Doç. Dr. Ahmet KAYA

Ege Üniversitesi Tire Kutsan Meslek Yüksekokulu Bilgisayar Programcılığı Bölümü
ahmet.kaya@ege.edu.tr

YÖNETİM BİLİŞİM SİSTEMLERİNDE YAZILIM TASARIM DESENLERİ¹

Özet

Yönetim bilişim sistemleri, karar destek sistemleri olarak sürekli güncellenmesi ve iyileştirilmesi gerekli yazılım platformlarıdır. Bu platformların yenilenme ve güncellenmeleri için yüksek maliyetli harcamalar zorunludur. Bu durum hem bu maliyeti karşılamakla yükümlü işletmeler, hem de bu yazılımı güncellemek zorunda olan yazılım firmaları bakımından bir dizi güçlükler doğurmaktadır. Bu bakımdan bu türden sistemlerin ileride yeniden ele alınma zorunluluğu, başlangıç kurguların ve düzenlemelerin tasarım evresinde yapılmasını zorunlu kılmaktadır. Bilgi sistemlerinde nesnelerin desteklenmesine katkı yapan, nesneler arasında ayırt edici özelliği bulunan arayüzlerin ve bu arayüzler arasındaki ilişkilerin tanımlanmasına yardımcı olan benzer arayüzlere sahip birçok sınıf için tanımlama ve genellemeler yapılmasını mümkün kılan işlemlere gereksinim vardır. Bu şekilde bir deseni, çözümü aynı şekilde olmamak üzere milyonlarca kez kullanmak mümkün olabilir. Bu çalışmada bu amaca uygun yazılım tasarım desenleri için genel bir tanıtım ve bir uygulama verilmiştir.

Anahtar Kelimeler: Yazılım tasarımı; Tasarım desenleri; Singleton uygulaması; Hiyerarşik desenler.

SOFTWARE DESIGN PATTERNS IN MANAGEMENT INFORMATION SYSTEMS

Abstract

Management information systems can be defined as decision support systems in the form of software platforms that need to be constantly updated and improved. Continuous renewal and updating of these platforms necessities a range of costs and expenditures. This poses a number of difficulties both for businesses

¹Bu çalışmanın yazılım desenleri bölümü, 2010 yılında Düzce Üniversitesi tarafından düzenlenen Ulusal Meslek Yüksekokulları Öğrenci Sempozyumunda sunulmuş, özet basım olarak yayımlanmıştır.

that have to cover this cost and for software companies that have to update this software. In this respect, the necessity of reconsidering such systems in the future necessitates that the initial constructs be made during the design phase. Hierarchical layouts, which contribute to the support of objects in information systems, have a distinctive character between objects, help define interfaces and relationships between interfaces, and provide definitions and generalizations for many classes with similar interfaces are important. In this way, it may be possible to use a pattern millions of times, not the same way as a solution. In this study, in accordance with this purpose, a general definition for software design patterns is made and an application is given.

Key Words: Software design, Design patterns, Singleton application, Hierarchical patterns.

GİRİŞ

21. Yüzyıl, bilgi çağıdır. Bilgi, işletme ve ekonominin en önemli ve en değerli girdisidir. Bilgiye ve teknolojiye dayalı firmaların gerçeğe yakın doğru bilgilere gereksinimleri vardır. Ekonomik hayatın her alanını etkisi altına alan teknolojik gelişmeler, bilgi sistemleri ile bütünleşen bir dinamikle hareket etmektedir. Büyük ve karmaşık sistemler, sürekli iyileştirilmeleri gereken bilgi araçlarıdır (Özvural vd, 598).

Bilgi ve bilginin etkin yönetimi her geçen gün önemi artan bir olgudur (Gödelek, 2006:158). Bilgi sistemleri de bu türden yazılım araçları ile gelişen ve sürekli güncellenen platformlardır. Bu türden iş ve işlemlerin hızlı kanaldan inşaa edilmesi son derece önemlidir. Yönetim bilgi sistemleri, yöneticilerin karar verme süreçlerine katkı sunan ve destekleyen karar destek sistemleridir (Kalıpsız vd, 2011).

Bununla birlikte, iş örgütlerinde karar almak ve yönetsel fonksiyonları desteklemek için bilgiyi toplayan, işleyen, depolayan, ve dağıtan parçaların oluşturduğu bütüne bilgi sistemleri adı verilir. Bilgi sistemleri, yöneticilere ve çalışanlara problem analiz etmelerinde, karmaşık sorunları çözmede ve yeni ürünler oluşturmalarında yardımcı olmaktadır (Alagöz, vd., 2013: 29).

Bu sistemlerin işlevleri kadar tasarım biçimleri de önemlidir. Zira karar mekanizmaları tasarlanırken tasarım deseni adı verilen düzeneklerin kurgulanması ve bu kurgulamalar temelinde bilgi sistemlerini geliştirmek, üretilen kararların etkinliği bakımından önemlidir. Bilgi sistemlerinin tasarım desenleri temelinde kurgulanması modüler bir yapının inşaa edilmesine katkı sunan bir sistematığın ortaya çıkmasına neden olacağından sistem; eklenebilir, güncellenebilir ve yeniden geliştirilebilir özelliklere sahip olur.

Bilgi sistemleri ile üretilen ve hayata geçirilen kararlardan olumlu veya olumsuz etkilenecek bireylerin durumları, mali getirileri, kayıpları ve risklerini göz önünde bulundurmaya gerektiren bir süreç kendiliğinden gelişmiş olur. Yöneticiler, ülkemizin ekonomik dinamiklerini de kapsayabilecek etki alanları geniş kararlar verirler. Yönetimsel her bir kararın yol açacağı bir dizi ekonomik maliyet ve değişim bulunmaktadır. Değişimler genelde rekabet üstünlüğü ve verimliliğini artırmaya yönelik olup yeni istihdam alanları yaratmak üzere

TİDSAD

kullanılabilir. Bu sayede değişimin gerektirdiği bilgilerle yeni bir dinamizmi hayata geçirmek, yeni kazanımlar ve fırsatlar yakalamak ve değişimlere adapte olmak mümkün hale gelir (Kaya, 2015:348-350), (Gates, 1999).

Hızlı değişimlere adapte olmak için işletmelerin bilişim sistemlerini hızlı bir şekilde geliştirmeleri ve bu sistemleri çalışır getirmeleri zorunludur. Bir sistemi her defasında yeni baştan ele almak ve geliştirmek hem zaman hem de maliyet bakımından mümkün değildir. Bunun yerine işletmenin sahip olduğu yazılım envanterleri üzerinde güncellemeler yapmak ve bunları hızlı bir şekilde kullanıma almak gerekir. Bunun için üretilen yazılımları, tasarım desenleri temelinde kurgulamak, işlemleri hızlandırmak anlamında bir zorunluluktur. Böylece benzer kod yapılarını dinamik tutan eşanlı kullanımlar mümkün hale gelir. Bundan sonraki bölümlerde anılan işlemleri mümkün hale getiren yazılım tasarım desenlerinin bölümleri, sınıflandırılması, organizasyonu, katalog hale getirilmesi tanıtılacak ve bir uygulama işlemi ile makale çalışması sonuçlandırılacaktır.

1. TASARIM DESENLERİ

Tasarım desenleri, nitelikli yazılım geliştirmenin adımlarından biridir. Nitelikli yazılım, firmaların gereksinimlerini en üst düzeyde karşılayan hızlı, modüler, eklenebilir ve güncellenebilir niteliklerde olması gerekir. Günümüzde yazılım sistemleri bir çok iş alanının temel bileşenleri arasında yer almaktadır (Bkz. Nalbant, Milsoft-ODTÜ).

Nesnelerin desteklenmesine katkı yapan, nesneler arasında ayırt edici özelliği bulunan, arayüzlerin ve bu arayüzler arasındaki ilişkilerin tanımlanmasına yardımcı olan, benzer arayüzlere sahip birçok sınıf için tanımlama ve genellemeler yapılmasını sağlayan hiyerarşik düzenlere tasarım desenleri adı verilir. Her bir desen, çevremizde sürekli olarak tekrar eden bir problemi ve problemin çözümünü anlatır. Bu şekilde bir deseni, çözümü aynı şekilde olmamak üzere milyonlarca kez kullanmak mümkün olabilir. Böylece aynı desenleri yeniden kurgulama gereksinimi ortadan kalkar.

Desen kavramının kullanımı 1970'li yılların sonlarında Christopher *Alexander* isimli bir mimar tarafından yapılmıştır. *Alexander*, desenlerin belgelenmesi için temel kabul edilen örnekler ile ilgili "A *Pattern Language* [1977]" ve "A *Timeless Way of Building* [1979]" isimli iki kitap yazmıştır. 1987 yılında, Ward Cunningham ve Kent Beck bir kullanıcı arayüzünün tasarımı ve small talk üzerine çalışıyordu. Bu amaçla yeni small talk programcılarına yol göstermesi amacıyla beş adet desen dili aracı geliştirmeye ve *Alexander*'in fikirlerinden yararlanmaya karar verildi. Elde edilen sonuçlar "*Using Pattern Languages for Object-Oriented Programs*" adlı makalede yazılıp OOPSLA'87 kongresinde Orlando'da sunuldu (Kaya ve ark., 2005: 321-325).

1990-1992 yılları arasında kadar Gang of Four (GoF) Dörtlü çete ekibinin üyeleri (Erich Gamma, Richard Helin, Ralph Johnson, ve John Vlissides) çeşitli desen katalogları hazırladılar. 1993 yılı Ağustos ayında Colorado şehrinde Kent Beck ve Grady Booch sponsorluğunda Hillside Group ilk toplantısını yapmıştır. Daha sonra desen toplantısı OOPSLA'93 yapılmış ve Nisan 1994'te Hillside Group ilk PloP konferansını planlamak üzere toplanmıştır. Kısa bir süre sonra GoF Design Patterns kitabı yayınlanmıştır.

2. DESENLERİN BÖLÜMLERİ

Yazılım tasarım desenleri kuşkusuz çok sayıda bölümden oluşmuştur. Bu bölümlerin her birini bu çalışma kapsamı içerisinde ifade etmeye olanak yoktur. Bu bakımdan desenlerin bölümlerini ana hatları ifade etmek uygun bulunmuştur. Buna göre, genel olarak bir desenin dört ana elemanı vardır (Kaya ve ark, 2005: 321-325):

- **Desen Adı:** Tasarım problemini, çözümleri ve sonuçlarını bir iki kelimede anlatmak için kullanılır. Desenin başlangıçta isimlendirilmesi tasarım sözlüğünü zenginleştirir. Yüksek seviyede soyutlama (abstraction) yapacak şekilde tasarım yapılmasını sağlar. Tasarım deseni için bir sözlüğün olması, tasarım ile ilgili olarak düşünmeyi, dökümantasyon yapmayı kolaylaştırır. İyi bir desen adı bulmak en zor işlerden biridir ve çok önemlidir.
- **Problem:** Desenin ne zaman uygulanacağını anlatır ayrıca problemi ve içeriğini açıklar. Belirli tasarım problemlerini anlatabilir. Esnek olmayan tasarımların belirtileri olan sınıf ve nesneleri tanımlayıcı kılar.
- **Cözüm:** Tasarımı oluşturan unsurları, bunlar arasındaki ilişkileri, sorumlulukları ve aralarındaki işbirliğini anlatır. Desen, çok farklı durumlara uygulanabilecek bir şablon olduğu için, çözüm kesinlik arz eden belli tasarım ve uygulamaları anlatmaz. Desen, daha ziyade tasarım problemini soyut olarak tanımlar ve bu problemin çözümü için unsurların nasıl düzenleneceğine karar verir.

Sonuçlar: Desenin uygulanması ile elde edilen sonuç ve faydaların anlatıldığı kısımdır. Tasarım kararları alınırken pek seslendirilmemelerine karşılık farklı tasarımların değerlendirilmesinde, uygulanan desenin maliyet ve tavizlerinin anlaşılmasında kritik öneme sahiptir. Desen sonuçlarının sistemin esnekliğinde, genişletilebilmesinde ve taşınabilirliğinde kritik etkisi vardır.

3. DESENLERİN SINIFLANDIRILMASI

Tasarım desenlerinin anlatımı için grafiksel notasyonlar önemli ve kullanışlı olmalarına karşın yetersizdir. Grafiksel notasyonlar sadece tasarım işleminin son ürününü gösterirler. Fakat bir tasarımı yeniden kullanabilmek için kararların, alternatiflerin ve verilen tavizlerin kaydedilmesi gerekir. Ayrıca tasarımı uygulamada görmeye yardımcı olan somut örnekler de önemlidir. Tasarım desenleri sabit bir format kullanılarak anlatılır. Her bir desen aşağıda verilen şablonlara uygun olarak bölümlere ayrılır.

- **Amaç (Intent):** Amaç bölümünde tasarım deseninin amaçları belirtilir.
- **Also known as:** Kullanılıyor ise, desenin bilinen diğer adı.
- **Motivation:** Bu bölümde, tasarım problemini, sınıf ve nesne yapılarının problemi nasıl çözdüğünü canlandıran senaryo bulunur. Bu bölümde, senaryo desenin daha iyi anlaşılması sağlanır.
- **Applicability:** Tasarım deseninin uygulanabileceği durumları, desenin ilgilendiği diğer tasarım örneklerinin neler olduğu belirtilir.

- **Structure:** Nesne modelleme teknikleri (Object Modeling Technique) kullanarak desendeki sınıfların grafiksel gösterimi yapılır. Ayrıca nesneler arasındaki işbirliğini ve talep zincirlerini canlandırabilmek için etkileşim diyagramları kullanılır.
- **Participants:** Tasarım modelinde yer alan sınıf ve/veya nesneler ve sorumlulukları belirtilir.
- **Collaborations:** Katılımcıların sorumluluklarını gerçekleştirebilmek için nasıl işbirliğine girecekleri belirtilir.
- **Consequences:** Desen hedeflerini, desen kullanmanın sonuçları ve verilen tavizler belirtilir.
- **Implementation:** Desenin gerçekleştirilmesinde dikkat edilecek güçlükler, ipuçları ve teknikler tanımlanır.
- **Sample Code:** Desenin nasıl gerçekleştirileceğini gösteren C++ ya da Smalltalk kod parçaları bulunur.
- **Known Uses:** Gerçek sistemlerde yer alan desen örnekleri bulunur. Farklı alanlardan en az iki örnek içermelidir.
- **Related Patterns:** Desen ile yakından ilişkili diğer tasarım desenleri ve farklılıkları belirtilir.

4. DESEN KATALOĞUNUN ORGANİZASYONU

Tasarım desen katalogları organize edilirken iki temel kriter göz önünde bulundurularak organize edilirler. Bu iki kriter şu şekilde planlanmaktadır:

Amaç (Purpose): Desenle neyin nasıl yapılacağı belirtilir. Bir desen, aşağıda belirtilen üç amaçtan birine sahip olabilir. Bunlar;

- a. Yaratılışsal (Creational): Nesne oluşturma süreci ile ilgilenir.
- b. Yapısal (Structural): Sınıf ve nesnelerin yapısı ile ilgilenir.
- c. Davranışsal (Behavioral): Sınıf ve nesne etkileşimleri ve sorumlulukları ile ilgilenir.

Alan (Scope): Alan, desenin ilk olarak sınıflara mı yoksa nesnelere mi uygulanacağına karar vermektedir. Sınıf desenleri bir sınıf ile alt sınıflar arasındaki ilişkiler ile ilgilidir. Bu ilişkiler kalıtım (inheritance) yolu ile sağlandığı için derleme anında statik özelliktir. Nesne desenleri ise çalışma zamanında değişebilen ve daha dinamik olan nesne ilişkileri üzerine yoğunlaşırlar. Hemen hemen her desende kalıtım söz konusu olabildiği için sadece sınıflar arası ilişkiler ile ilgilenen desenler, sınıf deseni olarak adlandırılır. Yaratılışsal sınıf desenleri, bazı nesne parçalarının oluşturulmasını alt sınıflara bırakırken, yaratılışsal nesne desenleri ise bu işlemleri diğer nesnelere bırakırlar. Yapısal sınıf desenleri kalıtımı sınıf oluşturmak için kullanırken yapısal nesne desenleri, nesnelerin nasıl birleştirileceğini anlatır. Davranışsal sınıf desenleri kalıtımı akış diyagramını ve algoritmaları anlatmak için kullanırken, davranışsal nesne desenleri bir nesnenin tek başına gerçekleştiremeyeceği işi nesne grubunun nasıl gerçekleştirdiğini anlatır.

Jurisdiction	Class	Characterization		
		Creational	Structural	Behavioral
	Class	Factory Method	Adapter (class) Bridge (class)	Template Method
	Object	Abstract Factory Prototype Singleton	Adapter (object) Bridge (object) Flyweight Glor Proxy	Chain of Responsibility Command Iterator (object) Mediator Memento Observer State Strategy
	Compound Builder		Composite Wrapper	Interpreter Iterator (compound) Walker

Tablo-1, Tasarım Desenlerinin Sınıflandırılması.

Desenler daha farklı şekillerde de sınıflandırılabilir. Bazı desenler birlikte kullanılırlar. Composite deseni Iterator yada Visitor ile birlikte kullanılır. Bazı desenler birbirinin alternatifidir. Prototype, Abstract Factory deseninin alternatifidir.

5. TASARIM DESENLERİ KATALOĞU

Tasarım desen kataloglarında her biri farklı işlevlere 23 adet tasarım aracı bulunmaktadır. Bu tasarım araçlarının her biri gömülü özelliklere sahip, çok sayıda işlevi bünyesinde barındıran niteliklere sahiptir. Bu 23 adet tasarım aracının isimleri ve işlevleri aşağıda listelenmiştir.

- **Abstract Factory:** Birbirleri ile ilişkili ya da birbirlerine bağımlı nesne ailelerinin somut sınıflarını belirtmeksizin oluşturulmaları için bir arayüz sağlar.
- **Adapter:** Bir sınıfın arayüzünün başka bir istemci arayüzünün beklediği şekle çevrilmesini sağlar. Uyumsuz arayüzler dolayısıyla çalışamayan sınıfların birlikte çalışmalarını sağlar.
- **Bridge:** Soyutlamanın (abstraction) ve uygulamanın birbirinden ayrılmasını sağlar.
- **Builder:** karmaşık nesnelerin oluşturulmasını gösterimlerinden ayırır. Böylelikle aynı oluşturma işlemi ile farklı gösterimler meydana getirilebilir.
- **Chain of Responsibility:** İsteğin karşılanması şansı birden fazla nesneye verilerek aynı isteğin tekrarlanması önlenir. Alıcı nesneler sıralanır ve istek karşılanana kadar talep zincir boyunca diğer nesneye aktarılır.
- **Command:** Bir istek, nesne olarak saklanır.
- **Composite:** Hiyerarşiyi göstermek amacıyla nesneler ile bir ağaç yapısı oluşturulur. Böylelikle istemcilerin her biri ayrı birer nesne gibi davranır.
- **Decorator:** Nesnelere dinamik olarak ek sorumluluklar eklenir. Fonksiyonelliği arttırmak için esnek alternatifler sağlar.
- **Facade:** Alt sistemdeki arayüzler kümesine tek bir arayüz sağlar. Alt sistemin daha kolay kullanılabilmesi için yüksek seviyede arayüz temin eder.

- **Factory Method:** Nesnelerin oluşturulması için bir arayüz tanımlar.
- **Flyweight:** Çok sayıdaki nesnenin verimli desteklenmesi için paylaşımı kullanır.
- **Interpreter:** Gramerin, soyut sözdizimsel ağacın ve basit diller için yorumlayıcının nasıl gösterileceğini tanımlar.
- **Iterator:** Nesnenin gösterimini açığa vurmaksızın nesne unsurlarına sırayla ulaşılmasını temin eden bir yol sağlar.
- **Mediator:** Nesne kümesinin birbiriyle nasıl etkileşimde bulunduğunu örten bir nesne tanımlar. Mediator, “*loose coupling*” özelliğinin gelişmesine yardım eder.
- **Memento:** Bir nesnenin iç durumunu encapsulation’ı bozmadan nesne daha sonradan bu duruma geri getirilebilecekmiş gibi saklar.
- **Observer:** Nesneler arasında one-to-many bağımlılığını tanımlar. Böylece bir nesnenin durumu değiştiğinde, ona bağlı tüm nesneler uyarılır ve otomatik olarak güncellenir.
- **Prototype:** Prototip kopyalayarak yeni nesnelerin oluşturulması için nesne çeşitlerini belirler.
- **Proxy:** Nesneye olan ulaşımı kontrol etmek için onun yerini tutan başka bir nesne tanımlanır.
- **Singleton:** Bir sınıfın tek bir örneğinin (instance) olmasını ve sınıfa ulaşım için kullanılan global bir nokta sağlar.
- **State:** Nesnenin dahili durumunun değişmesi halinde davranışın değişmesini sağlar.
- **Strategy:** Algoritma ailesini tanımlar. Bunların istemciden bağımsız olarak farklılık göstermesini sağlar.
- **Template Method:** Bir algoritmanın yapısını değiştirmeden alt sınıfların algoritmanın belli adımlarının tekrar tanımlanmasını sağlar.
- **Visitor:** Bir işlemin bir nesne yapısının elementleri üzerinde gerçekleştirilmesini sağlar. İşlem yaptığı unsurun sınıflarını değiştirmeksizin yeni bir işlem tanımlar.

6. SINGLETON UYGULAMASI

Singleton deseni bir programın yaşam süresince belirli bir nesneden sadece bir örneğinin (instance) olmasını garantiler. Aynı zamanda bu desen, yaratılan tek nesneye ilgili sınıfın dışından global düzeyde mutlaka erişilmesini hedefler. Örneğin bir veritabanı uygulaması geliştirildiğinde her programcı mutlaka belli bir anda sadece bir bağlantı nesnesinin olmasını ister. Böylece her gerektiğinde yeni bir bağlantı nesnesi yaratmaktansa var olan bağlantı nesnesi kullanılarak sistem kaynaklarının daha efektif bir şekilde kullanılması sağlanır. Bu örnekleri daha da artırmak mümkündür. Bu şekilde belli bir anda ilgili sınıfın bir örneğine ihtiyaç duyulduğunda bu desen kullanılabilir. O halde bir sınıftan sadece ve sadece bir nesne yaratılmasını garanti altına almak için nasıl bir çözüm yaratılmalıdır sorusunun cevabını bulmak gerekmektedir. Buna çözüm statik üye elemanlarını kullanarak Singleton tasarım desenine geçmeden önce sınıflar ve nesneler ile ilgili temel bilgiler hatırlandığında yeni bir

nesne oluşturmak için yapıcı metot (constructor) kullanılır. Yapıcı metotlar C# dilinde new anahtar sözcüğü kullanılarak aşağıdaki gibi çağrılabilir [Online 3].

```
public class SingletonDeseni
{
    private static SingletonDeseni nesne = new SingletonDeseni();

    private SingletonDeseni()
    {
    }
    public static SingletonDeseni Nesne
    {
        get
        {
            return nesne;
        }
    }
}
```

Yukarıdaki sınıf örneğinde SingletonDeseni sınıfı belleğe yüklendiği anda statik olan SingletonDeseni nesnesi yaratılacaktır. Bu nesne yaratılışının new anahtar sözcüğü ile yapıldığına dikkat edilmelidir. Eğer Main() gibi bir metodun içinden bu nesne yaratılmaya çalışılırdı derleme aşamasında hata alınırdı. Çünkü public olan herhangi bir yapıcı metot bulunmamaktadır. Ayrıca Main() gibi bir metodun içinden yaratılan bu nesneye,

SingletonDeseni nesne = SingletonDeseni.Nesne;

Şeklinde erişmek de mümkündür. Böylece yukarıdaki deyim her kullanıldığında geri dönen nesne, sınıfın belleğe ilk yüklendiğinde yaratılan nesne olduğu garanti altına alınmış olur. Dikkat edilmesi gereken diğer bir nokta, nesneyi geri döndüren özelliğin yalnızca get bloğunun olmasıdır. Böylece bir kez yaratılan nesne harici bir kaynak tarafından hiç bir şekilde değiştirilemeyecektir [Online 3].

Singleton Deseninin 2. versiyonu:

```
public class SingletonDeseni
{
    private static SingletonDeseni nesne = new Singleton();

    private SingletonDeseni()
    {
    }
    public static Singleton Nesne()
    {
        return nesne;
    }
}
```


Görüldüğü üzere nesne ilk olarak sınıf belleğe yüklendiğinde değil, nesne ilk defa kullanılmak istendiğinde yaratılmaktadır. İlgili nesneyi her istediğimizde yeni bir nesnenin yaratılmaması içinde $if(nesne == null)$ biçiminde bir koşul altında nesnenin yaratıldığına dikkat edilmelidir [3].

7. TASARIM DESENLERİNİN GEREKLİLİĞİ

Tasarım desenleri, yoğun bir emekle hazırlanmış bilgi parçacıklarını kullanmak, yeniden benzer çalışmaları üretmek üzere bir çaba harcamamak ve standart bir dil birliği sağlamak üzere tesis edilmiş kavramlar bütünüdür. Buna göre tasarım desenleri aşağıda belirlenen amaçları gerçeklemek maksadıyla kullanılmaktadır.

- **Yeniden kullanılabilir çözümler:** Önceden hazırlanmış tasarımların yeniden kullanılması, problemin çözümü için bir başlangıç noktası ve başka tecrübelerden faydalanma avantajı olarak belirtilebilir. Bu durumda tekrarlı olarak meydana gelen ortak bir problemin çözümünü yeniden keşfetmek zorunda kalınmaz.
- **Ortak bir terminoloji hazırlanır:** İletişim ve takım çalışması problemlere ortak bakış noktası ve ortak bir sözlük gerektirir. Tasarım desenleri projenin tasarım ve analiz bölümleri boyunca kullanılacak ortak bir referans noktası sağlar.
- **Diğer Sebepler:** Problem üzerinde ve nesneye yönelik tasarım işlemi esnasında yüksek seviyeli bir perspektif, geniş bir bakış açısı sağlar. Detaylar ile uğraşma zorluğu ortadan kalkar. Tasarım deseninin diğer faydaları şu şekilde sıralanabilir:
 1. Tasarım deseni kullanan yazılım gruplarında tasarım deseninin hem bireysel öğrenmeye hem de takımın gelişimine yardımcı olduğu görülmüştür. Çünkü yeni yazılımcılar, tecrübeli takım arkadaşlarının tasarım deseni kullanma isteklerini ve ona verdikleri önemi görerek bu güçlü kavramları öğrenmeye çalışırlar.
 2. Tasarım deseni, yazılımın geliştirme aşamasında değiştirilebilmesini sağlar.
 3. Tasarım deseni doğru bir şekilde öğrenildiğinde nesneye dayalı tasarım ilkelerinin anlaşılabilirliğini arttırmada kullanılabilir.
 4. Tasarım deseni sayesinde karmaşık problemler için bile büyük kalıtım hiyerarşisine gerek kalmaksızın tasarım yapmayı sağlar. Tasarım deseni kullanılmasa bile büyük kalıtım hiyerarşisinden kaçınmanın sonucunda gelişmiş bir tasarım ortaya çıkar.

SONUÇ ve ÖNERİLER

İşletmelerin en çok harcama yaptığı alanların başında teknolojik gelişmelere paralel olarak donanım ve özellikle yazılım araçlarına yapılan harcamalar gelmektedir. Bu harcamalar, sermaye düzeyleri güçlü olmayan işletmeler için sarsıcı ve maliyetli olabilmektedir. Özellikle yazılım maliyetlerinin her geçen gün daha yüksek düzeylere ulaşması, firmaları kendi bilgi işlem birimlerini kurması noktasına taşıyabilmektedir. Bazı işletmeler bakımından bu masraflar katlanılmaz bir noktaya ulaşabilmektedir. Bu aşamada çözüm, yazılımda tasarım desenlerini etkili bir biçimde kullanmasını bilen personel istihdam etmektir. Zira, tasarım desenleri yazılımcıya daha kolay ve hızlı bir şekilde yazılım geliştirme şansı verirken, işletmelere de nitelikli yazılım envanteri edinme olanağı sağlamış olmaktadır. Çünkü yazılım güncelleme işlemleri ile yazılım masrafları minimize olabilecek ve firmanın gereğinden fazla personel istihdam etme zorlukları ortadan kaldırılabilecektir.

Yazılım geliştirmek zor ve zahmetli bir iştir. Yazılım çalışması tamamlandıktan sonra, işlemin tamamen sona erdiğini söylemek mümkün değildir. Bir yazılım, bir süre kullanıldıktan sonra, ortaya çıkan gereksinimleri karşılamak amacıyla bir dizi güncellemeye gereksinim duyulabilir. Dolayısıyla yapılan çalışma ile ortaya konan zahmetlerin boşa gitmemesi için kod yazımını belli bir disiplin, planlama ve tasarım bütünlüğü içerisinde yapmak gerekir.

Yazılımda tasarım desenleri kavramı da bu amaca hizmet eden, çalışmaların bir plan ve bütünlük içerisinde, muhtemel güncellemelere imkan verecek esneklikte, eklenebilir ve modüler olmasına katkı sağlayan bir planlama metodudur. Tasarım desenlerinin sağladığı faydalar aşağıdaki gibi ifade edilebilir:

1. Karmaşık yazılımlar, basit bir düzeye indirgenmiş olur.
2. Yazılım, modüler bir özellik sağlanmış olur.
3. Yazılıma eklenebilirlik (güncelleme olanağı) sağlanır.
4. Yeniden uyarlama mümkün hale gelir.
5. Ortak bir dil oluşur.
6. Algoritma hiyerarşisi oluşur.
7. Hedefler ve sonuçlar belirlenir.
8. Katılımcıların sorumlulukları belirlenir.
9. Geniş bir bakış açısı elde edilir.
10. Bütün alt programlar için kullanılan arayüzler yerine tek bir arayüz kullanılır.

Bütün bu avantajları yanında, tasarım desenlerinin, yetersiz programlama araçları için kullanılması gereken bir metot olduğunu savunan görüşler vardır. Ancak bir programlama dili ne kadar gelişmiş olursa olsun tasarım desenlerinin kullanımı ile daha özellikli ve etkin bir hale geleceği düşünülmektedir.

KAYNAKLAR

- Alagöz, A, Öge, S., Koçyiğit, N. (2103), “Muasebe Bilgi Sistemi ve Karar Destek Sistemleri İlişkisinin Yönetmel Karar Alma Faaliyetlerine Etkisi“, Selçuk Üniversitesi, Sosyal Bilimler Enstitüsü Dergisi, Sayı:30, Sayfa:27-40.
- Gates, B.(1999). *Düşünce Hızında Çalışmak*, (Çev: Ali Cevat Akkoyunlu), Doğan Kitap, İstanbul.
- Gödelek, E. (2006), “Yönetmel Karar Destekleme Sistemleri:Pazarlama Uygulaması”, Dokuz Eylül Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, Cilt:8, Sayı:3, Sayfa:158-179.
- Kalıpsız, O. Buharalı, A. Biricik G. (2011). *Sistem Analizi ve Tasarımı*, Papatya Yayıncılık Eğitim, İstanbul.
- Kaya, Öktem, D., Toker, L., Dönmez, O. (2005), “Bilgisayar Bilimlerinde Yazılım Tasarım Desenleri ve Önemi”, Süleyman Demirel Üniversitesi Burdur Meslek Yüksekokulu, 3. Ulusal Meslek Yüksekokulları Sempozyumu, 28-30 Eylül 2005, Burdur.
- Kaya, A. (2015). “Bilişim Eğitiminde Değişim Yönetimi”, *International Conference on Research in Education and Science (ICRES)*, April 23-25, 2015, Antalya, Turkey.
- Kaya, A, Vardarcık, T. (2010), “Yazılım Tasarım Desenleri ve Singleton Uygulaması, Düzce Üniversitesi, Ulusal Meslek Yüksekokulları Öğrenci Sempozyumu, 21-22 Ekim, Özet Basım, Sayfa:44.
- Nalbant, S. “Yazılım Geliştirme Sürecinin Verimliliğini artırmak: Bir Bilgi Sistemi Örneği.MilSOFT Yazılım Teknolojileri A.Ş. ODTÜ, Ankara.
- Özvural, G.Ö., Gün, Ö., Ak, E. “Etkin bir yazılım Süreç Yönetimi için Süreç Yönetim Aracı Seçimi, TUBİTAK, Bilgem BTE Bilişim Teknolojileri Enstitüsü..
- [1] Online www.tr.wikipedia.org.
- [2] Online www.tasarimdesenleri.blogspot.com.
- [3] Online www.csarpnadir.com/makalegoster.asp/Mid=134.