PL/SQL Programları İçin Veri Tabanı Bağımlılık Analizi

Ersin Ersoy[1], Metin Altınışık[1], Hasan Sözer[2]

[1] Turkcell, İstanbul, Türkiye
{ersin.ersoy, metin.altinisik}@turkcell.com.tr
[2] Özyeğin Üniversitesi, İstanbul, Türkiye
hasan.sozer@ozyegin.edu.tr

Özet. PL/SQL dili ile yazılan programlar, veri tabanı üzerinde prosedür ve fonksiyon objeleri, ve bu objelerin bir kümesini barındıran paket objeleri olarak geliştirilmektedirler. Bu objeler yoğun olarak tablo ve görünüm gibi veri tabanı objelerini kullanmaktadırlar. Mevcut analiz araçları ile her bir objenin hangi diğer objelere bağımlılığı olduğunu görmek mümkündür. Ancak bu bilgi paket seviyesinde sağlanıp, paketler içindeki her bir prosedür ve fonksiyonun hangi veri tabanı elemanlarını kullanıldığı bilgisine ulaşılamamaktadır. Özellikle uzun yıllardır idame edilen programlarda, paketler çok fazla sayıda prosedür ve fonksiyon barındırmaktadır ve bu paketlerin belirli zamanlarda parçalanması idame edilebilirlik açısından fayda sağlamaktadır. Bu amaçla programların yeniden yapılandırılması, değişikliklere ilişkin etki analizlerinin yapılabilmesine destek sağlayacak bir analiz aracı geliştirilmiştir. Bu araç, paketler içerisinde yer alan prosedür ve fonksiyonların kullandıkları ortak veri tabanı tablolarını tespit edebilmekte ve böylece değişiklik etki analizi ile tasarım kararlarına destek olmaktadır. Geliştirdiğimiz analiz aracı, bir teknoloji şirketindeki müşteri ilişkileri yönetimi sistemine uygulanmıştır.

Anahtar Kelimeler: Yazılım Mimarisi, Etki Analizi, Yazılım İdamesi, PL/SQL.

Database Dependency Analysis for PL/SQL Programs

Ersin Ersoy[1], Metin Altınışık[1], Hasan Sözer[2]

[1] Turkcell, Istanbul, Turkey
{ersin.ersoy, metin.altinisik}@turkcell.com.tr
[2] Ozyegin University, Istanbul, Turkey
hasan.sozer@ozyegin.edu.tr

Abstract. PL/SQL programs are composed of procedure and function objects deployed on a database. These objects can be grouped into a set of package objects and they extensively use database objects such as tables and views. Existing analysis tools can detect which objects are dependent on which other objects. However, this information is available only at the package level. It is not possible to detect database dependencies of procedures and functions that are encapsulated in packages. Existing packages might include many procedures and functions and they might have to be refactored to improve software maintainability, especially in the case of legacy systems that are maintained for years. In this work, we developed a dependency analysis tool to support software refactoring and impact analysis. This tool detects database dependencies of procedures and functions taking place in packages. It supports change impact analysis and design decisions by detecting database tables commonly accessed by various objects. We applied our tool on a customer relations management system maintained by a technology firm.

Keywords: Software Architecture, Impact Analysis, Software Maintenance, PL/SQL.

1 Giriş

Modülerlik, yazılım tasarımında gözetilen önemli niteliklerden biridir [1]. Büyük ölçekli sistemlerde, idame ettirilebilirlik açısından bu nitelik daha da fazla önem arzetmektedir. Modülerliği sağlamak amacıyla, sistemler birbirleri arasındaki bağımlılığın az olduğu birimlerden oluşacak şekilde tasarlanmaktadırlar. Böylece sistem üzerinde yapılması gereken değişiklikler bu birimlerin kapsamında değerlendirilebilmektedir. Birimler arası bağımlılıklar ne kadar az olursa, değişikliklerin diğer birimlere bir etkisi olma ihtimali o kadar az olmaktadır. Sonuç olarak hata ortaya çıkma riski ve idame maliyetleri azalmaktadır.

Yazılım birimleri arasındaki bağımlılıkları sistematik olarak tespit edebilme problemi literatürde uzun yıllardır çalışılan bir konudur [2,3,4,5,9]. Bu sayede, yazılım sistemindeki bir birim üzerinde yapılan değişikliklerin diğer birimler üzerine olan etkileri analiz edilebilmektedir [6,7]. Tespit edilen bağımlılıklar yazılım mimarisnin dokümantasyonu veya yeniden yapılandırılması için kullanılabilmektedir [8,9,13,14].

Yazılım birimleri arasındaki bağımlılıklar farklı şekillerde olabilmektedir [14,15,16,17,18]. Genel olarak literatürde raporlanan çalışmalar ağırlıklı olarak direkt bağımlılıkları göz önünde bulundurmuştur. Bu bağımlılıklar statik [16] ve/veya dinamik [17] analizler ile tespit edilen, bir birimin diğer birimin elemanlarına doğrudan erişimi, kullanımı, çağrımı [15] şeklinde ortaya çıkmaktadır. Oysa bazı yazılım birimleri ortak kullanılan kaynaklardan ötürü dolaylı olarak bağımlıdırlar [14].

Dolaylı bağımlılık endüstride yaygın olarak kullanılan PL/SQL programlarında özellikle sık rastlanan bir durumdur [13,14]. PL/SQL programları veri tabanı üzerinde prosedür ve fonksiyon objeleri, ve bu objelerin bir kümesini barındıran paket objeleri olarak geliştirilmektedirler. Bu objeler birbirlerini doğrudan kullanmasalar da, bazı veri tabanı tabloları üzerinde ortak çalışabilirler. Mevcut analiz araçları ile her bir objenin hangi diğer objelere bağımlılığı olduğunu görmek mümkündür. Ancak bu bilgi paket seviyesinde sağlanıp, paketler içindeki her bir objenin hangi veri tabanı elemanlarını kullanıldığı bilgisine ulaşılamamaktadır.

Daha önceki çalışmamızda paketler arasındaki ve herhangi bir paket içinde yer almayan objeler arasındaki veri bağımlılıkları değerlendirilmiştir [13,14]. Biz bu çalışmamızda bir PL/SQL programı içerisindeki, herhangi bir paket içinde yer alsın veya almasın, tüm objeler için ortak kullanılan veri tabanı elemanlarını tespit eden bir veri bağımlılık analiz aracı geliştirdik. Bu araç PL/SQL programları için yazılım bağımlılık analizi, değişiklik analizi gerçekletirebilmeyi mümkün kılmakta ve böylece tasarım kararlarına destek olmaktadır.

Geliştirdiğimiz analiz aracı, bir teknoloji şirketindeki müşteri ilişkileri yönetimi sistemine uygulanmıştır. Bu uygulama sonucunda, bazı objelerin veri bağımlılıklarından ötürü mevcut duruma göre farklı paketler içinde yer alması gerektiği görülmüştür. Bu tespitler idame maliyetlerini azaltmak üzere yazılım mimarisinin tekrardan yapılandırılması için değerlendirilmektedir.

Bildirinin organizasyonu şu şekildedir: Bir sonraki bölümde literatürde yayınlanmış ilgili çalışmalar özetlenmektedir. 3. Bölüm'de kısaca PL/SQL programları tanıtılmaktadır. 4. Bölüm'de geliştirdiğimiz analiz aracı ve kullandığımız yöntemler anlatılmaktadır. 5. Bölüm'de modülerlik değerlendirmesi için kullanılan ölçütler açıkla-

mıştır. 6. Bölüm'de analiz aracın endüstriden bir vaka çalışması ile değerlendirilmesi sunulmuştur. 7. Bölüm'de sonuçlar özetlenerek bildiri sonlandırılmaktadır.

2 İlgili Çalışmalar

Bu bölümde, ilgili çalışmalar hakkında bilgiler verilmektedir. Özellikle de bu çalışmaya benzerlik gösteren ve PL/SQL programları üzerine uygulamaları bulunan çalışmalar özetlenerek bu çalışmadan farkları vurgulanmıştır. Literatürde yazılım bağımlılık analizi ile ilgili birçok farklı çalışma yayınlanmıştır. Bu çalışmalar, yazılım mimarisinin tekrar elde edilmesi [10], görselleştirilmesi [11] veya iyileştirilmesini [12] amaçlamıştır. Ancak önerilen tüm analiz araçları C/C++ ve Java dili ile geliştirilmiş yazılım sistemlerine ve yazılım modülleri arasındaki doğrudan bağımlılıklara odaklanmıştır. PL/SQL programları ve veri bağımlılığı gibi dolaylı bağımlıklıkların analizi üzerine çok az sayıda çalışma bulunmaktadır.

Daha önceki çalışmalarımızda [13] [14], PL/SQL prosedür ve fonksiyonların otomatik olarak gruplanması problemine odaklanılmıştır. Her iki çalışmada da herhangi bir paketten bağımsız olarak oluşturulmuş prosedür ve fonksiyonların belirli bir paket ya da paketler altında birleştirilmesi ile ilgili yazılım geliştiricilere yol gösterilmesi amaçlanmıştır. Bu çalışmada ise hali hazırda bir paket içinde yer alan prosedür ve fonksiyonların kullandıkları, yani bağımlı oldukları tablo ve görünümleri belirlemek ve bu bilgiler ışığında sistemin modülerliğini iyileştirmek amaçlı yapılabilecek çalışmalar ile ilgili tasarımcılara yol göstermek amaçlanmaktadır.

Yoğun olarak veriye erişim içeren uygulamalardaki modüllerin veri elemanlarına olan bağımlılıkları tespit etmek için statik kod analizi tekniklerini geliştirilmiştir [19]. Bu teknikler ile kullanılmayan elemanların belirlenerek temizlenmesi ve böylece verimlilik artışı amaçlanmıştır. Benzer bir başka çalışma da bankacılık sektörü özelinde yapılmıştır [20]. Ancak bu iki çalışmada da veri bağımlılık analizi paket seviyesinde yapılmaktadır. Analiz paketler içerisindeki prosedür ve fonksiyonlar seviyesinde olmadığından bizim çalışmamızdan ayrışmaktadır.

Bu çalışmalar dışında bizim çalışmamıza benzer kullanılabilecek Cocolab PL/SQL Analiz [21] ve Trivadis PL/SQL & SQL Kod Analiz [22] araçları bulunmaktadır. Bu araçlar prosedür ve fonksiyon seviyesinde veri bağımlılık analizi gerçekleştirebilmektedirler. Ancak her iki uygulamanın da analiz yapılacak bilgisayar üzerine kurulması ve bu şekilde çalıştırılması gerekmektedir. Bizim çalışmamız kapsamında geliştirilen aracın kendisi de PL/SQL dili kullanılarak geliştirildiğinden sadece bir kereye mahsus olarak veri tabanına yerleştirilmekte ve sonrasında yetkisi olan tüm kullanıcılar tarafından ek bir kuruluma gerek kalmadan kullanılabilmektedir.

Bir sonraki bölümde, PL/SQL programları hakkında temel bilgiler verilmektedir. PL/SQL programlarını diğer programlardan ayıran özeliklere kısaca değinilmektedir.

3 PL/SQL Programları

PL/SQL (Procedural Language/Structured Query Language) [23], SQL dilini prosüdürel dillerin özellikleri ile birleştiren 3. nesil bir programlama dilidir. Oracle[1] veritabanı yönetim sistemini kullanan büyük ölçekli finans kuruluşlarında, telekomunikasyon operatörlerinde ve devlet kurumlarındaki yazılım sistemlerinin büyük bir kısmı PL/SQL dili ile geliştirilmiştir.

Genel olarak PL/SQL programları paketler altında gruplanmış prosedür ve fonksiyonlardan oluşmaktadır. Ayrıca herhangi bir paket altında gruplanmamış tekil prosedür ve fonksiyonlar da olabilmektedir. Tüm prosedür, fonksiyon ve paketler derlendikten sonra ilgili veritabanının bir parçası olarak veritabanında kaydedilmektedirler ve bu veritabanına bağlanan uygulamalardan doğrudan çağrılabilmektedirler. Şekil 1'de bir PL/SQL prosedürünün 3 ana kısımdan oluşan temel blok yapısını gösteren bir örnek bulunmaktadır.

```
1 PROCEDURE OranEkle ( p_id IN NUMBER) IS
2 v_satis NUMBER;
3 v_toplam NUMBER;
4 v_oran NUMBER;
5 BEGIN
6 SELECT satis , toplamDeger
7 INTO v_satis , v_toplam
8 FROM t_sonuc_tab WHERE sonuc_id = p_id ;
9 v_oran := v_satis/ v_toplam ;
10 IF v_oran > 25 THEN
11 INSERT INTO t_karsilastirma_tab
12 VALUES ( p_id , v_oran ) ;
13 END IF ;
14 COMMIT;
15 EXCEPTION
16 WHEN ZERO DIVIDE THEN
17 INSERT INTO t_karsilastirma_tab
18 VALUES ( p_id , -1) ;
19 COMMIT;
20 WHEN OTHERS THEN
21 ROLLBACK;
22 END;
```

Şekil 1. PL/SQL bloklarının genel yapısı.

Tanımlama kısmı (Satır 1-4) uygulama akışında kullanılan değişkenlerin tanımlandığı kısımdır. Bu kısım eğer programda kullanılan herhangi bir değişken yoksa zorunlu değildir. Çalıştırılabilir kısım, BEGIN ifadesi ile başlayıp END ifadesi ile sonlanan (Satır 5-22) kısım olup, programın ana iş akışını içermektedir. Bu kısım tüm PL/SQL prosedürleri için zorunludur. Hata kontrol kısmı, EXCEPTION ifadesi ile başlar (satır 15-22) ve çalıştırabilir kısımdaki olası hataları yönetmektedir.

Kurumsal seviyedeki uygulamalar genelde çok büyük ve karmaşık uygulamalardır. Bu nedenle PL/SQL programlarında genelde bu karmaşıklığı daha iyi yönetmek için

[1] www.oracle.com

paket yapıları kullanılmaktadır. Paketler fonksiyon ve prosedürleri gruplamak için kullanılan objelerdir.

Paketler tanımlama ve gövde olmak üzere iki kısımdan oluaşmaktadır. Tanımlama kısmı paketin arayüz kısmıdır. Bu kısımda değişkenler, sabitler, hata tipleri, dışardan çağrılabilecek olan fonskiyon ve procedurlerin tanımı yapılır. Şekil 2'de standart bir paket tanımının genel bir görümü gösterilmiştir.

```
1 CREATE OR REPLACE PACKAGE
2 package name PCK MUSTERI
3 type records ,
4 index_by_t a b l e s
5 sabitler
6 hatalar
7 global değişkenler
8 procedure musteriEkle ( arg1 , . . . ) ;
9 function musteriSorgu ( arg1 , . . . )
10 return datatype ;
11 . . .
12 END package name;
```

Şekil 2. PL/SQL standart paket spesifikasyon kısmı.

Paketlerin gövde kısmında ise prosedür ve fonksiyonların asıl kod kısımları bulunmaktadır. Paket gövdesinin genel bir görünümü Şekil 3'de gösterilmiştir.

```
1 CREATE OR REPLACE PACKAGE BODY
2 package PCK_MUSTERI
3 PROCEDURE musteriEkle ( arg1 , . . . ) IS
4 BEGIN
5 . . .
6 EXCEPTION
7 . . .
8 END musteriEkle ;
9 FUNCTION musteriSorgula( arg1 , . . . )
10 RETURN data type IS
11 resultvariable data type
12 BEGIN
13 . . .
14 RETURN resultvariable;
15 EXCEPTION
16 . . .
17 END musteriSorgula ;
18 . . .
19 END PCK_MUSTERI ;
```

Şekil 3. PL/SQL standart paket gövde kısmı.

4 PL/SQL Veri Bağımlılık Analizi Aracı

Bu bölümde PL/SQL paketlerinin içerisinde yer alan fonksiyon ve prosedürlerin analiz edilmesi amacıyla geliştirdiğimiz analiz aracı anlatılmaktadır. Oracle veri tabanlarında objelerin birbiri ile ilişkilerini analiz etmek üzere geliştirilmiş olan veri sözlük

görünümleri, PL/SQL programları için sadece paket seviyesinde bilgi vermektedir. Paketlerin içinde yer alan her bir prosedür ve fonksiyon için bu bilgi ancak göz ile kontrol edilerek elde edilebilir. Büyük ölçekli yazılımlar için bu yöntem elverişli değildir. Tüm prosedür ve fonksiyonların veri bağımlılıklarını otomatik olarak elde edebilmek için bir araç geliştirdik. Geliştirdiğimiz araç bir PL/SQL programının kaynak kodunu statik olarak analiz etmektedir. Analiz, aşağıdaki algoritmada listelenen adımlardan oluşmaktadır:

Algoritma 1. PL/SQL prosedürleri ve fonksiyonları için veri bağımlılık analizi.

```
0   GİRDİ: Veri tabanı Şeması, S
1   ÇIKTI: BAĞIMLILIK LİSTESİ, B
2   B ← Ø
3   P ← S içindeki tüm paketler
4   Her p ∈ P için
5     F ← p içindeki prosedürler / fonksiyonlar
6     Her f ∈ F için
7       f.b ← f başlangıç satır numarası
8       f.s ← f bitiş satır numarası
9     Son
10    T ← p içinde geçen tablo ve görünüm bağımlılıkları
11    Her t ∈ T için
12      t.a ← tablo / görünüm adı
13      t.i ← işlem tipi
14      t.s ← t satır numarası
15      Her f ∈ F için
16        Eğer f.b < t.s < f.s ise
17          B ← B + (f, t.a, t.i)
18        Son
19      Son
20    Son
21  Son
```

Birinci adımda paket içindeki prosedürler, Oracle veri tabanının hali hazırda sağladığı kütüphane görünümlerinin otomatik analizi ile bulunmaktadır (Satır 5). Bundan sonraki her bir adımda da Oracle veri tabanının hali hazırda sağladığı kütüphane görünümleri (ALL_SOURCE, ALL_DEPENDENCIES vb.) kullanılmıştır. Sonrasında bulunan her bir prosedürün paket içindeki başlangıç ve bitiş satırları bulunmaktadır (Satır 6-9). Her bir prosedür, "PROCEDURE" ya da "FUNCTION" anahtar kelimesi sonrasında prosedür adı gelecek şekilde tanımlanmaktadır. Bitiş satırı için ise "END" anahtar kelimesi zorunludur. Bu kelimenin ardından ilgili prosedürün adı dayazılabilir (Örnek: **Şekil 3**), ancak bu zorunlu değildir (Örnek: **Şekil 1**). Bu nedenle son satırlarında "END" anahtar kelimesi sonrası prosedür adı yazılmayanlar prosedürleri bulmak için ek analiz gerektirmektedir. Sonraki adımda, paketin bağımlı olduğu (kullandığı) tablo ve görünümler bulunmaktadır (Satır 10). Bu aşamada, SELECT cümleciklerinde birden fazla tablo kullanılması durumunda, sadece ilk tablo dikkate alınmıştır. Ayrıca, PL/SQL dilinin yapısından dolayı bir komut cümleciği tek bir satırda olmak zorunda değildir (Örnek: **Şekil 1**). Dolayısıyla, doğru

komut cümleciğini bulmak için analiz ettiğimiz her satır kendinden önceki ve sonraki satırlarla birlikte ele alınmalıdır. Bu durum, tekrarlayan bilgilerin oluşmasına neden olmaktadır. Bu bilgiler bir sonraki adıma geçmeden önce temizlenmektedir. Bulunan her bir tablo ve görünüm bağımlılığının paket içinde hangi satırlarda ve hangi işlem ile (sorgulama, güncelleme, vs.) kullanıldığı bulunmaktadır (Satır 12-14). Son olarak, bulunan satır numaraları kullanılarak, bağımlılığın paket için hangi prosedürlerden kaynaklandığı bulunmakta ve bu bilgi kaydedilmektedir (Satır 15-18).

Bir sonraki bölümde, prosedürlerin veri bağımlılıklarından yola çıkarak, PL/SQL paketleri arasındaki bağımlılıkların ve paketlere ilişkin modülerliğin değerlendirmesi için kullanılan ölçütler açıklanmaktadır.

5 Bağımlılık ve Modülerlik Ölçütleri

Bir önceki bölümde tanıtılan aracımız tarafından tespit edilen veri bağımlılıklarını dikkate alarak yazılım mimarisi üzerinde bir bağımlılık (coupling) ve uyumluluk (co-hesion) alanizi gerçekleştirmek için çeşitli ölçütler kullanılmıştır. Öncelikle, her bir prosedür, *f* için dışa bağımlılık (*BD*) ölçütü aşağıdaki gibi hesaplanmıştır.

$$BD_f = \begin{cases} 0, & |T_f| = 0 \\ \dfrac{\sum_{\forall f' \notin p_f} |T_f \cap T_{f'}|}{|T_f|}, & |T_f| > 0 \end{cases}$$

Burada, T_f f prosedürü tarafından kullanılan tablo kümesini, p_f ise bu prosedürün dahil olduğu paketi ifade etmektedir. Hiçbir pakete dahil olmayan prosedürlerin yapay bir paketin parçaları oldukları varsayılarak heseplamalar yapılmıştır. Bir fonksiyon tarafından ve bu fonksiyon ile aynı pakete dahil olmayan fonksiyonlar tarafından ortak kulanılan tablo sayısı, fonksiyon tarafından kullanılan toplam tablo sayısına bölünerek hesaplama yapılmıştır. Bu ölçüt ile dışa bağımlılığı fazla olan prosedürler tespit edilebilmektedir.

Her paket çifti *p1* ve *p2* için bağımlılık (*B*) ölçütü aşağıdaki gibi hesaplanmıştır.

$$B_{p1,p2} = \sum_{f1 \in p1}^{|p1|} \sum_{f2 \in p2}^{|p2|} (f1 \text{ ve } f2 \text{ tarafından kullanılan ortak tablo sayısı})$$

Burada $|p|$, p paketi içerisinde yer alan prosedür sayısını, f ise her bir prosedürü ifade etmektedir. *B* ölçütünün hesaplanması için iki farklı paket içerisinde yer alan her bir prosedür çifti için ortak kullanılan tabloların sayısı hesaplanarak toplanmıştır.

Her bir paket için Uyumsuzluk ölçütü (UX) ise 0 ile 1 arasında bir değer verecek şekilde, LCOM (Lack of Cohesion Metric) ölçütünden esinlenerek [24] aşağıdaki gibi tanımlanmnıştır.

$$UX_p = \frac{\frac{\sum_{t \in T_p} F_p \text{ içinde } t \text{ tablosunu kullanan prosedür sayısı}}{|T_p|} - |F_p|}{1 - |F_p|}$$

Burada, T_p p paketi tarafından kullanılan tablo kümesini, F_p ise p paketi kapsamında yer alan prosedürlerin kümesini ifade etmektedir. UX değeri ne kadar 0 değerine yakın olursa paket o kadar uyumlu, 1 değerine ne kadar yakın olursa paket o kadar uyumsuz demektir. Bir sonraki bölümde, bu ölçütler kullanılarak analiz aracımızın bir endüstriyel vaka çalışması kapsamında kullanımı ve elde edilen sonuçlar anlatılmaktadır.

6 Endüstriyel Vaka Çalışması

Vaka çalışmamız bir iletişim ve teknoloji şirketindeki müşteri ilişkileri yönetimi veri tabanı şemasının analizi üzerinden yapılmıştır. Söz konusu şema içindeki 88 paket, bu paketlerin içindeki 671 prosedür bağımlı oldukları 203 adet tablo ve görünüm objesi baz alınarak analiz edilmiştir. Sonuç olarak **Tablo 1**'de örnek olarak gösterilmiş formatta analiz ettiğimiz şema için toplam 2032 bağımlılık tepit edilmiştir.

Tablo 1. Örnek olarak bir kısmı listelenmiş olan bağımlılık bilgileri.

ŞEMA	İSİM	REFERANS	PAKET	PROSEDÜR	İŞLEM
CRM	PK42_P332	T149	PK42	P332	INSERT
CRM	PK42_P330	T149	PK42	P330	SELECT
CRM	PK40_P310	T129	PK40	P310	DELETE
CRM	PK40_P310	T129	PK40	P310	INSERT
CRM	PK39_P300	T110	PK39	P300	UPDATE

Tespit edilen bağımlılıklardan yola çıkılarak, bir önceki bölümde tanıtılmış olan ölçütler hesaplanmıştır. Hesaplama sonuçları Tablo 2'de görülmektedir. Yer kısıtı nedeniyle matrisin belirli kısımları eklenmiştir. Bu tabloda her bir satır ve kolon birer pakete tekabül etmektedir. Ortaya çıkan 88x88 matristeki her bir satır ilgili paketin diğer paketlerle olan bağımlılık değerleridir. Köşegende ise uyumsuzluk değerleri bulunmaktadır. İyi seviyede bir modülerlik için düşük bağımlılık ve uyumsuzluk değeri arzulanmaktadır. Örnek olarak bu bakış açısından 30 nolu paket için sonuçların olumlu çıktığı görülebilir. Ancak 15 nolu paket için bir yeniden yapılandırma ihtiyacı olduğu görülmektedir.

Tablo 2. Hesaplama sonuçları.

	1	**2**	**3**	**5**	**15**	**26**	**27**	**28**	**29**	**30**	**31**
1	0.83	0.00	0.33	0.33	0.33	0.67	0.33	0.33	0.67	0.00	0.33
2	0.00	0.93	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.33
3	1.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00
4	0.50	0.00	0.50	0.25	0.50	0.50	0.50	0.50	0.50	0.00	0.50
5	0.00	0.50	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00
6	0.13	0.25	0.13	0.13	0.13	0.13	0.13	0.13	0.13	0.00	0.13
7	0.13	0.50	0.13	0.13	0.13	0.13	0.25	0.13	0.13	0.00	0.25
8	0.33	0.17	0.17	0.17	0.17	0.33	0.17	0.17	0.33	0.00	0.17
9	0.11	0.67	0.11	0.11	0.11	0.11	0.22	0.22	0.11	0.00	0.33
10	0.25	0.38	0.13	0.13	0.13	0.25	0.13	0.25	0.25	0.00	0.13
11	0.15	0.00	0.08	0.08	0.08	0.23	0.08	0.23	0.15	0.00	0.15
12	0.50	0.00	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.00	0.50
13	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.13	0.13	0.13	0.13	0.13	0.25	0.13	0.25	0.13	0.00	0.38
15	**1.00**	**0.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**	**1.00**
16	0.25	0.25	0.25	0.25	0.25	0.50	0.25	0.75	0.25	0.00	0.50
17	0.50	0.00	0.50	0.50	0.50	1.00	0.50	1.00	0.50	0.00	1.00
19	0.00	0.00	0.00	0.00	0.50	0.00	0.00	0.00	0.00	0.00	0.00
20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
21	0.13	0.13	0.13	0.13	0.00	0.13	0.13	0.13	0.13	0.00	0.13
22	0.20	0.20	0.20	0.20	0.13	0.40	0.40	0.60	0.20	0.00	0.60
23	0.14	0.43	0.14	0.14	0.20	0.14	0.29	0.43	0.14	0.00	0.43
24	0.00	0.00	0.00	0.00	0.14	0.00	0.00	0.00	0.00	0.00	0.00
25	0.40	0.00	0.20	0.20	0.00	0.92	0.20	0.40	0.40	0.00	0.40
26	0.50	0.50	0.50	0.50	0.20	0.50	0.75	0.50	0.50	0.00	1.00
27	0.14	0.00	0.14	0.14	0.50	0.29	0.14	1.00	0.14	0.00	0.29
28	1.00	0.00	0.50	0.50	0.14	1.00	0.50	0.50	0.60	0.00	0.50
29	0.08	0.17	0.08	0.08	0.50	0.17	0.17	0.17	0.08	0.00	0.87
30	**0.00**	**0.00**	**0.00**	**0.00**	**0.08**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
31	0.25	0.25	0.25	0.25	0.00	0.50	0.50	0.75	0.25	0.00	0.75

7 Sonuçlar ve Gelecek Çalışmalar

Geliştirdiğimiz analiz aracı ile daha önce oldukça üst seviye de kalan bağımlılık analizinin geliştirtirilmesi sonucunda paket içindeki prosedür ve fonksiyonların bağımlılıkları görünür hale gelmiştir. Bu detayda bağımlılık bilgisini görünür hale getirmek ile sistemin idame maliyetlerine katkı sağlayacak aynı zamanda bütünlüğü koruyacak aksiyonların alınması kolaylaşmıştır.

PL/SQL programlarında, bazı prosedürler ve fonksiyonlar bir paketin gövdesinde yer almakla birlikte, bu paketin spesifikasyon bölümünde tanımlanmamış olabilirler. Gerçekleştirdiğimiz analizlerde bu tip prosedürler ve fonksiyonlar göz ardı edilmiştir. Buna ek olarak, SELECT cümleciklerinde birden fazla tablo kullanılması durumunda sadece ilk tablo dikkate alınmıştır. Analiz ettiğimiz müşteri ilişkileri yönetimi şemasında bu durumların sayısı nispeten az olduğu için analiz sonuçlarına etkisi göz

ardı edilebilir ancak çalışmamızı belirttiğimiz durumların daha yaygın olduğu şemalara da uygulamak istediğimizden, bu konuları iyileştirme planlanmaktadır.

Yapmış olduğumuz ilk incelemelerde, bazı prosedüler ve fonskiyonların modülerlik açısından yer almaları gerektikleri paketlerde yer almadıkları fark edilmiştir. Bu sebeple, yazılım mimarisinin yeniden yapılandırılarak iyileştirilmesi gerektiği ortaya çıkmıştır. Gelecekte, prosedüler ve fonskiyonları bağımlılıklarına göre otomatik olarak gruplamak üzere kümeleme yöntemlerinin kullanılması planlanmaktadır. Bu şekilde, göz ile fark edemediğimiz başka iyileştirme alanlarınının da tespit edilmesi mümkün olacaktır.

Kaynaklar

1. Parnas, D.L.: On the criteria to be used in decomposing systems into modules.Communications of the ACM 15(12), 1053–1058 (1972).
2. Chen, Y.: Reverse engineering. In: Practical Reusable Unix Software, Krishnamurthy, B. (ed), chapter 6, pp. 177–208, New York: John Wiley & Sons (1995).
3. Chen, Y., Gansner E., Koutsofios, E.: A C++ data model supporting reachability analysis and dead code detection. IEEE Transactions on Software Engineering 24(9), 682–694 (1997).
4. Korn J., Chen Y., Koutsofios E.: Chava: Reverse engineering and tracking of Java applets. In: Proceedings of the 6th Working Conference on Reverse Engineering, pp. 314–325, Atlanta, GA, USA (1999).
5. Sangal, N., Jordan E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture.In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications, pp. 167–176, San Diego, CA, USA (2005).
6. Arnold, R.S.: Software change impact analysis. IEEE Computer Society Press, Los Alamitos, CA, USA (1996).
7. Li, B., Sun, X., Leung, H. and Zhang, S.: A survey of code-based change impact analysis techniques. Software Testing, Verification and Reliability, 23, 613–646 (2013).
8. Mens, T., Tourwe, T.: A survey of software refactoring. IEEE Transactions on Software Engineering 30(2), 126–139 (2004).
9. Ducasse, S., Pollet, D.: Software architecture reconstruction: A processs oriented taxonomy. IEEE Transactions on Software Engineering 35(4), 573–591 (2009).
10. Guo, G., Atlee, J. Kazman, R.:. A software architecture reconstruction method. In: Proceedings of the 1st Working Conference on Software Architecture, pp. 15-34, Deventer, The Netherlands, (1999).
11. Koschke, R.: Software visualization in software maintenance, reverse engineering, and reengineering: a research survey. Journal of Software Maintenance and Evolution: Research and Practice, 15(2), 87-109 (2003).
12. Ferrante, K.O.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems, 9(3), 319-349, (1987).
13. Ersoy E., Kaya, K., Altınışık, M., Sözer, H.: Using hypergraph clustering for software architecture reconstruction of data-tier software. In: Proceedings of the 10th European Conference on Software Architecture, pp. 326-333, LNCS, vol. 9839, Copenhagen, Denmark (2016).

14. Altınışık, M., Sözer, H.: Automated procedure clustering for reverse engineering PL/SQL programs. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1440–1445, ACM, Pisa, Italy (2016).
15. Offutt, J., Harrold M.J., Kolte P.: A software metric system for module coupling. Journal of Systems and Software, 20(3), 295–308 (1993).
16. Briand, L., Daly, J., Wust, J: A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 25(1) 91-121(1999).
17. Arisholm, E., Briand L., Foyen A.: Dynamic coupling measurement for object-oriented software. IEEE Transactions on Software Engineering, 30(8) 491-506 (2004).
18. Kirbas S., Hall T., Sen A.: Evolutionary coupling measurement: Making sense of the current chaos. Science of Computer Programming, 135, 4-19 (2017).
19. C. Nagy. Static analysis of data-intensive applications. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on, pages 435-438.IEEE, (2013).
20. Nagy, C., Ferenc, R. and Bakota, T., 2011, September. A true story of refactoring a large oracle pl/sql banking system. In Industrial Track of the 8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2011).
21. Cocolab PL/SQL Analiz, www.cocolab.com/app_plsql.html. Erişim Haziran 2017.
22. Trivadis PL/SQL Analiz, www.salvis.com/blog/2013/07/28/tvdca-0-5-1-beta-released/. Erişim Haziran 2017.
23. Oracle Database PL/SQL Language Reference, https://docs.oracle.com/cloud/latest/db112/LNPLS/toc.htm. Erişim Haziran 2017.
24. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493 (1994).