Yazılım Depoları Madenciliği ile Endüstriyel Yazılım Evrimi İncelemesi

Serkan Kırbaş, Alper Şen

Bilgisayar Mühendisliği Bölümü
Boğaziçi Üniversitesi
İstanbul, Türkiye
serkan.kirbas@boun.edu.tr
alper.sen@boun.edu.tr

Özet Yazılım evrimi alanında, açık kaynak yazılım geliştirmeden daha zengin veri içeren endüstriyel yazılımlar üzerinde daha fazla ampirik araştırma yapılmasına ihtiyaç vardır. Bu çalışmada, Yazılım Depoları Madenciliği (YDM) teknikleri kullanılarak endüstriyel bir yazılımın evrimi analiz edilmiştir. Veri kaynağı olarak kaynak kod depoları, hata depoları ve iletişim arşivlerini kullanmaya odaklanan mevcut çalışmaların aksine, bu araştırma Kurumsal Kaynak Planlama, Konfigürasyon Yönetimi Veritabanı ve İnsan Kaynakları (İK) sistemlerini kullanır. Bu sistemler, yazılımcı deneyimi, yazılım bakımı için harcanan süre, efor tipi gibi veriler içerir. Bu verileri bu çalışma kapsamında incelenen yazılımın kaynak kod deposundan alınan ve her bir değişikliğin detaylarını içeren verilerle birleştirdik. Daha sonra bu birleştirilen veriler üzerinde kontrol grafikleri, Spearman korelasyon ve ANOVA testleri gibi çeşitli istatistiksel analizler uyguladık. Çalışmamızın sonuçları, incelenen yazılımın bakım/evrim fazında yazılımcı deneyimi ve fazla mesainin verimlilik üzerinde etkileri olduğunu gösterdi. Buna karşın kaynak kodun karmaşıklığı ve bağımlılığı (coupling) üzerinde herhangi bir önemli (significant) etkisi gözlenmedi. Ayrıca, süreç ve ürün metrikleri arasında yapılan korelasyon analizi bazı önemli yazılım eğilimlerini ortaya çıkardı.

Anahtar Kelimeler. Yazılım Depoları Madenciliği, Yazılım Evrimi, Endüstriyel Yazılım, Örnek Olay İncelemesi, 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. Ancak, Yazılım Depoları Madenciliği (YDM) alanındaki çalışmalar, ana veri kaynağı olarak kaynak kod depoları, hata depoları ve iletişim arşivlerini kullanmaya

odaklanmıştır [1] [2] [3] [4]. Farklı yazılım depolarında saklanan verilerin çekilmesi ve birleştirilmesi yeni bilgi ve içgörülerin ortaya çıkarılmasını sağlayabilir. Çalışmamızda, endüstriyel yazılım geliştirme ortamında bulunan farklı yazılım depolarından faydalanmayı hedefledik.

Bu çalışmada, endüstriyel bir yazılım üzerinde ampirik bir araştırma gerçekleştirildi. Bakım fazındaki bir finansal yazılım bileşeninin son 2 yıllık verileri analiz edildi. Yazılımın nasıl evrildiği konusunda detaylı ve doğru bilgi edinebilmek için YDM teknikleri kullanıldı. Bu çalışma kapsamında kullanılan yazılım depoları şunlardır: Konfigürasyon Yönetimi Veritabanı (KYVT), Kurumsal Kaynak Planlaması (KKP), İnsan Kaynakları (İK) sistemi ve kaynak kod deposudur. Bu yaklaşım, ana veri kaynağı olarak daha önce sözü edilen yazılım depolarını kullanmaya odaklanmış mevcut YDM araştırmalarından farklıdır.

YDM teknikleri ile farklı yazılım depolarından toplanan verileri birleştirerek, çalışmamızda şu sorulara cevap vermeye çalışacağız:

S1 Yazılım evriminde hangi ürün ve süreç metrikleri arasında korelasyon vardır?
S2 Yazılımın bakımını yapan yazılımcıların deneyiminin verimlilik üzerinde etkisi var mıdır?
S3 Fazla mesainin verimlilik üzerinde etkisi var mıdır?
S4 Yazılımın bakımını yapan yazılımcıların deneyiminin kaynak kod karmaşıklığı ve bağımlılığı (coupling) üzerinde etkisi var mıdır?
S5 Fazla mesainin kaynak kod karmaşıklığı ve bağımlılığı (coupling) üzerinde etkisi var mıdır?

Çalışmamızın sonuçları, incelenen yazılımın bakım/evrimi fazında yazılımcı deneyimi ve fazla mesainin verimlilik üzerinde etkileri olduğunu gösterdi. Buna karşın kaynak kodun karmaşıklığı ve bağımlılığı üzerinde herhangi bir önemli (significant) etkisi gözlenmedi. Ayrıca, süreç ve ürün metrikleri arasındaki korelasyon analizi sonucunda, raporlanan hata sayıları ve hata düzeltme, destek faaliyetleri için harcanan efor için artış eğilimi saptandı. Sonuçlar, aynı zamanda kaynak kodun bazı parçalarının yeniden düzenlenmesi (refactoring) ihtiyacını ortaya çıkarmıştır.

2 İlgili Çalışmalar

Manny Lehman'ın araştırmaları ve ortaya koyduğu sekiz yasa yazılım evrimi araştırmalarının temelini oluşturmaktadır. Lehman'ın ampirik çalışmaları yazılım değişikliklerine ve yazılım evriminin doğasına odaklanmıştır. Yazılım sistem evrimini analiz etmek için oluşturulan yaklaşımlar kullanılan veri ve veri kaynaklarına göre değişiklik gösterirler [3]. Açık kaynak yazılım geliştirmenin yaygınlaşması ve bu açık kaynak projelerin kullandığı yazılım depolarının erişilebilirliği sayesinde YDM alanı popülerlik kazanmıştır. Birçok YDM araştırması kaynak kod için CVS, hata raporları için Bugzilla sistemlerini kullanmıştır [3]. Bunların dışında SVN [5], ClearCase [6], Git ve Mercurial [7] [8] [9] [10] gibi farklı kaynak kod depolarından veri kullanan başka çalışmalar da vardır. Bazı çalışmalarda [11] [12] [13] iletişim arşivleri de veri kaynağı olarak kullanılmıştır.

Literatürde endüstriyel yazılımlar üzerinde yapılmış çeşitli vaka incelemeleri de vardır. Gall [14] [15] Java ile yazılmış endüstriyel yazılımlar üzerinde CVS and Bugzilla kullanarak vaka çalışmaları yapmıştır. Ayrıca Gall [16] C dilinde yazılmış bir telekomünikasyon sisteminde vaka çalışması da yapmıştır. Gegick [17] güvenlik hata raporlarının belirlenmesi için Bugzilla hata deposunu kullanarak endüstriyel vaka çalışması gerçekleştirmiştir. Bieman [18] C++ ile yazılmış gerçek-zamanlı endüstriyel bir yazılım sistemi üzerinde vaka çalışması bildirmiştir.

Literatürde KYVT [19] [20], İK sistemleri [21] [22], ve KKP sistemleri [23] [24] [25] [26] üzerinde yapılmış veri madenciliği çalışmaları da mevcuttur. Ancak bu çalışmalarda elde edilen veriler kaynak kod depolarındaki versiyon geçmişi verileri ile birleştirilip analiz edilmemiştir.

Çalışmamızda, versiyon geçmişi, hata/istek raporları, yazılımcı efor ve deneyim verileri sırasıyla kaynak kod deposu, VYVT, KKP ve İK veri kaynakları kullanılarak elde edilmiştir. Çalışmamızda, yazılım evrimi konusunda tanımlanan iki açık noktaya [27] katkı yapılması hedeflenmiştir:

- Endüstriyel yazılımlar hakkında daha fazla deneysel araştırma ihtiyacı, ve
- Daha çeşitli yazılım depolarından veri kulanımı ve entegrasyonu

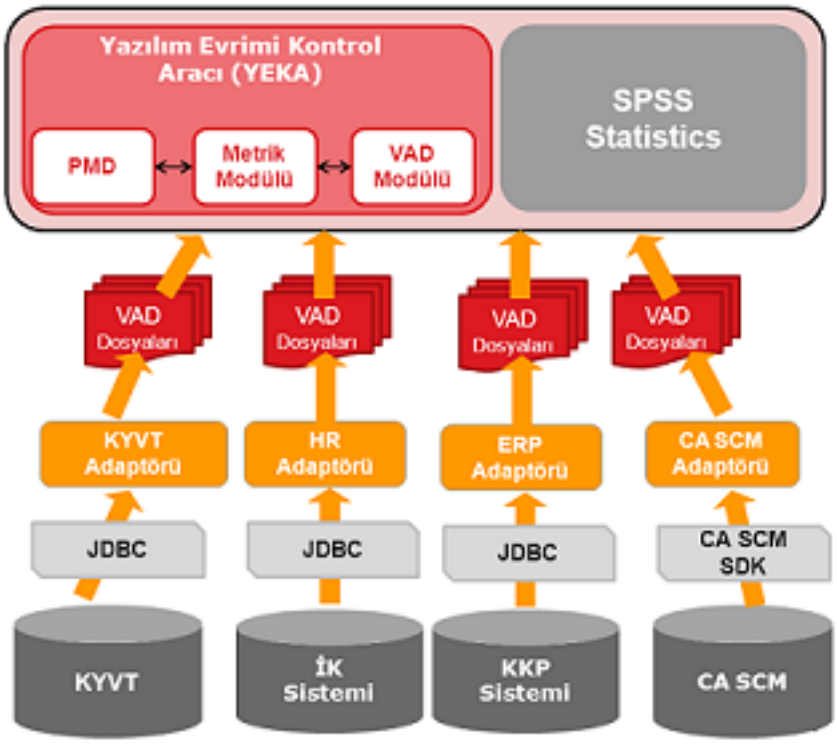
Çalışmamızda geliştirilen adaptörler, SPSS betikleri (scripts) ve Yazılım Evrimi Kontrol Aracı (YEKA) prototipi sayesinde yapılan analizler tek bir kereye mahsus olmamakta ve sürekliliği sağlanmaktadır. Analiz için sürdürülebilir bir çözüm sağlaması çalışmamızın diğer bir katkısıdır.

3 Yaklaşım, Metrikler ve Hipotezler

3.1 Yaklaşım

Şekil 1 yaklaşımımızın özetini sunar. Farklı veri kaynaklarından veri toplayabilmek için dört adaptör geliştirdik: KYVT adaptörü, KKP adaptörü, İK 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, İK ve KKP adaptörleri Java veri erişim teknolojisi olan JDBC (Java Database Connectivity) [28] kullanarak bu sistemlerin veritabanını sorgular. İlgili tablo ve sütun eşleşmeleri kullanıcı tarafından sağlanmaktadır. Adaptörlerin çıktıları belirli bir tarih aralığı için veri kaynağından alınan verileri içeren Virgülle Ayrılmış Değerler (VAD) (Comma-Separated Values) dosyalarıdır.

VAD dosyaları veri işleme ve ölçüm için Yazılım Evrimi Kontrol Aracına (YEKA) girdi olarak verilir. YEKA üç modülden oluşmaktadır: VAD Modülü, Metrik Modülü ve PMD [29]. VAD Modülü, VAD dosyalarını çözümler ve verileri diğer modüller için kullanılabilir hale getirir. Metrik Modülü metriklerin hesaplanmasını sağlar. Metrik Modülü, Java kaynak kodu ölçümlemesi için PMD aracının değiştirilmiş bir sürümünü kullanır. PMD, Soyut Sözdizimi Ağacı (Abstract Syntax Tree) temsilini kullanan popüler bir Java statik kaynak kod analiz



Şekil 1: İzlenen Yaklaşım

aracıdır. PMD ile analiz için kaynak kodun derlenmesine gerek olmadığından çok esnek ve verimli bir çözüm sağlar. Metrik Modülünün bir diğer önemli rolü ise farklı veri kaynaklarından toplanan ilişkili verilerin eşleştirilmesidir. Örneğin, KKP sisteminden alınan bir efor kaydı ile kaynak kod deposundan alınan bir ya da birkaç kaynak kod değişikliği birbiriyle ilişkilendirilmelidir. Bu eşleşme Metrik Modülü tarafından gerçekleştirilir. Çalışmamızda istatistiksel analizler için SPSS Statistics [30] aracı kullanılmıştır. Oluşturulan VAD dosyaları SPSS'e aktarılarak üzerinde kontrol grafikleri, Spearman korelasyonu, ANOVA testleri, Pareto analizi gibi çeşitli istatistiksel analizler uygulanır.

3.2 Metrikler

Bu çalışmada kullanılan metrikler üç kategoriye ayrıldı: ürün, süreç ve kaynak metrikleri. Aşağıdaki bölümlerde, ilk olarak taban metrikler daha sonra da bu taban metrikler kullanılarak oluşturulan türetilmiş (derived) metrikler anlatılmaktadır.

Ürün Metrikleri Kaynak kod evrimini izlemek amacıyla, literatürde nesne yönelimli programlama için tanımlanmış aşağıdaki metrikler taban metrik olarak kullanılmıştır:

- **CC : Karmaşıklık (Cyclomatic Complexity) [31]**
- **CBO : Objeler arası Bağımlılık (Coupling between Objects) [32]**
- **NOA : Nitelik Sayısı (Number of Attributes) [33]**
- **NOO : Metot Sayısı (Number of Operations) [33] [32]**
- **NCSS : Yorum satırları dışındaki Komut Sayıları (Non Commenting Source Statements) [34]**

Boyut ölçümü ve elde edilen metriklerin normalizasyonu için NCSS metriği kullanılmıştır. Ayrıca kontrol grafikleri üzerinde yazılım evrim verisindeki kontrol-dışı noktaların tespiti için NCSS kullanılmıştır. Diğer metrikler korelasyon analizi sonuçlarına göre seçilmiştir. Metot seviyesinde hesaplanabilen bir metrik olan CC metriğini sınıf seviyesinde kullanmak için, bir sınıfın tüm metotları için ayrı ayrı ölçülen CC değerlerinin ortalaması alınmıştır.

Sınıf seviyesindeki metrikler ile korelasyonlarını kontrol etmek için, iki metot seviyesi metriği de çalışmaya dahil edilmiştir: CCMetotMax ve NCSSMetotMax. Bu iki metrik ile sınıfların boyut ve karmaşıklık açısından en büyük metotlarının evrim eğilimlerini tespit etmeyi hedefledik:

- **CCMetotMax : Bir sınıfın tüm metotları içindeki maksimum CC değeri**
- **NCSSMetotMax : Bir sınıfın tüm metotları içindeki maksimum NCSS değeri**

Sadece değişikliklerin etkilerini ölçmek amacıyla, tüm kaynak kod metrikleri için Δ (Delta) metrikleri tanımlanmış ve analizler bu delta değerleri üzerinde gerçekleştirilmiştir.

Bu yaklaşımımızı şu şekilde formüle ettik. w_i yazılımın bakım fazındaki i nolu haftayı temsil eder, t_i i nolu haftanın başlangıç tarihini gösterir, c_{ij} i nolu haftada değiştirilen j nolu Java sınıfını gösterir, m w_i haftasında değiştirilen tüm sınıfların sayısını temsil eder. $CC(t_i, c_{ij})$, c_{ij} sınıfının t_i tarihindeki karmaşıklığını göstermektedir. Bu tanımlara dayanarak, belirli bir hafta w_i ve o hafta içerisinde değiştirilen sınıfların listesi 0'dan m'e kadar kullanılarak o hafta için Delta CC metriği şu şekilde hesaplanır:

$$\Delta CC(w_i) = \sum_{j=0}^{m} (CC(t_i, c_{ij}) - CC(t_{i-1}, c_{ij}))$$

Benzer şekilde, diğer metrikler için Delta tanımları yapılır.

Süreç Metrikleri Kaynak kod metriklerine ek olarak, yazılım bakım/evrim süreçleri için de metrikler tanımlanmıştır:

- **NoF : Hata sayısı (Number of Faults)**

- **NoER : Geliştirme İstekleri (Enhancement Requests) sayısı**
- **CorrEf: Hata Düzeltme (Corrective Maintenance) için harcanan Efor (adam-saat)**
- **EnhcEf: Geliştirme İstekleri (Enhancements) için harcanan Efor (adam-saat)**

NoF ve NOER ölçümleri KYVT'den elde edilmiştir. Efor metrikleri (CorrEf ve EnhcEf) KKP sisteminden elde edilir. Efor metrikleri için iki kategori tanımlamıştır: Fazla mesai ve normal mesai. Fazla mesai, günlük 8 saati geçen çalışmaları ve hafta sonu çalışmalarını kapsamaktadır. Verimlilik ölçümü için eforlar toplanırken, en az bir fazla mesai girişi içeren efor toplamlarını fazla mesai olarak sınıflandırıldık. Bunun dışındaki efor toplamları normal olarak sınıflandırıldı. Bu veriler, 3 ve 5 nolu araştırma sorularını cevaplandırmak için kullanıldı.

Ürün ve süreç taban metriklerini kullanarak, araştırma sorularını yanıtlamak için aşağıdaki türetilmiş metrikler tanımlanmıştır:

- **BirimCCArtışı : Birim karmaşıklık artışı**
- **BirimCBOArtışı : Birim bağımlılık artışı**
- **Verimlilik: Efor başına düşen kaynak kod büyüklüğü değişimi**

Verimlilik ölçümü için üç temel husus ele alınmıştır: kaynak kod ölçümleri ile efor ölçümlerinin eşleştirilmesi, verimlilik hesaplamasına dahil edilecek efor kategorileri, verimlilik ölçümünün yapılacağı periyot (haftalık, günlük, vb.). Değişiklik tarihi (hafta numarası), yazılımcı ve proje bilgilerini kullanarak girilen efor ile kaynak kod deposuna aktarılan (check-in) değişiklikleri eşleştirdik. Verimlilik hesaplamasına dahil edilecek efor kategorisi olarak kodlama seçildi. Testler ve yönetim için harcanan efor verimlilik hesaplamasında kullanılmadı. Verimlilik ölçümünün hafta bazında yapılmasına karar verdik, böylece kaynak kodun eforun harcandığı gün kaynak kod deposuna aktarılmadığı durumların büyük ölçüde üstesinden geldik.

Türetilmiş metriklerin hesaplanması için aşağıdaki denklemler kullanılmıştır:

$$BirimCCArtisi(w_i) = \Delta CC(w_i)/\Delta NCSS(w_i)$$

$$BirimCBOArtisi(w_i) = \Delta CBO(w_i)/\Delta NCSS(w_i)$$

$$Verimlilik(w_i) = \Delta NCSS(w_i)/\Delta EnhcEf(w_i)$$

Yazılımcı deneyiminin ve fazla mesainin türetilmiş metrikler üzerindeki etkisini bulmak için, yazılımcı ve fazla mesai özelliklerine göre gruplandırılma yapılmıştır. Bu ölçümler, 2., 3., 4. ve 5. araştırma sorularını cevaplandırmak için kullanılmıştır.

Kaynak Metrikleri Yazılımcı deneyimi için aşağıdaki metrik tanımlanmıştır:

- **YazılımcıDeneyimi : Yıl olarak Çalışma Deneyimi**

Bu metrik için yapılan ölçümler İK sisteminden elde edilmiş, 2. ve 4. araştırma sorularını cevaplamak için kullanılmıştır.

Daha önce yapılan çalışmalara dayanarak yazılımcı deneyimi için üç kategori tanımladık [35] [36]: deneyimsiz, orta derece deneyimli ve deneyimli. Üç yıldan daha az toplam deneyime sahip bütün yazılımcılar deneyimsiz olarak sınıflandırıldı. Üç yıldan on yıla kadar toplam deneyime sahip yazılımcılar orta derece deneyimli, on yıldan daha uzun çalışma deneyimine sahip olanlar ise deneyimli olarak sınıflandırıldı.

3.3 Sıfır Hipotezleri

2 ile 5 arasındaki soruları yanıtlamak için sırasıyla aşağıdaki sıfır hipotezleri tanımlanmıştır:

- Hipotez 1: Yazılımcı deneyiminin verimlilik üzerinde etkisi yoktur. (H1)
- Hipotez 2: Fazla mesainin verimlilik üzerinde etkisi yoktur. (H2)
- Hipotez 3: Yazılımcı deneyiminin kaynak kod karmaşıklığı ve bağımlılığı (coupling) üzerinde etkisi yoktur. (H3)
- Hipotez 4: Fazla mesai kaynak kod karmaşıklığı ve bağımlılığı (coupling) üzerinde etkisi yoktur. (H4)

Sıfır hipotezleri ANOVA testleri kullanılarak sınanmıştır.

4 Yöntem

4.1 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 versiyonlar 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 araç, Computer Associates (CA) şirketinin bir ürünü olan CA Software Change Manager (CA SCM) [37] aracıydı. CA SCM versiyon kontrolüne ek olarak değişim yönetimi (change management) işlevselliğine de sahiptir. Yazılımcıların, CA SCM'de değişiklik yapabilmesi için mutlaka değişiklik paketi (diğer SCM araçlarındaki "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. CA SCM üzerinde daha önce yapılmış herhangi bir YDM araştırması bulunmamaktadır.

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) kaynaklanıyor olsa da, benzer sistemler neredeyse tüm BT departmanları tarafından

kullanılmaktadır. KYVT şu verileri içerir [38]: bilgisayar sistemleri ve uygulama yazılımları gibi yönetilen kaynaklar; istek, hata ve değişiklik kayıtları gibi süreç ögeleri; ve yönetilen kaynaklar ve süreç ögeleri arasındaki ilişkiler. Çalışmamızda, hata raporları ve bu hata raporlarının yazılım bileşenleri ve proje görevleri ile ilişkisi hakkındaki verileri KYVT sisteminden elde ettik. Çalışmamızda kullanılan KYVT, hata ve istek kayıtları hakkında gerekli bilgileri depoladığından, ayrıca hata deposundan veri çekmek durumda kalmadık. Çalışmada kullanılan KYVT sistemi kurum içi geliştirilmiş bir sistemdir.

Kurumsal Kaynak Planlama (KKP) sistemleri finans/muhasebe, üretim, satış, servis ve müşteri ilişkileri yönetimini kapsayan, kurumların iç ve dış yönetim entegrasyonunu sağlayan bütünleşik yönetim sistemleridir. Çalışmamızda, yazılımcı efor verisi KKP sisteminden elde edilmiştir. Çalışmada kullanılan KKP sistemi kurum içi geliştirilmiş bir sistemdir.

İnsan Kaynakları (İK) sistemleri veya İnsan Kaynakları Yönetim Sistemleri (İKYS), çalışan bilgilerinin bir araya toplanması ve birleştirilmesi için özel olarak tasarlanmış sistemleri ifade eder. İK sistemleri bordro yönetiminden performans değerlendirmeye kadar birçok işlevi sağlayabilir. Çalışmamızda yazılımcıların deneyim bilgileri her çalışan hakkında en doğru verileri içeren İK sisteminden elde edilmiştir. İK sistemi de kurum içi geliştirilmiş bir sistemdir.

4.2 Veri Toplama

Çalışma kapsamında, kaynak kod verileri CA SCM versiyon kontrol sisteminden, hata/geliştirme verileri KYVT'dan, yazılımcı efor verileri KKP sisteminden ve yazılımcı deneyimi İK sisteminden toplanmıştır.

Kaynak kod verileri için, öncelikle kaynak kod deposunda belirtilen dönemde oluşturulan tüm versiyonlar adaptörler aracılığıyla çekildi. Daha sonra çekilen her sürüm üzerinde YEKA ile statik kod analizi yapıldı. PMD tarafından metot ve sınıf seviyesindeki metrikler hesaplandı. Kod metrikleri yanında, ilgili versiyonu 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.

İncelenen KKP sistemine her yazılımcı sorun, istek ya da proje için her gün harcanan eforu saat olarak girer. İzin verilen minimum değer 0.5 saattir. Örneğin bir problem çözme 0.5 saatten daha az sürerse, 0.5'e yuvarlanır. Bu çalışma kapsamında tek bir giriş için azami efor değeri 10 saat olarak gözlendi. Girilen her efor değeri kullanıcı tarafından sağlanan hata, istek ya da bir proje görev numarası (id) ile ilişkilendirilir. Böylece efor verisi her yazılımcı ve her gün için hata/geliştirme taskı seviyesinde toplanabilmiştir. Her efor girişi için aynı zamanda fazla mesai olup olmadığı belirtilmiştir. Detay seviyesi yüksek efor verileri sayesinde çalışmamızda detaylı analiz ve inceleme yapabildik.

Yazılımcı deneyimi verileri İK sisteminden çalışma kapsamında geliştirilen adaptörler aracılığıyla elde edildi. Yazılımcı deneyimi, çalışanın çalışmış ve çalışıyor olduğu şirket veya kurumlardaki iş deneyimleri toplanarak yıl olarak hesaplanır. Çalışma kapsamında kaynak kod deposu analizinde tespit edilen her yazılımcı için deneyim verisi İK sisteminden temin edildi.

4.3 Veri Analizi

Süreç sapmalarını tespit etmek ve kontrol-dışı noktalardan gelen verileri ayıklamak için kontrol grafikleri kullandık. Kontrol grafikleri istatistiksel süreç kontrol (İSK) araçlarından biridir [39]. İSK'ya göre iki ayrı tip varyasyon vardır: Yaygın neden varyasyonu (common cause variation) ve özel neden varyasyonu (special cause variation). Çalışmamıza sadece ilgili kaynak kod değişikliklerini dahil etmek amacıyla yazılım bakım/evrim sürecindeki özel neden varyasyonları tespit edildi ve bu noktalar analizden muaf tutuldu.

Metrikler arasındaki korelasyonun tespiti için, Spearman korelasyon analizi kullandık. Çalışma kapsamında bulunan korelasyon değerleri daha önceki araştırmalardan da yararlanarak şu şekilde değerlendirildi [40] [41]: 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.

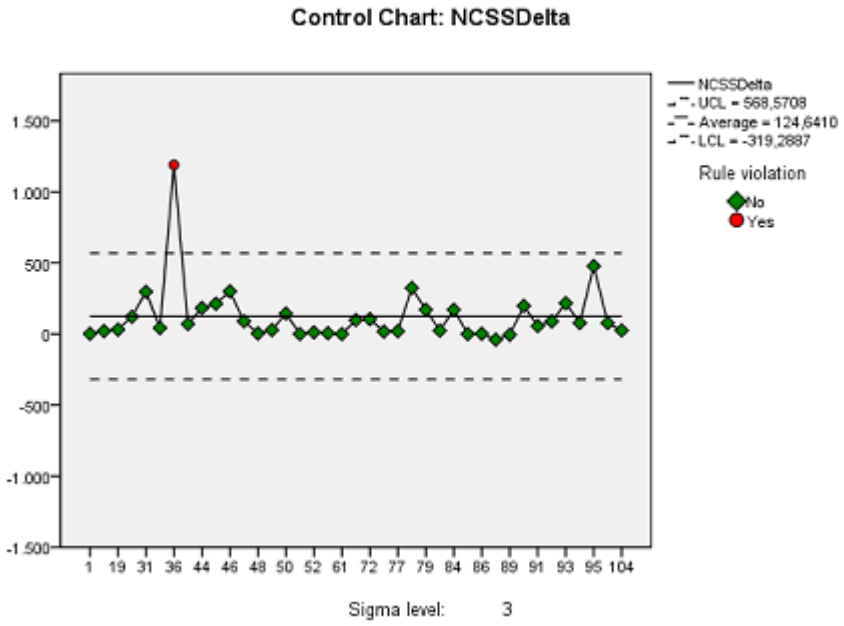
Yazılımcı deneyimi ve mesai tipinin (fazla mesai/normal) etkisini ölçmek için, varyans analizi (ANOVA) uygulanmıştır (alfa düzeyi 0.05). ANOVA deneysel araştırmalarda grupların ölçülen özellik bakımından birbirinden istatistiksel olarak farklı olup olmadığını kontrol etmek için yaygın olarak kullanılır. Ancak, ANOVA hangi grupların özellikle hangilerinden farklı olduğu bilgisini sağlamaz. Bu amaçla çalışmamızda Tukey HSD ve Games-Howell testlerini post-hoc testler olarak kullandık.

ANOVA analizinde grupların eşit varyanslara sahip olduğu varsayılır. Çalışmamızda bu varsayımı kontrol etmek için Levene testi kullanılmıştır. Bu varsayımın karşılanmadığı durumlarda, çalışmamızda Welch ve Brown-Forsythe testleri kullanılmıştır. Ayrıca bu gibi durumlarda post-hoc test olarak Tukey HSD yerine eşit olmayan varyanslar için tasarlanmış Games-Howell testi kullanılmıştır.

Bu çalışma kapsamında yapılan istatistiksel analizler için SPSS [30] aracı kullanılmıştır.

5 Örnek Olay İncelemesi

Bu çalışmada analizi yapılan finansal yazılım bileşeninin 2 yıla yayılan bakım/evrim verileri incelenmiştir. İlgili bileşenin yazılım geliştirmesi 10 ayda tamamlanmış ve bu süre sonunda ilk sürümü kullanıma alınmıştır. Sunulan çalışma bu noktadan sonra başlayan bakım/evrim fazının ilk 2 yılını kapsar. Yazılım bileşeni Java ile geliştirilmiş ve 6 yazılımcı bakımında görev almıştır.



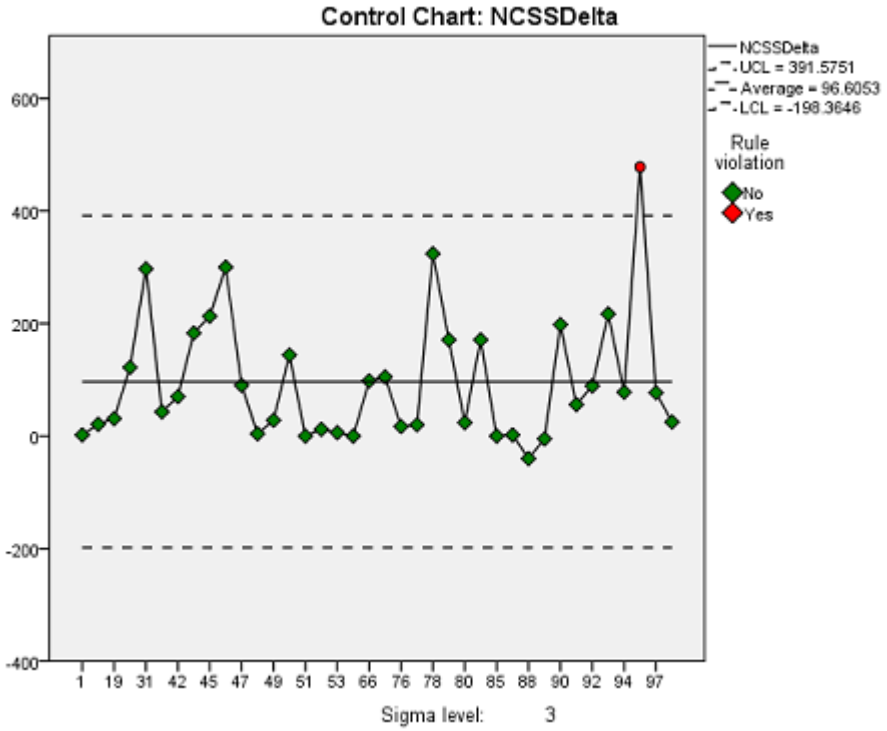
Şekil 2: Kontrol Grafiği - Kod Büyüklüğü Değişiminin Evrimi

5.1 Veri Doğrulama

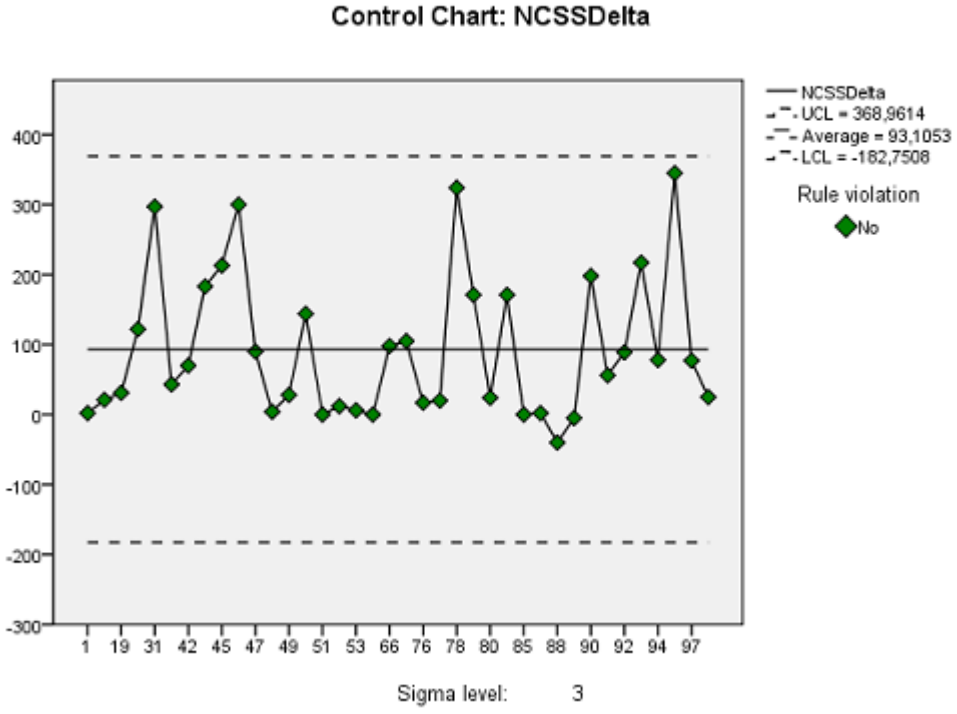
İlk olarak yazılımın kaynak kod büyüklüğünün değişimi kontrol edildi. Amacımız sadece ilgili değişikliklerin analiz için kullanılmasını sağlamaktı. Kaynak kod deposundan ölçümlenen değişikliklerin kod büyüklüğü (Delta-NCSS) için kontrol grafiği çizdik (Şekil 2).

Çizilen kontrol grafiğinde 36. haftanın ΔNCSS değerinin ÜKS (Üst Kontrol Sınırı) üzerinde olduğunu gördük. Bu nedenle bu nokta kontrol-dışı nokta olarak değerlendirildi. 36. haftada kaynak kod deposuna aktarılan dosyalar detaylıca incelendiğinde, şirketteki başka bir bileşenin kaynak kodunun uygulamanın kaynak kod deposuna kopyalanmış olduğu tespit edildi. Böylece aynı şirketteki projeler arasında kaynak kod tekrarını (duplication) ortaya çıkarmış olduk. Ilgili kaynak kod çalışılan yazılımın bakım projeleri kapsamında geliştirilmediğinden ilgili metrik veri noktasının analizden çıkarılmasına karar verildi.

Kontrol-dışı veri noktası çıkarıldıktan sonra, kontrol grafiği yeniden çizildi (Şekil 3). Bu sefer, hesaplanan yeni kontrol limitlerine göre 95. hafta kontrol-dışı nokta olarak tespit edildi. Detaylı incelemede, yeni oluşturulan bir Java sınıfının projenin varolan bir Java sınıfından kopyalandığı ortaya çıktı. Bu durum efor ve kaynak kod metrikleri arasındaki ilişkiyi analiz ederken hataya neden olabileceğinden, ilgili Java sınıfı analizden çıkarıldı.



Şekil 3: Kontrol Grafiği - İlk Kontrol-Dışı Nokta Uzaklaştırıldıktan sonra



Şekil 4: Kontrol Grafiği - İkinci Kontrol-Dışı Nokta Uzaklaştırıldıktan sonra

Tablo 1: Spearman Korelasyon Analizi Sonuçları

	HaftaNo	CorrEf	EnhcEf	NoF	ΔCC	ΔCCMtdMx	ΔNCSS	ΔNOA	ΔNOO	
CorrEf	ρ=.492**									
	p=.000									
EnhcEf	ρ=.012	ρ=.124								
	p=.898	p=.174								
NoF	ρ=.418**	ρ=.568**	ρ=.058							
	p=.000	p=.000	p=.524							
ΔCC	ρ=.042	ρ=.083	ρ=.394**	ρ=-.079						
	p=.646	p=.363	p=.000	p=.386						
ΔCCMtdMx	ρ=.069	ρ=.152	ρ=.389**	ρ=-.008	ρ=.928**					
	p=.450	p=.095	p=.000	p=.934	p=.000					
ΔNCSS	ρ=.004	ρ=.097	ρ=.489**	ρ=-.081	ρ=.817**	ρ=.858**				
	p=.968	p=.287	p=.000	p=.373	p=.000	p=.000				
ΔNOA	ρ=.070	ρ=.076	ρ=.401**	ρ=-.030	ρ=.733**	ρ=.739**	ρ=.741**			
	p=.446	p=.406	p=.000	p=.747	p=.000	p=.000	p=.000			
ΔNOO	ρ=.111	ρ=.102	ρ=.325**	ρ=-.048	ρ=.838**	ρ=.843**	ρ=.708**	ρ=.762**		
	p=.225	p=.263	p=.000	p=.599	p=.000	p=.000	p=.000	p=.000		
ΔCBO	ρ=.039	ρ=.062	ρ=.384**	ρ=-.094	ρ=.899**	ρ=.887**	ρ=.810**	ρ=.794**	ρ=.905**	
	p=.673	p=.496	p=.000	p=.301	p=.000	p=.000	p=.000	p=.000	p=.000	

Bu kontrol-dışı nokta da çıkardıktan sonra çizilen kontrol grafiğinde (Şekil 4) tüm değerler kontrol limitleri arasında görülmüştür. Bu çalışmada sunulan diğer analizler bu iki kontrol-dışı noktanın kaldırılması sonucu oluşan kaynak kod ölçümlerini kulllanmışlardır.

5.2 Korelasyon Analizi Sonuçları

1 numaralı araştırma sorusunu yanıtlamak için bu çalışmada dikkate alınan metrikler arasında yapılan Spearman korelasyon analizinin sonuçları Tablo 1'de gösterilmiştir. ΔCC, ΔNCSS, ΔNOA, ΔNOO, ve ΔCBO gibi tüm kaynak kod metrikleri ile geliştirme istekleri için harcanan efor (EnhcEf) arasında orta düzeyde korelasyon gözlemlenmiştir. Öte yandan hata düzeltme eforu (CorrEf) ve delta kaynak kod metrikleri arasında herhangi bir korelasyon tespit edilmemiştir. Bu gözlemler, kaynak kod üzerindeki değişikliklerin esas olarak geliştirme istekleri (enhancements) kapsamında yapıldığını göstermektedir.

Hafta numarası ve hata düzeltme eforu (CorrEf) arasında tespit edilen orta-derece korelasyon ilginç bir gözlem olarak değerlendirilebilir. Hafta numarası ve

Tablo 2: ANOVA Sonuçları - Yazılımcı Deneyimi

	Karelerin-Toplamı	Serbestlik-Derecesi	Karelerin-Ortalaması	F-değeri	P-değeri
Verimlilik	322.670	2	161.335	13.235**	0.0**
				7.883*	.003*
BirimCCArtışı	.036	2	.018	.296	.745
BirimCBOArtışı	.005	2	.003	2.529	.093

Tablo 3: Verimlilik - Yazılımcı Deneyimi için Games-Howell Sonuçları

	Deneyimsiz	Orta-Deneyimli
Deneyimsiz		
Orta-Deneyimli	.967	
Deneyimli	.002*	.002*

hata sayısı (NoF) arasında da orta-derece korelasyon tespit edildi. Bu hata düzeltme ve destek faaliyetleri için harcanan haftalık efor ve tespit edilen haftalık hata sayıları için artan bir eğilim ortaya koymaktadır. Ayrıca, ilginç olmamakla beraber hata düzeltme eforu (CorrEf) ile hata sayısı (NoF) arasında yüksek korelasyon tespit edilmesi analizi yapılan verilerin doğrulamasına katkı sağlamaktadır.

ΔCC - ΔCCMetotMax ve ΔNCSS - ΔNCSSMetotMax metrikleri arasında oldukça yüksek korelasyon (sırasıyla p değerleri: 0,873 ve 0,928) gözlemlendi. Bu projedeki sınıfların büyüklüğünün, özellikle sınıfın en büyük boyutlu metodunun boyutunu artırarak arttığını ortaya koymaktadır. Bu ilgili sınıfların metotlarının yeniden düzenlenmesi (refactoring) için bir ihtiyaç olarak yorumlanabilir. Fowler'ın önerdiği yeniden düzenleme (refactoring) tekniklerinin [42] ikisi burada uygulanabilir: Metot Ayıklama (Extract Method) ve Sınıf Ayıklama (Extract Class). Bu şekilde büyük boyutlu metotlar mevcut veya yeni sınıflar içinde daha küçük metotlara ayrılır ve böylece kaynak kod daha anlaşılır ve bakımı-kolay (maintainable) hale getirilir.

5.3 Hipotezlerin Testi ve Tartışma

S2: Yazılımın bakımını yapan yazılımcıların deneyiminin verimlilik üzerinde etkisi var mıdır? 1 numaralı Hipotezin ANOVA testinin sonuçları Tablo 2'de sunulmaktadır. Levene testi grup varyanslarının eşit olmadığını gösterir ($p = 0.004 < 0.05$). Bu nedenle Welch ve Brown-Forsythe testlerinde (tabloda sırasıyla * ve ** ile işaretlenmiş) elde edilen sonuçlar kontrol edilir. Her iki test de yazılımcı deneyiminin verimlilik üzerinde önemli (significant) etkisi (p değerleri 0.003 ve 0.0) olduğunu göstermektedir. Bu nedenle, ilgili sıfır hipotezi H1 reddedilir. Tablo 3'de gruplar arasındaki farkların ayrıntılarını gösteren (önemli değerler * ile işaretlenmiştir) Games-Howell testinin sonuçları verilmiştir. Testlere göre, deneyimli yazılımcıların verimlilik değerleri deneyim-

Tablo 4: Mesai Tipi için ANOVA Sonuçları

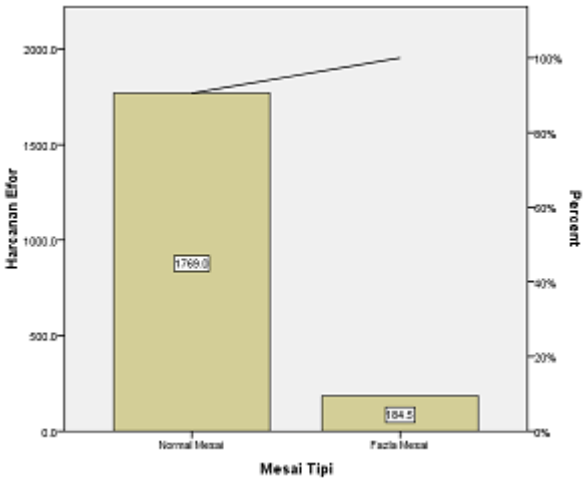
	Karelerin-Toplamı	Serbestlik-Derecesi	Karelerin-Ortalaması	F değeri	P-değeri
Verimlilik	143.034	1	143.034	4.846**	.040**
				4.846*	.040*
BirimCCArtışı	.010	1	.010	1.301	.262
BirimCBOArtışı	.000	1	.000	.060	.807

siz ve orta-deneyimli yazılımcılarınkinden önemli derecede (significant) farklıdır. Ancak, deneyimsiz ve orta-deneyimli yazılımcıların verimlilik değerleri arasında önemli bir fark tespit edilmemiştir. Bu, verilen görevlerin zorluk seviyesi, teknoloji ve uygulama deneyimi gibi farklı nedenlere bağlanabilir. Farklı faktörlerin verimlilik üzerindeki etkisi gelecekteki çalışmalarımızda ayrıca incelenecektir.

S3: Fazla mesainin verimlilik üzerinde etkisi var mıdır? 2 numaralı Hipotez için ANOVA testi sonuçları Tablo 4'de sunulmuştur. Levene testi ($p = 0.003 < 0.05$) grup varyanslarının eşit olmadığını gösterir. Bu nedenle Welch ve Brown-Forsythe testlerinde elde edilen sonuçlar kontrol edilir (sırasıyla * ve ** ile işaretlenmiş). Her iki test de fazla mesainin verimlilik üzerinde önemli bir etkisi ($p = 0.040 < 0.05$) olduğunu göstermektedir. Bu nedenle ilgili hipotez H2 reddedildi. Genel kanının aksine, analiz sonuçları fazla mesai grubu için verimliliğin diğer gruba göre daha yüksek olduğunu ortaya koymuştur. Bu durum için bir açıklama; yazılımcıların fazla mesai sırasında normaleden daha az kesintiye (interrupt) uğraması olabilir. Yazılımcılarla yaptığımız görüşmelerde normal çalışma saatleri içinde gün boyunca genellikle e-posta, gelen telefon çağrıları ve diğer acil istekler ile geliştirme işlerinin kesintiye uğradığını dile getirdiler. Bazı çalışmalar [43] [44] kesintiye uğramış bir göreve devam etmenin toparlanma süresi (recovery time) diye adlandırılan fazladan bir süreye malolduğunu ortaya koymuştur. Kesintilerin verimlilik üzerindeki etkileri gelecekteki çalışmalarımızda daha detaylı olarak incelenecektir.

Bu sonucu yorumlarken, fazla mesai eforunun genel efora oranı da (bu uygulama için %10) dikkate alınmalıdır (Şekil 5). Daha yüksek fazla mesai oranlarına sahip uygulamların sonuçları ile karşılaştırmak ilginç olacaktır. Sonuçlar üzerinde etkisi olabilecek bir diğer parametre de fazla mesai ücreti olabilir. Analiz yapılan şirkette yazılımcılara fazla mesai ücreti ödeniyordu. Fazla mesai ödenen yazılımcıların ödenmeyenlere göre daha motive çalışmaları beklenir. Bu konuyu da gelecekteki çalışmalarımızda araştırmayı planlıyoruz.

S4: Yazılımın bakımını yapan yazılımcıların deneyiminin kaynak kod karmaşıklığı ve bağımlılığı (coupling) üzerinde etkisi var mıdır? Tablo 2 Hipotez 3 için ANOVA sonuçlarını göstermektedir. Sonuçlar farklı yazılımcı deneyimi grupları arasında BirimCCArtışı ve BirimCBOArtışı metrikleri için



Şekil 5: Mesai Tipi Pareto Diyagramı

önemli (significant) bir fark göstermemektedir (p değerleri .745 ve .093). Bu durumda, sıfır hipotezi H3 reddedilemez.

S5: Fazla mesainin kaynak kod karmaşıklığı ve bağımlılığı (coupling) üzerinde etkisi var mıdır? Tablo 4 Hipotez 4 için ANOVA sonuçlarını göstermektedir. Sonuçlar fazla mesai ve normal mesai arasında BirimCCArtışı ve BirimCBOArtışı metrikleri için önemli (significant) bir fark göstermemektedir (p değerleri sırasıyla .262 ve .807). Bu nedenle, sıfır hipotezi H4 reddedilemez.

6 Sonuçlar ve Öneriler

Bu makalede, bir finansal yazılım bileşeninin evrimi için örnek olay incelemesi sunuldu. Çalışmada YDM teknikleri kullanılmıştır. Kaynak kod deposu, VYVT, KKP ve İK sistemleri veri kaynakları olarak kullanılarak versiyon geçmişi, hata raporları/geliştirme istekleri, yazılımcı deneyimi, efor ve mesai tipi verileri analiz edilmiştir. Bu yaklaşım, temel veri kaynağı olarak kaynak kod, hata depoları ve iletişim arşivlerine odaklanmış YDM kullanan mevcut yazılım evrimi araştırmalarından farklıdır. Bu çalışmada, çeşitli yazılım depolarından elde edilen veriler birleştirilip üzerinde kontrol grafikleri, Spearman korelasyon ve ANOVA testleri gibi istatistiksel analizler uygulanmıştır. Bu analizler sonucunda, süreç ve ürün metrikleri arasındaki ilişkiler, korelasyonlar, süreçteki sapmalar ve eğilimler gibi yazılım evrimi için değerli bilgiler ve içgörüler elde edilmiştir.

Çalışmamızda, projedeki sınıfların büyüklüğünün, özellikle sınıfın en büyük boyutlu metodunun boyutunun arttırılarak arttığı ortaya çıkmıştır. Bu ilgili sınıfların metotlarının yeniden düzenlenmesi (refactoring) için bir ihtiyaç olarak

yorumlanmıştır. Ayrıca, aynı şirkette faklı projeler arasında kaynak kod tekrarı (duplication) yapıldığı ortaya çıktı. Çalışmamızın sonuçları, incelenen yazılımın bakım/evrimi fazında yazılımcı deneyimi ve fazla mesainin verimlilik üzerinde etkileri olduğunu gösterdi. Buna karşın kaynak kodun karmaşıklığı ve bağımlılığı üzerinde herhangi bir önemli (significant) etkisi gözlenmedi. Ayrıca, süreç ve ürün metrikleri arasındaki korelasyon analizi sonucunda, raporlanan hata sayıları ve hata düzeltme, destek faaliyetleri için harcanan efor için artış eğilimi saptandı.

Gelecek çalışmalarımız için, atanan görevlerin zorluk seviyesi, ilgili teknoloji ya da uygulamadaki deneyim gibi verimlilik üzerinde etkisi olabilecek farklı faktörleri incelemeyi planlıyoruz. Ayrıca, Git, Mercurial, Subversion gibi diğer kaynak depoları için adaptörler geliştirmeyi ve YEKA geliştirmesini tamamlamayı planlamaktayız.

Kaynaklar

1. Hemmati, H., Nadi, S., Baysal, O., Kononenko, O., Wang, W., Holmes, R., Godfrey, M.W.: The msr cookbook: Mining a decade of research. In: Proceedings of the 10th Working Conference on Mining Software Repositories. MSR '13 (2013)
2. Hassan, A.: The road ahead for mining software repositories. In: Frontiers of Software Maintenance, 2008. FoSM 2008. (2008) 48–57
3. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maint. Evol. **19**(2) (March 2007) 77–131
4. D'Ambros, M., Gall, H., Lanza, M., Pinzger, M.: Analysing software repositories to understand software evolution. In: Software Evolution. Springer Berlin Heidelberg (2008) 37–67
5. Fischer, M., Pinzger, M., Gall, H.: Populating a release history database from version control and bug tracking systems. In: Proceedings of the International Conference on Software Maintenance. ICSM '03 (2003) 23–
6. Bevan, J., Whitehead, Jr., E.J., Kim, S., Godfrey, M.: Facilitating software evolution research with kenyon. SIGSOFT Softw. Eng. Notes **30**(5) (September 2005) 177–186
7. Sadowski, C., Lewis, C., Lin, Z., Zhu, X., Whitehead, Jr., E.J.: An empirical analysis of the fixcache algorithm. In: Proceedings of the 8th Working Conference on Mining Software Repositories. MSR '11 (2011) 219–222
8. McIntosh, S., Adams, B., Nguyen, T.H., Kamei, Y., Hassan, A.E.: An empirical study of build maintenance effort. In: Proceedings of the 33rd International Conference on Software Engineering. ICSE '11 (2011) 141–150
9. Khomh, F., Dhaliwal, T., Zou, Y., Adams, B.: Do faster releases improve software quality? an empirical case study of mozilla firefox. In: Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on. (2012) 179–188
10. Bird, C., Rigby, P., Barr, E., Hamilton, D., German, D., Devanbu, P.: The promises and perils of mining git. In: Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on. (2009) 1–10
11. German, D.M.: Mining CVS repositories, the softchange experience. In: Proceedings of the First International Workshop on Mining Software Repositories, Edinburg, Scotland, UK (2004) 17–21

12. Čubranić, D., Murphy, G.C.: Hipikat: recommending pertinent software development artifacts. In: Proceedings of the 25th International Conference on Software Engineering. ICSE '03 (2003) 408–418
13. ČubraniĆ, D., Murphy, G.C., Singer, J., Booth, K.S.: Learning from project history: a case study for software development. In: Proceedings of the 2004 ACM conference on Computer supported cooperative work. CSCW '04 (2004) 82–91
14. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Proceedings of the International Conference on Software Maintenance. ICSM '98 (1998) 190–
15. Gall, H., Jazayeri, M., Krajewski, J.: Cvs release history data for detecting logical couplings. In: Proceedings of the 6th International Workshop on Principles of Software Evolution. IWPSE '03 (2003) 13–
16. Gall, H., Jazayeri, M., Riva, C.: Visualizing Software Release Histories: The Use of Color and Third Dimension. In: ICSM, IEEE (1999) 99–108
17. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: An industrial case study. In Whitehead, J., Zimmermann, T., eds.: MSR. (2010) 11–20
18. Bieman, J.M., Andrews, A.A., Yang, H.J.: Understanding change-proneness in oo software through visualization. In: Proceedings of the 11th IEEE International Workshop on Program Comprehension. IWPC '03 (2003) 44–
19. Nadi, S., Holt, R., Mankovskii, S.: Does the past say it all? using history to predict change sets in a cmdb. In: Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on. (2010) 97–106
20. Anchuri, P., Zaki, M., Barkol, O., Bergman, R., Felder, Y., Golan, S., Sityon, A.: Infrastructure pattern discovery in configuration management databases via large sparse graph mining. In: Data Mining (ICDM), 2011 IEEE 11th International Conference on. (2011) 11–20
21. Ranjan, J., Goyal, D.P., Ahson, S.I.: Data mining techniques for better decisions in human resource management systems. Int. J. Bus. Inf. Syst. **3**(5) (June 2008) 464–481
22. Poon, S.K., Davis, J.G., Choi, B.: Augmenting productivity analysis with data mining: An application on it business value. Expert Syst. Appl. **36**(2) (March 2009) 2213–2224
23. Nemati, H., Barko, C.: Organizational data mining. In: Data Mining and Knowledge Discovery Handbook. Springer US (2005) 1057–1067
24. Symeonidis, A.L., Kehagias, D.D., Mitkas, P.A.: Intelligent policy recommendations on enterprise resource planning by the use of agent technology and data mining techniques. Expert Systems With Applications **25**(4) (2003) 589–602
25. Xu, L., Wang, C., Luo, X., Shi, Z.: Integrating knowledge management and erp in enterprise information systems. Systems Research and Behavioral Science **23**(2) (2006) 147–156
26. Zhang, H., Liang, Y.: A knowledge warehouse system for enterprise resource planning systems. Systems Research and Behavioral Science **23**(2) (2006) 169–176
27. Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., Jazayeri, M.: Challenges in software evolution. In: Proceedings of the Eighth International Workshop on Principles of Software Evolution. IWPSE '05 (2005) 13–22
28. JDBC: Java database connectivity (2013)
29. PMD: Web page of pmd tool (2013)
30. SPSS: Web page of spss (2013)
31. McCabe, T.J.: A complexity measure. IEEE Trans. Softw. Eng. **2**(4) (July 1976) 308–320

32. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6) (June 1994) 476–493
33. Lorenz, M., Kidd, J.: Object-Oriented Software Metrics. Prentice Hall (1994)
34. NCSS: Specification for the ncss and javancss-tool (2013)
35. Jørgensen, M., Sjøberg, D.I.K.: Impact of experience on maintenance skills. Journal of Software Maintenance **14**(2) (March 2002) 123–146
36. Jorgensen, M.: Experience with the accuracy of software maintenance task effort prediction models. Software Engineering, IEEE Transactions on **21**(8) (1995) 674–681
37. CASCM: Web page of ca software change manager (2013)
38. 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)
39. Burr, A., Owen, M.: Statistical methods for software quality: using metrics to control process and product quality. International Thomson Computer Press (1996)
40. WG, H.: A new view of statistics. SportScience (2003)
41. 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
42. Beck, K.: Refactoring: improving the design of existing code. The Addison-Wesley Object Technology Series. Addison-Wesley (1999)
43. Parnin, C., Rugaber, S.: Resumption strategies for interrupted programming tasks. Software Quality Control **19**(1) (March 2011) 5–34
44. Van Solingen, R., Berghout, E., van Latum, F.: Interrupts: just a minute never is. Software, IEEE **15**(5) (1998) 97–103