A Reference Model for Low-Level Design

Alt Düzey Tasarım için Referans Modeli

Ebru Özdoğru Kandırmaz[1] Ufuk Tiryaki[2]

Ali Hikmet Doğru[3]

[1,2] IBTECH A.Ş. Tübitak MAM Teknoloji Serbest Bölgesi, Gebze/Kocaeli
[3] Orta Doğu Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, Ankara
[1] ebru.ozdogrukandirmaz@ibtech.com.tr
[2] ufuk.tiryaki@ibtech.com.tr
[3] dogru@ceng.metu.edu.tr

Abstract. This research presents a reference model to improve the effectiveness of low-level design activities. This model is supported with methodological steps and has been realized as a result of real industrial problems and is explained here using an example case. This work shows that low-level design activity is very important to meet quality attribute requirements of systems as much as high-level design activity. As a case study, re-engineering process of a legacy banking system is presented.

Keywords: Low-level design, Software re-engineering, Quality attributes.

Özet. Bu çalışmada ayrıntı düzeylerinde tasarım çalışmalarını daha etkin hale getirmek için bir referans modeli sunulmaktadır. Gerçek endüstri problemlerinden yola çıkarak planlanan ve metodolojik adımlarla desteklenen bu model, bir saha vakası kullanılarak anlatılmıştır. Çalışma, istemlerde kalite öznitelik gereksinimlerinin karşılanmasında alt düzey tasarımın üst düzey mimari tasarım kadar önemli bir etkinlik olduğunu göstermektedir. Saha vakası olarak eski bir bankacılık sisteminin yeniden yapılandırma süreci anlatılmıştır.

Anahtar Sözcükler: Alt düzey tasarım, Yazılım değişim mühendisliği, Kalite öznitelikleri.

1 Giriş

Bankacılık sistemleri kurumsal sistemlerdir. Günümüz finans teknoloji şirketlerinde alçak düzeyli ve yordamsal dillerle geliştirilmiş anaçatı (mainframe) tabanlı sistemler kullanılmaya devam edilmektedir.

Gelişen teknoloji ve ihtiyaçlarla birlikte, bankalar sistemlerini yeniden yapılandırma çalışmaları içindedir. Yeni sistemlerin üç temel özelliğinden bahsedilebilir:

1. Yüksek düzeyli programlama dillerinin kullanımı,
2. Değişen ihtiyaçlara hızlı cevap verebilen yenilikçi mimari tasarımlar ve
3. Bilgiye kolay erişim ve mobil sistem destekleri.

Türkiye'deki sayısal bankacılık sisteminin Avrupa ülkeleri arasında ön sıralarda yer aldığı söylenebilir [1]. Bu motivasyonun bir sonucu olarak sistemlerin mimari ve tasarımsal konuları da oldukça ön planda yer almaktadır. Önemli olan her sahada olduğu gibi, bankacılık sahasında da yazılım mimarisi konuları ve bu çalışma özelinde ele alınmakta olan alt düzey tasarım konuları gelişme potansiyeli olan konulardır.

2 Gereksinimler ve Kalite Öznitelikleri

2.1 Sistem Gereksinimleri

İşlevsel olmayan gereksinimlerin çözümlenmesi ile sistemin kalite öznitelik gereksinimleri ortaya çıkar. Kalite öznitelik gereksinimleri mimari tasarımları yönlendirir [2, 3]. Bankacılık sisteminin işlevsel olmayan gereksinimleri şu şekilde özetlenebilir:

Müşteri Odaklılık. Müşteri kitlesini, gereksinimleri belirleyenler ve sistemi kullananlar olmak üzere iki gruba ayırabiliriz. Gereksinimleri belirleyen müşteriler bundan sonra iş kolları olarak adlandırılacaktır.

İş kolları, gereksinimlerin karşılanması konusunda rekabet ortamından dolayı sabırsızdır. Gereksinimler için belirlenen standartlaşmış bir referans dokümanı yoktur. İş kollarındaki personel değiştikçe aynı fonksiyon için sistemden beklenti de değişkenlik gösterir.

Sistemi kullanan müşteriler, bankacılık işlemlerini personel olarak kullanan veya sistemden hizmet alan uç kullanıcılar olarak tanımlanır. Hizmet alan müşteriler, değişen piyasa koşulları neticesinde paralarının değer kaybetmemesi ve hayat standartlarının bozulmamasını isterler. Sosyal medyayı etkin kullanırlar. Bu tür kullanıcılar açısından ortaya çıkacak hataların hızla tamir edilmesi ve uygulamanın pazara çıkış süresinin kısalması gerekir.

Zaman Odaklılık. Bu sahada söz konusu olan sistemler üretim ortamında sürekli kullanılan canlı sistemlerdir. Üretim ortamındaki hataların olabildiğince hızlı tamir edilmesi beklenmektedir. Müşterilere kesintisiz hizmet vermesi gereken fonksiyonları içerirler.

Küresel piyasalar, rekabet ortamı ve sık değişen yasal düzenlemeler, yazılım ihtiyaçlarının çok hızlı değişmesine neden olmaktadır. Geliştirmelerin pazara çıkması için çoğunlukla zaman kısıtları öne çıkmaktadır.

Performans Odaklılık. Gün geçtikçe sisteme yeni işlevler eklenmektedir. Günlük işlem hacmi çok yüksek hızda artmaktadır. Sistem yatay ve düşey yönde büyüme eğilimindedir.

Sistemin hızlı cevap vermesi önemlidir. Örneğin, piyasaların çok dalgalı olduğu saatlerde, döviz kuru değişikliklerini yakın gerçek zamanlı görebilip, hızlı işlem yapma gereksinimleri vardır. Piyasa hareketliliği sistemlerde ciddi miktarda yük yaratmaktadır.

2.2 Kalite Öznitelikleri

Tablo 1'de işlevsel olmayan gereksinimlerin mimari kalite özelliklerine göre sınıflandırılmış bir listesine yer verilmektedir.

Tablo 1. Mimari kalite özellikleri

Gereksinim	Öznitelik
Canlı (yaşayan) sistemler 7/24 hizmet edebilme Başarım/yük oranı	Kullanılabilirlik / Güvenilirlik
Gereksinimlerin sık değişmesi	Yeniden Kullanılabilirlik / Genişletilebilirlik / Bakım yapılabilirlik
Pazara çıkma süresinin kısalığı	Yeniden kullanılabilirlik
Gelişen sistem (Growing system)	Genişletilebilirlik / Ölçeklenebilirlik

3 Sistemleri Yeniden Yapılandırma Süreci

3.1 Yeniden Yapılandırma Etkinlikleri

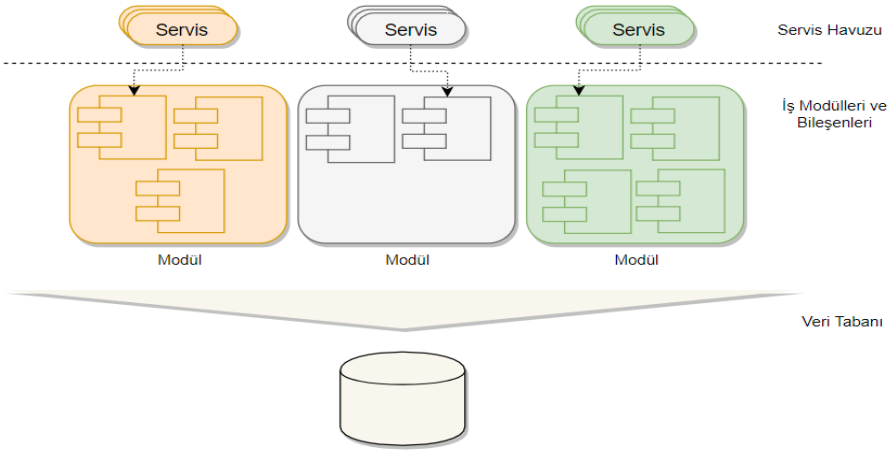
Yenilenen sistemleri olabildiğince çabuk 'canlıya almak' hedeflendiğinden dolayı üst düzey tasarıma odaklanılmaktadır. Alt düzey tasarımla ilgili konular genellikle sonraki çalışma adımları olarak düşünülmekte ve ötelenmektedirler.

Yazılım mimarisi, sistemi yukarıdan aşağıya doğru alt sistemlere bölerek yapılan çok katmanlı bir çalışmadır [6]. Üst düzey tasarım, tüm sistemin ana parçalarını ve ara yüzlerini belirleyebilmek amacı ile yapılır. Alt düzey tasarım ise, üst düzey tasarımda belirlenen sistemin ana bileşenlerinin içlerinin ayrıntılı tasarımının yapıldığı etkinliktir.

Bu çalışmada örnek olarak sunulmakta olan sistemde servis yönelimli bir mimari (SYM) seçilmiştir. SYMyi, yeniden kullanılabilir ve gevşek bağlı servislerin kullanılmasını temel alan üst düzey bir mimari tasarım kalıbı olarak değerlendirmek mümkündür [3].

Üst düzey mimari tasarım etkinliklerinde aşağıdaki konulara öncelik verilmiştir:

- Servis havuzu,
- Servislerin kullanım senaryoları ve bir araya getirilmesi,
- Temel iş modülü bileşenleri.



Şekil. 1. Yeni sistemin üst düzey mimarisi – Servis Yönelimli Mimari (SYM)

Servis Havuzu, servislerin uygulamalar tarafından kullanılabilmesi için gerekli tanımları da içerir. Sevisin iş modülü ile bileşen ve kaynak kod parçası ilişkisini barındırır.

İş katmanını oluşturan temel modüller, sistemin iş bakışı açısından bölümlemesini sunmaktadır. Bu bölümleme genelde kurumda organizasyonel yapılanmayı da hedefler. Modüllerin içindeki bileşenler, modüllerin iş mantığına göre ayrıştırılması ile oluşan kaynak kodlarıdır.

Örnek olarak sunulan sistemde, üst düzey mimari tasarımı sonrasında eski kodlar yeni yazılım diline ayrıntılı tasarım yapılmadan aktarılmaktadır. Bunların ayrıntı seviyesinde tasarlanıp yeniden yazılması daha sonraki etkinlikler olarak planlanmıştır. Bu süreç sonucunda, üst düzey bir dile geçişten bahsedilse de, yordamsal kaynak kodlarının devam ettirildiği yeni bir gevşek bağlı sistem geliştirilmiştir.

3.2 Gözlemlenen Sürdürülmekte Olan Sorunlar

Sistemlerin canlıya alınması sonrasında ise, iş kollarının gereksinimleri doğrultusunda yeni işlevlerin hızla sisteme eklenmesi istenmiştir. Kalıtım yolu ile aktarılmış kodların yeniden tasarlanması ve yazılması ötelenmektedir.

Ekiplere yeni gelen yazılımcılar kodlarda alışageldikleri uyarlamaları devam ettirmektedirler. Kodlar büyüdükçe ve alışkanlıklar yerleştikçe alt düzey tasarımsal aktivitelere yer vermek daha zor hale gelmektedir.

Değişen bir gereksinimi karşılayabilmek için, çalışan sisteme etki yaratmamak niyetiyle var olan kaynak kodlar kopyalanabilmekte, yazılım karmaşıklığı ve servis sayısı artmaktadır. Yazılımcıların aynı işlevsel modül üzerinde eş zamanlı çalışabilmesi zorlaşmaktadır.

Yeni geliştirilen uygulamalarda tasarımcılar da alışkanlıklardan dolayı üst düzey mimari tasarıma yoğunlaşmaktadırlar. Bu davranış zamanla kurum kültürüne dönüşmektedir.

4 Alt Düzey Tasarım için Referans Modeli

Alt düzey tasarımın etkinleştirilmesi amacı ile basit bir referans modeli önerilmektedir. Modelin şu amaçlarla kullanılması hedeflenmektedir:

1. Eski bir sistemin yeniden yapılandırma süreci sonrasında oluşmuş olan alt düzey mimari eksikliklerin ve dolayısıyla kalıtım yoluyla aktarılmış yordamsal kodların yarattığı yan etkilerin giderilmesi için yeniden yapılandırma çalışmalarının yönlendirilebilmesi
2. Sistemde tamamen yeni geliştirilecek yazılımlar için alt düzey mimari tasarımı etkinliklerinin zorunlu hale getirilebilmesi.

Referans modelinin odak noktası, nesneye yönelimli tasarımlar için paket tasarım prensipleri ve sınıf tasarımlarının ilk beş prensibi olarak bilinen ilkelere uyumlu bir çerçeve oluşturarak bağımlılıkları yönetebilen ve tasarıma uygun yazılım geliştirilmesini kolaylaştırmaktır [6,7]. Bu referans modeli temel olarak şu iki konuyu uygulayarak hedefe ulaşmayı amaçlamaktadır:

1. Doğru paket yapısının oluşturulması ve
2. Tasarım desenlerinin kullanılması.

4.1 Doğru Paket Yapısı

Tasarım etkinliklerinin sistemi yukarıdan aşağıya doğru alt sistemlere bölerek ilerleme doğasına uyumlu olarak, bileşen içerisindeki işlevlerin gruplanması gerekmektedir. İş bakış açısı ile paketlerin kendi içinde bütünlüğünü sağlamak en önemli ölçütlerden biridir. İş açısının oluşturacağı yönlendirme, doğal olarak modülerlik prensiplerini de destekliyor olmalıdır. Paketler, bağımlılık prensipleri gözetilerek oluşturulmalıdır [6, 7]. Paket bağımlılık prensipleri, yüksek tutunum (high cohesion) ve gevşek bağlaşım (low coupling) prensiplerine göre temellenirler [6, 7]:

Tutunum ile ilgili olanlar:

- Yayımlama Yeniden Kullanılabilirlik Denkliği Prensibi-YYP (Release Reuse Equivalency Principle): Yeniden kullanılabilir en küçük birim yayımlanabilen en küçük birimdir.
- Genel Kapalılık Prensibi-GKP (The Common Closure Principle): Birlikte değişen sınıflar birlikte paketlenirler.
- Genel Yeniden Kullanılabilirlik Prensibi-GYP (The Common Reuse Principle) : Birlikte Kullanılan sınıflar birlikte paketlenirler.

Bağlaşım ile ilgili olanlar:

- Çevrimsiz Bağımlılık Prensibi-ÇBP (The Acyclic Dependencies Principle) : Paketler arasında çevrim oluşturan bağımlılıklar bulunmamalıdır.
- Dengeli Bağımlılık Prensibi-DBP (The Stable Dependencies Principle) : Paketler az değişiklik gösteren paketlere bağımlı olmalıdır.

- Dengeli Soyutlama Prensibi-DSP (The Stable Abstractions Principle) : Çok az değişiklik gösteren paketler soyut paketler olmalıdır. Soyut sınıflar bu paketlerde toplanabilir. Az değişiklik gösteren paketler somut paketler tarafından genişletilebilirler.

Paket yapısının doğru oluşturulması aşağıdaki faydaları sağlar:

- Yazılım Kalite öznitelik gereksinimlerinin daha etkin olarak karşılanabilmesi,
- Bileşendeki işlevlerin daha çabuk kavranabilmesi,
- İşlevlerin bağımsız konuşlandırılabilmesi ve lisanslanabilmesi,
- Yazılımcıların aynı modül içinde eş zamanlı çalışabilmesi.

İşlev paketlerinin altında alt işlevler var ise onlar için de alt düzey paket yapısı oluşturulması faydalıdır. Gün sonu işlemleri ve veri tabanı erişimleri için bir alt paket bulundurulur.

4.2 Tasarım Desenleri

Tasarım desenleri, yaygınca tekrarlanan bir tasarım sorununu çözebilen yeniden kullanılabilir çözüm kalıplarıdır [5]. Bu özellikleri ile yazılım kalite metrikleri üzerinde olumlu etkileri bulunmaktadır [15]. Baştan belirlenmiş tasarım desenlerinin içerildiği bir şablonun kullanılması, ilk beş tasarım prensibine uygun yazılım geliştirmeyi en başta destekleyen bir etki yaratmaktadır. Bu prensipler, değişikliklere daha kolay uyum sağlayabilen ve daha kolay bakımı yapılabilen yazılımlar geliştirebilmeyi hedefleyen mimari odaklı prensiplerdir. İngilizce isimlerinin baş harfleri kullanılarak oluşturulmuş S.O.L.I.D kısa adı ile anılırlar ve şu şekilde tanımlanırlar [6,7]:

- Tek Sorumluluk Prensibi (**S**ingle Responsibility Principle): Sınıflar tek bir işlevle sorumlu olmalıdır.
- Açık/Kapalılık Prensibi (**O**pen/Close Principle): Sınıflar genişlemeye açık değişikliğe kapalı olmalıdır.
- Liskov Yerine Geçme Prensibi (**L**iskov Substition Principle): Somut sınıflar ana sınıfın yerine davranışı farklılaştırmadan geçebilmelidir.
- Arayüz Ayrımı Prensibi (**I**nterface Segregation Principle): Arayüzler ince olmalı, az işlev tanımlamalıdır.
- Bağımlılıkların Ters Çevrilmesi Prensibi (**D**ependency Inversion Prensibi): Bağımlılıklar soyutlamalar üzerinden tanımlanmalı, somut sınıflara dayanmamalıdır.

Tablo 2'de yazılım kalite özniteliklerinin paket ve S.O.L.I.D prensiplerince nasıl eşleştiğinin bir listesi sunulmuştur:

Tablo 2. Kalite öznitelikleri ve S.O.L.I.D prensipleri eşlemeleri

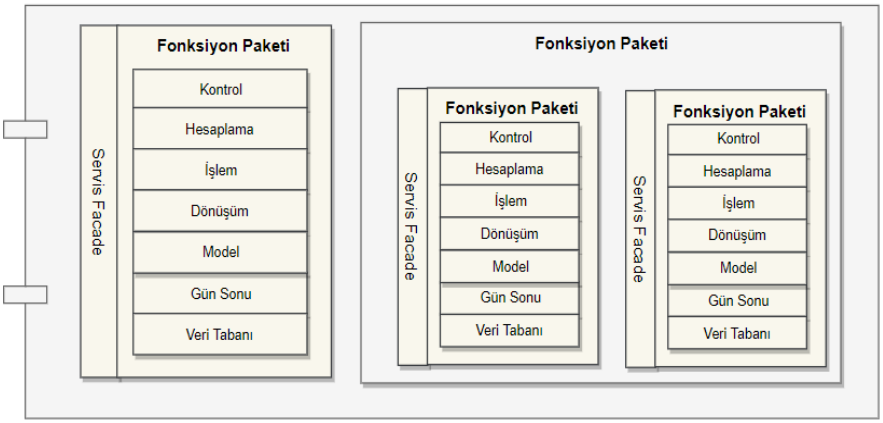
Öznitelik	Paket Prensipleri	S.O.L.I.D Prensipleri
Kullanılabilirlik Güvenilirlik	-	-
Yeniden kullanılabilirlik	YYP, GKP, GYP	S, I

Genişletilebilirlik	ÇBP, DBP, DSP	O, L, I, D
Bakım yapılabilirlik	YYP, GKP, GYP, ÇBP, DBP, DSP	S, O, L, I, D
Ölçeklenebilirlik	-	S, I

Bu tasarım desenlerinin kullanımının, özellikle geç zamanlarda ortaya çıkan değişiklik baskıları sonucunda yapılacak işlemlerde yan etkilerin azalması ve bu değişikliklerin yönetiminin kolaylaşması yönünde yararlı olduğu gözlemlenmiştir.

4.3 Referans Modeli

Bileşen Paket Ayrıştırması. Referans Modelinin uygulandığı modül bileşenlerinin paket prensiplerine uygun ayrıştırılması Şekil 2 ‘de verilmiştir. Paket ayrıştırılması zaman içinde geliştirilen projelerde oluşturulmuş benzer ve ortak yapılar olarak belirlenmiştir. Bu tür yapılar için yaygın örnekler aşağıda listelenmektedir:



Şekil. 2. Bileşen paket bölümlemesi

- Fonksiyon / Alt Fonksiyon Paketi: Bileşende iş bakış açısı ile kendi içinde bütünlüğü olan işlev gruplarıdır.
- Servis Facade: Bir fonksiyon / alt fonksiyon paketinin SYM’ye uyumlu olarak, sistemin diğer bileşenlerine sunduğu servislerin kaynak kodda giriş noktalarının sunulduğu paketlerdir.
- Kontrol: İşlevlerle ilgili gerekli kontrol mantıklarının içerildiği yapılardır. Bu kontrol yapıları kurumdaki hemen hemen her uygulama modülünde bulunmaktadır. Örnek olarak, verilerin doğruluk kontrolü gibi mantıklar burada kurgulanabilir.
- Hesaplama: İşlevlerle ilgili hesaplamaların içerildiği yapılardır. Uygulamalarda, hemen her modülde hesaplamalarla ilgili mantıklar bulunmaktadır. Faiz hesaplamaları, yatırım getiri hesaplamaları gibi bir takım özel hesaplamalar örnek olarak verilebilir.

- İşlem: İşlevlerle ilgili iş mantıklarının barındırıldığı yapılardır. Kredi başvuru işlemi gibi örnekler verilebilir.
- Dönüşüm: Veri modellerinin oluşturulması ve başka bir modele dönüştürülmesi ile ilgili yapılar.
- Model: İşlevlerle ilgili verilerin modellendiği yapılar.
- Gün Sonu: Bankacılık uygulamalarında, günün sonunda toplu iş görevleri çalıştırılarak işlemler yaptırılır. Vadeli mevduat bitişlerinin belirlenerek faiz getirilerinin hesaplara yatırılması örneği verilebilir.
- Veri Tabanı: İşlevlerle ilgili olarak veri tabanı tablolarına erişim sınıfları.

Önerilen Tasarım Desenleri. Referans modelinde belirlenmiş işlevler için belli başlı tasarım desenlerinin kullanılması öngörülmektedir. Bu işlevlerde hangi desenin kullanılacağı, zaman içinde geliştirilen projelerde oluşturulmuş çözümlerin incelenmesi ile belirlenmiş ve referans modeli kapsamında genelleştirilmiştir. Bu sayede, referans modelinin uygulamalarda kullanılabilecek hazır çözüm kalıpları şablonunu oluşturduğundan bahsedilebilir.

Aşağıda önerilen tasarım kalıplarının bir listesi verilmektedir:

- Bağımlılık enjeksiyonu (dependency injection-DI) ve kontrolün ters çevrilmesi (Inversion of Control - IoC) desenlerinin kullanılması [4]: Bağımlılıklar soyut sınıflar (arayüzler) ile tanımlanarak azaltılır. Bağımlı olan nesneye bağımlı olunan nesne çalışma zamanında geçirilmiş olur.
- Servislerin kaynak kodunda giriş noktalarının Facade tasarım örüntüsü ile sunulması [5]: Bu örüntü tasarım ve iş mantığını gerçekleyen kaynak kodlarının değiştirilmesini kolaylaştırmaktadır. Servis havuzunda değişiklik gerektirmez. Servisin hizmet ettiği uygulamaları, kod ve tasarım iyileştirmeleri etkinliklerinin yan etkilerinden korumaktadır. Her işlev paketi içinde bir service Facade kullanımı önerilir.
- Kontrol işlemlerinde Sorumluluk Zinciri (Chain of Responsibility-CoR) tasarım deseni kullanılır [5].
- Hesaplama işlemlerinde Dekoratör (Süsleyici/Decorator) tasarım deseni kullanılır [5].
- İşlevsel gereksinimleri karşılayan işlemler için Dekoratör (Süsleyici/Decorator) tasarım deseni kullanılır [5].
- Veri modelini bir veri tipinden oluşturma veya veri modelleri arasındaki dönüşüm işlemleri için Kurucu (Builder) tasarım deseni kullanılır [5].
- Kontrol ve hesaplama işlemleri gibi sınıfların yaratılması için Fabrika tasarım deseni (Factory) kullanılır [5].
- Veri tabanı erişimi için Uyarlayıcı (Adapter) tasarım deseni kullanılır [5]. Uyarlayıcılar veri tabanına yazma ve veri tabanından okuma uyarlayıcıları olarak çeşitlendirilir. Uyarlayıcılar kullanılarak yazılım veri tabanından soyutlanabilmektedir.

Tablo 3'te referans modelinde kullanımı belirlenen tasarım desenlerinin S.O.L.I.D prensipleri açısından bir değerlendirilmesi sunulmuştur:

Tablo 3. Referans Model'de önerilen tasarım deseninin karşıladığı S.O.L.I.D prensibi

Önerilen Tasarım Deseni	S.O.L.I.D Prensipleri
Bağımlılıkların enjeksiyonu ve IoC	I, D
Facade	I, D
Sorumluluk Zinciri	S, O, L, I, D
Dekoratör	S, O, L, I, D
Fabrika	S, O, I, D
Builder	S, I, D
Uyarlayıcı	S, I, D

5 Uygulamada Referans Modelinin Değerlendirilmesi

Bu çalışmanın yön verildiği kurumda, bir alandaki iş modüllerine ait bileşenlerde alt düzey tasarım etkinlikleri için referans modeli uygulama süreci son birkaç senedir devam etmektedir.

Üst düzey tasarım çalışmasında eski bileşenlerde yazılım geliştirilmesine karar verildiğinde referans modeli bu bileşenlerde uygulanmaya başlanmış ve kalıtımsal yollarla aktarılan kodlara yeniden yapılandırma (refactoring) çalışması yapılmıştır. Yeni bir bileşen yaratıldığında ise bileşende kalıtımla aktarılan kod bulunmadığından referans modeli en baştan uygulanmıştır.

5.1 Yazılım Kalite Ölçüt Analizi:

Yazılımın kalitesi, önceden belirlenerek ölçülen özelliklere göre değerlendirilir [9]. Kurumda belirlenmiş ölçütler kapsamında, bir kod kalite analiz aracı olan SONAR kullanılarak yazılımların ölçümlemesi yapılmaktadır.

Türev Teminat projesi kapsamında, Risk modülünde referans modeli uygulanmaya başlanmıştır. Projenin geliştirilmesinden önceki Risk modülündeki yazılım kalite ölçüt değerleri, referans modeli uygulanarak geliştirilen proje sonrasındaki değerler ile karşılaştırılmıştır. Referans modeli kullanımının yazılım kalite ölçüt değerlerindeki etkisi Tablo 4'te özetlenmiştir:

Tablo 4. Yazılım kalite ölçüt analizi

Ölçüt	Etki	Sayısal değişim (%)
Kaynak kod satır sayısı	Artış	15.82
Sınıf sayısı	Artış	7.09
Sınıf kod satır sayısı	Azalma	34.16
McCabe karmaşıklık ölçütü	Azalma	8.09
McCabe sınıf karmaşıklık ölçütü	Azalma	26
Paket çevrimleri	Yeni yaratılmamış	-

Paket bağımlılıkları	Yeni yaratılmamış	-
Paket çevrim sayısı	Azalma	43.17
Bulgu sayısı	Azalma	11.75
Teknik borç (Technical debt)	Yeni yaratılmamış	-

Tabloya göre kaynak kod satır sayısında artış olmasına rağmen, sınıf kod satır sayısında, karmaşıklık değerlerinde, paket çevrim sayılarında ve bulgu sayılarında azalma görülmektedir. Sınıf sayısının artması ve sınıf kod satır sayısının az olması sınıfların S.O.L.I.D prensipleri olan 'Tek Sorumluluk Prensibi' ve 'Arayüz Ayrımı Prensibi' ile uyumludur. Sınıf kod satır sayısının, karmaşıklıkların, paket çevrim ve bağımlılıkları sayılarının az olması durumunda bakımı daha kolay yapılabilir ve modüler bir yazılımdan söz edilebilir.

Bulgu sayısındaki azalma ve teknik borç yaratılmama ölçütü ise, referans model kullanımı ile hata üretme oranının azaldığı konusunda bir fikir vermektedir.

5.2 Üretim Ortamına Geçiş Zamanlaması:

Sektör genelinde bankalarda ortak fon altyapısına geçiş projeleri geliştirilmiştir. Projeler Türk Lirası (TL) Fonu Geçişi ve Yabancı Para (YP) Fonu işlevlerinin eklenmesi projeleri olarak iki fazda planlanmıştır.

TL Fon Geçiş Projesinde, mevcutta sistemde olan TL Fonu modülü referans modeli uygulanarak ortak fon yapısına geçirilmiştir. Bu kapsamda müşteriye yatırım fonu alış ve satış akışının yönetildiği modüller üzerinde referans modeli ile uyumlu bir kod ve veri tabanı modeli oluşturulmuştur.

YP Fonu Geçiş Projesinde, ilk proje kapsamında referans modeli uygulanarak geliştirilmiş olan Fon Modülü üzerinde, referans modelinin kullanımı devam ettirilerek YP fon işlevleri ortak fon yapısı için geliştirilmiştir. Yabancı Para Fonu projesi, TL fonu projesinin aksine tamamen referans modeli uygulanmış bir modül üzerinde geliştirilmiştir.

Tablo 5'te, iki projede yakın sayılarda ekran ve tablo etkisi olmasına rağmen, yabancı para fonu geçişinin daha az maliyetle gerçekleştiği görülmektedir.

Tablo 5. Proje Maliyet Karşılaştırması

Proje	Maliyet (Adam/Gün)	Ekran sayısı	Veritabanı tablo sayısı
TL Fon projesi	960	24	18
YP Fon Projesi	320	21	14

5.3 İşlevsel Olmayan Gereksinimlerin Karşılanması:

İşlevsel olmayan gereksinimlerin referans modeli tarafından karşılanması iki yöntemle incelenebilir:

1. Tablo 1'de işlevsel olmayan gereksinimlerin kalite öznitelikleri ile ilişkisi verilmiştir. Tablo 2 ve Tablo 3'te bahsedilen kalite özniteliklerinin referans modelinin temel aldığı prensipler tarafından karşılanmasına yer verilmiştir. Bu kapsamda, işlevsel olmayan gereksinimlerin referans modeli ile karşılanmasının etkisi tablolar değerlendirilerek yapılabilir.
2. Tablo 4 ve Tablo 5'te verilen analizler bakımından uygulamada referans modelinin işlevsel olmayan gereksinimlere etkisi değerlendirilebilir.

Aşağıda bu değerlendirmelere yer verilmiştir:

- Başarım/Yük Oranı: Tablo 4'te verilen bulgu sayısının azalmasına bağlı olarak sistemin kullanılabilirliğinde artış olmaktadır. Servislerin daha az hatalı çalışmasına bağlı olarak sistemin başarım oranında artış gözlemlenebilir.
- Gereksinimlerin sık değişmesi: Tablo 1, Tablo 2 ve Tablo 3'ten yola çıkılarak referans modelinin sık değişen gereksinimleri ele alma konusunda olumlu etkisi olduğu söylenebilir. Tablo 4 sonuçları için yapılan değerlendirmeler bağlamında, referans modeli daha kolay bakımı yapılabilir bir sistem oluşturulması sağlamaktadır. Bu kapsamda, değişikliklerin yapılması kolaylaşmaktadır.
- Pazara çıkma süresinin kısalığı: Tablo 1, Tablo 2 ve Tablo 3'ten yola çıkılarak referans modelinin pazara çıkma süresinin kısalmasında olumlu etkisi olduğu söylenebilir. Tablo 5'te sunulan sonuçlara bağlı olarak, referans modeli kullanımı ile sürenin kısaldığı görülmektedir. Yazılım kalite ölçüt analizindeki değerlendirmeler kapsamında, yapılan geliştirme ve değişikliklerin belli bir alanı etkilediğinden bahsedilebilir. Bu hem geliştirme hem de test süresinde kısalma sağlamaktadır.
- Gelişen sistem: Tablo 1, Tablo 2 ve Tablo 3'ten yola çıkılarak referans modelinin bu gereksinimi karşıladığı söylenebilir.

5.4 Gözlemlenen Kazanımlar:

Referans modelinin kullanımı yazılımlarda belli prensiplerin uygulanmasını zorunlu hale getirmektedir. Bu zorunluluk ile birlikte şu görünen kazanımları elde ettiğimizden bahsedebiliriz:

- Yordamsal kodların sayısı azalmaya başlamıştır. Tablo 4'teki sınıf sayısının artması ve sınıf kod satır sayısının azalması bu konuda bir gösterge olarak düşünülebilir. Bununla birlikte, yazılımcılarda yapısal düşünme, yukarıdan aşağıya tasarım bakış açısı ve tasarım desenleri kullanım becerilerinin gelişmekte olduğu gözlemlenmektedir.
- Bileşenler içinde yeniden kullanılabilir sınıflar oluşmaktadır. Tablo 5'teki değerlendirmeleri temel alarak, referans modeli oluşturulduktan sonra geliştirilen proje, ilk projedeki bazı sınıfları kullanarak üretim ortamına geçiş sürecini hızlandırmıştır.
- Kodların daha kolay anlaşılabildiği ve tasarımın koddan daha kolay saptanabildiği gözlemlenmektedir. Tablo 4'teki kalite ölçütlerindeki iyileşmeler bu gözlemi desteklemektedir. Çevik süreçlerin uygulandığı ve sürdürülen detaylı dokümantasyonun

olmadığı kurumda bakım süreçleri kolaylaşmaktadır. Hatalara neden olan kod blokları, hata ayıklayıcı program daha az kullanılarak ve daha kısa sürede tespit edilebilmektedir.

- Yeni geliştirmelerin çok büyük bir kod bloğundaki iş akışını dikkate almaya gerek kalmadan gerçeklenebildiği ve yazılımın daha kolay genişletilebildiği gözlemlenmektedir. Daha az yan etki üreten kod geliştirmenin kolaylaşmasına bağlı olarak, bulgu sayılarının ve etki testi yapılma süresinin azaldığı gözlemlenmiştir. Tablo 4'teki kalite ölçütlerindeki iyileşmeler ve Tablo 5'teki süreler bu gözlemi desteklemektedir.

6 Geçiş Sürecinin Yazılımcılar Üzerindeki Etkileri

Yeni bir modele geçişin yazılım geliştirme kültüründe değişiklik ve yeni konuların öğrenilmesi ihtiyaçlarını zorladığından, benimsenmesinin zaman aldığından bahsedilebilir.

Organizasyonda uzun yıllardır çalışan yazılımcılar için modele uyum sürecinde başlangıçta zorluklar gözlemlendiği söylenebilir. Pazara çıkma zaman baskısının da etkisi ile alışılageldik şekilde yazılım geliştirme yaklaşımının daha hızlı bir çözüm olarak kabul görmesi, geçiş sürecinin daha uzun zamanda gerçekleşmesine neden olmaktadır.

Organizasyona yeni katılan yazılımcıların referans modelini uygulamalarının, alt düzey tasarım etkinliklerine daha kolay uyum sağlamalarında yardımcı olduğu söylenebilir. Bununla birlikte, idame ettirilmekte olan ve kalıtımla aktarılmış kodların etkisi onlarda da gözlemlenmektedir.

Zaman içerisinde referans modelinin kullanılmasının devam ettirilmesi ve kalıtımsal yolla aktarılan kodların azalması ile birlikte referans modelinin daha etkin kullanılmaya başlandığından söz edilebilir.

7 İlgili Çalışmalar

Çalışmanın kapsamında sunulan referans modelinin kullanımı ile hem eski sistemlerin alt düzey mimari eksikliklerinin tamamlanması hem de yeni tasarlanacak sistemlerde alt düzey tasarım etkinliklerinin uygulanması hedeflenmektedir. Referans modeli, bu hedefe yazılım prensipleri ve tasarım desenlerinin belirlendiği şablon bir model sunarak ulaşmaya çalışmaktadır.

Yeniden yapılandırma çalışmaları, sistemlerin yeniden tasarlanması veya eski sistemden tersine mühendislik yöntemi ile yeniden yapılandırma olarak gerçekleştirilebilmektedir.

Eski sistemlerin yeniden yapılandırma sürecine yönelik olarak, referans modelinin ele aldığı gibi tasarım desenleri kullanılarak ileri doğru yeniden yapılandırma yaklaşımını kullanmakta olan savunma ve medikal gibi farklı sektörlere ait çalışmalar bulunmaktadır [9, 10, 11, 12]. Yeniden yapılandırma çalışmalarında tasarım desenleri kullanımlarının yazılım kalitesini olumlu yönde etkilediği ile ilgili çalışmalar da bulunmaktadır [13]. Bu çalışmalarda yazılımların genişleyebilirlik ve bakım yapılabilirlik özel-

likleri vurgulanmaktadır. Diğer yandan, tersine mühendislik yöntemi kullanılarak, yordamsal kodlardan nesneye yönelik kodları üretebilen çalışmalardan da bahsedilmektedir [14].

Referans modeli, yeniden yapılandırma çalışmalarının yanı sıra sıfırdan geliştirilecek yazılımlar için de alt düzey mimari etkinliklerinin yapılmasını sağlayabilecek bir model olarak önerilmesi açısından bu çalışmalardan farklılık göstermektedir.

8 Sonuç ve Gelecek Çalışmalar

Alt düzey tasarım etkinliği, sistemin kalite gereksinimlerini karşılamakta önemli bir etmendir. Daha esnek ve yeniden kullanılabilir yapı taşlarının kullanılması ile desteklenmektedir.

Kurumsal sistem yeniden yapılandırma sürecinde alt düzey tasarım etkinliklerinin ötelenmesi, yaşayan sistemde birçok olumsuz yan etki yaratmaktadır.

Alt düzey tasarım etkinliklerinin gerçeklenmesi için bir referans modelinin kullanımı süreci kolaylaştırmaktadır. Önerilen referans modeli, Paket ve S.O.L.I.D [6,7] prensipleri ile belli başlı tasarım desenleri [5] kullanımını temel almaktadır.

Referans modelinin kullanıldığı iş modülünde pazara çıkma zamanında kısalma, değişen gereksinimlere hızlı uyum sağlayabilme ve hataların daha az yan etki ile çözülebilmesi kazanımları olmuştur. Yaşayan sistem için de kurum kültürünün oluşmasında olumlu etkisi bulunmaktadır.

Referans modelinin kullanımı, kalıtımsal yollarla aktarılan yordamsal kodlara dayanan sistemler için bir dönüşüm süreci gerektirir. Yeni kurulan bir sistemde kodlar sıfırdan yazıldığından modelin kullanımına uyum göreceli olarak daha kolay olmaktadır.

Mevcut süreçte modele uyum, tasarımcıların kontrolü ve yönlendirmesine bağımlıdır. Kişiye bağımlılığın azaltılması amacı ile modelin otomasyona geçirilmesi gelecekte planlanabilir. Model şablonunu oluşturarak bileşenler içinde kullanılmasını sağlayan bir yazılım, tasarımcılar üzerindeki yükü hafifleterek zaman kazanılmasına yardımcı olabilir. Yazılımcıların modele uyum yeteneklerini kendi başlarına geliştirmelerine olanak sağlayabilir.

Referans modeli, kurumda oluşan gereksinimden geliştirilmiş ve uygulanmış bir modeldir. Ancak, kapsadığı nitelikler ile diğer sektörlerde de uygulanabilme potansiyeline sahiptir. Bu bağlamda gelecek çalışma konuları olarak, modelin diğer sektörlerde uygulama denemelerinin yapılarak sonuçların karşılaştırılması değerlendirilebilir.

Kaynakça

1. https://www2.deloitte.com/content/dam/Deloitte/global/Documents/About-Deloitte/central-europe/ce-digital-banking-maturity-study-emea.pdf?nc=1, son erişim 2018/09.
2. O'Brien L., Bass L., Merson P.: Quality Attributes and Service-Oriented Architectures, CMU/SEI-2005-TN-014.
3. Kaur P., Sing H.: An Analytical Review of Quality Attributes of Service-Oriented Architecture, Trends in Information Management (TRIM), ISSN: 0973-4163, pp. 40-50, (2014).
4. https://martinfowler.com/articles/injection.html. , son erişim 2018/09.

5. Gamma E., Helm R., Johnson R., and Vissides J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
6. Design Principles Page Papers by Bob Martin, http://condor.depaul.edu/dmumaugh/OOT/Design-Principles, son erişim 2018/11.
7. http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod, son erişim 2018/10.
8. IEEE, 'Standards for a Software Quality Metrics Methodology', P-1061-1998 (R2009), IEEE Press, New York, 2005.
9. Barkataki S., Harte S., Dinh T., Reengineering A Legacy System Using Design Patterns and Ada-95 Object-Oriented Features, http://www.sigada.org/conf/sa98/papers/barkataki.pdf, son erişim 2018/11
10. Re-engineering Legacy Code with Design Patterns:A Case Study in Mesh Generation Software,https://pdfs.sematicscholar.org/1aa8/ba2e3cd596f3c501e0966530ab24c6bb190e.pdf, son erişim 2018/11.
11. Jung-Eun Cha, ChulüHong Kim, Young-Jong Yang, Architecture Based Software Reengineering Approach for Transforming from Legacy System to Component Based System through Applying Design Patterns, International Conference on Software Engineering Research and Applications, SERA 2003: Software Engineering Research and Applications pp 266-278.
12. Usability of Software Architecture Design Pattern in Medical Process Re-engineering Model, http://www.ijaiem.org/Volume2Issue6/IJAIEM-2013-06-26-084.pdf, son erişim 2018/11.
13. On the Role of Design Patterns in Quality-Driven Re-engineering, http://www.stargroup.uwaterloo.ca/pubs/conf/Tahvildari_et_al_CSMR02.pdf, son erişim 2018/11.
14. A SURVEY OF OBJECT IDENTIFICATION IN SOFTWARE RE-ENGINEERING, http://www.sis.uta.fi/cs/reports/pdf/A-1998-6.pdf, son erişim 2018/11.
15. A Quality Model for Design Patterns, https://www.researchgate.net/publication/249885094_A_Quality_Model_for_Design_Patterns, son erişim 2018/11