Değişiklik Bağlaşımı ve Yazılım Hataları İlişkisinin İncelenmesi

Serkan Kırbaş[1], Alper Şen[1], Bora Çağlayan[2], Ayşe Bener[2]

[1] Bilgisayar Mühendisliği Bölümü
Boğaziçi Üniversitesi
İstanbul, Türkiye,
serkan.kirbas@boun.edu.tr, alper.sen@boun.edu.tr
[2] Bilgisayar Mühendisliği Bölümü
Ryerson Üniversitesi
Toronto, Kanada,
bora.caglayan@ryerson.ca, ayse.bener@ryerson.ca

Özet Bir yazılım sistemi değişirken, yazılım depoları evrilen yazılım sisteminin ve ilgili yazılım sürecinin ayak izlerini tutarlar. Özellikle sürüm kontrol sistemlerinden revizyon geçmişi kullanılarak sistemin hangi bölümlerinin yaygın olarak beraber değiştirildiği bulunabilir. Bu kavram değişiklik bağlaşımı olarak tanımlanmaktadır. Bu çalışmada endüstriyel bir yazılım geliştirme ortamında değişiklik bağlaşımı ile yazılım hatalarının ilişkisi araştırılmıştır.

Keywords: Yazılım Depoları Madenciliği, Değişiklik Bağlaşımı, Yazılım Hataları, Endüstriyel Yazılım, Ölçümleme

1 Giriş

Yazılım sistemleri, rekabet, inovasyon, maliyet azaltma ve yasal düzenlemeler gibi birçok etkenden dolayı değişikliklere maruz kalırlar. Bir yazılım sistemi değişirken, yazılım depoları evrilen yazılım sisteminin ve ilgili yazılım sürecinin ayak izlerini tutarlar. Yazılım depolarındaki zengin verilerin sürekli analizi ile, yazılım evrimi ve evrim/bakım süreci hakkında içgörü elde edilir. Bu da yazılım geliştiricilerin ve yöneticilerin karar alma süreçlerini destekler ve yönlendirir. Yazılım Depoları Madenciliği (YDM) çalışmalarının ortaya koyduğu kavramlardan biri de değişiklik bağlaşımıdır (Change Coupling ya da Evolutionary Coupling). Değişiklik bağlaşımı, yazılım sisteminin evrimi sırasında sık sık birlikte değiştiği gözlenen iki veya daha fazla yazılım parçasının (artifact) arasındaki örtülü ilişkiler olarak tanımlanabilir. Günümüzde yazılımların boyutlarının arttığını ve her geçen gün daha da karmaşıklaşarak diğer yazılımlarla etkileştiğini göz önüne alırsak değişiklik bağlaşımı konusunun ve etkilerinin incelenmesinin önemi ortaya çıkar. Özellikle her yıl dünyada milyarca dolar harcanan yazılım hataları üzerindeki etkilerinin incelenmesi önemlidir.

Değişiklik bağlaşımı kavramı üzerinde, mimari bozulma gibi tasarım problemlerine işaret edebileceğini [1] [2] ve hata tahminlemede kullanılabileceğini [3]

gösteren çalışmalar literatürde mevcuttur. Buna karşın bazı çalışmalarda [4] [5] değişiklik bağlaşımı ölçütlerinin hata tahminlemede iyi sonuçlar vermediği ortaya konmuştur. Değişiklik bağlaşımının yazılım hataları ile ilişkisi konusunda daha fazla ampirik araştırma yapılmasına ve somut etkilerinin ortaya konulmasına ihtiyaç vardır. Bu çalışma kapsamında değişiklik bağlaşımının yazılım hataları ile ilişkisi üzerinde çalışmalar yapılmıştır. Belirlenen değişiklik bağlaşımı ölçütleri ile hatalar arasındaki korelasyon incelenmiştir. Çalışmanın araştırma sorusu şu soru ile ifade edilmiştir:

- **Değişiklik bağlaşımı ile yazılım hataları arasında nasıl bir ilişki vardır?**

Çalışmamızın sonuçları, incelenen yazılımın bakım/evrimi fazında değişiklik bağlaşımının yazılım hataları ile ilişkili olabileceğini göstermiştir. Değişiklik bağlaşımı ölçütleri ve hata ölçütleri arasındaki korelasyon analizi sonucunda, raporlanan hata sayıları/hata yoğunluğu ve değişiklik bağlaşımı ölçütleri arasında pozitif korelasyon saptanmıştır. Analiz sonuçları, genel olarak düşük ve orta düzeyde korelasyon olduğunu göstermektedir.

2 İlgili Çalışmalar

Değişiklik bağlaşımı kavramı ilk olarak 1997 yılında ortaya atıldı [6]. Bu ilk çalışmada değişiklik bağlaşımı bilgisi kullanılarak beraber değişen sınıflar (classes) görsel olarak sunulmuştur ve sistemin evrimi sırasında sık değiştirilen sınıf kümeleri tespit edilmiştir. Bu çalışma aynı kümeye ait sınıfların semantik olarak ilgili olduğunu göstermiştir. 1998 yılında modül seviyesinde değişiklik bağlaşımının incelendiği başka bir çalışma [1] değişiklik bağlaşımının sistem mimarisi üzerinde yararlı içgörüler elde etmeye yardımcı olduğunu ortaya koymuştur. 2000 yılında yapılan bir çalışmada [4] bir modülün değişiklik tarihçesinin hangi özelliklerinin muhtemel olarak hata sayılarını tahminlemede kullanılacağını değerlendirmek için çeşitli istatistiksel modeller geliştirildi. Çalışmanın sonuçları, değişiklik bağlaşımı ölçütlerinin kullanıldığı modellerin tahminleme performanslarının diğer modellere göre daha düşük olduğunu göstermiştir. 2003 yılında yapılan diğer bir çalışma [2] endüstriyel bir yazılım sistemi üzerinde sınıf seviyesinde değişiklik bağlaşımları incelemiştir. Bu çalışma, değişiklik bağlaşımının kötü tasarlanmış arayüzler ve kalıtım (inheritance) hiyerarşileri gibi mimari zayıflıkların tespit edilmesi için kullanılabileceğini ortaya koyması bakımından önemlidir. 2004 yılındaki bir çalışmada [7], değişiklik görevi atanan yazılım geliştiricilere ilgili olabilecek kaynak kodları veri madenciliği teknikleri kullanarak öneren bir yaklaşım geliştirilmiştir. Bu yaklaşım açık kaynak kodlu projelere uygulanarak önemli bağımlılıkları (dependencies) ortaya çıkardığı gösterilmiştir. 2005 yılındaki önemli bir çalışma [8] kaynak kodun bir bölümü (dosya, sınıf, özellik, metod seviyelerinde) değiştiğinde değişmesi olası diğer bölümleri tahminleyen bir teknik sunmuştur. Bu çalışma ilişki kural madenciliği kullanılarak değişiklik bağlaşımlarının saptandığı ilk çalışmadır. 2005 yılındaki diğer bir çalışma [9] modüller arasındaki değişiklik bağlaşımlarının Kiviat diyagramlar üzerinde gösterilerek

kod içyapı düzenlemesi (code refactoring) için aday modüllerin tespit edilebileceğini göstermiştir. 2006 yılında değişiklik bağlaşımı bilgisi ve veri madenciliği teknikleri yazılım sistemlerinde loglama ve güvenlik gibi kaynak kodun farklı noktalarına dağılmış işlerin (crosscutting concerns) belirlenmesinde kullanılmıştır [10]. Yine 2006'da yayınlanan başka bir çalışma [11] değişiklik bağlaşımı verisini sürüm kontrol sistemindeki iki dosya arasındaki mesafenin hesaplanmasında kullanmış ve animasyonlu paneller dizisi şeklinde gösterilmesini sağlamıştır. 2006'da Mozilla projesi üzerinde yapılan başka bir çalışma [5] değişiklik bağlaşımı ölçütlerinin hata tahmininde iyi sonuçlar vermediğini ortaya koymuştur. Bu çalışmada değişiklik bağlaşımının hata yoğunluğunu tahminleme yeteneği denenmiştir.

Değişiklik bağlaşımı ile yazılım hataları arasındaki ilişkiyi inceleyen ilk çalışma 2009 yılında yapılmıştır [3]. Üç büyük yazılım sistemi üzerinde yapılan incelemede değişiklik bağlaşımı ile yazılım hataları arasında bir korelasyon tespit edilmiştir. Bu araştırmada sadece proje içindeki sınıflar arasındaki değişiklik bağlaşımları incelenmiştir. 2012 yılındaki başka bir çalışma [12] beraber değiştirilmiş sınıflar üzerinden bağlaşım oluşturmuş sıralı sürüm çizelgelerini (graph) çıkarır. Sonrasında bu çizgeler ile hata tahminleme yapılabileceğini gösterir. 2013 yılında yapılan bir araştırma [13], değişiklik bağlaşımı bilgisini dikkate alarak, mevcut hata yeri belirleme (Bug Localization) yöntemleri için iyileştirme önermiştir. Çalışmada üretilen yöntem iki açık kaynak kod projesine (Eclipse SWT ve Android ZXing) uygulanmış ve doğrulanmıştır. 2013 yılında yayınlanan başka bir çalışma [14] projelerin modülleri arasındaki değişiklik bağlaşımlarının modülleri içindeki değişiklik bağlaşımlarına göre hatalar üzerinde daha etkili olduğunu ortaya koymuştur.

3 Yöntem

3.1 Veri Kaynakları

Kaynak kod depoları, kaynak kodu ve kaynak koda yapılan değişiklikleri saklamak ve yönetmek için kullanılır. Değişikliklerin tarihçesi, değişikliği yapan yazılımcı, değişiklik tarihi, oluşan dosya sürümleri ve hatta ilgili ister veya proje görevinin bilgisi kaynak kod depolarından elde edilebilir. Kaynak kod depolarındaki bu zengin veri, YDM araştırmalarının temelini oluşturmaktadır. Bu çalışmada, kaynak kod deposu yönetmek için kullanılan yazılım, Computer Associates (CA) şirketinin bir ürünü olan CA Software Change Manager (CA SCM) [15] yazılımıydı. CA SCM sürüm kontrolüne ek olarak değişiklik yönetimi (change management) işlevselliğine de sahiptir. Yazılımcıların, CA SCM'de değişiklik yapabilmesi için mutlaka değişiklik paketi (diğer SCM yazılımlarındaki "değişiklik grubu" (change set) kavramına benzer) seçmesi gerekir. Değişiklik paketi, aynı hata düzeltme veya geliştirme görevi kapsamında birlikte yapılan tüm ilgili değişiklikleri içerir ve bir arada tutar.

Birçok şirket bilgi sistemlerinin tüm bileşenleri ile ilgili bilgileri Konfigürasyon Yönetimi Veritabanında (KYVT) depolar. KYVT terimi, BT hizmet yönetimi için en iyi uygulamaları (best practice) tanımlayan Bilgi Teknolojisi Altyapı Kütüphanesinden (Information Technology Infrastructure Library, ITIL)

Şekil 1: İzlenen Yaklaşım

kaynaklanıyor olsa da, benzer sistemler neredeyse tüm BT departmanları tarafından kullanılmaktadır. KYVT şu verileri içerir [16]: bilgisayar sistemleri ve uygulama yazılımları gibi yönetilen kaynaklar; değişiklik kayıtları gibi süreç ögeleri; ve yönetilen kaynaklar ve süreç ögeleri arasındaki ilişkiler. Çalışmamızda, hata raporları ile yazılım bileşenleri ve proje görevleri arasındaki ilişkileri KYVT sisteminden elde ettik. Çalışmada kullanılan KYVT sistemi kurum içi geliştirilmiş bir sistemdir.

Çalışmamızda hata kayıtları hakkında gerekli bilgi şirketin Hata Verisi Deposundan (HVD) elde edilmiştir. Çalışmada kullanılan HVD kurum içi geliştirilmiş bir sistemdir.

3.2 Veri Toplama

Çalışma kapsamında, kaynak kod verileri CA SCM sürüm kontrol sisteminden, hata verileri Hata Verisi Deposundan (HVD), hatalar ile kaynak kodlar arasındaki ilişkiler ise KYVT'dan toplanmıştır. Şekil 1 yaklaşımımızın özetini sunar. Farklı veri kaynaklarından veri toplayabilmek için üç adaptör geliştirdik: KYVT adaptörü, HVD adaptörü ve CA SCM adaptörü. Adaptörler basit bir arayüze sahiptir ve verilen tarih aralığına göre veri kaynağını sorgular. CA SCM adaptörü belirtilen tarih aralığında yapılan kaynak kod değişiklikleri ve kaynak kodun kendisini alabilmek için CA SCM SDK (Yazılım Geliştirme Kiti) kullanır. KYVT ve HVD adaptörleri Java veri tabanı erişim teknolojisi olan JDBC (Java Database Connectivity) [17] kullanarak bu sistemlerin veritabanını sorgular. İlgili tablo ve sütun eşleşmeleri kullanıcı tarafından sağlanmaktadır. Adaptörlerin belirli bir tarih aralığı için veri kaynağından aldığı veriler Yazılım Evrimi Veri Tabanına (YEVT) kaydedilir.

YEVT üzerindeki veri, işleme ve ölçüm için Yazılım Evrimi Kontrol Aracına (YEKA) girdi olarak verilir [18]. YEKA şu modüllerden oluşmaktadır: Ölçüt Mo-

dülü, Veri Analizi Modülü, Veri Erişim Modülü ve Veri Modelleri Modülü. Veri Erişim Modülü, veri tabanına bağlantı, erişim ve verinin okunması/yazılması işlerinden sorumludur ve verileri diğer modüller için kullanılabilir hale getirir. Ölçüt Modülü ölçütlerin hesaplanmasını sağlar. Ölçüt Modülünün bir diğer önemli rolü ise farklı veri kaynaklarından toplanan ilişkili verilerin eşleştirilmesidir. Örneğin, HVD'den alınan bir hata kaydı ile kaynak kod deposundan alınan bir ya da birkaç kaynak kod değişikliği birbiriyle ilişkilendirilmelidir. Bu eşleşme Ölçüt Modülü tarafından gerçekleştirilir. Çalışmamızda istatistiksel analizler için SPSS Statistics [19] yazılımı kullanılmıştır. Veri tabanından alınan veri SPSS'e aktarılarak üzerinde Spearman ve Pearson korelasyonu gibi çeşitli istatistiksel analizler uygulanır. SPSS dışında özellikle YEKA modülleri içerisindeki istatistiksel işlemler için Weka yazılımının [20] Java kütüphanesi kullanılmıştır.

Kaynak kod verileri için, öncelikle kaynak kod deposunda belirtilen dönemde oluşturulan tüm dosya sürümleri adaptörler aracılığıyla alındı. Daha sonra alınan her dosya sürümü üzerinde YEKA ile statik kod analizi yapıldı. Ölçüt modülü ile ölçütler hesaplandı. Kod ölçütleri yanında, ilgili dosya sürümünü oluşturan yazılımcı, oluşturma tarihi ve ilgili hata/istek/proje numarası da kaynak kod deposundan sağlandı.

KYVT üzerinde her yazılım ürünü ayrı bir yapılandırma öğesi (YÖ) (configuration item) olarak tanımlanır ve her değişiklik kaydedilip ilgili YÖ ile ilişkilendirilir. Çalışmamızda belirlenen süre içinde yazılım ürünü ile ilgili tüm değişiklikler (hata düzeltme veya geliştirme) adaptörler aracılığıyla toplandı. İncelenen KYVT üzerinde değişiklik için üç farklı tip tanımlanmıştır: Hata, İstek, Proje. Bu nedenle hata düzeltme ve geliştirme kolayca ayırt edilebildi.

Tablo 1: İncelenen Sistem Hakkında Bilgiler

Finansal Yazılım Sistemi	
Yazılım Bileşeni Sayısı	274
Toplam Yazılımcı Sayısı	460
Toplam Satır Sayısı	2.3 milyon
Toplam Dosya Sayısı	150 bin
Toplam Dosya Sürümü Sayısı	192 bin
Veri Toplanan Zaman Dilimi	2009-2013

Bu çalışma kapsamında bir finansal yazılımın 274 bileşeni için 5 yıla yayılan bakım/evrim verileri toplanmıştır. İlgili yazılım bileşenleri PL/I ve COBOL dilleri ile geliştirilmiş ve toplam 460 yazılımcı görev almıştır. İncelenen bileşenlerin toplam büyüklüğü 2.3 milyon satırdır ve 150 bin dosyadan oluşur. İncelenen sistem ile ilgili detaylı bilgi Tablo 1'de verilmiştir.

3.3 Ölçütler

Değişiklik Bağlaşımı Ölçütleri Değişiklik bağlaşımı, yazılım sisteminin evrimi sırasında sık sık birlikte değiştiği gözlenen iki veya daha fazla yazılım parçası (artifact) arasındaki örtülü ilişkiler olarak tanımlanır. Yazılım parçaları ne kadar fazla birlikte değiştirilirseler aralarındaki değişiklik bağlaşımı bu ölçüde güçlenir. Üzerinde fikir birliğine varılmış değişiklik bağlaşımı ölçütleri bulunmamakla birlikte daha önceki araştırmalarda kullanılmış bazı ölçütler mevcuttur. Çalışmamızda bu bölümde detayları verilen 2 adet değişiklik bağlaşımı ölçütü kullanılmıştır.

Çalışma yapılan şirkette, kaynak koda yapılan her değişiklik istekler (modification request (MR)) ile hayata geçirilir. Bir MR, bir veya daha fazla kaynak kod dosyasının bir veya daha fazla yazılım geliştirici tarafından değiştirilmesini kapsayan kavramsal bir yazılım değişimini temsil eder. Bu değişiklikler hata düzeltme ya da yeni işlevsellik olabilirler.

Değişiklik bağlaşımı ölçütlerinin hesaplanmasındaki yaklaşımımızı şu şekilde ifade ettik. MR değişiklik istekleri kümesini, mr tek bir değişiklik isteğini temsil eder. f, mr kapsamında değiştirilen bir kaynak kod deposu dosyasını temsil eder. Bu tanımlara dayanarak, belirli bir dosya için değişiklik bağlaşımı kümesi şu şekilde hesaplandı:

$$SECF(f) = \{f_i | mr \in MR \wedge f_i \in mr \wedge f_i \neq f\}$$

Belirli bir f dosyası ile değişiklik bağlaşımı olan toplam dosya sayısı:

$$NoECF(f) = |SECF(f)|$$

Belirli bir f dosyasının belirli bir mr kapsamındaki değişiklik bağlaşımı kümesi:

$$SECFMR(f, mr) = \{f_i | f_i \in mr \wedge f_i \neq f\}$$

Belirli bir f dosyası ile bütün ilgili mr'lar kapsamında değişiklik bağlaşımı olan dosya sayılarının kümülatif toplamı:

$$NoECFMR(f) = \sum_{i=0}^{n} |SECF(f, mr_i)|$$

Değişiklik bağlaşımı ölçümü için üç temel husus ele alınmıştır: ölçümün yapılacağı detay seviyesi (granularity), dosyaların gruplanması, ölçümlemenin yapılacağı birimin sınırları. Yazılım hataları çalışma yaptığımız sistemler için dosya seviyesinde eşleştirildiğinden, değişiklik bağlaşımı ölçütlerini de dosya bazında hesaplanmıştır. Dosyaların gruplanması daha önce açıklandığı gibi MR bazında yapılmıştır. Bir dosyanın içinde bulunduğu alt-sistem ile olan bağlaşımı, dosya seviyesindeki ikili değişiklik bağlaşımı ölçütlerinin toplanması ile hesaplanmıştır.

Yazılım Büyüklüğü Ölçüm Ölçütleri Büyüklük ölçümü ve elde edilen ölçütlerin normalizasyonu için LOC ölçütü kullanılmıştır.

- **LOC: Toplam Satır Sayısı (Lines of Code)**

Hata Ölçütleri Hata ölçümlemesi için de aşağıdaki ölçütler tanımlanmıştır:

- **NoF : Hata sayısı (Number of Faults)**
- **DD : Hata Yoğunluğu (Defect Density)**

Hata yoğunluğu ölçütü şu şekilde hesaplanacaktır:

$$DD = NoF/LOC$$

3.4 Deney Tasarımı

Araştırma sorusuna cevap bulabilmek için Pearson ve Spearman korelasyon analizleri kullanılmıştır. Çalışma kapsamında üretilen değişiklik bağlaşımı ölçütlerinin ölçüm değerleri ile hata sayıları korelasyon analizine girdi olarak verilmiştir. Bu analizler sonucunda, değişiklik bağlaşımı ve yazılımın hataları arasındaki ilişki -1 ile 1 arasında bir değer ile ifade edilir. Yüksek korelasyon 1 ve -1'e yakın değerler ile gösterilir. 1 aynı yönde pozitif korelasyonu temsil ederken -1 tersi yönde negatif bir korelasyonu gösterir. 0'a yakın değerler ise hiçbir korelasyon olmadığının göstergesidir. Korelasyon sonuçlarının anlamlı (significant) sayılması için önem (significance) seviyesi 0.05 olarak alınmıştır. Bu değerin altındaki sonuçlar önemli olarak değerlendirilmiştir. Analizler SPSS yazılımı [19] kullanılarak yapılmıştır. SPSS istatistiksel analiz için yaygın olarak kullanılan bir yazılım olduğundan ve yazarların daha önce deneyimleri olmasından dolayı seçilmiştir.

Toplanan değişiklik bağlaşımı (DB) ve hata ölçütleri üzerinde her yazılım bileşeni için ayrı ayrı yapılan korelasyon analizlerinin sonucu elde edilen *rho*, *p* ve *StdErr* değerleri incelenmiştir.

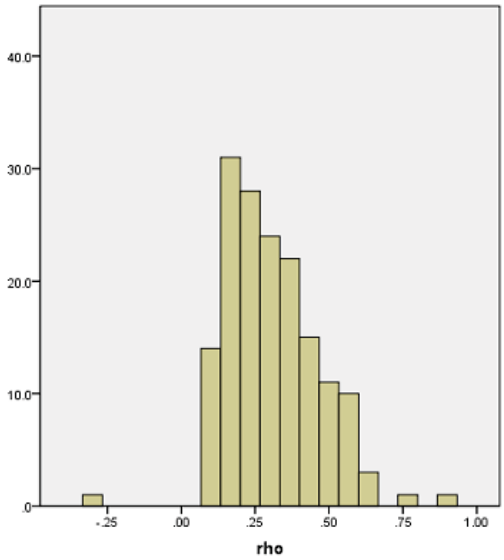
Çalışma kapsamında bulunan korelasyon değerleri daha önceki araştırmalardan da yararlanarak şu şekilde değerlendirildi [21] [22]: 0.1'den daha düşük korelasyon değerleri önemsiz, 0.1 ve 0.3 arasındaki değerler düşük, 0.3 ve 0.5 arasındakiler orta, 0.5 ve 0.7 arasındaki değerler yüksek, 0.7 ile 0.9 arasındakiler çok yüksek, ve 0.9'dan daha büyük değerler mükemmel.

4 Analiz Sonuçları

4.1 Korelasyon Analizi Sonuçları

DB ölçütlerinin hesaplanabildiği 274 yazılım bileşeninin 161'inde DB ölçütleri ile hata sayıları (NoF) arasında Spearman analizi kullanılarak anlamlı (significant) bir korelasyon ($p < 0.05$) tespit edildi. Bu 161 yazılım bileşeninin Spearman *rho* değerlerinin dağılımı Şekil 2'deki histogram üzerinde görülebilir. DB ölçütleri ile hata sayıları arasında genel olarak düşük ve orta düzeyde korelasyon gözlemlenmiştir. Az sayıda bileşen için ise yüksek düzeyde korelasyon tespit edilmiştir.

Korelasyon tespit edilmeyen 113 projenin 46'sında DB ölçütlerinin değerinin 0 ya da düşük değerler (<10) olduğu tespit edildi. Aynı veri üzerinde Pearson



Şekil 2: Değişiklik Bağlaşımı-Hata Sayıları Korelasyonu (Spearman) - Rho Değerleri Histogramı

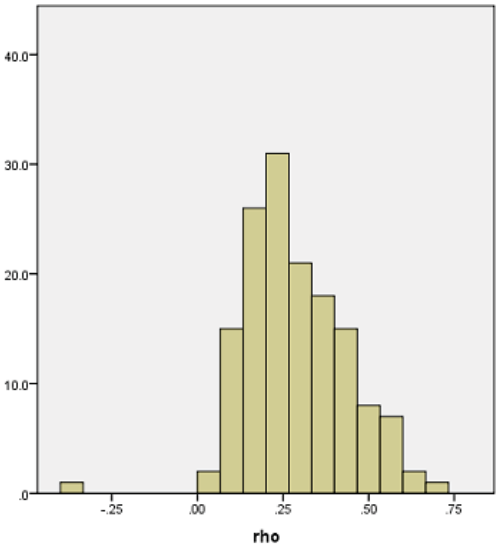
korelasyon analizi de uygulandı. Spearman ile korelasyon tespit edilen 161 projeden 52'sinde anlamlı bir korelasyon tespit edilmedi. Buna karşılık Spearman ile korelasyon tespit edilmeyen 7 projede Pearson ile korelasyon tespit edildi.

DB ölçütleri ile diğer bir hata ölçütü olarak seçtiğimiz hata yoğunluğu (DD) üzerinde Spearman analizi yapıldığında 147 bileşen için anlamlı (significant) bir korelasyon ($p < 0.05$) tespit edildi. Bu bileşenlerin Spearman rho değerlerinin dağılımı Şekil 3'deki histogram üzerinde görülebilir. Daha az sayıda bileşen için korelasyon tespit edilmiş olsa da bu histogram ile Şekil 2'deki histogram arasında büyük benzerlik görülmektedir. Daha önceki sonuçlarla benzer olarak DB ölçütleri ile hata yoğunluğu arasında genel olarak düşük ve orta düzeyde korelasyon gözlemlenmiştir. Az sayıda bileşen için ise yüksek düzeyde korelasyon tespit edilmiştir.

4.2 Önemli Gözlemler

Analiz sonuçları DB ölçütleri ile hata ölçütleri arasında pozitif bir korelasyon olabileceğini göstermiştir. Ama bu sonuçlardan direkt olarak neden-sonuç ilişkisi çıkarılamaz. Değişiklik bağlaşımının yazılım hatalarına neden olduğu sonucuna varmak için daha detaylı analizler yapılması gerekmektedir.

DB ölçütlerini hesaplarken kaynak kod deposundaki revizyonları iki tür gruplama olanağı mevcuttur: MR bazında ve Kaynak Kod Giriş İşlemi (Transaction) bazında. İncelenen sistem için aynı anda kaynak kod girişi yapılan dosya sayısı



Şekil 3: Değişiklik Bağlaşımı-Hata Yoğunluğu Korelasyonu - Rho Değerleri Histogramı

fazla olmadığından daha sağlıklı sonuç olabilmek için MR bazında gruplama yapmak tercih edilmiştir.

Bunun dışında kaynak kod deposu kullanımının düşük olduğu bileşenler için DB ölçütlerinin hesaplanamadığı ya da gerçek bağlaşımları yansıtamadığı gözlemlenmiştir. DB ölçütleri hesaplanırken bu tip kaynak kod deposu projelerinin kapsam dışında bırakılmasını tavsiye ediyoruz.

Çalışmanın başında kaynak kod deposundan CA tarafından sağlanan Java kütüphanesini kullanarak dosya sürüm tarihçesini almaya çalıştık. Fakat ciddi performans sorunları yaşandığından direkt olarak veritabanından istenen verilerin elde edilmesi yönünde çözüm geliştirildi.

5 Geçerliliğe Tehditler

Çalışmada HVD'den alınan hataların kaynak kod deposu üzerindeki dosyalar ile eşleştirilmesi, hatalar ve hatalara bağlı açılan kayıtlara (üretime alma, kod gözden geçirme, teste alma, vb.) iliştirilen KYVT üzerinde tanımlı konfigürasyon öğeleri ile yapılır. Buradaki iki varsayım şu şekildedir:

- İlgili hata kapsamında değiştirilen her dosyaya karşılık gelen bir konfigürasyon öğesi KYVT üzerinde tanımlıdır
- İlgili hata kapsamında değiştirilen her dosyaya karşılık gelen konfigürasyon öğesi hataya iliştirilmiştir

Bu iki varsayımın geçerliliği özellikle bazı kayıt türleri için (üretime alma, kod gözden geçirme) garanti edilebilir fakat her hata için bütün ilişkilerin bulunması garanti edilemez.

Bunun dışında yapılan bütün değişikliklerin kaynak kod deposuna aktarıldığı (commit) varsayılmıştır. Derleme, teste alma ve üretime alma süreçlerine konulan kontrollerle bu garanti altına alınmaya çalışılmıştır. Diğer bir varsayım yazılımcıların bir MR ile ilgili değişen dosyaları aynı değişiklik paketine koyduğudur. Bu varsayım, değişiklik bağlaşımı metrklerinin hesaplanmasından kullanılmıştır.

6 Sonuçlar ve Öneriler

Bu makalede, bir finansal yazılım sistemi üzerinde değişiklik bağlaşımı ve yazılım hataları ilişkisi konusunda yapılan çalışmanın detayları ve sonuçları sunulmuştur. İlgili yazılım sistemini oluşturan 274 adet yazılım bileşeni için ayrı ayrı ve değişiklik bağlaşımı ve hata ölçütleri (hata sayısı ve hata yoğunluğu) kullanılarak korelasyon analizi yapılmıştır. Çalışmada Yazılım Depoları Madenciliği (YDM) teknikleri kullanılmıştır. Çalışmamızın sonuçları, incelenen yazılımın bakım/evrimi fazında değişiklik bağlaşımının yazılım hataları ile ilişkili olabileceğini göstermiştir. Değişiklik bağlaşımı ölçütleri ve hata ölçütleri arasındaki korelasyon analizi sonucunda, raporlanan hata sayıları/hata yoğunluğu ve değişiklik bağlaşımı ölçütleri arasında pozitif korelasyon saptanmıştır. Analiz sonuçları, genel olarak düşük ve orta düzeyde korelasyon olduğunu göstermektedir.

Gelecek çalışmalarımız için, değişiklik bağlaşımı ölçütlerini kullanarak hata tahminleme modelleri oluşturmayı ve bu modellerin mevcut modellere göre performanslarını incelemeyi planlıyoruz. Ayrıca, Git, Subversion gibi diğer kaynak depolarını kullanabilmek için YEKA'da iyileştirmeler planlamaktayız.

Kaynaklar

1. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Software Maintenance, 1998. Proceedings., International Conference on, IEEE (1998) 190–198
2. Gall, H., Jazayeri, M., Krajewski, J.: Cvs release history data for detecting logical couplings. In: Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of, IEEE (2003) 13–23
3. D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. In: Reverse Engineering, 2009. WCRE'09. 16th Working Conference on, IEEE (2009) 135–144
4. Graves, T.L., Karr, A.F., Marron, J.S., Siy, H.: Predicting fault incidence using software change history. Software Engineering, IEEE Transactions on **26**(7) (2000) 653–661
5. Knab, P., Pinzger, M., Bernstein, A.: Predicting defect densities in source code files with decision tree learners. In: Proceedings of the 2006 international workshop on Mining software repositories, ACM (2006) 119–125
6. Ball, T., Kim, J.M., Porter, A.A., Siy, H.P.: If your version control system could talk. In: ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering. (1997)

7. Ying, A.T., Murphy, G.C., Ng, R., Chu-Carroll, M.C.: Predicting source code changes by mining change history. Software Engineering, IEEE Transactions on **30**(9) (2004) 574–586

8. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: Mining version histories to guide software changes. Software Engineering, IEEE Transactions on **31**(6) (2005) 429–445

9. Pinzger, M., Gall, H., Fischer, M., Lanza, M.: Visualizing multiple evolution metrics. In: Proceedings of the 2005 ACM symposium on Software visualization, ACM (2005) 67–75

10. Breu, S., Zimmermann, T.: Mining aspects from version history. In: Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on, IEEE (2006) 221–230

11. Beyer, D., Hassan, A.E.: Animated visualization of software history using evolution storyboards. In: Reverse Engineering, 2006. WCRE'06. 13th Working Conference on, IEEE (2006) 199–210

12. Steff, M., Russo, B.: Co-evolution of logical couplings and commits for defect estimation. In: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on, IEEE (2012) 213–216

13. Tantithamthavorn, C., Ihara, A., Matsumoto, K.I.: Using co-change histories to improve bug localization performance. In: Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2013 14th ACIS International Conference on, IEEE (2013) 543–548

14. Kouroshfar, E.: Studying the effect of co-change dispersion on software quality. In: Software Engineering (ICSE), 2013 35th International Conference on, IEEE (2013) 1450–1452

15. CASCM: Web page of ca software change manager (2013)

16. Carlisle, F., Eisinger, J., Johnson, M., Kowalski, V., Mukerji, J., Snelling, D., Vambenepe, W., Waschke, M., Wiles, V.: Configuration management database (cmdb) federation specification (2010)

17. JDBC: Java database connectivity (2013)

18. Kirbas, S., Sen, A.: Yazılım depoları madenciliği ile endüstriyel yazılım evrimi İncelemesi. In: UYMS. (2013)

19. SPSS: Web page of spss (2013)

20. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter **11**(1) (2009) 10–18

21. Hopkins, W.G.: A New View of Statistics. Will G. Hopkins (1997)

22. Lu, H., Zhou, Y., Xu, B., Leung, H., Chen, L.: The ability of object-oriented metrics to predict change-proneness: a meta-analysis. Empirical Software Engineering **17** (2012) 200–242

EK: Kullanılan Kısaltmalar

KISALTMALAR	
BT	Bilgi Teknolojisi
CA	Computer Associates
CA SCM	Computer Associates Software Change Manager
DB	Değişiklik Bağlaşımı (Change Coupling / Evolutionary Coupling)
DD	Defect Density (Hata Yoğunluğu)
HVD	Hata Verisi Deposu
ITIL	Information Technology Infrastructure Library (BT Altyapı Kütüphanesi)
JDBC	Java Database Connectivity
KYVT	Konfigürasyon Yönetimi Veri Tabanı
LOC	Lines of Code (Toplam Satır Sayısı)
MR	Modification Request (Değişiklik İsteği)
NoF	Number of Faults, Hata Sayısı
SDK	Yazılım Geliştirme Kiti
YDM	Yazılım Depoları Madenciliği
YEVT	Yazılım Evrimi Veri Tabanı
YEKA	Yazılım Evrimi Kontrol Aracı
YÖ	Yapılandırma Öğesi (Configuration Item)