelmektedir.

Bahsedilen kitabı incelediğimizde görüyoruz ki Tasarım Kalıpları tasarımın amacına yönelik, Creational (Oluşumsal), Structural (Yapısal) ve Behavioral (Davranışsal) olmak üzere temelde üç gruba ayrılmaktadır. Ayrıca tasarımın ilgilendiği hedef noktası olaraksa bunları sınıf ve nesne olmak üzere ikiye ayırabiliriz. Tablo 1'de Tasarım Kalıplarının bir katalog halinde gruplandırılmış şeklini görmekteyiz.

Oluşumsal Tasarım Kalıpları nesne oluşturma işlemlerinden kaynaklanan karmaşıklıkları gidermek ve yazılımı daha anlaşılır ve yönetilebilir kılmak için, nesne oluşturma işlemlerini belli bir sınıfa atamak üzere tasarım çözümleri sunar. Böylece yeni bir nesne oluşturulurken, Client denen istemci sistem bu nesneyi kendi oluşturmak yerine bu amaç için görevlendirilen sınıftan istediği nesneyi talep eder. Bunun en büyük

avantajı, bu istemci sistemin Oluşumsal Tasarım Kalıpları kullanılarak, objesi oluşturulan sınıflardan soyutlanmasıdır. Yani aslında bu kalıplar Coupling (sistem içerisindeki bağımlılıklar) denen yazılım kalite metriğini olumlu yönde etkilemektedir.

Yapısal Tasarım Kalıpları sınıf ve nesnelerin belli bir tasarım ortaya çıkarmak üzere nasıl birlikte kullanıldığı ve nasıl bir bağımlılık gösterdikleri ile ilgilidir. Burada önemli olan, nesne tabanlı tasarımda kullanılan sınıflar arasındaki ilişkilerin iyi anlaşılmasıdır. Çünkü özünde bu tasarım kalıpları Inhertiance, Aggregation, Composition ya da Use gibi temel sınıflar arası ilişkilere dayanır.

Davranışsal Tasarım Kalıpları daha çok bir işin nasıl yapıldığı ya da algoritması ve bir tasarım problemini çözmek için bu işin nesneler arasında nasıl paylaşıldığı ile ilgilidir. Bu kalıplar içinse sınıflar arası ilişkilerin yanı sıra, nesnelerin bir biri ile nasıl iletişim kurduğu, yani aslında bir manada işin nesneler arasında bir akış şeması gibi nasıl gerçekleştirildiği önemlidir.

Daha öncede bahsettiğimiz gibi bu kitap belli bir sunum şeklini kullanmaktadır ve bu sunum genel olarak Tasarım Kalıplarının aktarılmasında geniş kabul görmüştür. Her kalıp aşağıdaki yönleriyle bu kitapta tartışılmaktadır. Tasarım Kalıbının:

- İsmi ve Sınıfı
- Amacı
- Bilinen Diğer Adları
- Motivasyon Örnekleri
- Uygulama Alanları/Uygulanma Durumları
- Mimari Yapısı
- Mimarinin Parçaları
- Parçalar Arası İlişkiler
- Sonuçlar
- Uygulama
- Örnek Kod
- Bilinen Kullanımları
- İlgili Diğer Kalıplar

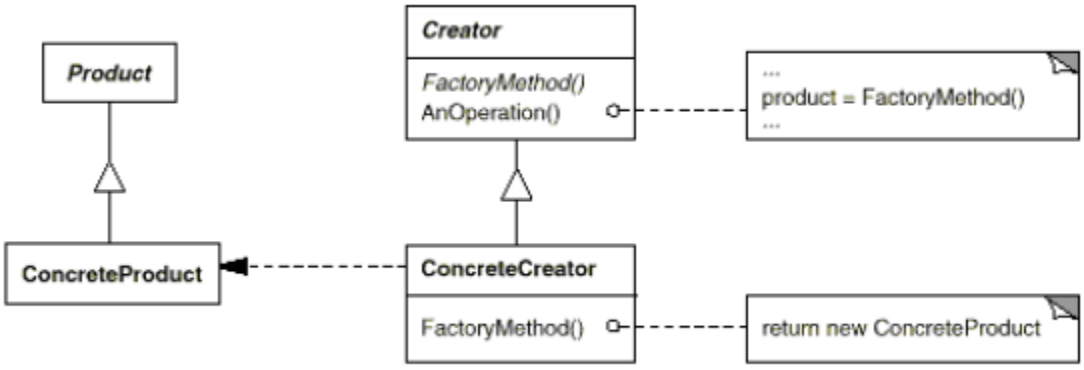
Tecrübelerimizle gözlemlediğimiz, bir Tasarım Kalıbının bu içeriğe göre bir ders müfredatı olarak etkili bir şekilde verilmesini engelleyen bazı etmenler vardır. Kitap içeriği incelendiğinde görmekteyiz ki Motivasyon kısmında verilen örnekler öğrenciler için çok üst düzey kalmaktadır. Özellikle öğrencilerin yeterince nesne tabanlı programlama ve endüstriyel boyutta yazılım geliştirme tecrübesi olmayınca burada verilen örnekler tasarım kalıbının anlaşılmasında ya da öğrencilerin motive edilmesinde çok etkisiz kalmaktadır. Mimari yapı kısmında kullanılan UML diyagramlarını okumak da aynı şekilde tecrübe gerektirmektedir. Her ne kadar, bu ders için bir Nesne Tabanlı Programla dersi ön koşul olarak istense de, öğrencilerin bu yazılım mimarisini anlamadıkları yada bu yapı içerisindeki nesneler yada sınıflar arasındaki ilişkileri yorumlamakta çok güçlük yaşadıkları gözlemlenmiştir. Bunun yanında bazı tasarımların öğretilmesinin özellikle daha güç olduğunu belirten çalışmalarda mevcuttur [14].

Örnek Kod kısmında verilen örnekler genellikle bu kalıpları en minimal seviyede oluşturmak ve tasvir etmek için tasarlanmıştır. Dolaysıyla öğrenciler bu örnek kodları inceleyip başarılı bir şekilde çalıştırsalar dahi gerçek anlamda kullanılan tasarım kalıbının rolünü yada kullanım amacını daha büyük projeler içerisinde anlayamamaktadır. Ayrıca Tasarım Kalıplarının kitapta verilen bilinen kullanımları da, bahsedilen projeler hakkında bilgi ve tecrübesi olmayanlar için çok açıklayıcı olmamaktadır.

Dolaysıyla bu çalışmamızda, Factory Method ve Singleton tasarım kalıplarının öğretilmesine yönelik örnek bir uygulama projesi verilmektedir. Bu tip projelerle öğrencilerden farklı kalıpları birlikte kullanmaları istenerek daha etkili bir öğrenme ortamı oluşturulabileceği düşünülmektedir.

4. FACTORY METHOD TASARIM KALIBI(FACTORY METHOD DESIGN PATTERN)

Factory Method oluşumsal tasarım kalıplarının en yaygın olarak kullanılanlarındandır [18]. Bu kalıpta, çeşitli nesnelerin oluşturulma görevi bu Factory Method denen yapıya verilir. Adındanda anlaşılacağı gibi bu yapı genellikle bir fonksiyon ya da metottur ve hangi türden bir nesne istendiğinibelirten bir parametre alır. Böylece Client yeni bir nesneye ihtiyaç duyduğu zaman, bunu nesnesini istediği sınıfı kullanarak kendisi oluşturmak yerine, Factory Method'dan isteyerek elde eder. Bu hem zaman zaman çok karmaşık olabilecek nesne oluşturma işini bir noktada sınırlandırılmış olur (Factory Method fonksiyonu içinde), hem de Client, nesnesi oluşturulacak olan sınıflardan soyutlanmış olur. Artık Client'ın yapması gereken sadece Factory Method'dan bir nesne üretmesini istemektir. Bunun dışında hangi nesnenin oluşturulduğu veya nasıl oluşturulduğu sadece Factory Method tarafından idare edilir.



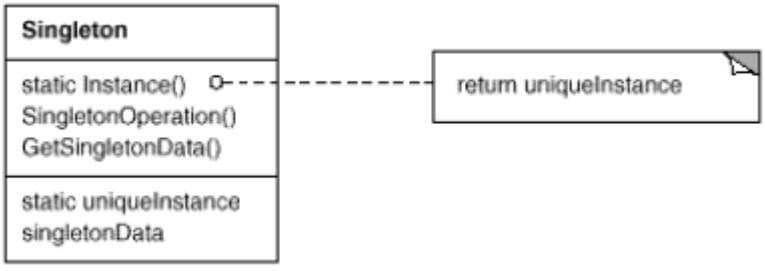
Şekil 1 Factory Method Kalıbının Yapısı(Structure of Factory Method Design Pattern)

Şekil 1'de Factory Method kalıbının yapısı gösterilmektedir [21]. Görüldüğü gibi Factory Method fonksiyonunu içeren sınıf soyut bir yapı olarak tanımlanıp tanımlanan somut sınıflar içerisinde değişik Factory Method fonksiyonları elde etmek de mümkündür. Burada şunu da ifade etmeliyiz ki büyük çaplı yazılım tasarımlarında genellikle birden fazla kalıp bir arada kullanılır.

5. SINGLETON TASARIM KALIBI(SINGLETON DESIGN PATTERN)

Bu tasarım kalıbının kullanım amacı belli bir sınıfın her zaman sadece bir nesnesi olduğunu kontrol altına almaktır. Buda Factory Method gibi oluşumsal bir kalıptır ki nesnelerin gerektiği yada talep edildiği zaman oluşturulmasını (sadece bir kere) ve bu nesnelere tek bir erişim noktası sağlanmasını amaçlar.

Bu kalıpla istenen fonksiyonellik aslında global bir değişken ile de elde edilebilir ama genel olarak bilinen bir çok nedenden dolayı özellikle nesne tabanlı programlamada global değişken kullanmaktan kaçınmak gerekir. Ayrıca eğer bir nesnenin oluşturulması maliyetli bir işse (computationally expensive) her defasında yeni bir nesne oluşturmak yerine daha önceden oluşturulmuş ve bir yerde saklanan nesneyi kullanarak işlem yapmak, devam eden bölümlerde açıklandığı gibi, ciddi performans kazanımlarıda sağlayacaktır. Bu yönüyle Singleton, Flywwight kalıbına çok benzemektedir. Flyweight kalıbında, eğer bir nesne olduğu gibi tekrar kullanılamıyorsa, baştan yenisini oluşturmak yerine, nesneler arasındaki ortak özellikleri barındıran nesneler oluşturup farklılık gösteren özellikleri istendiği zaman değiştirerek kullanıp yeni nesne oluşturmanın önüne geçilmiş olur. Singleton kalıbının Şekil 2'de gösterildiği gibi çok basit bir yapısı vardır [21].



Şekil 2 Singleton Kalıbının Yapısı(Structure of Factory Method Design Pattern)

6. TASARIM KALIPLARININ ÖĞRETİLMESİ: GÜÇLÜKLER(DIFFICULTIES WITH TEACHING DESIGN PATTERNS)

Bu çalışmada anlatılan proje, tespitler, çıkarımlar ve sonuçlar Syracuse Üniversitesi ve Fırat Üniversitesinde Tasarım Kalıpları dersinin verilmesinden elde edilen tecrübelerimize dayanmaktadır. Özellikle Syracuse Üniversitesindeki öğrenci profiline bakıldığında, bahsedilen içeriğin olduğu şekliyle bir seminer dersi olarak verilmesi yeterli olmaktadır. Çünkü buradaki öğrencilerin dünyanın önde gelen teknoloji şirketlerinde çalışıyor olma ya da çalışabilme tecrübesi vardır. Öyle ki bu dersi alan bazı öğrenciler kısmi-zamanlı öğrenci statüsünde çalıştıkları teknoloji firmasının desteği ile eğitimlerine devam etmiştir. Aynı şekilde öğrencilerin büyük kısmı mezuniyetlerinin ardından aralarında Microsoft'un da bulunduğu bu tip şirketler tarafından istihdam edilmiştir. Buradaki tecrübemiz göstermiştir ki bu öğrenciler genellikle verilen kalıpları kolaylıkla kavrayabilmekte ya da bunları gerçek projeler de uygulayabilmektedirler

Öte yandan aynı ders ve müfredatı Fırat Üniversitesinin Uluslararası Yazılım Mühendisliği Programında verdiğimiz dönemden elde ettiğimiz tecrübe ile ise yukarıdaki durumun tersi gözlenmiştir. Yani buradaki öğrenci profili göz önüne alınınca öğrencilerin bu dersi seminer şeklinde etkili öğrenemediği tespit edilmiştir. Çünkü bu öğrencilerin kendi ülkelerinde aldıkları Bilgisayar Bilimleri Lisans müfredatı muhtemelen Tasarım Kalıplarını Yüksek Lisans ve Doktora dersi olarak etkili bir şekilde öğrenmeyi desteklemekte yetersiz kalmıştır. Dolaysıyla bu öğrenciler için uygulamalı proje desteği ile öğrenmenin gerekliliği ortaya çıkmıştır. Ayrıca bu öğrencilerin hiç birinde endüstriyel çapta yazılım geliştirme tecrübesi gözlenmemiştir.

Bu iki öğrenci profili arasındaki diğer bir fark ise İngilizce dilindeki yeterliliktir. Seminer şeklinde verilen bu ders ileri seviye İngilizce yeterliliği gerektirmektedir ki bu yukarıdaki iki durum arasında farklılık göstermektedir.

Bu durum göz önüne alınınca Tasarım Kalıpları ve benzeri derslerin öğretilmesinde uygulamalı proje yoluyla öğretimin özellikle bazı durumlarda çok daha etkili olduğu tespit edilmiştir. Bu amaçla Factory Method ve Singleton kalıplarını öğretmek amacı ile öğrencilerden Tablo 2'de bazı koşulları verilen ve kısmen tamamlanmış olan projeyi istenen tasarım kalıplarını uygulayarak tamamlamaları istenmiştir.

7. ÖRNEK TASARIM(EXAMPLE DESIGN)

Öğrencilerden aşağıda proje ifadesinin bir kısmı verilen ve kısmen tamamlanan proje üzerinde çalışmaları ve istenen bu iki tasarım kalıbını verilen projenin tasarımında kullanmaları istenmiştir. Bu proje ile tasarım kalıplarının yanı sıra temel nesne tabanlı programlama prensiplerinin de öğretilmesi amaçlanmıştır. Bu projeyi istenen şekilde tamamlamak için öğrencilerin, abstract class, interface, base class, derived class, aggregation, composition ve inheritance gibi nesne tabanlı tasarımda kullanılan ifadelerin ne anlama geldiğini, ne zaman ve ne amaçla kullanıldığını öğrenmeleri gereklidir. Proje ifadesinin bir kısmı aşağıda verilmiştir.

Hazırladığımız bu proje ifadesi ile öğrencilerden kısaca, Command Line Argument olarak verilecek olan bir metni, istenen bir karakteri kullanarak ve istenen boyutta ekrana çizecek ve bir hedef dosyaya yazacak olan bir yazılım projesi tasarlamaları ve uygulamaları istenmiştir. Ayrıca öğrencilerin bu tasarımda Factory Method ve Singleton Tasarım Kalıplarını kullanmaları zorunlu kılınmıştır. Bunun yanında Singleton tasarım kalıbıyla nasıl ve hangi durumda bu projede bir performans kazancı sağlanabileceğinin araştırılması istenmiştir.

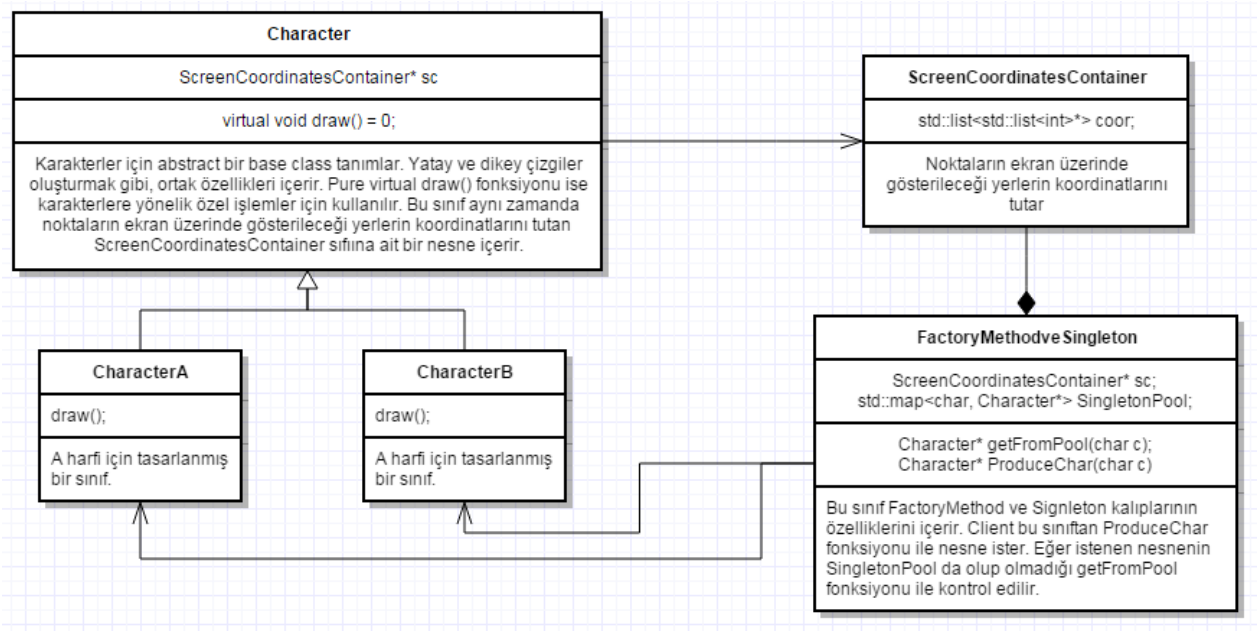
Proje ifadesinden de anlaşılacağı gibi bu proje bir öğrenci için orta ölçekli sayılabilecek bir projedir. Yani öğrencilerden bu projeyi istenen koşulları sağlayacak şekilde tamamlayanların hem nesne tabanlı tasarım ilkelerini hem de bahsedilen iki tasarım kalıbını iyi bir şekilde kavramış oldukları söylenebilir. Şekil 3'tebu projenin örnek bir tasarımı gösterilmiştir.

Şekil 3'teki tasarımdan da anlaşılacağı gibi Factory Method ve Singleton sınıfına Factory Method ve Singleton kalıplarının sorumlulukları yüklenmiştir. Böylece ProduceChar(char c) factory metodu istenen nesneyi oluşturmadan önce bunun Singleton havuzunda olup olmadığını getFromPool(char c) metodunu kullanarak kontrol eder. Böylece giriş metnindeki aynı karakterler için her defasında yeni bir nesne oluşturmak yerine bu havuzdaki daha önceden oluşturulmuş nesneler kullanılır. Öğrencilerden istenen performans testi bu nokta ile ilgilidir. Yani eğer bir karakter nesnesi oluşturmak maliyetli bir iş olsaydı giriş metnindeki her karakter için bu nesneleri oluşturmak genel maliyeti ya da performansı oldukça etkileyecekti.

Table 2. Proje İfadesi(Project Statment)

SENG536 Software Metrics/Design Patterns *Güz 2016*

Proje #1: Karakter Görselleştirme Son Teslim Tarihi: 8 Ekim Pazar
Versiyon 1.1
Amaç:
Bu projenin amacı nesne tabanlı programlamada kullanılan bazı temel tasarım kavramlarını ve sınıflar arası ilişkileri sizlere öğretmektir. Aşağıda daha detaylı açıklandığı gibi, projenizin temel amacı verilen bir metni Konsol Ekranına çizmek ve aynı zamanda bir dosyaya yazmaktır. Factory Method ve Singleton tasarım kalıplarını proje dizaynınıza entegre etmeniz beklenmektedir.Bahsedilen bu iki kalıbın detayları derste kullandığımız "Design Patterns-Elements of Reusable Object-Oriented Software" adlı kitapta yer almaktadır. Projenizde aynı zamanda modüler bir yapı oluşturmanız beklenmektedir. Bu modüler yapıyı elde etmek için, karakterler arasındaki (yatay ve dikey çizgi oluşturmak gibi) ortak fonksiyonları tanımlayan bir karakter temel sınıfı oluşturup, daha sonra her karakter için oluşturulan bu temel sınıftan türeyen ve karakterlere özel fonksiyonları (yatay ve dikey çizgilerin bu karakter için kaçkere ve hangi noktalarda oluşturulması gerektiği gibi) tanımlayan somut bir sınıf oluşturabilirsiniz.
İstenen Şartlar:
Karakter Görselleştirme Projeniz:
1. C++ standart kütüphanesini kullanmalı ve Visual Studio 2013 kullanarak derlenebilmelidir. Visual Studio 2013 verilen linkten elde edilebilir ve proje bu versiyonda oluşturulabilir.
2. Bütün kullanıcı konsoluna giriş ve konsoldan çıkış işlemleri için C++ std::iostream kütüphanesinin servisleri kullanılmalıdır. Ayrıca dinamik hafıza yönetimi için new ve delete C++ operatörleri kullanılmalıdır.
3. Command Line argümanı olarak bir metin belirtilmeli ve karakterlerin çizilmesi istenen yükseklik ve genişliğini gösteren beşten büyük bir sayı girilmelidir. Projeniz daha sonra belirtilen bu metni her bir karakteri istenen boyutta çizecek şekilde ve istenen herhangi bir karakteri çizgileri oluşturmak için kullanarak ekranda görselleştirmelidir.
4. Command Line argümanı olarak çizgilerin oluşturulmasında kullanılacak spesifik bir karakter belirtilmelidir. Ayrıca çizimde kullanılacak bir renk de Command Line argümanı olarak girilmelidir. Eğer hiç bir karakter belirtilmemişse Projeniz çizimlerde kullanılacak rastgele bir renk seçmelidir.
5. Karakter nesnelerini oluşturmak için Factory Method kalıbı kullanılmalıdır. Bütün spesifik karakter sınıfları tek bir temel sınıftan türetilmelidir.
6. Nesne oluşturma işlemi için Singleton kalıbı kullanılmalıdır. Ayrıca uzun giriş metinleri için Singleton kalıbını kullanmanın performansı nasıl etkilediğini inceleyip açıklamanız beklenmektedir.
7. Oluşturulan çizim ayrıca bir dosyaya yazılmalıdır.



Şekil 3 Örnek Proje Tasarımı(Example Project Design)

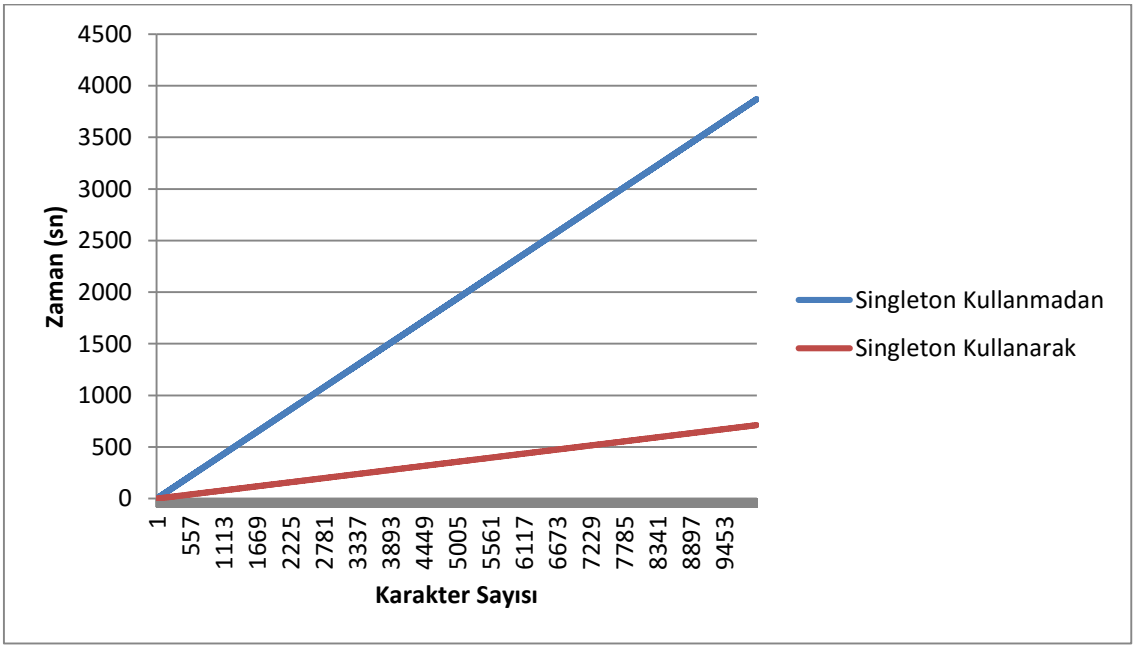
```
Character(ScreenCoordinatesContainer* _sc, intlength = SIZE) :sc(_sc){
		int j = 0;
		for (size_t i = 0; i < 10000; i++)
		{
			for (size_t k = 0; k < 10000; k++)
			{
				j++;
			}
		}
	}
```

Şekil 4. Character Sınıfı Kurucu Üyesi (Constructor for Character Class)

Bu durumun demonstrasyonunu yapmak için Character adlı taban sınıfının kurucu üyesine, yeni nesne oluşturmanın maliyetini artıracak şekilde bir döngü eklenmiştir. Bu kurucu üye Şekil 4'te gösterilmiştir.

Dolaysıyla bu kurucu üye ile birlikte, Singleton tasarım kalıbını kullanmadan her karakter için yeni bir nesne oluşturulması demek, örneğin eğer giriş metninde 100 tane 'a' harfi mevcut ise, maliyeti yüksek olan bu işlemi 100 defa çalıştırmak anlamına gelir. Singleton tasarım kalıbı bu durumda Şekil 5'te gösterildiği gibi ciddi bir performans kazancı sağlar.

Ayrıca Factory Method tasarım kalıbını kullanmadan nesne oluşturma işlemini doğrudan Client programına vermek, Client programını gereksiz bir şekilde somut karakter sınıflarına bağımlı kılacaktır. Bunun yanında bu nesne oluşturma işleminden dolayı Client programı istenmeyen bir şekilde büyüyecek ve karmaşıklaşacaktır.



Şekil 5 Singleton Performans Testi(Singleton Performance Evaluation)

Şekil 5'te, Tablo 2'de ifadesi ve Şekil 3'te tasarımı verilen projenin Singleton tasarım kalıbı kullanarak ve Singleton tasarım kalıbı kullanmadan ortaya koyduğu performansın bir değerlendirilmesi yapılmıştır. Şekil 5, bu projenin Singleton tasarım kalıbı ile ve Singleton tasarım kalıbı kullanmadan değişik karakter sayısına sahip giriş metinleri için toplamda ne kadar zaman aldığını göstermektedir. Görüldüğü gibi, karakter sayısı artıkça bu iki durum arasındaki performans farkı da artmaktadır. Yani, performanstaki bu kazanç giriş metninin uzunluğu ile doğru orantılı olarak artmakla birlikte, örneğin100 karakterlik bir giriş metni için Singleton tasarım kalıbı kullanılmadan oluşturulan programın bu işlem için aldığı süre yaklaşık 38 saniye iken, Singleton ile birlikte bu süre 7 saniyeye düşmektedir. Bu fark daha uzun giriş metinlerinde Şekil 5'te gösterildiği gibi daha da artmaktadır.

8. SONUÇ (CONCLUSION)

Tasarım kalıpları büyük yazılımların tasarımını, anlaşılmasını ve bu yazılımla ilgili tersine-mühendislik (reverse engineering) işlemlerini kolaylaştırır [23]. Dolaysıyla Yazılım Mühendisleri için bu konuda uzmanlaşmak çok önemlidir. Tasarım Kalıplarının soyut yapısından dolayı, bahsedilen bu detayların etkili bir şekilde öğretilmesi zorlaşmakta ve öğrencilerin Tasarım Kalıplarının önemini anlaması zorlaşmaktadır.Bu çalışmamızda Factory Method ve Singleton tasarım kalıplarının detaylı bir analizi verilmiştir ve bu detayların etkili bir şekilde öğretilmesine olanak sağlayacak örnek bir uygulama projesi tasarlanmıştır. Bu kalıplar genellikle gerçek endüstriyel yazılım geliştirmede sıklıkla kullanıldığı için, basit örnekler bu kalıpların kullanımını açıklamakta yetersiz kalmaktadır. Özellikle yeterli yazılım geliştirme tecrübesi olmayan öğrenciler için ise bu daha kritik bir hale gelmektedir. Bu çalışmamızda, Tasarım Kalıplarının biri ABD'de diğeri Türkiye'de olmak üzere iki farklı üniversitede yüksek lisans ve doktora dersi olarak verirken elde ettiğimiz tecrübelere yönelik bir çalışma sunmaktayız. Özellikle Türkiye'de karşılaştığımız durum, değişik tasarım kalıplarının önemli noktalarının izahı için yukarıda bahsedilen örneğe benzer projeler oluşturulmasının gerekliliğini ortaya koymaktadır. Bu amaçla burada Factory Method ve Singleton kalıplarının önemini ve birlikte kullanımını açıklayan bir tasarım sunulmuştur. Bu tasarım ile birlikte öğrencilerin tasarım kalıplarını benimsemelerinde ve gerçek kullanım amaçlarını açıklamalarında, orta ölçekli gerçek uygulama projelerinin, seminer ve basit iskelet kodlara nazaran (skeleton code) çok daha etkili ve motive edici olduğu tespit edilmiştir. Bu çalışma aslında, Tasarım Kalıpları için orta ölçekli projeler içeren bir proje kataloğu hazırlamayı öngörmektedir. Böylece bu dersin anlatımında bu katalogdaki gerçek olabilecek kadar büyük aynı zamanda kolay anlaşılabilecek kadar basit tasarımlar kullanılabilecektir. Bu katalog Tasarım Kalıplarının, özellikle yüksek lisans ve doktora öğrencilerine faydalı olabilecek şekilde işlenmesini sağlayacaktır.

REFERANSLAR (REFERENCES)

[1] Author. **Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE** 36th Annual. IEEE, 2012.

[2] Author. **25th International Conference on Software Engineering and Knowledge Engineering (SEKE).** 2013.

[3] G. Antoniol, G. Casazza, M. Di Penta, R. Fiutem, "Object-oriented design patterns recovery", *Journal of Systems and Software*, 59(2), 181-196, 2001.

[4] A. B. Tucker, "Strategic directions in computer science education", *ACM Computing Surveys (CSUR)*, 28(4), 836-845, 1996.

[5] T. D. Meijler, S. Demeyer, R. Engel, "Making design patterns explicit in face", *Software Engineering—ESEC/FSE'97 Springer Berlin Heidelberg,* 94-110, 1997.

[6] M. Shaw, D. Garlan,**Software architecture: perspectives on an emerging discipline**. Englewood Cliffs: Prentice Hall, 1996.

[7] Mary Shaw. "Patterns for software architecture", **Pattern Languages for Program Design 2**, Editör: J. M. Vlissides, J. O. Coplien, and N. L. Kerth (Eds.),Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA,255-269, 1996.

[8] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S. T. Halkidis, "Design pattern detection using similarity scoring",*Software Engineering, IEEE Transactions on*, 32(11), 896-909, 2006.

[9] W. Zimmer, "Relationships between design patterns", **Pattern languages of program design**, Editör: J. O. Coplien and D. C. Schmidt (Eds.). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 345-364, 1995.

[10]R. K. Keller, R. Schauer, S. Robitaille, P. Pagé, "Pattern-based reverse-engineering of design components", **Proceedings of the 21st international conference on Software engineering,** 226-235, 22 May 1999.

[11] F. Bergenti, A. Poggi,"Improving UML designs using automatic design pattern detection", *12th International Conference on Software Engineering and Knowledge Engineering (SEKE),* 336-343, 2000.

[12] G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino, M. Risi,"Design pattern recovery by visual language parsing",*Software Maintenance and Reengineering*, *CSMR, Ninth European Conference on. IEEE,*102-111, 2005.

[13] Y. Gueheneuc, H. Sahraoui, F. Zaidi,"Fingerprinting design patterns",*Reverse Engineering, Proceedings. 11th Working Conference on. IEEE*, 172-181, 2004.

[14] O. Astrachan, G. Mitchener, G. Berry, L. Cox, "Design patterns: an essential component of CS curricula" **ACM SIGCSE Bulletin**, 30(1), 153-160, 1998.

[15]O. Astrachan,D. Reed, "AAA and CS 1: the applied apprenticeship approach to CS 1",**ACM SIGCSE Bulletin**, 27(1),1-5, 1995.

[16] O. Astrachan, R. Smith,J. Wilkes,"Application-based modules using apprentice learning for CS 2",**ACM SIGCSE Bulletin. ACM**, 233-237, 1997.

[17] E. Wallingford, "Toward a first course based on object-oriented patterns", **ACM SIGCSE Bulletin. ACM**, 27-31, 1996.

[18] B. Ellis, J. Stylos, B. Myers, "The factory pattern in API design: A usability evaluation",**Proceedings of the 29th international conference on Software Engineering. IEEE Computer Society,**302-312, 2007.

[19]C. Alexander, **The timeless way of building**, New York: Oxford University Press, 1979.

[20]C. Alexander, S. Ishikawa, M. Silverstein, **A pattern language: towns, buildings, construction**, Oxford University Press, 1977.

[21] E. Gamma, R. Helm, R. Johnson, J. Vlissides,**Design patterns: elements of reusable object-oriented software**. Pearson Education, 1994.

[22]J. O. Coplien, D. C. Schmidt,**Pattern languages of program design**, ACM Press/Addison-Wesley Publishing Co., 1995.

[23] H. Albin-Amiot, P. Cointe, Y.-G. Gueheneuc, N. Jussien, "Instantiating and detecting design patterns: putting bits and pieces together," *Automated Software Engineering, (ASE). Proceedings. 16th Annual International Conference on*, 166-173, 2001.