T.C.

MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YAZILIM HATA LOGLARI KULLANILARAK VERİ MADENCİLİĞİ UYGULAMASI GERÇEKLEŞTİRİLMESİ

GÖKHAN ACUN

Yüksek Lisans Tezi

Tez Danışmanı

Yrd. Doç. Dr. Turgay Tugay Bilgin

İSTANBUL – 2014

T.C.

MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YAZILIM HATA LOGLARI KULLANILARAK VERİ MADENCİLİĞİ UYGULAMASI GERÇEKLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

GÖKHAN ACUN

Tez Danışmanı

Yrd. Doç. Dr. Turgay Tugay Bilgin

İSTANBUL – 2014

ÖZET

Yüksek Lisans Tezi, Yazılım Hata Logları Kullanılarak Veri Madenciliği Uygulaması Gerçekleştirilmesi, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Bu tez çalışmasında bir kurumsal kaynak planlama yazılımına ait günlük hata loglarını veritabanından çekip birtakım önişleme işlemleri gerçekleştirerek, elde edilen veri üzerinde birliktelik analizi algoritması çalıştıran bir yazılım geliştirilmiştir. Bu yazılım sayesinde, bir hatanın oluşmasına yol açan koşulların birliktelik analizi gerçekleştirilerek elde edilen sonuçları yönetim kademesine e-posta gönderen bir yapı kurulmuştur. Ayrıca, hataya sebep olan sık örüntüler tespit edilmiş ve görselleştirilmiştir.

Bu tez beş bölümden oluşmaktadır. Birinci bölümünde tez ile ilgili genel kavramlar açıklanmıştır. İkinci bölümde literatür taraması sonucu derlenen çalışmalara yer verilmiştir. Üçüncü bölümde günlük hata loglarının veritabanından çekilmesinden, veri önişleme adımının uygulanmasından, Apriori algoritmasının ve Graphviz yazılımının uygulanmasından sonra ilgili kişilere hata analizi hakkında bilgi verilmesinden bahsedilmektedir. Dördüncü bölümde de çalışma sonunda elde edilen sonuçlar değerlendirilmiştir.

Bu tez 2014 yılında tamamlanmıştır ve 84 sayfadan oluşmaktadır.

Anahtar Kelimeler: Veri Madenciliği, Log Madenciliği, Hata Madenciliği, Data Mining, Log Mining, Bug Mining

ABSTRACT

Master Thesis, Performing a Data Mining Application using Software Bug Logs, T.C. Maltepe University, Institute of Natural Sciences, Department of Computer Engineering.

In this master thesis, a software has been developed, performing some data preprocessing steps taking daily bug logs about an Enterprise Resource Planning software from a database. This software executes association rule mining on data. After association rule mining, E-mail about the results of the conditions of the bugs is sent to management position. Moreover, the frequent patterns, causing bug, has been detected and visualized.

This thesis has five sections. In the first section, general concepts of the thesis have been mentioned. In the second section, the previous studies about the general concepts have been presented. In the third section, it is explained that daily bug logs are taken from database, data-preprocessing has been implemented, Apriori algoritm and Graphiz software have been applied and bug analysis has been informed to related people through e-mail. In the last section, the results of the study have been discussed and interpreted.

This thesis has been completed in 2014, and consists of 84 pages.

Keywords: Veri Madenciliği, Log Madenciliği, Hata Madenciliği, Data Mining, Log Mining, Bug Mining

TEŞEKKÜR

Tez konusunu seçmemde beni yönlendiren, tez süreci boyunca destek ve yardımlarını esirgemeyen, değerli bilgilerinden istifade ettiğim danışman hocam Yrd. Doç. Dr. Turgay Tugay BİLGİN'e, maddi ve manevi desteğini benden hiçbir zaman esirgemeyen çok değerli aileme ve çalışmalarım sırasında emeği geçen herkese teşekkürlerimi sunarım.

İÇİNDEKİLER

ŞEKİLLER

Sayfa No

ÇİZELGELER

Sayfa No

1. GİRİŞ

Bu tez çalışmasında yazılım hata logları kullanılarak bir veri madenciliği uygulaması gerçekleştirilmiştir. Yazılım sistemlerinde yer alan hata depoları (repository) eğer doğru değerlendirilirse, önemli bir bilgi kaynağıdır. Veri madenciliği, kullanışlı olabilecek ilginç bilgileri keşfetmek için bu depolar üzerine uygulanabilir. Bu bilgiler sayesinde geliştirim ekipleri gelecek yazılım kurgularını ve sürümlerini doğru planlayabilirler.

Bu çalışmada bir kurumsal yazılım şirketinde kullanılan yazılım sisteminin hataları değerlendirilmiştir. Bir tabloda saklanmakta olan bu hatalar tez kapsamında geliştirilen yazılım aracılığıyla günlük olarak çekilerek daha önceden belirlenen bir hata kritiklik sınıflandırması uygulanmıştır. Daha sonra hataların belli bir yüzdedeki birliktelikleri bulunmuştur. Bu birliktelikler graf yapısı aracılığıyla görselleştirilmiştir. Bu görsel, belli kişilere e-posta yoluyla gönderilebilmiştir. Böylece istenilen kişiler günlük olarak kritik hatalar hakkında bilgilendirilebilmiştir.

Uygulamanın günlük olarak çalışabilmesi için Windows servis desteğiyle otomatik bir yapı kurulmuştur. Ayrıca ilgili uygulamanın Windows servis dışında uygulama formu üzerinden yönetilebilen bir türevi de kurgulanmıştır.

1.1 Veri Madenciliği

Veri madenciliği, bilgisayar bilimlerinin disiplinler arası bir sahasıdır [1]. Veritabanlarında bilgi keşfi (Knowledge-Discovery in Databases) olarak da bilinir [1]. Veri madenciliği; yapay zeka, makine öğrenmesi, istatistik ve veritabanı sistemlerinin kesişiminde yöntemleri olan ve büyük miktarlardaki veri setlerinde örüntüler keşfetmenin hesaplamalı bir sürecidir. Veri madenciliği sürecinin nihai amacı bir veri setinden bilgi üretmek ve onu daha ileri kullanımlar için anlaşılabilir bir veriye dönüştürmektir [1]. Son yıllarda popüler olmaya başlamıştır. Birliktelik kuralı madenciliği, veri madenciliğinin en aktif araştırma sahalarından birisidir.

Veri madenciliği; genellikle örüntü eşleme ya da diğer neden-sonuç tekniklerini kullanarak daha önceden bilinmeyen verilerden bilgi çıkarma sürecidir. Veri noktaları arasındaki bağlantıları saptamak için depodaki büyük miktardaki veri üzerinde işleyen hesaplama gücünü kullanır. Veri madenciliğinin siber güvenliğin yanı sıra ulusal güvenliği sağlamada da birçok uygulaması vardır. Ayrıca şüpheli insanların bulunmasında, bankalara daha iyi performans sağlamada, yeni müşteriler kazanmada, müşterilerin satın alma davranışlarını incelemede de faydaları vardır. Kredi kartı dolandırıcılığı ve biyometrik uygulamalar için uygulanabilir [2].

Veri madenciliği, veritabanı ve yapay zeka teknolojilerinin bir kombinasyonudur. Yapay zeka kavramı son on yılda ciddi bir sıçrama yapmıştır, ancak halihazırda var olan bilgisayar bilimlerine de önemli bir katkı sağlayabileceğini de kanıtlamıştır. Aslında birçok uzman, veri madenciliğinin internet ve veri ambarcılığından sonra endüstrideki en sıcak üçüncü saha olduğuna inanmaktadır [3].

Veri madenciliği gerçekte veriyi analiz etme sürecinden sonraki adımdır. Ancak sadece sorguları ve kullanıcı tanımlı ilişkileri elde etmek yerine bir adım ileri giderek verideki anlamlı ilişkileri de bulmaktadır. Bu ilişkiler veriyi incelemede daha derin bir bakış açısı kazandırmaktadır.

Örneğin, bir bilgisayar tarafından oluşturulmuş grafik, kullanıcının anlaması için zor olabilir, ama veri madenciliği kullanıcıya verideki akımların tam olarak ne olduğunu daha net bir şekilde gösterir. Veri madenciliği, ek bir bilgiye ihtiyaç duymadan veritabanındaki veriye değer katmaktadır. Bu durum hem teknik olan hem de teknik olmayan kullanıcıların daha iyi cevaplar alabilmelerini, daha doğru karar verebilmelerini ve şirketlerin ciddi bir mali tasarruf yapabilmelerini sağlamaktadır [3].

1.1.1 Önemli Kullanımlar

Bilim ve Mühendislik: Veri madenciliği, bilim ve mühendislik alanlarında son yıllarda geniş şekilde kullanılmaktadır, örneğin, biyoinformatik, genetik, tıp, eğitim ve elektrik mühendisliği vb. Önemli amaçlarından birisi, insan DNA sırasındaki çeşitlilik ve hastalık eğilimi arasındaki ilişkiyi anlamaktır. DNA yapılarındaki değişikliklerin kanser gibi yaygın hastalıkları nasıl etkilediğini araştırmaktır. Bu görevi gerçekleştirmek için kullanılan tekniğe çok faktörlü boyut indirgeme (multifactor dimensionality reduction) denir [4].

İş Dünyası: Veri madenciliği, şirketlerin insan kaynakları departmanlarına fayda sağlamaktadır. İnsan kaynakları departmanları, başarılı çalışanları veri madenciliği sayesinde tespit edebilirler. Mesela başarılı çalışanların mezun olduğu üniversiteleri tespit edebilirler ve işe almalarda odaklarını bu yöne yönlendirebilirler. Ayrıca kazanç ve marj payı hedefleri gibi hedeflerini üretim planları ve iş gücü seviyesi gibi operasyonel kararlara çevirebilirler.

Veri madenciliği, ayrıca bir şirkete müşteri ilişkileri yönetiminde de ciddi derecede değer katar. Herhangi bir şirket çağrı merkezinden rastgele bir müşteri seçip, e-posta göndermekten daha ziyade bir teklife cevap vermesi muhtemel olan müşterileri belirleyip çabalarını bu yönde harcayabilir. Kampanyalarda kaynaklarını optimize edebilmek için daha gelişmiş yöntemleri kullanabilirler, bu sayede bir bireyin bütün potansiyel teklifler arasından hangisine cevap vereceğini tahmin edebilirler. Ayrıca veri kümeleme de müşteri gruplarını ve kümelerini keşfetmede kullanılabilir.

1.2 Yazılım Madenciliği

Yazılım madenciliği, yazılım kurgularını anlamayı sağlayan yazılımları modernleştirme alanında bir bilgi keşfi uygulamasıdır. Bu süreç tersine mühendislik kavramıyla ilişkilidir [5]. Varlık-bağıntı modeli, herhangi bir yazılımdan elde edilen bilgiyi sunmanın sıkça kullanılan bir yöntemidir. Object Management Group (OMG),

herhangi bir kodun bilgi keşfini gerçekleştirme amacıyla yazılımsal varlıkları ve ilişkileri için bir ontoloji tanımlayan Bilgi Keşfi Metamodeli tanımı geliştirmiştir [5].

Yazılım mimarileri iş ve ticaret için ciddi derecede değer sağlar. Bu yüzden yazılım madenciliği veri madenciliğiyle de yakın ilişki içindedir. Yazılım sistemlerinden bilgi çıkarırken sistem tarafından işlenen veri değil de sistemin yapısı ve davranışları gözlemlenmektedir. Yazılım mühendisliği bireysel veri setleri yerine veritabanı şemaları gibi meta verileri dikkate almaktadır.

Yazılım madenciliği, işleyişini farklı seviyelerde gerçekleştirebilir. Örneğin, program seviyesinde olabilir Bu sayede bağımsız değişkenleri ve ifadeleri inceleyebilir. Yazılımın prosedürleri ve fonksiyonlarının çağrılmasını, alt sistemler ve arayüzleri gibi yazılımın mimari seviyesini, veri depolama yapısını, uygulamanın akış yapısını veya yazılımı ticari olarak inceleyebilir.

Yazılım depo madenciliği, hem temel hem de değerli bilgiyi çıkarmayı içermektedir. Bu madencilik türü aktif bir alandır ve yazılım geliştirimi ve bakımını desteklemek için kullanılır. Araştırmacılar, yöneticiler gibi çeşitli kişiler, farklı amaçlarla gerçek bilgi üretebilmek için bu depolar üzerinde veri madenciliği gerçekleştirirler [6].

1.3 Hata Madenciliği

Yazılımdaki kusurlara hata (bug) denir [6]. Hatalar, yazılım geliştirirken beklenmedik şekillerde oluşan davranışlardır. Yazılım testçileri ve kalite mühendisleri tarafından yazılımlar test edilirken bu tarz beklenmedik şekilde oluşan davranışlar hata olarak işaretlenir. Hatalar; Bugzilla, Perforce, JIRA vb. araçlarla yönetilirler ve izlenirler.

Hataları bulmak bazen zordur. Statik analiz araçları daha önceden tanımlı hataları bulmada sorun yaşamazlar, ancak yeni hata tiplerini öngörmek ve bulmak, ilgili programlama dilini derinlemesine kavrayan bir anlayış gerektirir [7]. Derinlemesine olmadan daha genel olarak yapılan gözden geçirmeler daha fazla zaman gerektirirler.

Hata raporlarındaki metinsel veriler, hata tamiri sürecinde yazılım geliştiricilerine önemli bilgiler sağlar [8]. Hata veritabanı madenciliği hakkındaki daha önceki çalışmalar üç temel hususa odaklanmaktadır [8]:

- Hata tamirini doğru kişiye atamak
- Tekrarlı hata raporlarını bulmak
- Raporlanmış bir hataya doğru şiddeti atamak

Bir yazılım hata deposu, sadece yazılım hataları hakkında bilgi içermez, yazılım geliştiricilerinin, kalite mühendislerinin (testçiler), yöneticilerin ve diğer ekip üyelerinin yazılım hatalarını çözmek için verdikleri çaba hakkında da bilgi içermektedir. Bu bilgi yararlı bilgi örüntüleri çıkarabilmek için analiz edilebilir [9].

Bazen yönetici ve ekip liderleri bazı soruları kendilerine sormak zorunda kalırlar. "Bu yeni gelen yazılım hatasını kime atasam acaba?" veya "Benzer yazılımsal hatayı daha önce çözmüş bir kişi var mı?" bu tarz sorulardır. Aslında ekip liderleri hataları çözerken ekip üyelerinin iş yükünü ve daha önce ilgili hataya yakın benzerlikte hataları çözüp çözmediğini göz önünde bulundururlar. Bu da elle gerçekleştirilen bir süreçtir. Yani yazılım geliştiricilerine hata raporlarını atamak, zaman kaybına sebep olan sıkıcı bir iştir.

Herhangi bir yazılım hata deposu yazılım hataları hakkında bilgi içermektedir. Yazılım hata raporları hatanın raporlandığı tarih, hatanın tanımı, hatanın özeti (başlığı), hatanın atandığı yazılım geliştirici, hatayı atayan kişi, ekip üyelerinin yorumları vb. hata niteliklerini barındırır [9].

Hata izleyicileri, sürüm kontrol sistemleri ve mesaj panoları, yazılım projelerinin gelişmesine yardımcı olur [9]. Bu tarz araçlar yazılım projeleri hakkında bilgiler içermektedir. Böyle bilgiler sürekli bir yazılım gelişimine rehberlik sağlaması için veri madenciliği gerçekleştirilebilir. Hata raporlama sistemleri, kullanıcıların hata

raporlarını gönderebilmesi, tanımlayabilmesini, izleyebilmesini, yorum yapabilmesini ve hata raporlarını sınıflandırabilmesini sağlar. Bugzilla, hata raporlama aracı için bir örnektir.

Bugzilla gibi bir hata-izleme sistemi; geliştirim ekipleri, test ekipleri ve son kullanıcılar gibi çeşitli kaynaklardan toplanmış hata raporlarını tutarlar. Hata raporlayıcıları, bir hata raporlama sistemine hata raporlarını gönderdiğinde gönderilen hataların güvenlik problemine sebep olup olmadığını göstermek için güvenlikle ilgili hata raporu (security bug reports - SBR) ya da güvenlikle ilgili olmayan hata raporu (not-security bug reports - NSBR) diye hata raporlarını etiketleme ihtiyacı duyarlar [10]. Güvenlikle ilgili olan hatalar olmayanlardan hata tamirinde daha yüksek önceliğe sahiptirler. Bununla birlikte hata raporlayıcıları güvenlikle ilgili bilgi eksikliği yüzünden bazen SBR'leri NSBR şeklinde etiketlendirebilirler. Bu yanlış anlama güvenlik hatalarını tanımlama ve tamir etmedeki gecikmeye bağlı olarak yazılımcı ve sistemcilere zarara sebep olmaktadır.

Hata raporlarının analizi, yazılım depoları madenciliği alanı içinde önemli bir alt sahadır. Hataları sınıflandırma ve hatalara öncelik atama sürecine dair değişik bilgiler ortaya çıkarmayı amaçlar. Bunun için kusurları izleme sistemlerinde kullanılabilen zengin veriyi keşfeder. Hata verisi, Bugzilla ve JIRA gibi sistemlerden kolaylıkla erişilebiliyorken yaygın bir veritabanı şekli ve veri seti daha kolay kopyalamaya izin verdiği için gelecek araştırmaları ciddi şekilde arttırabilir [11].

1.4 Log Madenciliği

Yazılım sistemleri log dosyalarında kendi aktiviteleriyle ilgili bilgiler toplamaktadır. "loglamak" terimi tamamlanmış aktivitelerin kaydını tutmak için bir log defterine kayıt girişi yapmak demektir. Loglar olarak adlandırılan log dosyalarındaki bilgiler, yazılım sisteminin hareketleri ya da olayları, durum bilgileri ve hata bilgilerinden oluşur. Herhangi bir log satırı temel olarak tarih ve zaman bilgisi, kullanıcı bilgisi, uygulama bilgisi ve olay bilgisi içermektedir. Loglar genellikle sistem izleme, sistemin hatalarını ayıklama ve hata teşhisi için toplanırlar.

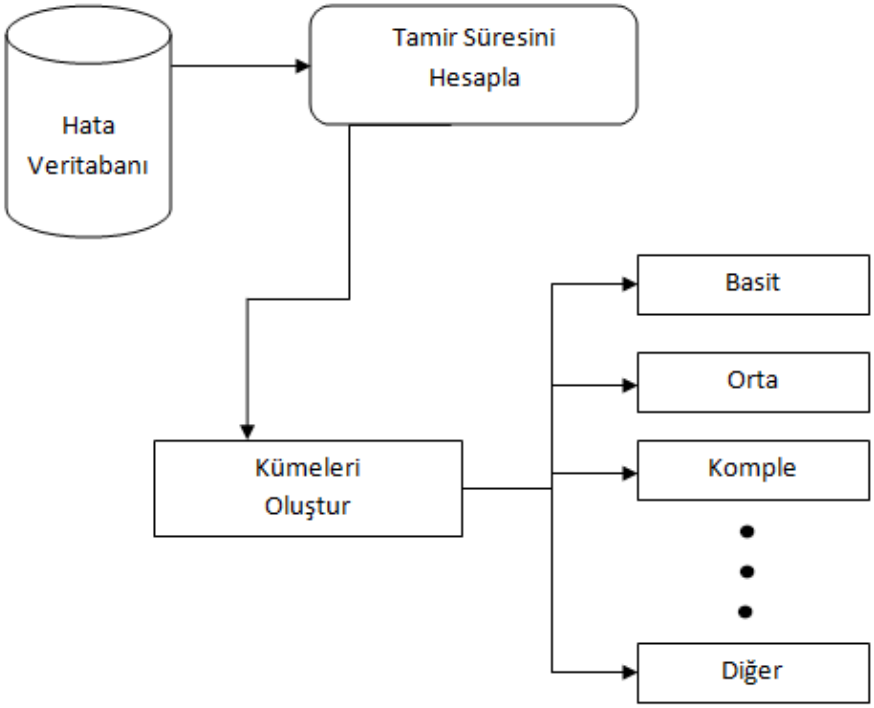
Log dosyaları, bir sistemin işleyişi hakkında önemli bilgiler barındırmaktadır. Bu bilgi; genellikle hata ayıklama, operasyonel profilleştirme, anormallikleri bulma, güvenlik tehditlerini saptama, performansı ölçme vb. gibi işlemler için kullanılır. Log dosyalarını elle okuma (manually) hala en geniş kullanılan tekniklerden birisidir. Ancak buna rağmen bu derece büyük bir bilgi kitlesi böyle bir teknikle okunamaz. Çünkü bilgi miktarı çok büyüktür. Son yıllarda çeşitli veri madenciliği ve makine öğrenmesi algoritmaları log dosyalarındaki bilgiyi analiz etmek için kullanılmaktadır [12].

Log madenciliği, loglar üzerinde veri madenciliğinin uygulanması işlemidir ve bilişim araştırmasının yeni bir sahasıdır. Log verisine ön koşul tanımlama ve algoritmayı kullanarak madencilik işlemi uygulama log madenciliğinin en yaygın iki sahasıdır [13]. Ayrıca log madenciliği, web loglarında gizlenmiş kullanıcı örüntülerini bulabilmek için kullanılabilir.

2. LİTERATÜR İNCELEMESİ

2.1 Hata Madenciliği

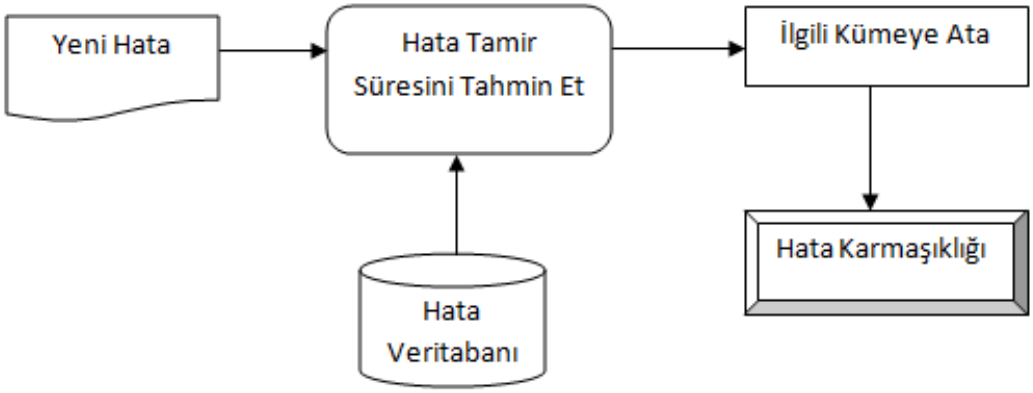
Nagwani ve Bhansali "A Data Mining Model to Predict Software Bug Complexity Using Bug Estimation and Clustering" isimli makalelerinde bir hatanın karmaşıklığını tahmin etmeye dair bir model tasarlamışlardır [6]. Kullandıkları teknik üç aşamalı bir yöntemden oluşmaktadır. İlk aşamada hata deposundaki hatalar için belli bir tamir süresi hesaplanmıştır. Sonra da bu tamir sürelerine bağlı olarak hatalar belli karmaşıklık kümelerine ayrılmıştır.



Şekil 2.1 Bir yazılım hatasının tahmini için durum diyagramı [6]

Bir yazılım hatasını tahmin etmek için hata kümeleme durum diyagramı Şekil 2.1'de görülmektedir. İkinci aşamada, yeni bir hata geldiğinde daha önceden tahmin süreleri hesaplanmış hatalar referans alınmıştır. O hatalara göre yeni hatanın tamir zamanı

tahmin edilmiştir. Üçüncü aşamada da yeni hata, Şekil 2.2'de görüldüğü gibi kendisi için tahmin edilen zamana göre hangi kümeye atanması gerekiyorsa, o kümeye atanmıştır.



Şekil 2.2 Yeni bir hatanın tamir süresine göre kümelenmesinin durum diyagramı [6]

Breckel "Error Mining: Bug Detection through Comparison with Large Code Databases" adlı makalesinde büyük bir kod veritabanında karşılaştırmalar yaparak hataları otomatik olarak saptamaya çalışmıştır [7]. Bu yaklaşım da şöyle gösterilebilir:

Aşağıdaki C kodu parçası diziler üzerinde işleyen daha büyük bir fonksiyonun parçası olsun:

```
for(j = 0; j < - n; j++) array[ i ] = j; [7]
```

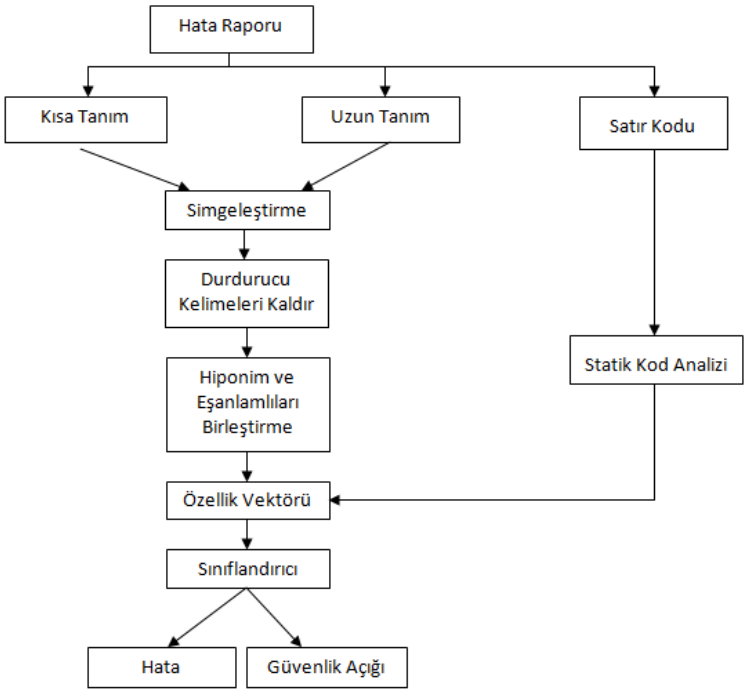
Yukarıdaki kodda hatalı kısımlar kırmızı renkle gösterilmiştir. İlgili algoritma çalıştığında yukarıdaki kodu ele almaktadır. Kod veritabanında yukarıdaki koda benzeyen aşağıdaki kodu bulur.

```
for(i = 0; i < k; i++) arr[i] = i; [7]
```

Daha sonra her iki kodu karşılaştırır ve aralarındaki farklılıkları ve benzerlikleri inceler. En sonunda üstteki koddaki hatalar ile ilgili olarak bir yorum getirir. Yani kırmızıyla belirtilen kısımları bulacak ve bunları potansiyel birer hata olarak işaretleyecektir.

Kritik ve hassas sistemler için yazılım güvenlik açıklarını tanımlama işlemi, karmaşık yazılım sistemlerine bağlı olarak gittikçe önem kazanıyor. Bazı yazılım hataları fark edilmelerinden çok uzun zaman sonra güvenlik açığı olarak tanımlanırlar. Bu güvenlik açıkları gizli etkili güvenlik açıkları (hidden impact vulnerabilities) olarak bilinir [8].

Wijayasekara, Manic, Wright ve McQueen, "Mining Bug Databases for Unidentified Software Vulnerabilities" adlı makalelerinde gizli etkili güvenlik açıkları için kullanılan hata veritabanlarına veri madenciliği uygulanmasının gerektiğini belirtmişlerdir [8]. Şekil 2.3, çalışmanın amaçlanan metodolojisini sunmaktadır.



Şekil 2.3 Amaçlanan güvenlik açığı keşfinin akış şeması [8]

Şekil 2.3, hata veritabanlarındaki hata raporlarının kısa ve uzun tanımlarını kullanmaktadır. Buradaki amacı gizli etkili güvenlik açıklarını bulmaktır. Bunun için ilk etapta gelişmiş metin madenciliğini kullanmıştır. İlgili şekil, raporlanmış bir hatadan kısa ve uzun tanımları çıkarır, ondan sonra metin madenciliği sayesinde bir özellik vektörü üretir. Metin madenciliği aşaması; simgeleştirme (tokenizing), durdurucu kelimelerin (stop-words) kaldırılması, eşanlamlı ve hiponim (hyponym) kelimelerin birleştirilmesi vs gibi aşamalar içermektedir. Ayrıca hatanın raporlandığı statik kodun analizi de yapılabilir ve özellik vektörüne eklenebilir. Son olarak sınıflandırıcı, hatanın özellik vektörüne göre raporlanmış bir hatayı normal bir hata veya güvenlik açığı olan hata olarak sınıflandıracaktır.

Hatalar kısa ve uzun tanımlı olarak sürece dâhil olurlar. Sembolleştirme, hata tanımlarındaki ünik olan kelimeleri tespit etmek için kullanılmıştır. Sembolleştirme sürecinde özel karakterler ve numaralar kaldırılmıştır. Büyük harf kullanımı ve diğer metin formatlama da ayrıca metinden çıkarılmıştır. Çizelge 2.1, hatanın her kategorisindeki ünik kelimelerin sayısını göstermektedir. Metin madenciliği sürecinin her adımından sonra özellik vektörünün boyunun azaltılması Çizelge 2.1'de görülebilmektedir [8].

Bu sayede veriler daha sade ve analiz edilmeye müsait bir hale getirilmiştir.

Çizelge 2.1 Veri setinin her sınıfında anahtar kelime sayısı [8]

Hata Tipi		Simgeleştirme Adımından Sonra Benzersiz Kelimelerin Toplam Sayısı	Durdurucu Kelimeler Çıkarıldıktan Sonra	Eşanlamlı ve Hiponim Kelimeleri Birleştirdikten Sonra	Porter Köklerine Ayırma İşleminden Sonra	Hataların %10'undan Daha Azında Geçen Kelimeler Çıkarıldıktan Sonra
Gizli Etkili Olmayan Güvenlik Açıkları	Kısa Tanım	335	308	272	268	8
	Uzun Tanım	2595	2468	1848	1794	159
Gizli Etkili Güvenlik Açıkları	Kısa Tanım	325	297	264	260	8
	Uzun Tanım	2144	2026	1564	1525	210
Normal Hatalar	Kısa Tanım	6161	6039	4536	4349	90
	Uzun Tanım	9981	9843	8067	7825	158
Toplam		21541	20981	16551	16021	633

Lampanfi, Perez ve Demeyer; "The Eclipse and Mozilla Defect Tracking Dataset: "A Genuine Dataset for Mining Bug Information" adlı makalelerinde Eclipse ve Mozilla Kusur İzleme Veri Setini (The Eclipse and Mozilla Defect Tracking Dataset) incelemeyi amaçlamışlardır [11]. Bu set, bütün hata önceliklendirme yaşam döngüsünü kapsamak için tasarlanmıştır. Sadece gerçek kusurları barındıran örnek bir veritabanıdır. Bu veri seti; hata şiddetini tahmin etmek için, hata tamir zamanına çalışmak için ve atanmış bileşenleri tanımlamak için kullanılmıştır.

Bu makalede herhangi bir raporlanmış hatanın yaşam döngüsü boyunca geçirdiği değişim ve güncellemelere odaklanılmıştır [11]. Bir hatanın sadece tek bir ekran görüntüsünü dikkate almak yerine daha değişik ve ilginç bulgularının bütün yaşam döngüsü sorgulandığında elde edilebileceği düşünülmüştür. Sunulmuş olan Eclipse ve Mozilla Kusur İzleme Veri Seti, hem Eclipse hem de Mozilla'nın dört popüler ürününün hata raporlarının setini içerir. Bu veri setindeki nitelikler şunlardır [11]:

1. Öncelik: Öncelik, hatanın ne kadar süre içinde tamir edilmesi gerektiğini gösterir. Bu nitelik tipik olarak P1 en yüksek önceliği gösterecek şekilde P1 ve P5 arasında çeşitlenir.

2. Ciddiyet: Yazılım sistemi üstünde hatanın etki şiddeti. Bu nitelik; *önemsiz, az önemsiz, normal, önemli, kritik* ve *bloklayıcı* olarak çeşitlenir.

3. Ürün: Hatanın ilişkili olduğu özel yazılım uygulaması

4. Bileşen: Raporlanmış hata için ürünün ilgili alt sistemi

5. Hata Durumu: Bu nitelik, hatanın şu anki durumunu göstermektedir. Bu niteliğin değerleri; *onaylanmamış, yeni, atanmış, yeniden açılmış, hazır, çözülmüş, doğrulanmış* arasında çeşitlenir.

6. Karar: Bu nitelik, hataya ne olduğunu gösterir. Bu niteliğin değeri; *tamir edildi, geçersiz, alışılmış tamir, çift, worksforme, tamam değil* arasında çeşitlenir.

7. Kime Atandığı: Atanmış hatayı alan geliştirici tanımı

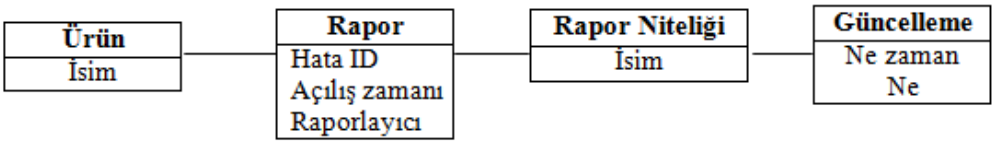
8. Bilgi Verilecekler: Bu hatanın süreciyle ilgilenen ve bilgi alan kullanıcı

9. Kısa Tanım: Hatayı anlatan tek satırlık bir özet.

10. Sürüm: Hatanın içinde bulunduğu ürünün sürümü.

11. İşletim Sistemi: Hatanın bulunduğu işletim sistemi.

Her niteliğin kendi raporunun bütün yaşam döngüsü boyunca gerçekleştirilmiş farklı "değişimleri" derlenmiştir. Her böyle değişim için hata raporunun ilgili niteliğinin yeni değerinin ne zaman atandığı tutulmaktadır. Ayrıca değiştirilemez iki rapor niteliği vardır, raporlayıcı ve raporlanmış hatanın açılış zamanı. Bu nitelikler Şekil 2.4'te görüldüğü gibi her hata raporu için *Rapor* örneklerinde (instance) bulunur [11].



Şekil 2.4 Veri seti modeli [11]

İlk olarak, her proje için ürünlerin sayısı seçilir ve raporlanmış hata sayısı ve bileşen sayısına göre ürünlerin boyutu ölçülür. Ondan sonra, niteliklerin her birinin hata raporlarının yaşam döngüsü boyunca ne kadar sıklıkla güncellendiğinin sayısı hesaplanır. Çizelge 2.2 seçilmiş farklı yazılım ürünlerini ve o ürünlerin bileşen ve hata sayılarını göstermektedir.

Çizelge 2.2 Farklı ürünlerin hata ve bileşen sayıları [11]

Ürün	**Hata Sayısı**	**Bileşen Sayısı**
Eclipse Platform	24775	22
Eclipse JDT	10814	6
Eclipse CDT	5640	20
Eclipse PDE	5655	5
Mozilla Core	74292	137
Mozilla Firefox	69879	47
Mozilla Thunderbird	19237	23
Mozilla Bugzilla	4616	21

Hata sınıflandırıcıları, hangi hataların tamir edileceğine, kimin yapacağına ve ne zaman tamir edileceğine karar verebilmesi için gelen hata raporlarını analiz ederler. Burada, hata sınıflandırıcıları yanlış ve kayıp bilgiyi güncelleme imkânına sahiptirler. Özellikle, bir sınıflandırıcı, hatalı "bileşen" nitelikli bir hata raporu aldığında güncellemeler o nitelik için olur. Bugzilla bu güncellemelerin kaydını tutar,

bundan dolayı hata veritabanında depolanmış güncelleme bilgilerini kullanarak bir hata raporunun niteliklerinin her biri için değerleri çıkarılabilir. Çizelge 2.3, bütün ürünler için olan böyle atamaların toplam sayısını sunmaktadır [11].

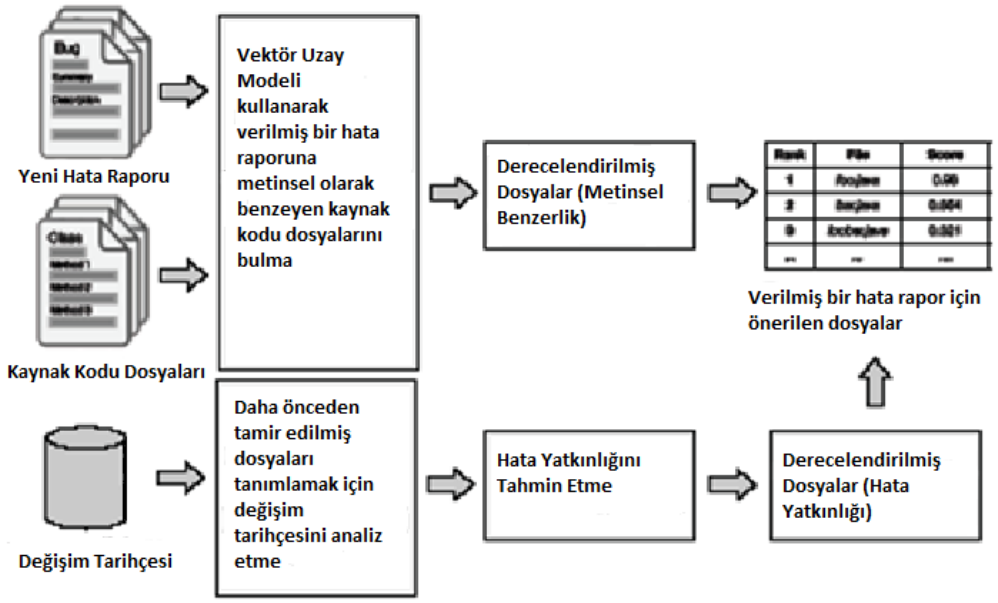
Çizelge 2.3 Raporlardan seçilen farklı nitelikler ve yeniden atamaların sayısı [11]

Hata Niteliği	**Atama Yüzdesi**	
	Eclipse	**Mozilla**
Hata Durumu	%100	%100
Çözümü	%100	%100
Kime Atandığı	%66	%29
Bilgi Verilecek Kişiler	%65	%90
Kısa Tanım	%27	%15
Bileşen	%18	%17
Ürün	%5	%7
Ciddiyet	%6	%6
Sürüm	%6	%21
Öncelik	%3	%6
İşletim Sistemi	%5	%10

Çizelge 2.3'te farklı nitelikler ve azalan sırada sıralanmış güncelleme frekansları gösterilmektedir. Beklendiği gibi hata durumu (bug_status), karar (resolution), raporlanmış bir hatanın en sık güncellenen nitelikleridir. Bu nitelikler, hangi hataların çözüldüğünün ve hangilerinin de çözülmeye çalışıldığının kaydını tuttuğu için en sık güncellenmeye meyillidirler. Çizelge 2.3'te ayrıca en az sıklıkta güncellenen niteliklerin işletim sistemi (op_sys), öncelik (priority) ve sürüm (version) gibi nitelikler olduğu gözlemlenmektedir.

Hata yerelleştirme, bir yazılım hatasının yazılımsal konum açısından tespit edilme sürecidir. Tantithamthavorn, Teekavanich, Ihara ve Matsumoto; "Mining A Change History to Quickly Identify Bug Locations : A Case Study of the Eclipse Project"

adlı makalelerinde hata yerelleştirme performansını arttırabilmek için yazılımın değişim tarihçesini inceleyen bir yaklaşım amaçlamışlardır [14]. Çalışmanın temel fikri şu şekildedir: Son zamanlarda hatayı düzeltmek için üstünde işlem yapılmış bir dosyanın yakın zamanda tekrar üstünde işlem yapılma olasılığı vardır. Hatanın oluştuğu sanılan kaynak kodu dosyasını tahmin etmek için yazılımın değişim tarihçesi incelenir ve metinsel kurgularından faydalanılır. Bu yaklaşım, Eclipse Platform ve Eclipse JDT'den elde edilen 1212 adet hata raporu üzerinde uygulanıp değerlendirilmiştir. Deneysel sonuçlar ilgili çalışmanın hata yerelleştirme performansını geliştirdiği ve hatalı dosyaları etkili bir şekilde bulabildiğini göstermiştir.



Şekil 2.5 Hata yerelleştirme performansı için amaçlanan yaklaşımın bakış açısı [14]

Şekil 2.5, amaçlanan yaklaşımın bakış açısını göstermektedir. Yeni bir hata raporu için, hata raporunun başlık ve tanımına metinsel olarak benzeyen kaynak kodu dosyalarını bulmak için Vektör Uzay Modeli kullanılmıştır. Daha önce üstünde hata tamiri işlemi uygulanmış dosyalar üzerinde yazılımsal değişim tarihçesi analizi uygulanmıştır. Ardından bu dosyaların hata bulundurma eğilimleri tahmin edilmiştir.

Son olarak, bu elde edilen verilerin skorları birleştirilerek yazılım geliştiricilerine hata bulundurması muhtemel yüksek skorlu kaynak kodu dosyaları bildirilmiştir.

Metinsel olarak benzeyen dosyaları tespit ederken şöyle bir yol izlenmiştir. İlk olarak, hata raporu ve kaynak kodu dosyalarından anlamlı kelimeler çıkarılmıştır. Bütün noktalama işaretleri ve sayılar kaldırılmıştır. Bütün kelimeler simgelere ayrılmıştır ve küçük harfe dönüştürülerek normalize edilmiştir. Ayrıca yaygın İngilizce kelimeleri (a, an, the gibi) ve genel programlama terimleri (int, double, char gibi) kaldırılmıştır. Ardından belli eşitsizlikler uygulayarak her hata raporu için ve kaynak kodu dosyaları için bir vektör değeri elde edilmiştir. Üçüncü olarak da her hata raporu ve kaynak kodu dosyası için *VSMScore(q,d)* değerini hesaplamak için başka bir eşitsizlik kullanılmıştır. Son olarak, her hata raporu için belli bir derecenin üstünde olan kaynak kodu dosyaları elde edilmiştir.

Daha önceden hata tamiri işlemi uygulanmış dosyaları tanımlamak için değişim tarihçesi analiz edilmiştir. Genel olarak, yazılım geliştiricileri bir hatayı tamir ettiklerinde log kayıtlarında hata raporunun numarasını da belirtirler. Daha önceden tamir edilmiş dosyaları tanımlamak için yazılım sürüm arşivleri ve tamir edilmiş hata raporlarının logları arasındaki mantıksal ilişki kullanılmıştır. Bu sayede daha önce tamir edilmiş dosyaların kronolojik sırada bir listesi elde edilebilmiştir.

2.2 Log Dosyaları Madenciliği

Şirket, kurum, kuruluş vb. gibi büyük organizasyonlardaki çalışanlar arasındaki iletişim, ofis telefonları ya da ofis içi e-posta sistemleri gibi kanallar dışındaki yöntemlerle de gerçekleşebilmektedir. Ayrıca bu organizasyonlar her türlü suistimale karşı tedbirli olabilmeleri için ve güvenlik endişeleri sebebiyle çalışanlara karşı sıkı politikalar uygulayabilmektedir. Dahili e-posta ve iletişimi gözlemlemek bir bakıma böyle organizasyonların ilgilendiği konulardır. Şuanda bu tarz bir işlem için güvenilir otomatikleştirilmiş bir çözüm bulunmamaktadır. İnsan müdahalesi kaçınılmaz görünür ama pahalıdır. Güvenli bir işyeri ortamını sağlamada pratik bir yöntem şüpheli iletişim ya da kullanıcılara dikkat kesilen otomatik bir filtreleme sistemidir.

Esichaikul, Guha ve Niyamosatha, "Mining Email Transaction Logs to Locate Significant Messages and Users" adlı makalelerinde bir sistem içindeki e-posta logları üzerinde veri madenciliği gerçekleştirmişlerdir [15]. Bu sayede önem derecesi yüksek kişileri ve mesajları basit ve sezgisel bir model aracılığıyla tespit etmeye çalışmışlardır. Bu model, işlem loglarında graf analizi uygulayarak mesajlara ve kullanıcılara önemlilik skoru atamaya dayanmaktadır.

Çizelge 2.4 Bir e-posta log tablosu örneği [15]

E-posta ID	**Gönderici**	**Mesaj Tipi**	**Cevabı Olduğu Mesaj**	**Zaman**	**Konu**	**Alıcı**	**Skor**
1	A	Orjinal	0	1:30	Buluşma	B,C,D	**2.2**
2	C	İletilmiş	1	2:10	Buluşma	E,F,C	**1.6**
3	A	Orjinal	0	4:45	Öğle Yemeği	C	**0.8**
4	E	Cevap	2	5:30	Buluşma	C	**0.5**
5	F	Cevap	2	8:30	Buluşma	C	**0.5**
6	C	İletilmiş	3	9:20	Öğle Yemeği	B,D	**0.6**

Çizelge 2.4'teki log örneği herhangi bir organizasyonun e-posta işlem loglarından alınmıştır. Şu alanlardan oluşmaktadır:

1-) Mesaj ID: Her mesaj için ünik pozitif bir tamsayıdır.

2-) Gönderen: Gönderenin ID'sini gösterir.

3-) Mesaj Tipi: "Orjinal", "İletilmiş" ya da "Cevap" seçeneklerinden birini almaktadır.

4-) Cevap Olduğu Mesaj: Hangi mesajın cevabı olduğunu göstermektedir.

5-) Zaman: Mesajın gönderildiği sistem zamanını göstermektedir.

6-) Konu: Mesajın konu başlığıdır.

Var olan mesajlar üzerinde belli işlemler uygulanarak her mesaj için bir skor üretilmiştir. Mesajların skorlarını belirlerken, ilgili mesajın başka bir mesaja cevap niteliğinde olup olmaması skoru etkilemiştir. Bu sayede örneğin, daha fazla cevaplanan mesaj daha yüksek skorlu olmuştur.

Ngok ve Gong, "Log Mining to Support Web Query Expansions" adlı makalelerinde arama motorlarındaki sorguları inceleyerek gereksiz arama sonuçlarını azaltma ile ilgili bir model geliştirmişlerdir [16]. Bunu yaparken de log madenciliğinden faydalanmışlardır. Yapılan çalışma, sorgu koşullarının düzenlenerek ve geliştirilerek daha doğru arama sonuçlarının elde edilmeye çalışılmasına dayanmaktadır.

Halihazırdaki arama motorlarında kullanıcıların aramalarına cevap verme yöntemleri çok tatmin edici değildir. Bunlar ya aranan kelimeyle çok fazla alakasız sonuç, ya sadece birkaç sonuç döndürmektedir ya da hiç sonuç döndürmemektedir. Bunun dışında bazen sonuçlar geldiğinde aranan kelimeyle ilişkisi epey düşüktür. Bunun iki sebebi vardır. Girilmiş arama sorgusu oldukça kısadır veya sorgunun anlamı kullanıcının ihtiyacını yeterince iyi anlatamayabilir.

Çizelge 2.5 Sorgu logu kaydı örneği [16]

IP	**Sorgu İfadesi**	**Tarih (TimeStamp)**	**URL Adresi**
192.168.1.1	Java WordNet API	12.3.2006 23:00:01	http://www.wordnet.com

Kullanıcıların sorgu hareketleri arama motorları tarafından loglanmaktadır. Bu log kayıtları, genellikle Çizelge 2.5'te de görüleceği üzere kullanıcıların IP adresleri,

sorgu ifadeleri, sorgu tarihleri (timestamp) ve kullanıcıların web sayfasında tıkladıkları adresleri içermektedir.

Vouk ve Nagappan "Abstracting log lines to log event types for mining software system logs" adlı makalelerinde bir log dosyasını soyutlamak için yeni bir yaklaşım sunmuşlardır [12].

```
Jen kullanıcısı için işleme başlama

Tom kullanıcısı için işleme başlama

Henry kullanıcı için işleme başlama

Tom kullanıcısı için işleme başlama

Peter kullanıcısı için işleme başlama
```

Çizelge 2.6 Veri özeti adımından sonra kelimelerin sıklık tablosu [12]

Kelime	**1**	**2**	**3**	**4**	**5**
Başlama	5	0	0	0	0
işleme	0	5	0	0	0
İçin	0	0	5	0	0
Jen	0	0	0	1	0
Tom	0	0	0	2	0
…	0	0	0	0	…
Kullanıcı	0	0	0	0	5

Çizelge 2.6'da görüldüğü gibi, "başlama", "işleme", "için", "kullanıcısı" kelimeleri sırayla 1, 2, 3 ve 5 pozisyonlarında 5 kere geçiyor. Çünkü bu kelimeler log

satırlarının statik alanlarıdır. Yani her log kaydında bulunmaktadır. “Jen”, “Henry” ve “Peter” kelimeleri bir kere geçiyor ve “Tom“ iki kere geçiyor. Bunlar diğer kelimelerin sıklığından çok daha azdır. Buradan şöyle bir sonuç çıkarılabilir. Log dosyasında bu satırları oluşturmuş mesaj aşağıdaki formatta olabilir: “$kullaniciadi kullanıcısı için işleme başlama”. Bu ifade gerçekte log mesajını oluşturmuş ifadedir. Sabit mesaj tipi sahasına ait olan kelimeler, değişken parametre sahasındaki kelimelerden daha sık geçmektedir. Log dosyalarını olay tiplerine soyutlamak için bu özellikten faydalanılmıştır.

Jiang, Hassan, Flora ve Hamann; “Abstracting Execution Logs to Execution Events for Enterprise Applications” adlı makalelerinde işlem loglarını kullanarak uygulamaların işleyişlerini gözlemlemeyi amaçlamışlardır [17]. Log satırları, bir olayın farklı dinamik parametrelerle gerçekleşmesini ifade etmektedir. Bu çalışmada da bu log satırlarından dinamik parametreler çıkarılarak gerçekleşen işlem tiplerini tespit eden bir yaklaşım amaçlanmıştır [17].

Log satırları arasındaki benzerlikleri saptamak için yeni bir yaklaşım türetilmiştir. Ardından bu satırlar parametreleştirip soyutlanmıştır. Buradaki yaklaşım binlerce ya da milyonlarca log satırını işleyerek ölçebilmektedir.

Çizelge 2.7 Örnek log satırları [17]

Log Satırları
1. Hesaplamayı başlat 2. Ödeme gerçekleştir, ürün=çanta, çeşit=1,miktar=100 3. Ödeme gerçekleştir, ürün=kitap, çeşit=3,miktar=150 4. Hesapla, toplam miktar 250 adettir. 5. Hesaplama tamamlandı.

Çalışma üç adımdan oluşmaktadır: Anonimleştirme, Simgeleştirme ve Kategorileştirme. Bu bölümün geri kalanında Çizelge 2.7 ile ilgili küçük bir örnek gösterilmektedir.

Anonimleştirme adımında, Çizelge 2.7'de dinamik parçalara karşılık gelen log satırlarındaki dinamik simge alanları için sezgisel tarama (heuristics) kullanılır. Bu dinamik alanlar tespit edildiğinde genel simgelerle yer değiştirilir ($v). Sezgiler eklenebilir ya da buradaki yaklaşımdan çıkarılabilir. Aşağıdaki iki sezgi (heuristics) log satırlarındaki dinamik parçaları tanımak için kullanılır:

1- "kelime=değer" gibi atama çiftleri

2- … değer adettir gibi ifadeler

Çizelge 2.8'de log satırlarının anonimleştirme adımı uygulandıktan sonraki durumu gösterilmiştir. İkinci ve üçüncü satırlarda eşittir işaretinden sonra içerikler $v genel terimiyle yer değiştirilmiştir. Dördüncü satırda "250 adettir" ifadesi "=$v" terimiyle yer değiştirilmiştir. İlk ve son satıra yapılmış herhangi bir değişiklik yoktur**.**

Çizelge 2.8 Anonimleştirme adımından sonra örnek loglar [17]

Log Satırları
1. Hesaplamayı başlat 2. Ödeme gerçekleştir, ürün=$v, çeşit=$v,miktar=$v 3. Ödeme gerçekleştir, ürün=$v, çeşit=$v,miktar=$v 4. Hesapla, toplam miktar $v adettir. 5. Hesaplama tamamlandı.

Simgeleştirme adımında, her log satırındaki kelime ve parametrelerin sayısına göre anonimleştirilmiş log satırları farklı gruplara (kutulara) ayrılmıştır. Log satırlarını farklı kutulara ayırma Kategorileştirme Adımı boyunca arama yerini daraltır. Kutular, arama yerini sınırlayarak güncel bir şekilde büyük log dosyalarının işlenilmesine izin verir. Genel terimlerin ($v) sayısını sayarak bir log dosyasındaki parametrelerin sayısı tahmin edilir. Aynı sayıda dinamik simge barındıran log satırları ve aynı sayıda parametre içeren log satırları aynı kutuya yerleştirilir. Çizelge 2.9, anonimleştirme ve simgeleştirme adımlarından sonra örnek log satırlarını göstermektedir. Sol kolon bir kutunun adını belirtmektedir. Her kutu bir sayısal işaret ile ifade edilmektedir: kelime sayısı ve bu kutu içindeki diğer log satırlarıyla ortak parametre sayısı. Çizelge 2.9'da sağ kolon log satırlarını göstermektedir. Her satır kutuyu ve ilgili log satırlarını göstermektedir. İkinci ve üçüncü log satırları 8 kelime ve 3 parametre içermektedir. Böylece ikinci ve üçüncü log satırları 8_3 kutusunda birlikte gruplanmıştır. Benzer şekilde ilk ve son log satırları, hem 3 kelime içerdiği hem de hiç parametre içermediği için 3_0 kutusu içinde birlikte gruplanırlar**.**

Çizelge 2.9 Simgeleme adımından sonra loglar [17]

Kutu Adı	**Log Satırları**
3_0	1. Hesaplamayı başlat 5. Hesaplama tamamlandı
5_1	4. Hesapla, toplam miktar $v adettir
8_3	2. Ödeme gerçekleştir, ürün=$v, çeşit=$v,miktar=$v 3. Ödeme gerçekleştir, ürün=$v, çeşit=$v,miktar=$v

Kategorileştirme adımında her kutudaki log satırları karşılaştırılır ve ilgili işlem tiplerine soyutlanır. Sonuçlandırılmış işlem olayları gelecek referanslar için bir işlem veritabanında depolanır. Algoritma, her kutu kendi içinde değerlendirilerek devam etmektedir. Bu adımdan sonra her log satırı bir işlem olayına soyutlanmalıdır. Çizelge 2.10, kategorileştirme adımından sonra çalışma örneğinin sonuçlarını çizelgeleştirmektedir. Algoritma, 3_0 kutusuyla başlar. Başlangıç olarak bu kutuya uyan herhangi bir işlem olayı yoktur. Bu yüzden ilk log satırına uyan işlem olayı isim olarak da 3_0_1 ilk işlem olayı olur (3 kelimesi olan ve hiç parametresi olmayan kutuya uyan ilk yürütme olayı). Ondan sonra algoritma beşinci log satırını içeren 3_0 kutusundaki bir sonraki log satırına hareket eder. Şu anda sadece bir yürütme olayı vardır: 3_0_1. Beşinci log satırı 3_0_1 yürütme olayına benzemediği için, beşinci log satırı için 3_0_2 yeni bir işlem olayı oluşturulur. 3_0 kutusundaki bütün log satırlarının işlenmesiyle 5_1 kutusuna geçilir. 5_1 kutusuna başlangıçta uyan herhangi bir işlem olayı olmadığı için dördüncü log satırı 5_1_1 olarak yeni bir işlem olayına atanır. 8_3 kutusundan log satırları işletildiğinde benzer log uygulamaları aynı algoritmayla işler.

Çizelge 2.10 Kategorizasyon adımından sonra loglar [17]

İşlem Olayı	**Log Satırları**
3_0_1	1. Hesaplamayı başlat
3_0_2	5. Hesaplama tamamlandı
5_1_1	4. Hesapla, toplam miktar $v adettir
8_3_1	2. Ödeme gerçekleştir, ürün=$v, çeşit=$v,miktar=$v
8_3_1	3. Ödeme gerçekleştir, ürün=$v, çeşit=$v,miktar=$v

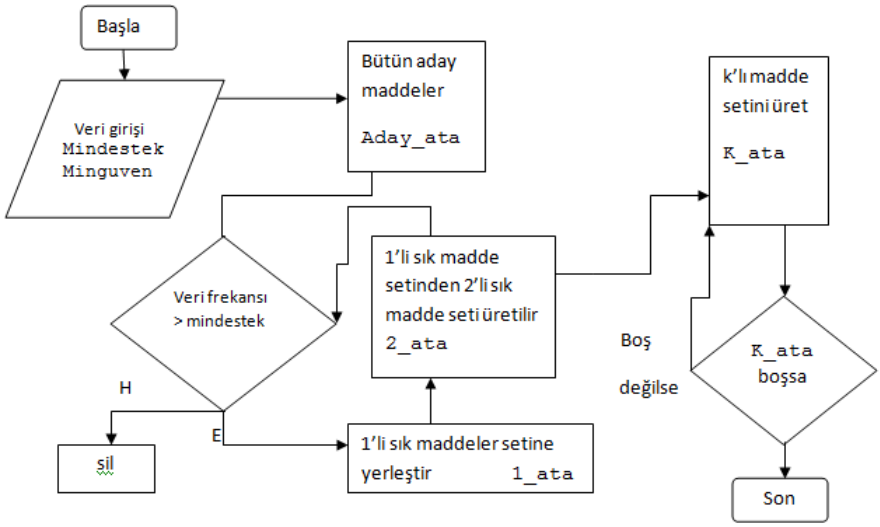
2.3 Apriori Algoritması

Apriori, işlem yapılabilen veritabanları üzerinde sık öğe seti madenciliği ve birliktelik kuralı öğrenimi için kullanılan klasik bir algoritmadır. Apriori tarafından

saptanan sık öğe setleri, veritabanında genel akımları vurgulayan birliktelik kurallarını saptamak için kullanılabilir. Market sepet analizi gibi alanlarda uygulamaları vardır.

2.3.1 Genel Çerçeve

Apriori algoritması, birliktelik kuralı madenciliğinde en klasik ve en etkili algoritmalardan birisidir [18]. Ayrıca en yoğun kullanılan yöntemlerden birisidir. Katman katman çalışarak döngüsel bir yöntemle kurallarını çıkarır. Öğe seti, öğelerin kümesine denmektedir ve N öğelerini barındıran bir öğe seti, N öğe seti olarak adlandırılmaktadır. Öğe setlerinin sıklığı, öğe setlerinin gördüğü işlem sayısı anlamına gelmektedir. Temel işleyişi Şekil 2.6'daki gibidir:



Şekil 2.6 Apriori algoritmasının akış şeması [18]

1. Öncelikle sisteme minimum destek ve minimum güven değerleri atanmalıdır. Mindestek = s ve minguven = c.

2. Ardından ayrı ayrı 1'li olarak öğe setlerinin adedi hesaplanır. 1'li öğe setlerinin sık sayılabilme şartı her birinin destek sayısının minimum destek sayısından fazla olmasıdır.

3. Veritabanından ayrı ayrı bütün öğeler okunur ve her bir öğenin 1'li öğe setleri kullanılır. Ardından her öğe setinin desteği üretilir. Son olarak 1'li öğe setlerinden 2'li öğe setleri kümesi türetilir.

4. Veritabanı taranır ve 2'li aday öğe seti kümesi türetilebilir ve ondan da 3'lü aday öğe setleri kümesi aynen önceki gibi türetilir.

5. Veritabanı taranır ve Sık Öğe Seti kümesinin daha yüksek seviyede sonuçlanan minimum desteğiyle karşılaştırılır. Son olarak sonraki aday öğe seti yeni bir aday öğe seti gelmeyene kadar türetilir.

Apriori algoritmasının temel prensipleri şu şekilde ifade edilebilir. Birliktelik kuralları üç parametreyle tanımlanır [18].

1. *Destek S(XY)*: Kuralların ne kadar sıklıkta olduğunu ifade etmektedir ve Minsup, minimum desteğin ne kadar olduğunu göstermektedir.

2. *Güven C(XY)*: Kuralların ne kadar yoğunlukta olduğunu ifade etmektedir ve Minconf, minimum güveni ifade etmektedir.

3. *Sık öğe seti :* Veritabanındaki kullanıcılar tarafından verilmiş minimum destek ve güveni karşılayan birliktelik kuralını tanımlamak için kullanılır.

Bununla birlikte Apriori algoritması uygulamaları çözerken bazı sakıncaları da barındırmaktadır. Eğer sistem farklı n tane öğe setine sahipse, veritabanında tarama yaparken n kere tarama yapmak zorunda kalmaktadır. Bu durum kural üretimi için sistemde ciddi bir zaman kaybına sebep olmaktadır. Böyle bir sürecin zaman karmaşıklığı m öğe setli n işleme sahip olan $O(e^n)$ olacaktır [19]. Yüksek eşik kriteri olması durumunda bazı öğe setleri daha ilk adımlarda ayrılır. Bu durum da önemli tahmin ve kuralları dışarıda bırakacaktır.

2.3.2 Örnekler

Birinci örnekte, her satırın bir işlem olduğu ve her hücrenin, işleme ait bağımsız bir öğe olduğu Şekil 2.7'deki gibi bir veri seti bulunmaktadır.

kalem	kağıt	defter
kalem	kağıt	dosya
kalem	kağıt	defter
kalem	kağıt	dosya

Şekil 2.7 Örnek veri seti [20]

Bu veritabanından saptanabilen birliktelik kuralları şu şekilde olacaktır:

1. Kalem'li setlerin %100' ü ayrıca kağıt da içermektedir.

2. Kalem'li ve kağıt'lı setlerin %50' si ayrıca defter de içermektedir.

3. Kalem'li ve kağıt'lı setlerin %50' si ayrıca dosya da içermektedir.

Bir diğer örnekte, büyük bir süpermarket her öğe için stok tutma birimine göre satış verisini izlemiştir: "yağ" ya da "ekmek" gibi öğe setleri sayısal bir stok tutma birimi tarafından tanımlanmıştır. Süpermarketin her işlemin birlikte alınmış eşyaları gösterdiği bir işlem veritabanı vardır.

İşlemler veritabanında {1,2,3,4}, {1,2}, {2,3,4}, {2,3}, {1,2,4}, {3,4}, ve {2,4} setlerinden oluşmaktadır. Bu veritabanında sık öğe setlerini saptamak için Apriori kullanılmıştır. Bu şekilde veritabanının en az üç işleminde birlikte geçen bir öğe setinin sık olduğu söylenilebilir: 3 değeri *eşik değeri*dir [20].

Apriori algoritmasının ilk adımı, ilk seferinde veritabanını tarayarak destek olarak adlandırılan her üye öğesinin bağımsız şekilde yer alma sayılarını hesaplamaktır. Şekil 2.8'deki sonuç elde edilir [20].

Öğe	**Destek**
{1}	3
{2}	6
{3}	4
{4}	5

Şekil 2.8 Bir elemanlı öğe setlerinin destek adetleri

Bütün tekli öğe setleri en az 3'lük bir desteğe sahiptir, bu yüzden bu öğe setleri sıktır denilebilir. Şekil 2.9'da görüldüğü gibi bir sonraki adımda çiftli öğelerin listesi oluşturulmuştur:

Öğe	**Destek**
{1,2}	3
{1,3}	1
{1,4}	2
{2,3}	3
{2,4}	4
{3,4}	3

Şekil 2.9 İki elemanlı öğe setlerinin destek adetleri

{1,2}, {2,3}, {2,4} ve {3,4} çiftleri 3' ün minimum desteğini karşılar ve aşar, bu yüzden onlar sık olarak değerlendirilebilir. {1,3} ve {1,4} sık olmadığı için {1,3} ve {1,4} içeren herhangi bir daha büyük set sık olamaz. Bu şekilde setler çıkarılabilir: Şimdi veritabanında sık üçlüler aranacaktır, ama bu iki çiftlerin birini içeren bütün üçlüler zaten dışarıda tutulabilir:

Öğe	**Destek**
{2,3,4}	2

Şekil 2.10 Üç elemanlı öğe setlerinin destek adetleri

Örnekte sık üçlüler yoktur. Şekil 2.10'da görüldüğü gibi {2,3,4} minimal eşiğin altındadır ve diğer üçlüler dışarıda tutulur çünkü onlar zaten eşiğin altında olan süper set çiftleridir. Böylece veritabanında sık öğe setleri saptanmıştır ve bazı öğelerin nasıl sayılmadığı gösterilmiştir.

2.4 Graflar

Graflar, köşeler (edge) ve bu köşeleri birbirlerine bağlayan kenarlardan (edge) oluşan ayrık ağ yapılarıdır. Bu dairesel köşeleri birbirine bağlayan bağlantılara da kenar denilmektedir. Genel olarak bir graf, doğru ya da eğriler kenarları gösterecek şekilde ve daireler de köşeleri gösterecek şekilde şemasal bir form aracılığıyla gösterilir. Graflar, Ayrık Matematik'in araştırma sahalarından birisidir [21]. Graf kenarları, yönlü kenarlar veya yönsüz kenarlar olabilir.

Graflar karmaşık yapılı çeşitli verileri modellemek için çok önemli araçlardır. Ayrıca farklı disiplinlerdeki problemler graflar aracılığıyla çözülebilmektedir. Köşeler, kök (node) veya nokta (point) olarak da adlandırılabilir. Graflar, graf teorisinin temel konusudur [21].

Örneğin, iki kişi birbirini tanıyorsa ve birbirleriyle sosyal medya üzerinden bir bağlantı oluşturuyorlarsa, bu bağ yönsüz grafa bir örnek olacaktır. Ancak sosyal medyada bir ünlünün hayran sayfaları üzerinde başka kişilerin üye olduğu düşünülürse bu yapı da yönlü grafa bir örnek olarak belirtilebilir. Çünkü hayranları bir ünlüyü tanıyabilirler, ancak ünlünün bütün hayranlarını tanımasına imkan yoktur.

2.4.1 Graf Tanımı

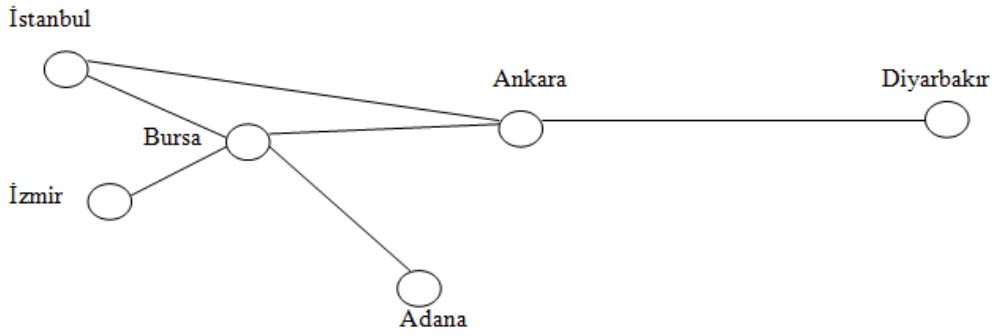
Herhangi bir graf, G = (V,E) sıralı çiftler seti olarak tanımlanabilir [21]. Bu tanımdaki V harfi kökler ve köşeler setini ifade etmektedir. E harfi de tüm kenarların iki elemanlı alt kümelerinin setlerini ifade etmektedir. Karışıklıktan kaçınmak için bu çeşit graflar yönsüz ve basit olarak tanımlanabilir.

Bir kenara ait köşeler, kenarın ucu (end), uç noktaları (endpoints) ya da kenarın uç köşeleri (end vertices) olarak adlandırılırlar. Bir köşe, graf içinde olup da herhangi bir kenara ait olmayabilir.

Bir grafın sırası (order) |V| (köşelerin sayısı)'dir [21]. Bir grafın boyutu da kenar sayısı olan |E| olur [21]. Bir köşenin derecesi ona bağlanan kenarların sayısıdır. Eğer bir kenar bir köşeye her iki tarafından da bağlıysa, yani bir döngü varsa derece iki kere sayılır. Birbirlerine bir kenar aracılığıyla bağlı köşelere bitişiktir denir [21].

2.4.2 Graf Örneği

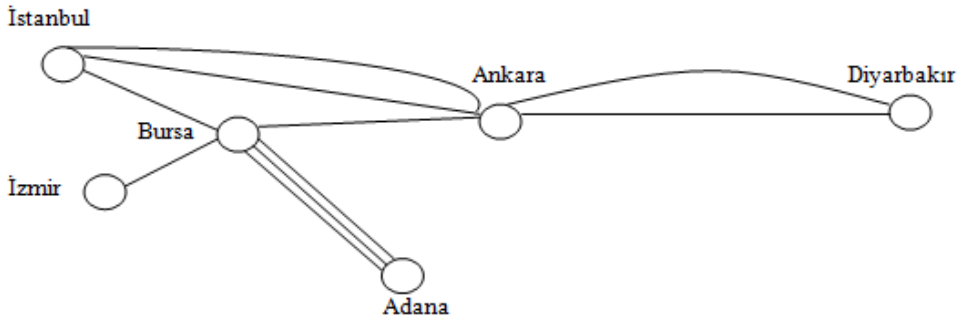
Farklı şehirlerdeki banka şubeleri arasında iletişim bağlantılarından oluşan bir ağ olsun. Şekil 2.11'de görüldüğü gibi şubenin konumları dairelerle ve iletişim bağlantıları da çizgilerle gösterilmiştir. Bu ağ yapısı graf kenarlarının iletişim bağlantılarını gösterdiği ve graf köşelerinin de şubelerin veri merkezlerini gösterdiği bir graf olarak modellenebilir.



Şekil 2.11 Bir bilgisayar ağı

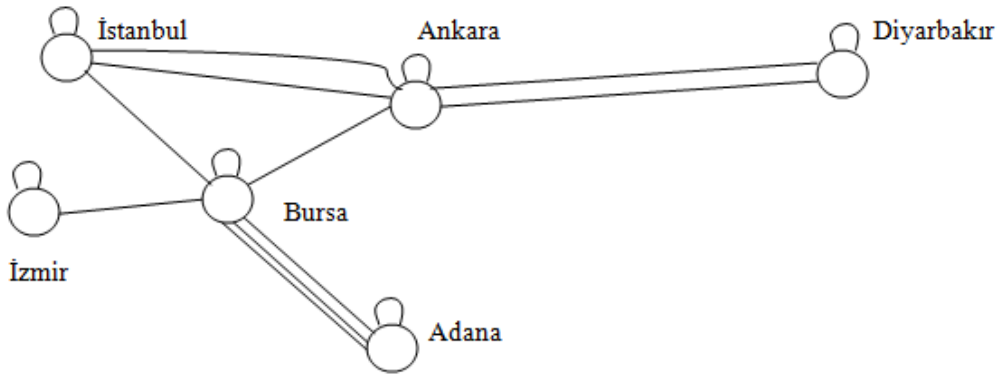
Bu bilgisayar ağını gösteren grafın her kenarı iki farklı köşeyi bağlamaktadır. Yani, hiçbir kenar herhangi bir köşeyi yine kendisine bağlamamıştır. Ayrıca iki farklı kenar aynı köşeleri birbirine bağlamamaktadır. Her kenarın iki farklı köşeyi bağladığı ve iki farklı kenarın aynı köşeleri bağlamadığı bir grafa basit graf denir [22].

Bir bilgisayar ağı Şekil 2.12'de olduğu gibi veri merkezleri arasında çoklu bağlantı da içerebilir. Böyle ağları modellemek için aynı köşe çiftlerini bağlayan birden daha fazla kenarlı graflara ihtiyaç vardır. Aynı köşeleri bağlayan çoklu kenarlara sahip olabilen graflar çoklu graf olarak adlandırılır. {u,v} aynı sırasız köşe çiftlerine bağlı m tane farklı kenar olduğunda {u,v}'nin m çeşitliliğinin bir kenarı olduğu söylenebilir [22].



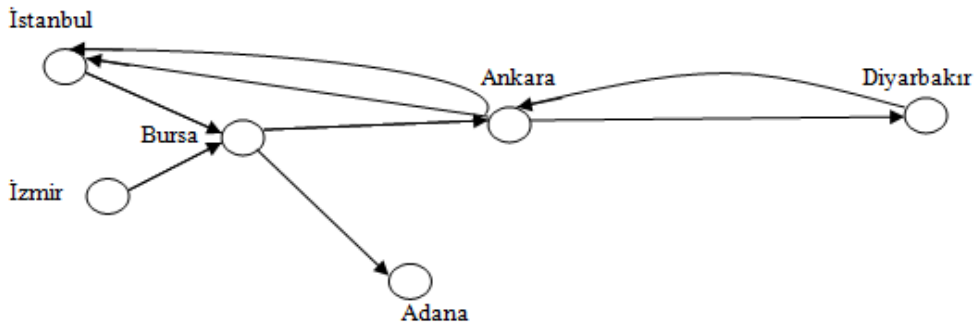
Şekil 2.12 Veri merkezleri arasında çoklu bağlantılı bilgisayar ağı

Bazen bir iletişim bağlantısı herhangi bir veri merkezini teşhis amaçlı olarak geri bildirim döngüsüyle kendisiyle bağlayabilir. Bu tarz bir ağ Şekil 2.13'de gösterilmiştir. Bu ağı modellemek için bir köşeyi kendisine bağlayan kenarları dahil etmek gerekir. Bu tarz kenarlara döngü denir ve bazen bir köşede birden daha fazla döngüye sahip olunabilir.



Şekil 2.13 Teşhis bağlantılı bir ağ yapısı

Bundan önceki graflar yönsüz graflardı. Döngüsel graf olmadığı sürece yönsüz bir grafta maksimum kenar sayısı n (n-1)/2 olur [21]. Kenarları da yönsüzdüler. Bununla birlikte bir graf modeli inşa etmek için grafın kenarlarına yönler atamak da gerekli olabilir. Örneğin, bir bilgisayar ağında bazı bağlantılar sadece tek bir yönde işleyebilir (böyle bağlantılar tekli çift katlı hatlar olarak isimlendirilir) Bu yapı bazı veri merkezlerine gönderilen veride çok fazla trafiğin olduğu, az trafiğin olduğu ya da zıt yönde giden trafiğin olmadığı durumdur. Böyle bir ağ Şekil 2.14'te gösterilmiştir.



Şekil 2.14 Çok yönlü bağlantılı bir bilgisayar ağı

Hatlı bir yönlü graf gösterildiğinde bir kenarın yönünün u'da başlayıp v'de bittiğini gösteren u'dan v'ye bir ok kullanılır. Yönlü bir grafın döngüleri olabilir ve aynı köşede başlayıp biten çoklu yönlü grafı olabilir. Yönlü bir graf ayrıca u ve v köşelerini her iki yönde bağlayan yönlü kenarlara da sahip olabilir, yani bir yönlü graf u'dan v'ye bir kenara sahip olduğunda v'den de u'ya bir ya da daha fazla kenara da sahip olabilir. Yönlü bir grafın döngüsü ve çoklu yönlü kenarı yoksa basit yönlü graf olarak isimlendirilir.

Bazı bilgisayar ağlarında iki şube arasında çoklu iletişim sunulabilir. Bir köşeden ikinci bir köşeye birden fazla yönlü kenara sahip olabilen yönlü graflar böyle ağları modellemek için kullanılır.

Bazı modellerde diğerleri yönlüyken kenarları yönsüz olan bir grafa ihtiyaç duyulur. Hem yönlü hem de yönsüz kenarı olan graflara karışık graf denir [22]. Örneğin, bir karmaşık graf her iki yönde işleyen bağlantıları içeren bir bilgisayar ağını modellemek için kullanılabilir.

2.4.3 Graf Uygulamaları

Sosyal ağlarda graflar, insanlar ve insan grupları arasındaki farklı ilişki çeşitlerine dayalı sosyal yapıları modellemek için yoğun şekilde kullanılır. Bu sosyal yapılar ve onları sunan graflar sosyal ağlar olarak bilinir. Bu graf modellerinde bireyler ya da

organizasyonlar köşelerle gösterilir. Bireyler ve organizasyonlar arasındaki ilişkiler de kenarlarla gösterilir. Sosyal ağların çalışması oldukça aktif disiplinler arası bir alanıdır ve insanlar arasındaki ilişkilerin farklı çeşitleri onları kullanarak incelenir.

İletişim ağlarında aygıtları göstermek için köşeleri kullanarak farklı iletişim ağlarını ve ilişki bağlantılarının özel bir çeşidini görselleştirmek için kenarlar modellenebilir.

Bilgi ağlarında graflar, özel bilgi çeşitlerini bağlayan çeşitli ağları modellemek için kullanılır. Bir graf kullanılarak World Wide Web modellenebilir. Ayrıca farklı dökümanlardaki alıntılar modellenebilir.

İletişim ağlarında graflar, farklı çeşitlerdeki iletişim ağlarını, yol, hava ve raylı ağların farklı türlerini modellemede kullanılabilir.

2.4.4 Diğer Graf Çeşitleri

Yönlerinin olup olmamasına göre, aynı köşeler arasında birden fazla bağlantı olup olmamasına göre ve döngülerin olup olmamasına göre çok farklı graf çeşitleri vardır.

Her kenara bir değer (ağırlık) atanmışsa o grafa ağırlıklandırılmış graf denir [21]. Bu tarz ağırlıklar probleme bağlı olarak örneğin, ücretler, uzunluklar vb. sayısal değerleri gösterebilir.

Düzenli bir graf, her köşenin aynı sayıda komşuya sahip olduğu graftır [21]. Her köşenin derecesi ya da değerliliği aynıdır. k derece köşeli düzenli bir graf, k dereceli düzenli graf ya da k-düzenli grafı olarak adlandırılır.

Her köşenin birbiriyle bağlantısının olduğu graf tam graftır [21].

Bir G grafının V köşe seti sonsuz olabilir. Sonsuz köşe setli bir graf ya da sonsuz kenarlı bir graf sonsuz graf olarak adlandırılmıştır. Tam tersi sonlu köşe setli ve sonsuz kenar setli bir graf da sonlu graf olarak adlandırılmıştır [21].

2.5 Graphviz Yazılımı

Graf görselleştirme; soyut graf ve ağların diyagramları olarak yapısal bilgiyi sunmanın bir yoludur. Graphviz, açık kaynak graf görselleştirme yazılımıdır. İlgili programlama dili DOT, basit metinsel formattaki graf tanımlarını okur ve Bitmap, Ölçülebilir Vektör Grafiği (Scalable Vector Graphic - SVG) ya da PostScript (ps) gibi farklı şekillerde kullanışlı diyagramlar üretir [23].

Metinsel tanım genellikle dış veri kaynağından otomatik olarak oluşturulur, ayrıca elle de (manually) oluşturulabilir ve düzenlenebilir. Graphviz kullanıcı tanımlı özelliklere göre grafların biçimlendirilmesini ve çizimini sağlar. Graphviz'in sağladığı yararları kullanarak ağın içindeki herhangi bir kök hakkında hızlıca bilgi elde edilebilir. Bir analist, bu grafları şüpheli ağ davranışlarını daha kolay tanımlamak için kullanır. Ayrıca bunlar bir analiz kaynağı olarak kullanılabilir.

Graphviz'in standart kök/kenar tabanlı grafları oluşturabilmesinde çeşitli avantajları vardır [24]. Öncelikle jpeg gibi standart grafiksel dosya formatlarını içeren çıktılara izin veren esnek bir yapısı vardır. İkinci olarak da Graphviz paketi; kök rengi, kök şekli, kenar rengi ve kenar genişliği gibi grafların birçok niteliğini değiştirmeye izin verir. Bu nitelikleri kolaylıkla değiştirebilmek graflara ekstra bilgi ekleyebilmeyi sağlar. Son avantajı da birçok programlama arayüzü otomatik bir şekilde onu kullanabilir. Graphviz graf taslağını otomatik bir süreçle oluşturur.

2.5.1 Dot Programlama Dili

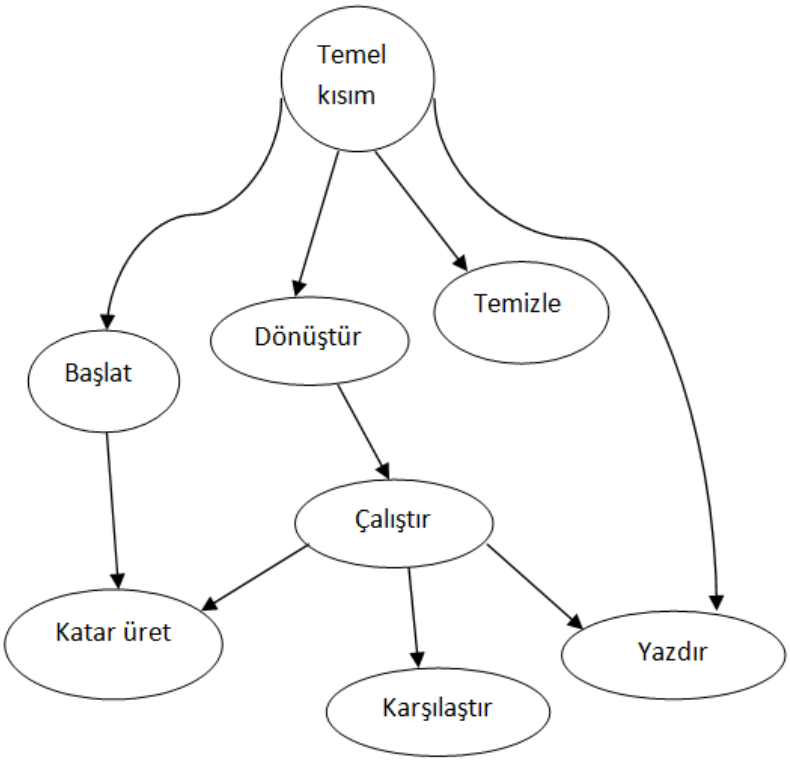
Dot hiyerarşik olarak yönlü graflar çizen bir yapıdır. Eğer kenarlarda yönlülük varsa kullanmak için varsayılan olarak bu araç kullanılır. Dot aynı yöndeki kenarları hedefler (üstten aşağıya ya da soldan sağa) ve ardından kenar uzunluklarını azaltmaya çalışır ve kenar kesişimlerinden kaçınır. Bu yapı, kaynak dosya olarak graf metin dosyalarını okur ve hedef dosya olarak da GIF, PNG, SVG, PDF ya da PostScript olarak grafik formatında dosyalar üretir [25].

Dot grafları dört aşamada üretmektedir [25]. Bu aşamaları bilmek dotun işleyişini anlamayı kolaylaştıracaktır.

Dot, DOT dilinde yazılmış girişleri işleyecektir. Bu dil üç çeşitte nesne tanımlar: graflar, kökler ve kenarlar. Üretilen şekiller yönlü graflardan (`digraph`) veya yönsüz graflardan (`graph`) oluşabilir.

Şekil 2.15'te dot dilinde üretilen bir graf örneği görülmektedir. İlk satır grafın adı ve tipini göstermektedir. Ondan sonraki satırlar kök, kenar ya da alt grafların ilişkisini göstermektedir.

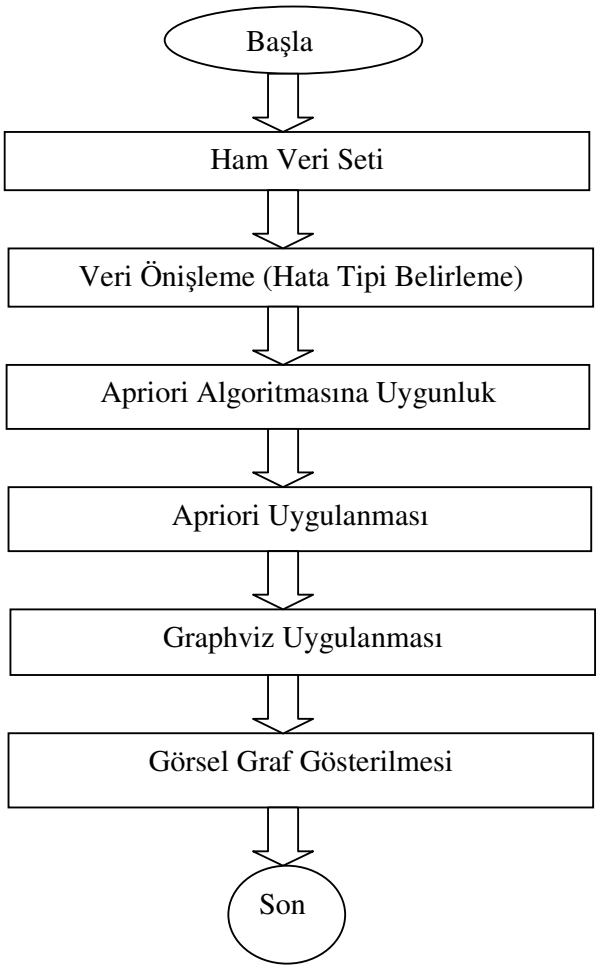
```
Digraph G {
     Temel_kisim -> donustur -> calistir;
     Temel_kisim -> baslat;
     Temel_kisim -> temizle;
     Calistir -> katar_uret;
     Calistir -> yazdir;
     Baslat -> katar_uret;
     Temel_kisim -> yazdir;
     Calistir -> karsilastir;
}
```



Şekil 2.15 Küçük bir graf örneğinin çizilmesi [25]

3. YAZILIM LOGLARI İLE KRİTİK HATA TESPİT SİSTEMİ

Bu bölümde kurumsal bir yazılım şirketinin yazılım hataları analiz edilerek kritiklik tespit sistemi yapılmıştır. Uygulamaya başlamadan önce Şekil 3.1'deki gibi bir işleyiş planlanmıştır. Öncelikle ham veri setine önişleme uygulanarak veri madenciliğine uygun hale getirilmiştir. Daha sonra veri setine Apriori algoritması uygulanarak sık örüntüler tespit edilmiştir. Tespit edilen sık örüntüler Graphviz yazılımı sayesinde graf görseli halinde sunulmuştur.



Şekil 3.1 Sistemin temel akış şeması

3.1 Geliştirme Ortamı ve Kullanılan Araçlar

Uygulamada hataların analizi için C#.NET programlama dili ile bir yazılım geliştirilmiştir. Bu yazılım hataları veritabanından toplanıp, birliktelik analizi gerçekleştirilerek bir Windows servisi aracılığı ile sonuçları e-posta olarak gönderilmektedir.

Bunun dışında hata birlikteliklerini analiz aşamasında da Apriori algoritması ve Graphviz yazılımından faydalanılmıştır.

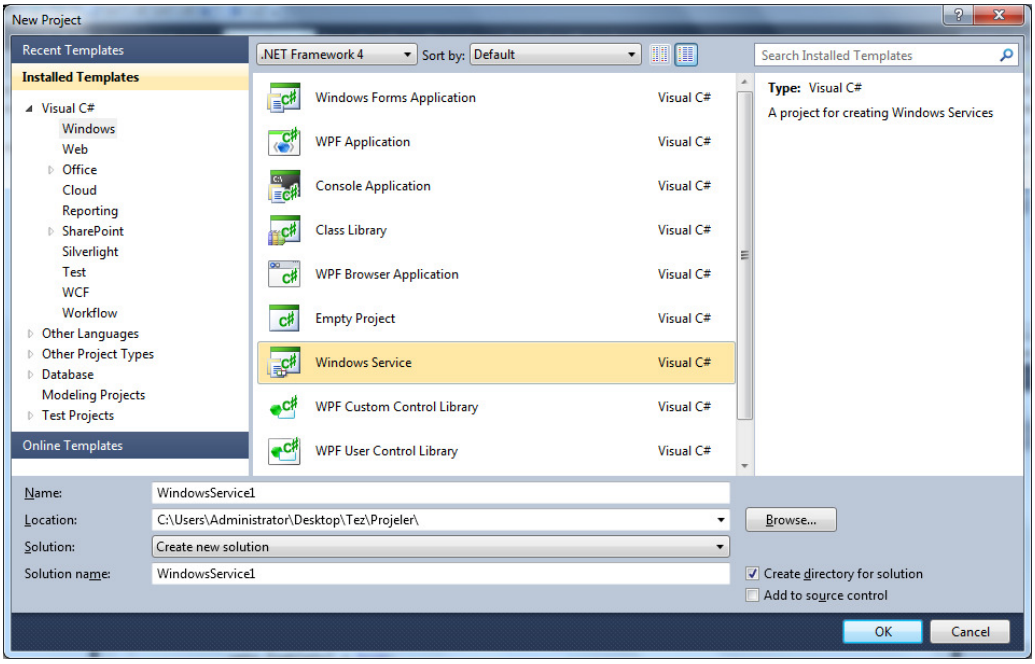
3.1.1 ColdFusion Programlama Dili

ColdFusion 1995'te J. J. Allaire tarafından keşfedilmiş hızlı bir ticari web uygulama geliştirim platformudur [26]. (Bu platformla kullanılan programlama dili de ayrıca ColdFusion olarak adlandırılır, tam isim olarak da CFML denilmektedir) ColdFusion bir veritabanına basit HTML sayfaları bağlamayı kolaylaştırmak için tasarlanmıştır. İkinci versiyonla birlikte (1996) bir betik dili olmasının yanısıra IDE de içeren bir platform olmuştur [26]. 2010 yılından beri ColdFusion'un versiyonları (2005'te Adobe Systems tarafından satın alınmıştır) kurumsal entegrasyon ve zengin internet uygulamalarının gelişimi için gelişmiş özellikler içermektedir.

ColdFusion'un öne çıkan özelliklerinden birisi ColdFusion Markup Language (CFML) betik dilidir,. CFML, amaç ve özellik açısından ASP, JSP ve PHP'nin betik bileşenlerine benzetilir, ama etiket yazımı daha çok HTML'ye benzer. Betik yazımı da JavaScript'e benzer. "ColdFusion" genellikle "CFML" ya da "CFM" ile eşanlamlı kullanılır, ama CFML'in başka desteklediği programlama dilleri ColdFusion ve ColdFusion'un yanı sıra ek CFML uygulama sunucuları vardır, örneğin sunucu taraflı Actionscript ve CFScript olarak bilinen JavaScript formatında yazılabilen gömülü betikler bunlardandır [26].

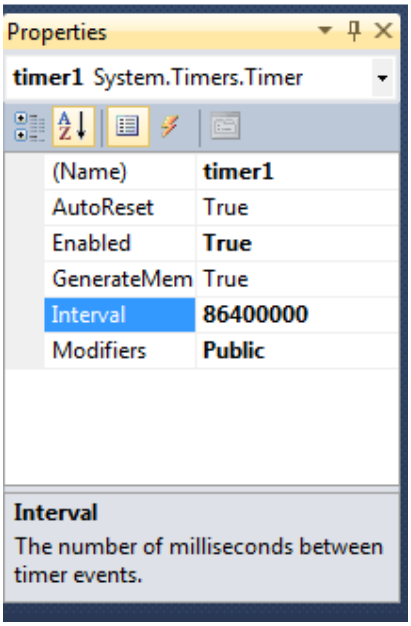
3.1.2 Windows Servisi

Windows servisleri bilgisayarın arka planında her halükarda çalışan yapılardır. Erişim ve yetkileri fazladır. Onlar sayesinde sistem performansına, sistemdeki hatalara, ağ erişimlerine, veritabanı işlemlerine hükmedilebilir.



Şekil 3.2 Örnek bir windows servisi oluşturma ekranı

Şekil 3.2'de görüldüğü gibi C# için yeni bir windows servisi oluşturulurken yeni uygulama oluşturma ekranından windows servisi şablonu seçilerek windows servisi oluşturulmuştur. İşlemin periyodik olarak günlük çalışması istendiği için ilgili uygulamaya standart zamanlayıcıdan farklı bir zamanlayıcı nesnesi eklenmiştir. Şekil 3.3'te görüldüğü gibi bu eklenen zamanlayıcıda periyot zamanı 1 gün olarak (86400000 ms) atanmıştır. Ayrıca uygulamaya bir adet olay logu nesnesi de eklenmiştir.

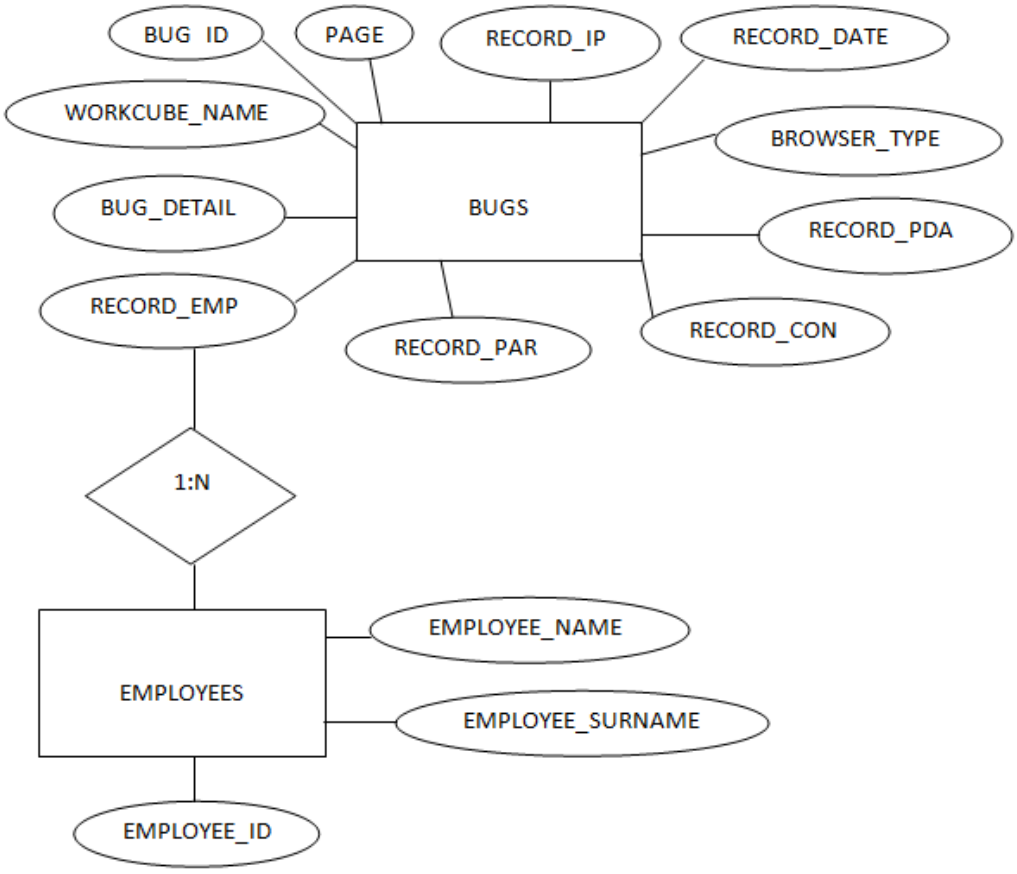


Şekil 3.3 Zamanlayıcı nesnesine periyot zamanının atanması

Windows servisi uygulaması derlenmiş ve oluşturulmuştur. Komut satırından özel komutuyla Windows servisi olarak atanmıştır. Bu aşamadan sonra bilgisayarın Denetim Masası -> Yönetimsel Araçlar -> Hizmetler alanında bir windows servisi olarak görülebilecektir. Diğer Windows servislerine benzer şekilde çalıştırılabilir ve durdurulabilir.

3.1.3 Veritabanı Tasarımı

`BUGS` tablosunun tasarımı gerçekleştirilirken başka bir tablodan da destek alınmıştır. Bu tablo `EMPLOYEES` tablosudur. `EMPLOYEES` tablosu sistemdeki çalışanların bilgilerini tutmaktadır. `BUGS` tablosundaki kayıtlar hatanın yapıldığı her işlemi tutmaktadır. Bir çalışan birden fazla hata yapabileceği için bir çalışan `RECORD_EMP` sütununda birden fazla defa bulunabilir. Bu yüzden `EMPLOYEES` tablosuyla `BUGS` tablosu arasında 1:N ilişkisinin bulunduğu söylenebilir.



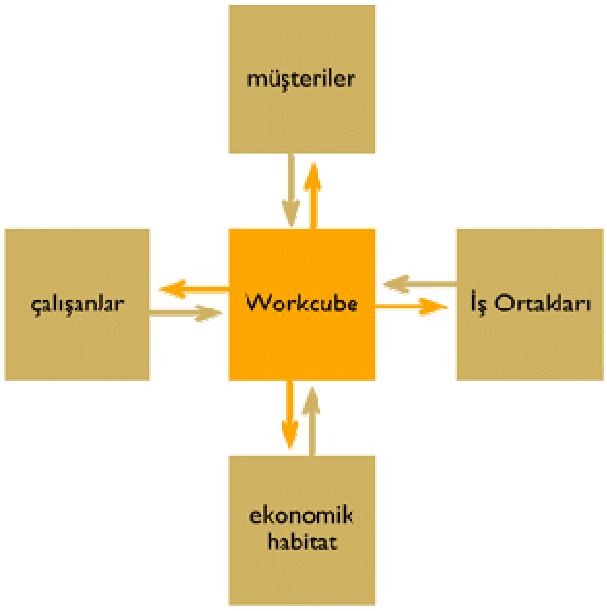
Şekil 3.4 Veritabanı tasarımının varlık-ilişki şeması

3.1.4 Workcube Hakkında

Workcube, %100 web tabanlı ve ölçeklenebilir bir yazılımdır [27]. Workcube, kendi bünyesinde 4500'ün üzerinde iş objesi bulundurmaktadır [30]. Workcube'ün geliştirim çalışmalarına 2001 yılında başlanmıştır ve Adobe ColdFusion, Java ve W3C standartlarıyla geliştirilmiştir [27]. Workcube; Apache, IIS ve HTTP web sunucuları üzerinde çalışabilmektedir [28]. Ayrıca Workcube' ün modüler bir yapısı vardır ve genel olarak ERP (Enterprise Resource Planning – Kurumsal Kaynak Planlama), CRM (Customer Relation Management – Müşteri İlişkileri Yönetimi), PMS (Project Management System – Proje Yönetim Sistemi), PAM (Physical Asset Management – Fiziki Varlık Yönetimi), LMS (Learning Management System – Eğitim Yönetimi Sistemi), CMS (Content Relation Management – İçerik Yönetim

Sistemi), B2B (Business to Business – İşletmeden işletmeye e-ticaret) ve B2C (Business to Consumer – İşletmeden son kullanıcıya e-ticaret) temel başlıkları altında bulunan 52 modülden oluşmaktadır [29]. Workcube, %100 tarayıcı tabanlı ses ve video iletişim sunucusu Flash Communication Server, e-posta ve faks sunucularıyla uyumlu çalışır [27]. Workcube, Windows 2000/2003, Linux Redhat/Suse, Solaris, HP UX işletim sistemleriyle ve Adobe JRun, Adobe ColdFusion, IBM WebSphere, BEA WebLogic uygulama sunucularıyla çalışabilmektedir [31]. MS SQL, DB2 ve ORACLE veritabanları Workcube'ü desteklemektedir [31]. Workcube yazılımı yaklaşık 20-25 kişilik bir ekip tarafından geliştirilmektedir.

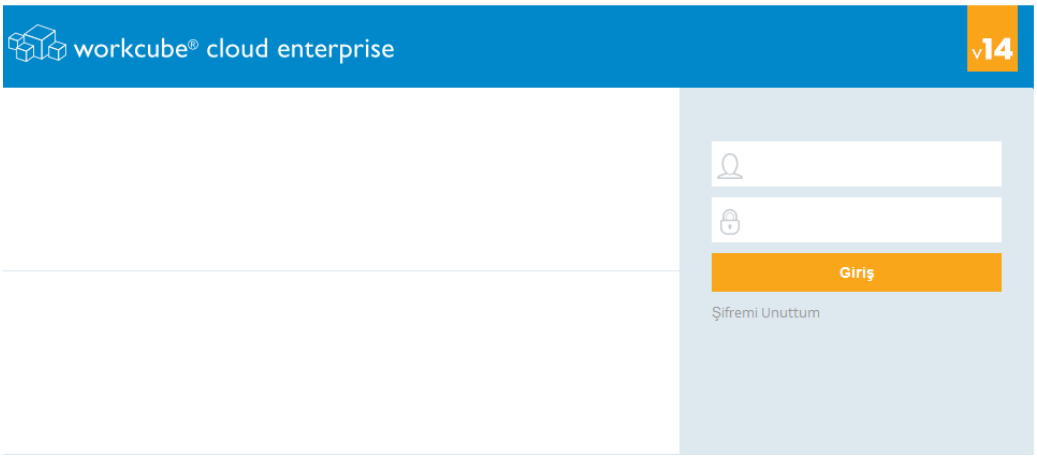
Workcube yazılımı, bir ürünün üretiminden bir e-ticaret sitesi aracılığıyla son kullanıcıya ulaştırılmasına kadar olan bütün aşamalarını içinde bulundurmaktadır. Ürün, bu yaşam döngüsünü farklı modüller içinde devam ettirmektedir.



Şekil 3.5 Workcube yazılımının iletişim noktaları

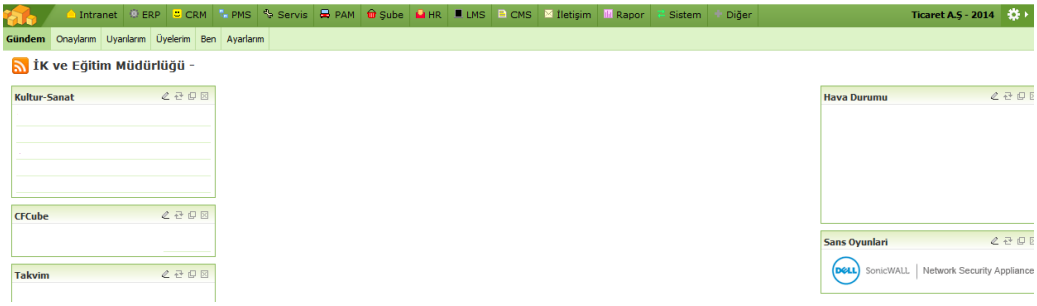
Şekil 3.5'te görüldüğü gibi Workcube yazılımı, kendisine bağlı olan üç grubu temel almaktadır. Bu grubun ilki, kendi çalışanlarıdır. Bu kişiler Workcube yazılımını kullanan ilgili işletmenin çalışanlarından oluşmaktadır. Workcube yazılımına

girmeye sadece bu kişiler yetkilidir. Şekil 3.6'da görüldüğü gibi Workcube yazılımına kullanıcı adı ve şifre bilgileriyle girilir.



Şekil 3.6 Workcube yazılımı giriş ekranı

Sisteme girilince kullanıcı anasayfaya yönlendirilir. Ekranın üst bölümünde iki adet menü bulunmaktadır. Üstteki menü sistemin genel menüsüdür. Alttaki menü ise, kullanıcının o anda Workcube'ün hangi modülünde bulunduğunu göstermektedir. (Şekil 3.7'de görüldüğü gibi Anasayfa, Gündem modülünde yer almaktadır.) Workcube yazılımının anasayfası kullanıcı bazlı olarak özelleştirilebilmektedir.

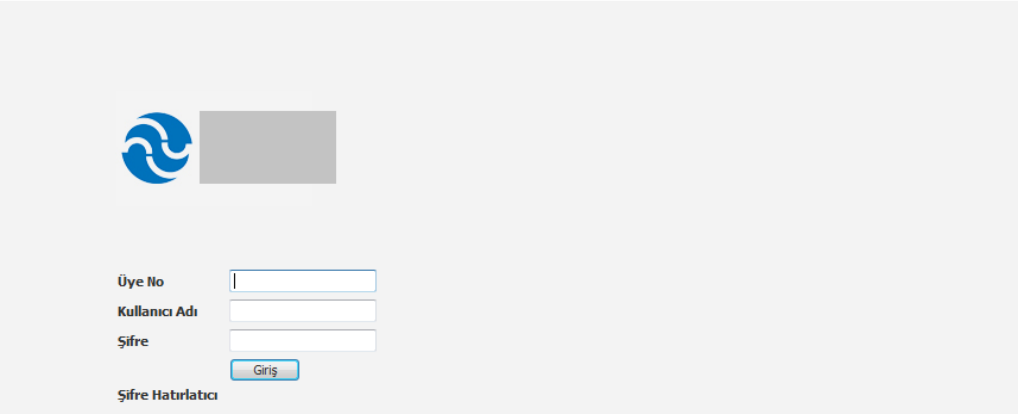


Şekil 3.7 Workcube yazılımı anasayfası

Workcube sisteminde kayıtlı bir diğer grup, üyelerdir. Üyeler yazılımı kullanan işletmenin etkileşim halinde olduğu her türlü üyeleri, müşterileri, bayileri, son kullanıcıları vb. kapsamaktadır. Üyeler; kurumsal ve bireysel üyeler olarak ikiye

ayrılmıştır. Workcube yazılımı, bünyesinde barındırdığı ürünlerin bir e-ticaret sitesi aracılığıyla satılabilmesine olanak sağlamaktadır. Bu siteler B2B (Business-To-Business) ya da B2C (Business-To-Consumer) olabilmektedir. Bu sayede işletmeler, kendi ürünlerini internet üzerinden satabilmektedirler.

Kurumsal üyeler, ilgili işletmenin kurumsal bağlantısının olduğu tüm işletme ve şirketleri kapsar. Şekil 3.8'de görüldüğü gibi kurumsal üyeler Workcube yazılımına bağlı B2B portallerine girme ve kurumsal olarak alışveriş yapma yetkisine sahip olabilirler.



Şekil 3.8 Workcube yazılımına bağlı olarak çalışan örnek bir B2B sitesi giriş sayfası

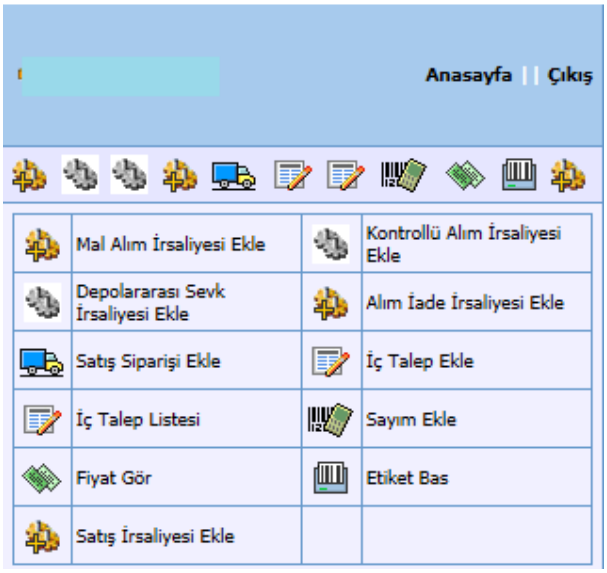
Bireysel üyeler, bağımsız üyelerdir. Şekil 3.9'da görüldüğü gibi bireysel üyeler yazılıma kayıtlı olan son kullanıcılara yönelik e-ticaret sitelerine (B2C) girebilme ve alışveriş yapabilme haklarına sahip olabilirler.

Şekil 3.9 Workcube yazılımına bağlı olarak çalışan örnek bir B2C sitesi

Şekil 3.10'da görüldüğü gibi Workcube üzerinden kariyer portaller oluşturularak adayların İnsan Kaynakları modülünde yer alan CV bankasına form bırakabilmeleri mümkündür.

Şekil 3.10 Workcube yazılımına bağlı olarak çalışan örnek bir kariyer portali

Son olarak Şekil 3.11'de görüldüğü gibi Workcube yazılımına bağlı olarak PDA portali oluşturulabilmektedir. Workcube yazılımını kullanan işletmenin çalışanları PDA portaline giriş yetkisine sahiptirler. Özellikle satış çalışanları, el terminalleri üzerinden satış yaptıklarında bu portali kullanabilmektedirler. Tüm portaller merkezi veritabanına bağlı olarak çalışmaktadır. Bu da Workcube yazılımına tümleşik bir yapı kazandırmaktadır.



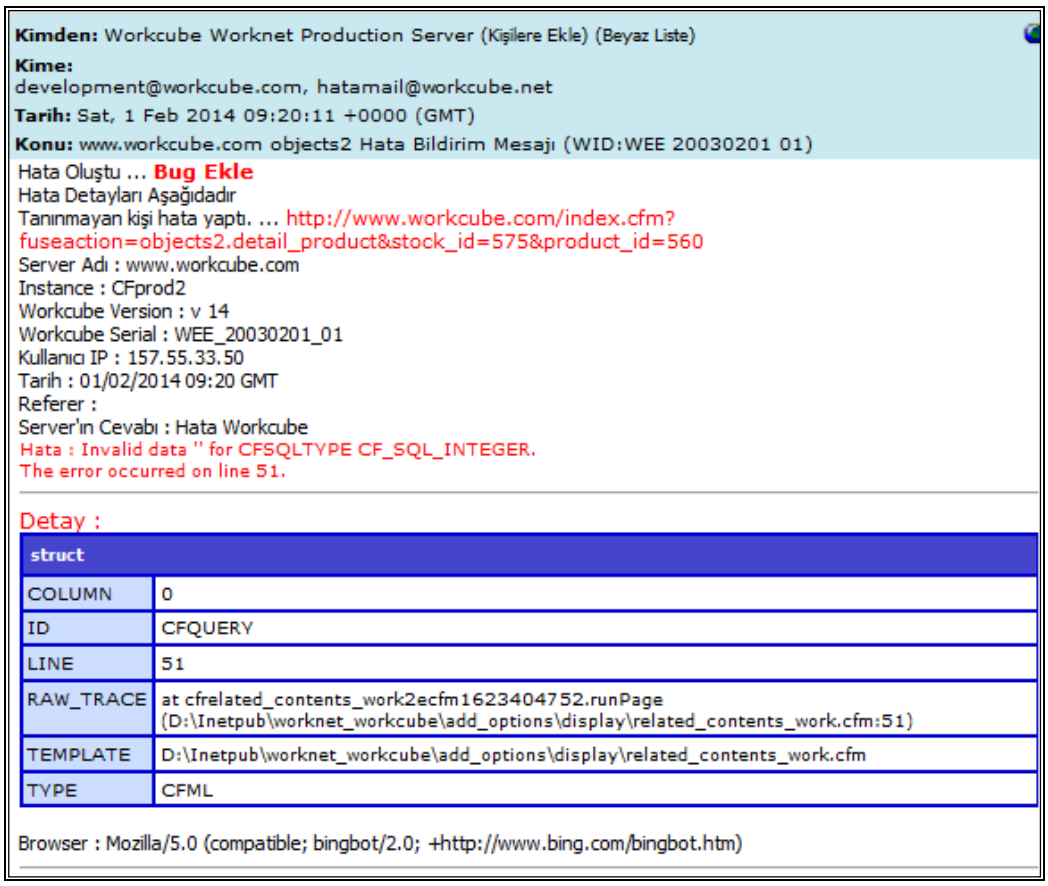
Şekil 3.11 Workcube yazılımına bağlı olarak çalışan örnek bir PDA portali

3.2 Ham Veri Seti

Bu çalışmada metal, mobilya ve perakende sektörü için geliştirilmiş bir ticari programa ait hata logları kullanılmıştır. Ticari program Adobe ColdFusion programlama diliyle geliştirilmiştir.

Normalde program içindeki herhangi bir sayfada yazılımsal bir hata gerçekleştiğinde Şekil 3.12'deki hata e-postası sistemde tanımlı belli kişilere gönderilmektedir. Bu e-posta içinde hatayla ilgili önemli bilgilere yer verilmiştir. Bu sayede sistemin yazılım geliştiricileri yazılım hataları hakkında detaylı bilgi elde edebilirler. Hatayı doğru yorumlayarak hataları ayıklayabilirler ve tamir edebilirler.

Bu çalışmada bu hata e-postaları bir tabloda depolanmıştır. Bunun için tabloda tutulması planlanan alanlar belirlenmiştir. Şekil 3.13'te görüldüğü gibi bu alanlardan oluşan bir tablo oluşturulmuştur. Böylece sistem içinde herhangi bir sayfada yazılımsal bir hata oluştuğunda sistem normal işleyişinde olduğu gibi hatanın detaylı bilgilerini e-posta olarak göndermemiştir. Onun yerine hataları ilgili tabloya yazmıştır. Ardından bu tablodan yapılan çalışmada yararlanılmıştır.



Kimden: Workcube Worknet Production Server (Kişilere Ekle) (Beyaz Liste)

Kime:
development@workcube.com, hatamail@workcube.net

Tarih: Sat, 1 Feb 2014 09:20:11 +0000 (GMT)

Konu: www.workcube.com objects2 Hata Bildirim Mesajı (WID:WEE 20030201 01)

Hata Oluştu ... **Bug Ekle**
Hata Detayları Aşağıdadır
Tanınmayan kişi hata yaptı. ... http://www.workcube.com/index.cfm?fuseaction=objects2.detail_product&stock_id=575&product_id=560
Server Adı : www.workcube.com
Instance : CFprod2
Workcube Version : v 14
Workcube Serial : WEE_20030201_01
Kullanıcı IP : 157.55.33.50
Tarih : 01/02/2014 09:20 GMT
Referer :
Server'ın Cevabı : Hata Workcube
Hata : Invalid data '' for CFSQLTYPE CF_SQL_INTEGER.
The error occurred on line 51.

Detay :

struct	
COLUMN	0
ID	CFQUERY
LINE	51
RAW_TRACE	at cfrelated_contents_work2ecfm1623404752.runPage (D:\Inetpub\worknet_workcube\add_options\display\related_contents_work.cfm:51)
TEMPLATE	D:\Inetpub\worknet_workcube\add_options\display\related_contents_work.cfm
TYPE	CFML

Browser : Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)

Şekil 3.12 Veri madenciliği uygulanan örnek bir hata e-postası

Alan Adı	Sayfa	Tarayıcı Çeşidi	Kaydeden IP	Kaydeden	Hata Detayı
www.deneme.com	xxx.yyyy	MSIE	192.168.18.1	Ali Veli	Error Executing Database Query

Şekil 3.13 Yazılımsal hataların tutulduğu BUGS tablosu

Buna göre test sistemi üzerinde kullanıcıların gerçekleştirdiği veya denk geldiği bir yazılım hatası ilgili tablomuza hata hakkında kayıt atılmasını sağlamıştır. Tablonun alanları şu şekilde tanımlanmıştır:

1. BUG_ID: Hataya ait olan ünik alandır. Birincil anahtardır. (integer)

2. DOMAIN_NAME: Yazılım sistemi bir adet ana adres çubuğu alan adı ve birden fazla alt adres çubuğu alan adları şeklinde kurgulanmıştır. Dolayısıyla hatanın oluştuğu alan adı bu kısımda tutulmaktadır. (nvarchar)

3. BUG_DETAIL: Bu alanda hata hakkında bilgi veren birkaç cümlelik bir açıklama yer almaktadır. (nvarchar)

4. PAGE: İlgili sistem ünik sayfa adlarından oluşmaktadır. Dolayısıyla bu alan hatanın olduğu sayfanın adres çubuğu adını verir. (nvarchar)

5. BROWSER_TYPE: İlgili hatanın gerçekleştiği tarayıcının adı ve sürüm numarası tutulmaktadır. (nvarchar)

6. RECORD_DATE: İlgili hatanın gerçekleştiği tarih değeri tutulmaktadır. (datetime)

7. RECORD_IP: İlgili hatanın gerçekleştiği bilgisayarın IP adresi tutulmaktadır. (nvarchar)

8. RECORD_EMP: İlgili hatanın şirketin hangi çalışanı tarafından gerçekleştirilen bir olay sonucu oluştuğu bilgisi tutulmaktadır. (integer)

9. RECORD_PAR: İlgili hatanın şirketin hangi kurumsal üyesi tarafından gerçekleştirilen bir olay sonucu oluştuğu bilgisi tutulmaktadır. (integer)

10. RECORD_CON: İlgili hatanın şirketin hangi bireysel üyesi tarafından gerçekleştirilen bir olay sonucu oluştuğu bilgisi tutulmaktadır. (integer)

11. RECORD_PDA: İlgili hatanın şirketin hangi çalışanı tarafından PDA ortamında gerçekleştirilen bir olay sonucu oluştuğu bilgisi tutulmaktadır. (integer)

3.3 Veri Önişleme Modülü

Bu bölümde, bir önceki bölümde bir tabloda toplanmış verinin ön işlemeye tabi tutularak veri madenciliğine uygun hale getirilişimden bahsedilmiştir. Bir sonraki adımda uygulanacak Apriori algoritması için veri üzerinde bazı revizeler yapılması gerekmektedir.

Bu revizeler bir anlamda verideki gereksiz bilgilerin temizlenmesinden oluşmaktadır. Aşağıda belirtilen alanlar revizeli haliyle, diğerleri de var olan haliyle Apriori algoritmasının uygulanacağı bir metin dosyasına aktarılır. Bu metin dosyasının verileri tuttuğu belli bir formatı vardır. Burada da verileri aktarırken bu formata sadık kalınmıştır.

Revize yapılan alanlar Bölüm 3.3.1 ve Bölüm 3.3.2'de açıklanmıştır:

3.3.1 Sayfa (PAGE)

Bölüm 3.2'de de belirtildiği gibi sistem üzerindeki sayfa adları PAGE alanında tutulmuştur. Ancak bu alanın tabloda tutulduğu ilk halinde adres çubuğu linki olarak uzun şekli yer almaktaydı. Çizelge 3.1'de görüldüğü gibi her kayıt için aynı olan kısımlar kesilerek sayfa adı sade bir hale getirilmiştir.

Çizelge 3.1 Sayfa alanının sadeleştirmeden önceki ve sonraki hali

SAYFA ADININ TABLODA TUTULDUĞU HALİ
http://ep.workcube/index.cfm?fuseaction=settings.popup_form_add_process_cat
http://ep.workcube/index.cfm?fuseaction=report.detail_inventory_report
REVİZE EDİLMİŞ HALİ
settings.popup_form_add_process_cat
report.detail_inventory_report

3.3.2 Hata Tanımı (`BUG_DETAIL`)

Bir diğer alan da `BUG_DETAIL` alanıdır. Bu alan, hatanın birkaç cümlelik açıklamasıdır. Alanın tablodaki kayıtlarında yer alan tüm hata tanımları soyutlanmış ve statik alanlarına göre ünik bir şekle indirgenmiştir. Bu sayede her hataya ayrı birer tip atanmıştır. İlgili hatalar ve karşılık gelen ünik değerleri Çizelge 3.2'de görülmektedir.

Çizelge 3.2 Örnek hata tipleri

Hata Tanımı	Hata Tipi ID
`Could not find the included template …` (Dosya bulunamaması ile ilgili hata)	**1**
`Error Executing Database Query.` (SQL sorgusunda yaşanan hata)	**2**

`Element … is undefined in …` (Tanımsız değişken hatası)	**3**
`Variable … is undefined in …` (Tanımsız değişken hatası)	**4**
`The resource … was not found …` (Kaynak bulunamadı hatası)	**5**
`Context validation error for …` (İçerik geçerlilik hatası)	**6**
`Invalid CFML construct …` (Geçersiz CFML yapısı)	**7**
`Invalid column index …` (Geçersiz kolon indeksi)	**8**
`Invalid list index …` (Geçersiz kolon indeksi)	**9**
`Attribute validation error for …` (… için nitelik geçerlilik hatası)	**10**

`Division by zero.`
`Division by zero is not allowed …`
(Sıfıra bölme. Sıfıra bölmeye izin verilmiyor.)
	11

`Invalid tag nesting configuration. A query driven queryloop tag is nested inside a queryloop tag that also has a query`	

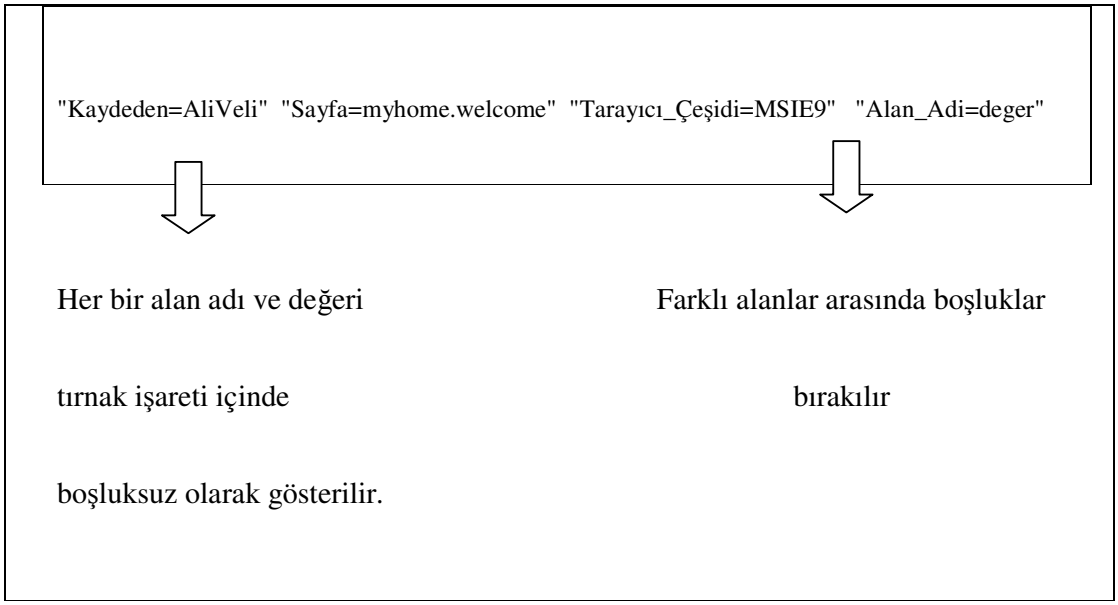
attribute. This is not allowed. Nesting these tags implies that you want to use grouped processing (Geçersiz etiket biçimlendirmesi. Bir sorgu sürümlü Queryloop etiketi bir sorgu niteliğine sahip olan queryloop etiketi içinde konumlanmıştır. Buna izin verilmiyor. Bu etiketleri konumlandırma senin gruplanmış işlemeyi istediğini vurguluyor.)	**12**
Invalid token … on (Geçersiz bir çerez)	**13**
An error occurred when performing … (… işlemini gerçekleştirirken hata oldu)	**14**
The value … cannot be converted to a number (… değeri sayıya çevrilemez)	**15**
The tag <cfif> … requires an end tag. The <cfif> tag requires an end tag to nest within … (<cfif> etiketinin kapanış etiketine ihtiyacı vardır)	**16**
Parameter 1 of function IsDefined, which is now … must be a syntactically valid variable name (IsDefined fonksiyonunun parametresi yazımsal olarak geçerli bir değişken olmalıdır.)	**17**

function keyword is missing in FUNCTION declaration. (function anahtar kelimesi FUNCTION tanımında eksik)	**18**
Invalid data (Geçersiz veri)	**19**
An exception occurred when performing a file operation copy on files （Dosyalar üzerinde dosya kopyalama yapılırken hataya rastlandı）	**20**
NULL is an invalid date or time string. (NULL geçersiz bir tarih veya zaman alanı)	**21**
Date value passed to date function DateAdd is unspecified or invalid. Specify a valid date in DateAdd function. （DateAdd fonksiyonuna aktarılan parametre değeri geçersiz）	**22**
Decryption has failed （Şifre çözme başarısızlığa uğradı）	**23**
cannot convert the value … to a boolean （Değer boolean değere dönüştürülemiyor.）	**24**
No data was received in the uploaded file (Yüklenen dosyada herhangi bir kaydı yok.)	**25**

`function has an invalid parameter` (Fonksiyonun geçersiz bir parametresi var.)	**26**

`Cannot perform web service invocation …`
`The fault returned when invoking the`
`web service`
(Web servis çağırırken hata oluştu.)
	27

`Web service operation … with parameters` (Parametreli … Web servis çağırırken hata oluştu.)	**28**
`Routines cannot be declared more than once.` (Yordamlar birden fazla tanımlanamaz.)	**29**
`Datasource … could not be found` (Veri kaynağı bulunamadı.)	**30**
`A CFML variable name cannot end with` (CFML değişken adı … ile bitemez.)	**31**



Şekil 3.14 Apriori algoritmasının talep ettiği veri formatı

Şekil 3.14, verilerin aktarılmasının ardından metin dosyasının görünümünü göstermektedir.

Sonuç olarak veri önişleme adımında sistem, "hatalar.txt" adında bir metin dosyası oluşturmuştur. Her veri, veritabanındaki alan adıyla tırnak işareti ("") içinde birleştirilmiş ve eşittir (=) işaretiyle de ayrılmış vaziyette oluşturulacaktır. Tırnak işareti içindeki ifadeler içinde boşluk karakteri bulunmaması gerekmektedir.

3.4 Apriori Algoritmasının Uygulanması

Apriori algoritması ilgili algoritmayı uygulayacak bir çalıştırılabilir dosya (.exe dosyası) kullanılarak uygulanır. Sistem, ilgili Apriori dosyasını parametreleriyle birlikte çağırmıştır.

```
apriori.exe -m3  -s5  -v\";\"  -k\"->\"  -q0 hatalar.txt
-o kural.txt
```

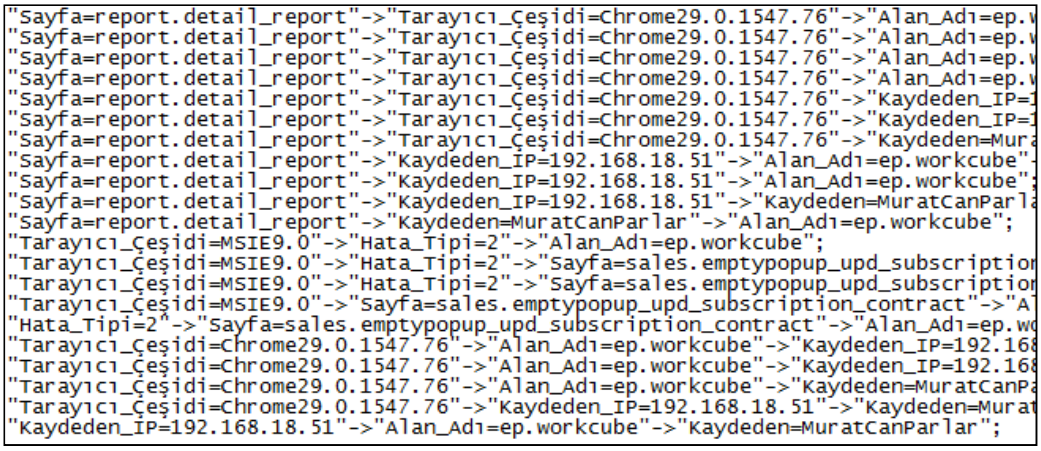
`-m3` : Algoritmadan bulunması istenen birlikteliklerin sayısı

`-s5` : Algoritmanın bulması gereken minimum birliktelik yüzdesi

Uygulamada yer alan aşağıdaki kod bloğu Apriori algoritmasının işletilmesini sağlar. Bunun için C# kütüphanesinde yer alan `System.Diagnostics` kütüphanesinden faydalanılmıştır. Bu kütüphaneye bağlı bir proses oluşturulmuş ve bu prosesle Apriori algoritmasının çalıştırılabilir dosyası (executable file) komut satırından çalıştırılıyor gibi kodun içinden çağrılmıştır ve çalıştırılmıştır. Apriori hedef ve kaynak dosyalarında varsayılan konumu C:\Windows\System32 olarak düşündüğü için dosyaların yolları belirtilmiştir.

```
System.Diagnostics.Process process1 = new System.Diagnostics.Process();

System.Diagnostics.ProcessStartInfo startInfo1 = new
System.Diagnostics.ProcessStartInfo();

startInfo1.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;

startInfo1.FileName = "apriori.exe";

startInfo1.Arguments = "-m3  -s20  -v\";\"  -k\"->\"  -q0 C:/hatalar" + bugun
+ ".txt  -o C:/kural" + bugun + ".txt";
```

Algoritma, `hatalar.txt` dosyasına uygulandığında `kural.txt` dosyası elde edilmiştir. Şekil 3.15'te görüldüğü gibi bu dosyada en çok birlikte geçen 3 örüntünün tüm kayıtlar içinde en az %5' inde geçenleri bulunmaktadır.



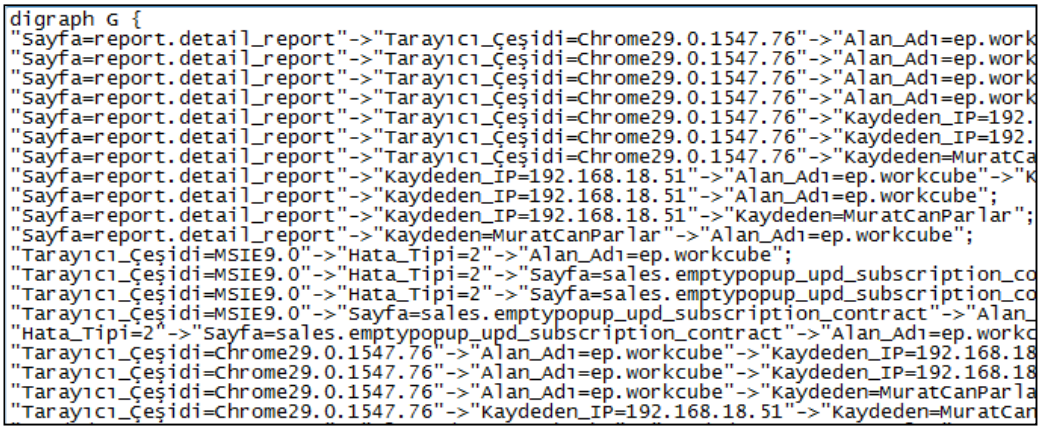
Şekil 3.15 Apriori algoritmasının ürettiği ve sık birliktelikleri gösteren kural dosyası

3.5 Graphviz Yazılımının Uygulanması

Graphviz yazılımı, önceki bölümde elde edilen kural dosyasını görselleştirmeyi sağlayacaktır.

İlgili algoritma, bir çalıştırılabilir dosya (.exe dosyası) kullanılarak uygulanır. Ancak tıpkı Apriori algoritmasında olduğu gibi Graphviz yazılımının da uygulanacağı dosyanın belli bir formata çevrilmesi gerekmektedir. Bunun için öncelikle `kural.txt` dosyası içindeki veriler aşağıdaki bloğun içine dahil edilerek `graf_kural.txt` dosyası oluşturulur.

```
digraph G {

// kural.txt dosyasından gelen kısım

}
```



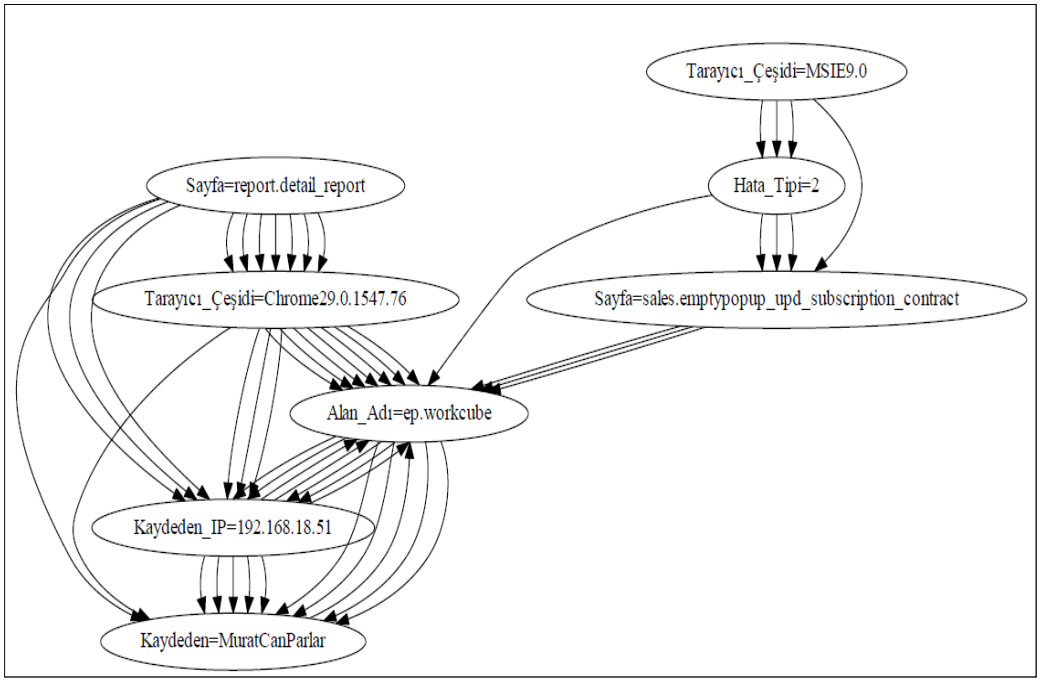
Şekil 3.16 Graphviz yazılımı uygulanacak dosya

Şekil 3.16'da görülen `graf_kural.txt` dosyası hazırlandıktan sonra sistem bu dosyaya aşağıdaki komutu uygulayacaktır.

```
dot -Tpdf graf_kural.txt  -o graf.pdf
```

`-Tpdf` : Oluşturulacak hedef dosyanın uzantısını belirten parametredir.

Komut çalıştırıldığında sistem, hedef dosyayı belirtilen uzantıda oluştuacaktır. Bu dosya `graf.pdf` adında bir pdf dosyası olacaktır. Bu dosya örüntüler arasındaki ilişkiyi Şekil 3.17'de görüldüğü gibi görselleştirmektedir.



Şekil 3.17 Graphviz yazılımı uygulandığında elde edilen graf yapısı

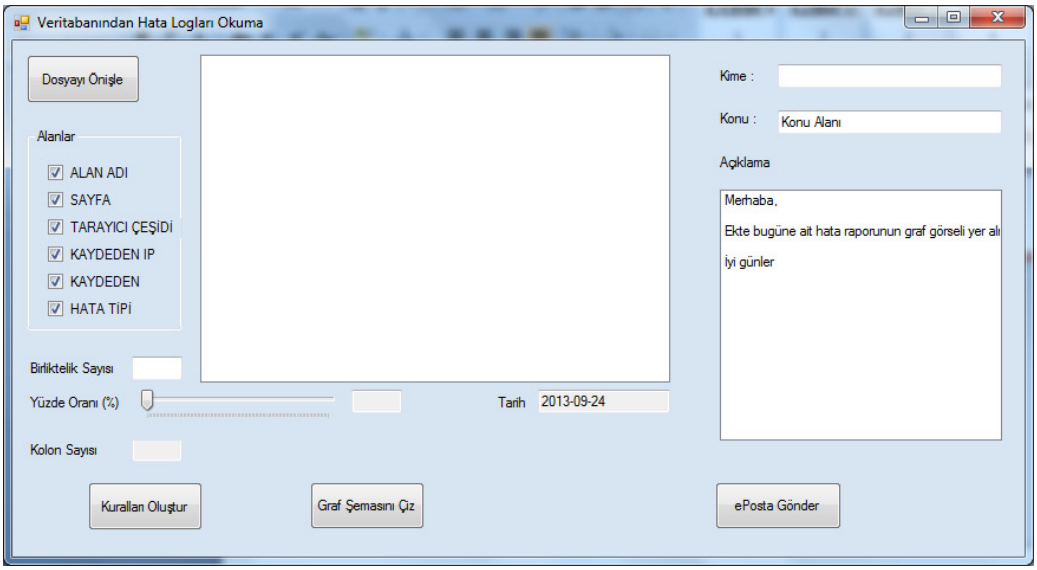
Son aşamada sistem elde edilen bu görseli belirli kişi ya da kişilere e-posta yoluyla gönderebilecektir. İlgili görsel, e-postaya ek dosya olarak eklenebilir.

3.6 Uygulama Türleri

Bu çalışmada iki farklı uygulama tasarlanmış ve gerçekleştirilmiştir. Bu uygulamalardan ilki "Windows servisi tabanlı hata analizi uygulaması", diğeri de "Grafiksel kullanıcı arayüzü tabanlı hata analizi uygulaması"'dır. Windows servisi tabanlı uygulama, çalıştığı bilgisayarın windows servisinin desteği altında işlem görmektedir. Bu uygulama periyodik olarak arkaplanda çalıştığı için herhangi bir arayüze sahip değildir. Diğer uygulamada ise uygulamanın ekran arayüzleri bulunmaktadır. Bu uygulamanın işlemesi kullanıcının seçimine bağlıdır. Periyodik olarak çalışma özelliği bulunmamaktadır.

3.6.1 Grafiksel Kullanıcı Arayüzü Tabanlı Hata Analizi Uygulaması

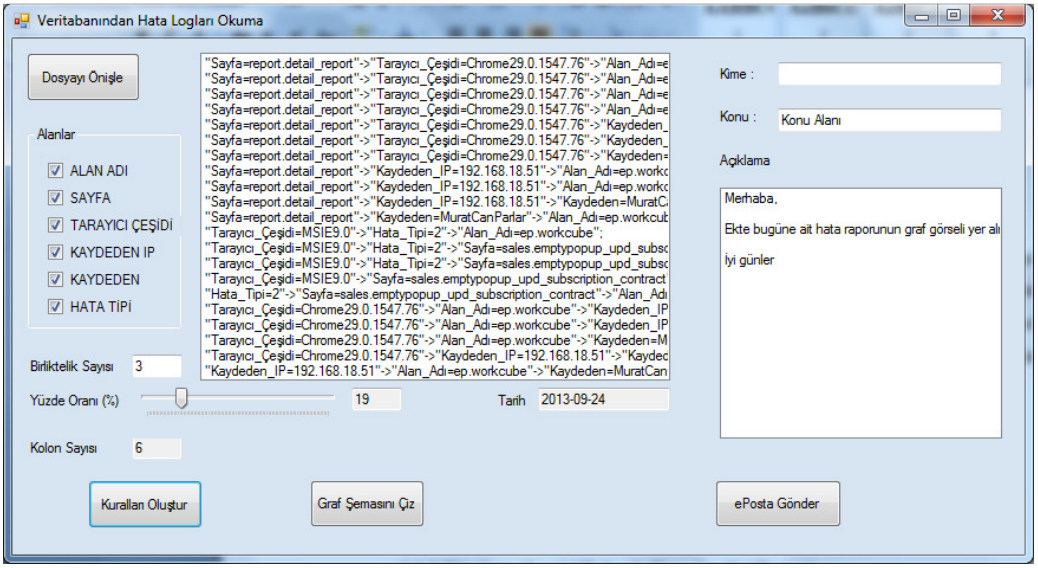
Bu uygulamanın çalıştırılabilir dosyası (executable file) çağrıldığında Şekil 3.18'deki pencere açılacaktır. Ekran üzerinde dört adet buton bulunmaktadır. Bu butonlar temel olarak Şekil 3.1'deki işleyişle paralel hareket etmektedir. Öncelikle kullanıcıdan sayfada tanımlanması gereken bazı bilgileri girmesi beklenir. Ekranın sağ bloğunda yer alan "Alanlar" çerçevesi içinde hataların hangi alanlarının analiz edileceği bilgisi seçilir. Örnek olarak hepsi seçilmiştir.



Şekil 3.18 Ara yüz tabanlı uygulamanın ekran görüntüsü

"Dosyayı Önişle" butonuna basıldığında Bölüm 3.3'de anlatılan işlemler gerçekleştirilerek ekranın "Tarih" değerinde yer alan güne ait hatalar belli bir metin dosyasına atanacaktır. İşlem sorunsuz gerçekleştiğinde bunu "İşlem Başarıyla Gerçekleşti" uyarısıyla bildirecektir. Sonrasında "Birliktelik Sayısı" alanından çıkarılması gereken kuralların maksimum değeri girilmelidir. Bu değer "Alanlar" kısmında seçilmiş alan sayısından fazla olmamalıdır. Ayrıca "Yüzde Oranı" alanından kural dosyasında çıkarılması gereken birlikteliklerin genel verinin minimum yüzde kaçında yer alacağı belirtilmelidir.

Ardından “Kuralları Oluştur” butonuna basılır. Sistem kullanıcıdan ilk butonda oluşmuş hataların dosyasını seçmesini ister. İlgili dosya seçilir. Ekrana işlemin sorunsuz gerçekleştiğini bildiren bir uyarı göstermektedir. Bunun dışında arkaplanda yer alan beyaz çerçeve içerisine de kuralların yer aldığı metin dosyasındaki birliktelikleri yansıtacaktır (Şekil 3.19).

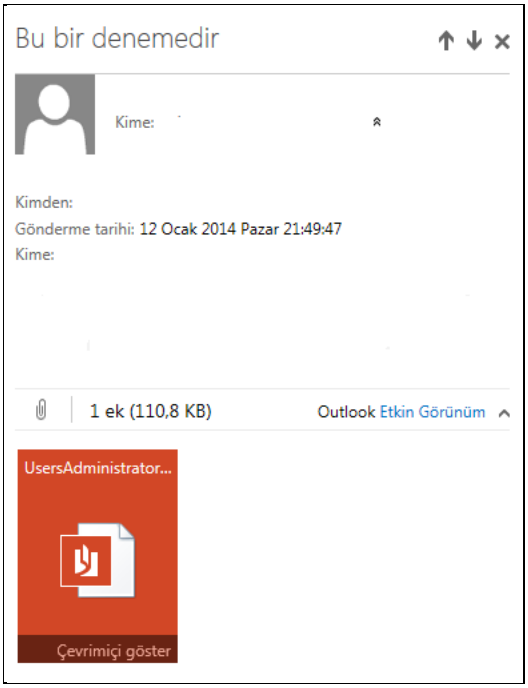


Şekil 3.19 Birliktelik kuralları oluşturulduktan sonra ekran görüntüsü

Bu işlemden sonra kurallar oluşturulmuş ve birliktelikler çıkarılmıştır. Artık bu birliktelikler görselleştirilmelidir. Ekranın alt bölümünde yer alan “Graf Şemasını Çiz” butonuna tıklanır. Bu buton ilgili kuralları görselleştirerek yeni bir windows formu üzerinde PDF okuyucu aracılığıyla Şekil 3.17’deki graf şeklini gösterecektir.

Son olarak bu görsel istenilen kişiyle paylaşılabilecektir. “Kime” kısmında görselin gönderileceği kişinin e-posta adresi girilmelidir. Bunun yanısıra “Konu” alanına bir e-posta başlığı girilmelidir. Bu çerçevede “ePosta Gönder” butonuna basılır. Sistem ilgili kişiye e-postayı göndererek bunu kullanıcıya bildirir.

E-postanın gönderildiği kişi gelen e-postayı açtığında Şekil 3.20’deki ekran sayesinde graf görselini elde edebilecektir.



Şekil 3.20 E-posta aracılığıyla gönderilen ek dosyanın gelen kutusu görünümü

3.6.2 Windows Servisi Tabanlı Hata Analizi Uygulaması

Bu uygulama, daha önceden atanmış belli bir tetiklemeye dayanarak çalıştığı için grafiksel kullanıcı arayüzü uygulamasında dört ayrı butonla gerçekleştirilen fonksiyonları tek çatı altında birleştirmiştir. Windows servisi tabanlı bir uygulama olduğu için Bölüm 3.1.2'deki işlemler uygulanarak altyapısı hazırlanmıştır. Uygulamaya eklenen zamanlayıcı nesnesinin tetiklenme olayına fonksiyon kodları birleştirilerek eklenmiştir. Bu sayede windows servisin her tetiklenmesinde tüm işlemler bir arada gerçekleştirilerek en sonunda ilgili kişiye e-posta gönderilmiştir.

Grafiksel kullanıcı arayüzü uygulamasında "Birliktelik Sayısı", "Yüzde Oranı", "Kolon Sayısı", "E-posta Gönderilecek Kişi" gibi bazı bilgiler dinamik olarak belirlenebiliyordu. Ancak bu uygulamada bu bilgiler statik olarak tanımlanmıştır.

Aşağıda, tez kapsamında geliştirilen windows servisi tabanlı uygulamaya ait kodlardan bir bölüm verilmiştir. Bu kod bloğunda öncelikle hangi güne ait logların

işlem göreceği belirlenmiş ve "bugun" değişkenine atanmıştır. Sonrasında log kayıtlarına ait analiz dosyalarının konumlandırılacağı dosya yolu belirtilmiştir. Daha sonra veritabanına gerekli bilgilerle bağlanılarak veritabanı sorgusu aracılığıyla o günkü hata loglarına erişilmiştir. Kod satırlarında daha detaylı açıklamalar yer almaktadır.

```
private void timer1_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
{
            /* 1. blok */
            /* SQL'den Günlük Hata Loglarının Çekilmesi, Veri Önişleme
Adımları ve hatalar.txt dosyasının oluşturulması */

            //int kolon_sayisi;
            //int kayit_sayisi;
            object Missing = System.Reflection.Missing.Value;
            //String bugun = DateTime.Now.Year.ToString() + '-' +
DateTime.Now.Month.ToString() + '-' + DateTime.Now.Day.ToString(); /* O günün
tarihini (yyy-aa-gg) formatında alan komut */
            String bugun = "2013-09-24";
            //int sonraki = DateTime.Now.Day + 1; /* Ertesi günün tarihini
(yyy-aa-gg) formatında alan komut */
            String yarin = "2013-09-25";
            System.IO.StreamWriter file = new
System.IO.StreamWriter("C:/Users/Administrator/Desktop/Tez/Projeler/PencereHiz
meti2/PencereHizmeti2/bin/Debug/hatalar" + bugun + ".txt");
            SqlConnectionStringBuilder Ayarlar = new
SqlConnectionStringBuilder();
            Ayarlar.DataSource = "192.168.18.2"; /* Veritabanı sunucusunun IP
adresi */
            Ayarlar.InitialCatalog = "calisma";  /* Veritabanının adı */
            Ayarlar.UserID = "gokhanacun";
            Ayarlar.Password = "Gn236-";
            SqlConnection YeniBaglanti = new
SqlConnection(Ayarlar.ConnectionString);  /* Veritabanına bağlantı nesnesi
oluşturuluyor */
            YeniBaglanti.Open();
            SqlCommand Uygula = new SqlCommand("SELECT B.WORKCUBE_NAME,
B.PAGE, B.BROWSER_TYPE, B.RECORD_IP, (SELECT E.EMPLOYEE_NAME + ' ' +
E.EMPLOYEE_SURNAME FROM workcube_cf.dbo.EMPLOYEES E WHERE E.EMPLOYEE_ID =
B.RECORD_EMP) KAYDEDEN, B.BUG_DETAIL FROM BUGS B WHERE (B.BUG_DETAIL <> '' OR
B.BUG_DETAIL IS NOT NULL) AND B.PAGE LIKE '%=%' AND B.RECORD_EMP IS NOT NULL
AND B.RECORD_DATE < '" + yarin + "' AND B.RECORD_DATE >= '" + bugun + "' ORDER
BY B.BUG_ID DESC", YeniBaglanti);
            /* Üstteki komutta veritabanından o günkü hata log kayıtları
çekiliyor */
            SqlDataReader DataOku = Uygula.ExecuteReader();
}
```

4. DENEYSEL SONUÇLAR

Tez çalışması kapsamında bir örnek teşkil etmesi için hata veritabanındaki hatalardan kayıt tarihi 24.09.2013 olanlar incelenmiştir. Veritabanında o güne ait 38 tane hata bulunmaktadır. Bu hataların veritabanındaki görünümleri Şekil 4.1'de yer almaktadır.

Tez kapsamında geliştirilen uygulama %20'den fazla ve en az üçlü geçen birliktelikleri bulmak üzere çalıştırılmıştır. Uygulama ilk etapta veritabanından bu 38 kaydı çekmiştir ve `hatalar.txt` dosyasını oluşturarak bu 38 kaydı Apriori algoritması için hazır duruma getirmiştir. Apriori algoritması %20'den fazla ve en az üçlü birliktelikleri `kural.txt` dosyasına kaydetmiştir.

Buna göre 1 adet beşli birliktelik, 6 adet dörtlü birliktelik ve 14 adet üçlü birliktelik saptanmıştır. Daha sonra bu birlikteliklere Bölüm 3.5'te anlatıldığı gibi Graphviz yazılımı ile graf çizdirilmiştir. Bu yazılım nihai olarak ilgili birliktelikleri Şekil 3.17'de olduğu gibi görselleştirmiştir.

Bu görselden aşağıdaki gibi çıkarımlar yapılabilmektedir:

1- "report.detail_report sayfasında oluşan hatalar mutlaka Chrome29.0 tarayıcısında gerçekleşmiştir."
2- "report.detail_report sayfasında ve Chrome29.0 tarayıcısında olan hatalar bazı durumlarda Murat Can Parlar kullanıcısı tarafından gerçekleştirilmiştir."
3- "sales.emptypopup_upd_subscription_contract sayfasında oluşan hatalar MSIE9.0 tarayıcısında gerçekleşmiştir. Bunlardan büyük çoğunluğu 2. Hata tipidir. (Çizelge 4.2, "Error Executing Database Query")"

	WORKCUBE_NAME	PAGE	BROWSER_TYPE	RECORD_IP	KAYDEDEN	BUG_DETAIL
1	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 9.0	192.168.18.39	Gökhan Acun	Error Executing Database Query. [Macromedia][SQLS...
2	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 9.0	192.168.18.39	Gökhan Acun	Error Executing Database Query. [Macromedia][SQLS...
3	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 9.0	192.168.18.39	Gökhan Acun	Error Executing Database Query.
4	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 9.0	192.168.18.39	Gökhan Acun	Error Executing Database Query.
5	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Variable order_back_demand_total_11_9_2013 is un...
6	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Variable order_back_demand_total_11_9_2013 is un...
7	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Variable DAY_ORDER is undefined.
8	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Variable TOTAL_20 is undefined.
9	ep.workcube	ep.workcube/index.cfm?useaction=product.list_pr...	MSIE 9.0	192.168.18.39	Gökhan Acun	Variable BROWSERDETECT is undefined.
10	ep.workcube	ep.workcube/index.cfm?useaction=project.projects	MSIE 10.0	192.168.18.75	Osman Selvi	Variable BROWSERDETECT is undefined.
11	ep.workcube	ep.workcube/index.cfm?useaction=product.list_pr...	MSIE 9.0	192.168.18.39	Gökhan Acun	Variable BROWSERDETECT is undefined.
12	ep.workcube	ep.workcube/index.cfm?useaction=product.list_pr...	MSIE 9.0	192.168.18.39	Gökhan Acun	Variable BROWSERDETECT is undefined.
13	ep.workcube	ep.workcube/index.cfm?useaction=project.projects	MSIE 10.0	192.168.18.75	Osman Selvi	Variable BROWSERDETECT is undefined.
14	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Variable TOTAL_20 is undefined.
15	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Invalid CFML construct found on line 272 at column 1...
16	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Invalid CFML construct found on line 272 at column 1...
17	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	Chrome 29.0.1547.76	192.168.18.51	Murat Can Parlar	Invalid CFML construct found on line 272 at column 1...
18	ep.workcube	ep.workcube/index.cfm?useaction=objects.empty...	MSIE 9.0	192.168.18.50	Gökcem Bulut	Element ACTION_ID is undefined in ATTRIBUTES.
19	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 10.0	192.168.18....	Fatih Ayık	Invalid tag nesting configuration. A query driven query...
20	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 9.0	192.168.18.17	Şeyma Santaş	Element POSITION_CODE is undefined in EP.
21	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 10.0	192.168.18....	Fatih Ayık	Variable FINISDATE is undefined.
22	ep.workcube	ep.workcube/index.cfm?useaction=test.bombos	MSIE 10.0	10.0.0.15	Emin Çokyaşa...	Could not find the included template Emin/bos.cfm. N...
23	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18.22	Esra Varlı	Error Executing Database Query. [Macromedia][SQLS...
24	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 10.0	192.168.18....	Fatih Ayık	Error Executing Database Query. [Macromedia][SQLS...
25	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18.22	Esra Varlı	Error Executing Database Query. [Macromedia][SQLS...
26	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18.22	Esra Varlı	Error Executing Database Query. [Macromedia][SQLS...
27	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18.22	Esra Varlı	Error Executing Database Query. [Macromedia][SQLS...
28	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18.22	Esra Varlı	Error Executing Database Query. [Macromedia][SQLS...
29	ep.workcube	ep.workcube/index.cfm?useaction=test.bombos	MSIE 10.0	10.0.0.15	Emin Çokyaşa...	Variable SAAT2 is undefined.
30	ep.workcube	ep.workcube/index.cfm?useaction=test.bombos	MSIE 10.0	10.0.0.15	Emin Çokyaşa...	Variable SAAT2 is undefined.
31	ep.workcube	ep.workcube/index.cfm?useaction=ehesa	MSIE 9.0	192.168.18.15	Şenay Gargacı	Invalid list index 2. In function ListGetAt(list, index [, d...
32	ep.workcube	ep.workcube/index.cfm?useaction=report.detail_re...	MSIE 9.0	192.168.18.17	Şeyma Santaş	Element RECORDCOUNT is undefined in GET_POSI...
33	ep.workcube	ep.workcube/index.cfm?useaction=objects.empty...	MSIE 9.0	192.168.18.17	Şeyma Santaş	Error Executing Database Query. [Macromedia][SQLS...
34	ep.workcube	ep.workcube/index.cfm?useaction=objects.empty...	MSIE 9.0	192.168.18.90	Şafak Bilir	Element CANDIDATE_POS_1 is undefined in GET_E...
35	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18....	Merve Kılınçar...	Error Executing Database Query. [Macromedia][SQLS...
36	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18....	Merve Kılınçar...	Error Executing Database Query. [Macromedia][SQLS...
37	ep.workcube	ep.workcube/index.cfm?useaction=sales.emptypo...	MSIE 9.0	192.168.18....	Merve Kılınçar...	Error Executing Database Query. [Macromedia][SQLS...
38	ep.workcube	ep.workcube/index.cfm?useaction=objects2.empt...	MSIE 9.0	192.168.18....	Merve Kılınçar...	Invalid list index 2. In function ListGetAt(list, index [, d...

Şekil 4.1 24.09.2013 tarihinde oluşan hataların veritabanı görünümü

```
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube"->"Kaydeden_IP=192.168.18.51"->"Kaydeden=MuratCanParlar";
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube"->"Kaydeden_IP=192.168.18.51";
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube"->"Kaydeden=MuratCanParlar";
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube";
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Kaydeden_IP=192.168.18.51"->"Kaydeden=MuratCanParlar";
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Kaydeden_IP=192.168.18.51";
"Sayfa=report.detail_report"->"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Kaydeden=MuratCanParlar";
"Sayfa=report.detail_report"->"Kaydeden_IP=192.168.18.51"->"Alan_Adı=ep.workcube"->"Kaydeden=MuratCanParlar";
"Sayfa=report.detail_report"->"Kaydeden_IP=192.168.18.51"->"Alan_Adı=ep.workcube";
"Sayfa=report.detail_report"->"Kaydeden=MuratCanParlar"->"Alan_Adı=ep.workcube";
"Tarayıcı_Çeşidi=MSIE9.0"->"Hata_Tipi=2"->"Alan_Adı=ep.workcube";
"Tarayıcı_Çeşidi=MSIE9.0"->"Hata_Tipi=2"->"Sayfa=sales.emptypopup_upd_subscription_contract"->"Alan_Adı=ep.workcube";
"Tarayıcı_Çeşidi=MSIE9.0"->"Hata_Tipi=2"->"Sayfa=sales.emptypopup_upd_subscription_contract";
"Tarayıcı_Çeşidi=MSIE9.0"->"Sayfa=sales.emptypopup_upd_subscription_contract"->"Alan_Adı=ep.workcube";
"Hata_Tipi=2"->"Sayfa=sales.emptypopup_upd_subscription_contract"->"Alan_Adı=ep.workcube";
"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube"->"Kaydeden_IP=192.168.18.51"->"Kaydeden=MuratCanParlar";
"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube"->"Kaydeden_IP=192.168.18.51";
"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Alan_Adı=ep.workcube"->"Kaydeden=MuratCanParlar";
"Tarayıcı_Çeşidi=Chrome29.0.1547.76"->"Kaydeden_IP=192.168.18.51"->"Kaydeden=MuratCanParlar";
"Kaydeden_IP=192.168.18.51"->"Alan_Adı=ep.workcube"->"Kaydeden=MuratCanParlar";
```

Şekil 4.2 24.09.2013 tarihinde oluşan hata birliktelikleri

5. DEĞERLENDİRME VE ÖNERİLER

Bu çalışmada, hata birliktelikleri tespit edilerek hatalar hakkında çıkarımlarda bulunulmuştur. Bu çıkarımlar sistem üzerinde oluşan hataların genel çerçevesini sunmaktadır. Bu sayede hataların en çok hangi koşullarda oluştuğu tespit edilebilir ve bu koşullar düzeltilerek veya iyileştirilerek hatalar minimize edilebilirler. Böylece, daha stabil bir yazılım sistemi elde etmek mümkün olacaktır.

Proje kapsamında geliştirilen uygulamanın çalıştırılması sonucunda graf biçiminde görseller elde edilmiştir. Bu graf görseli belli bir klasörde hangi güne ait olduğunu belirtecek şekilde tarih biçimiyle isimlendirilmiştir. (örn. graf21032013.pdf, graf22032013.pdf, graf23032013.pdf vb.) Bu sayede, her ne kadar bu görseller ilgili kişilere e-posta yolula gönderilmiş olsa da arşiv amaçlı ve geriye dönük inceleme yapabilmek amacıyla bir klasörde saklanması sağlanmıştır.

Gelecekteki çalışmalarda graf görsellerinin belli bir klasörde tutulması yerine bir dosya yönetimi sistemi aracılığıyla istenilen zamanda istenilen tarihe ait görsel elde edilebilecek şekilde daha gelişmiş bir arşiv yapısında saklanması planlanmaktadır. Ayrıca, graf görselini otomatik olarak yorumlayabilecek bir yapı inşa edilmesi planlanmaktadır. Bu yapı, graf görselini yorumlayarak görsel hakkındaki önermeleri otomatik olarak sunabilecektir.

Bu çalışmadaki analizde sadece firma çalışanlarının sebep olduğu hatalar incelenmiştir. Gerçek uygulamada, ilgili sistemdeki kurumsal veya bireysel üyelerin de oluşturdukları hatalar loglanabilmektedir. Geliştirilen yazılımın kapsamı genişletilerek sistemdeki diğer birey ve öğelerin oluşturduğu hatalar da incelenebilir. Bu sayede sisteme bağlı çalışan e-ticaret sitelerinde en çok hangi sayfalarda ne tür hatalara denk gelindiğine dair çıkarımlarda bulunulabilir. Bu alanlarda uğraş verecek araştırmacılara hata analizi uygulamasını web tabanlı bir yapıya getirilerek e-ticaret sitelerindeki hataları gerçek zamanlı analiz edebilecek hale dönüştürmeleri önerilebilir.

6. KAYNAKLAR

1. http://en.wikipedia.org/wiki/Data_mining, (Erişim Tarihi: 08.01.2014)

2. Thuraisingham B., "Data Mining for Malicious Code Detection and Security Applications", Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on, ISBN : 978-0-7695-3801-3, Milan, 2009

3. Bora S.P. "Data mining and ware housing", Electronics Computer Technology (ICECT), 2011 3rd International Conference on, ISBN : 978-1-4244-8678-6, Kanyakumari, 2011

4. http://conceptsfordw.blogspot.com.tr/2009/04/notable-uses-of-data-mining_07.html (Erişim Tarihi: 09.02.2014)

5. http://en.wikipedia.org/wiki/Software_mining, (Erişim Tarihi: 08.01.2014)

6. Nagwani N. K., Bhansali A., "A Data Mining Model to Predict Software Bug Complexity Using Bug Estimation and Clustering", 2010 International Conference on Recent Trends in Information, ISBN 13: 978-1-4244-5956-8, Kochi, Kerala, 2010.

7. Breckel A., "Error Mining: Bug Detection Through Comparison with Large Code Databases", Mining Software Repositories (MSR), 9th IEEE Working Conference on, ISBN : 978-1-4673-1760-3, Zürich, 2012

8. Wijayasekara D., Manic, M., Wright, J.L., McQueen, M., "Mining Bug Databases for Unidentified Software Vulnerabilities", Human System Interactions (HSI) 5th International Conference on, ISBN : 978-1-4673-4498-2, Perth, WA, 2012

9. Nagwani N. K., Verma S., "Predicting Expert Developers for Newly Reported Bugs Using Frequent Terms Similarities of Bug Attributes", ICT and Knowledge Engineering (ICT & Knowledge Engineering), 2011 9th International Conference on, ISBN : 978-1-4577-2161-8, Bangkok, 2012

10. Gegick, M., Rotella, P., Tao Xie, "Identifying security bug reports via text mining: An industrial case study", Mining Software Repositories (MSR), 7th IEEE Working Conference on, ISBN : 978-1-4244-6802-7, Cape Town, 2010

11. Lamkanfi, A., Perez, J., Demeyer, S., "The Eclipse and Mozilla defect tracking dataset: A genuine dataset for mining bug information", Mining Software Repositories (MSR), 10th IEEE Working Conference on, ISBN : 978-1-4799-0345-0, San Francisco, CA, 2013

12. Nagappan, M., Vouk, M.A., “Abstracting log lines to log event types for mining software system logs”, Mining Software Repositories (MSR), 7th IEEE Working Conference on, ISBN : 978-1-4244-6802-7, Cape Town, 2010

13. Lei-Yue Yao, Jian-Ying Xiong, “The research and implementation of a correlative degree mining algorithm based on IIS logs”, Granular Computing, 2009, GRC '09. IEEE International Conference on, ISBN : 978-1-4244-4830-2, Nanchang, 2009

14. Tantithamthavorn C., Teekavanich R., Ihara A., Matsumoto K., “Mining A change history to quickly identify bug locations : A case study of the Eclipse project”, Software Reliability Engineering Workshops (ISSREW), IEEE International Symposium on, Pasadena, CA, USA, 2013

15. Esichaikul, Vatcharaporn, Guha, S., Niyamosatha, T., “Mining email transaction logs to locate significant messages and users”, Advanced Information Management and Service (IMS), 2010 6th International Conference on, ISBN : 978-1-4244-8599-4, Seoul, 2010

16. Ngok P., Zhiguo Gong, “Log mining to support web query expansions”, Information and Automation, 2009. ICIA '09. International Conference on, ISBN : 978-1-4244-3607-1, Zhuhai, Macau, 2009

17. Zhen Ming Jiang, Hassan, A.E. ; Flora, P. ; Hamann, G., “Abstracting Execution Logs to Execution Events for Enterprise Applications”, Quality Software,. QSIC '08. The Eighth International Conference on, ISBN : 978-0-7695-3312-4, Oxford, 2008

18. Liu Y., Liu Q., Zheng D., Deng Q., “Application and improvement discussion about Apriori algorithm of association rules mining in cases mining of influenza treated by contemporary famous old Chinese medicine”, Bioinformatics and Biomedicine Workshops (BIBMW), 2012 IEEE International Conference on, ISBN : 978-1-4673-2746-6, Philadelphia, 2012

19. Anand H.S., Vinodchandra S.S., “Applying correlation threshold on Apriori algorithm”, Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on, ISBN : 978-1-4673-5037-2, Tirunelveli, 2013

20. http://en.wikipedia.org/wiki/Apriori_algorithm (Erişim Tarihi: 19.01.2014)

21. http://en.wikipedia.org/wiki/Graph_(mathematics)(Erişim Tarihi: 13.04.2014)

22. Kenneth H. Rosen, “Discrete Mathematics and its Applications”, http://web.karabuk.edu.tr/hakankutucu/Discrete_Mathematics_and_Its_Applications_7th_Edition_Rosen.pdf ,13.04.2014

23. Schneider A., Walter S., Langer J., Heinkel U., "Automatic Visualization of Abstract System Specifications", Quality Software, 2006. QSIC 2006. Sixth International Conference on, ISBN : 0-7695-2718-3, Pekin, 2006

24. Fox C., Wilson D., "Visualization in Interrogator using Graphviz", Information Assurance Workshop, 2006 IEEE, ISBN : 1-4244-0130-5, West Point, NY, 2006

25. Emden R. Gansner, Eleftherios Koutsofios, Stephen North, "Drawing graphs with dot", 2010

26. http://en.wikipedia.org/wiki/Adobe_ColdFusion (Erişim Tarihi: 20.01.2014)

27. http://www.workcube.com/workcube-nedir (Erişim Tarihi: 13.04.2014)

28. http://eng.workcube.com/workcube-robust-enterprise-technology-platform (Erişim Tarihi: 13.04.2014)

29. http://eng.workcube.com/what-is-workcube (Erişim Tarihi: 13.04.2014)

30. http://eng.workcube.com/thank-you-for-showing-interest-in-workcube (Erişim Tarihi: 13.04.2014)

31. http://www.workcube.com/neden-coldfusion (Erişim Tarihi: 13.04.2014)

.

ÖZGEÇMİŞ

Gökhan Acun, 1986 yılında Sinop'da doğdu. Öğrenimlerini sırasıyla Piripaşa İlköğretim Okulu, Kağıthane Anadolu Lisesi'nde (Ortaokul - Lise) tamamladı. 2004 yılında Trakya Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı ve 2008 yılında mezun oldu. Bir süre (6 ay) İngiltere'nin Eastbourne şehrinde St. Giles College dil okulunda ingilizce eğitimi aldı. Nisan 2010 tarihinden bu yana Workcube E-İş Sistemleri şirketinde yazılım geliştirici pozisyonunda çalışmaktadır. Eylül 2011 tarihinde Maltepe Üniversitesi Bilgisayar Mühendisliği Bölümü'nde yüksek lisans eğitimine başlamıştır.