Tanımları Türkçe Olarak Yazılmış Yazılım Hatalarının Otomatik Sınıflandırılması

Ömer Köksal

ASELSAN A.Ş.
koksal@aselsan.com.tr

Özet. Yazılım projelerinin başarılı bir şekilde tamamlanabilmesi için hataların, yazılım testlerinde hızlı bir şekilde bulunması kadar bulunan hataların sınıflandırılıp incelenmesi de önemlidir.
Literatürde, hata sınıflandırma için önerilen pek çok yöntem ve hata şemaları bulunmakla birlikte sanayide standart olarak kullanılan bir sınıflandırma yöntemi bulunmamaktadır. Hata sınıflandırma genellikle elle yapılmaktadır. Elle yapılan hata sınıflandırma süreci yazılım mühendisinin ve/veya yazılım testçinin tercihine bağlıdır, zaman alıcıdır ve standartlaştırılması zordur.
Bu makalede, Türkçe olarak yazılmış yazılım hatalarının, daha önceden belirlenen hata şemalarına göre otomatik olarak sınıflandırılmasını sağlayan yeni bir yöntem sunulmaktadır. Bu yöntem, Türkçe olarak yazılmış yazılım hatalarını sınıflandırma özelliği ile – bilindiği kadarı ile – bir ilktir. Ayrıca, yöntemin uygulanması için geliştirilen yazılım ile otomatik olarak yapılan sınıflandırma, elle sınıflandırmaya göre çaba ve zaman tasarrufu sağlamaktadır. Önerilen yöntemde, yazılım hata sınıflandırma işleminin kişisel tercihlerden bağımsız hale getirilmesi mümkün kılınmıştır. Sınıflandırma amacına göre farklı hata şemalarının kullanılabilmesi, yöntemin farklı yazılım proje tiplerine uyarlanabilmesini sağlamıştır.

Anahtar Kelimeler. Otomatik Hata Sınıflandırma, Sınıflandırma Algoritması, Yazılım Geliştirme, Yazılım Test, Yazı Sınıflandırma, Yazılım Süreçleri

1 Giriş

Yazılım projelerinin boyutu ve karmaşıklığı her geçen gün artmaktadır. Daha büyük ekiplerle gerçekleştirilen büyük çaplı ve karmaşık yazılım projelerinde yazılım hatalarından kaçınmak imkânsızdır. Bu projelerin başarılı bir şekilde bitirilmesi için yazılım hatalarının hızlı bir şekilde bulunması ve düzeltilmesi gereklidir. Hataların hızlı bir şekilde bulunmasının yanı sıra sınıflandırılması da önemli bir konudur. Hataların sınıflandırılarak analiz edilmesi, bu hataların neden yapıldığının anlaşılmasına ve gerekli tedbirlerin alınmasına yardımcı olacaktır. Ayrıca, hataların sınıflandırılıp analiz edilmesi sonraki projelerde bu hataların tekrar edilmemesi açısından önemlidir.

Hata takip sistemleri, bulunan hataların detaylarının tutulması ve bakım aktivitelerinin verimli bir şekilde yapılabilmesi açısından önemlidir [1]. Pek çok yazılım geliştirme yönteminde, yazılım testçileri entegrasyon aşamasına dahil edilirler. Testçiler

hata bulduklarında hata açıklamalarını hata takip yazılımlarına girerler. Yazılım mühendisleri bu açıklamaları hataları düzeltmek için kullanırlar. Hata tanımlarının ve açıklamalarının doğru ve detaylı bir şekilde verilmesi hataların çabuk çözülebilmesi için çok önemlidir [2–4]. Hata takip sistemlerine girilen bu tanımlar ve açıklamalar hataların sınıflandırılması için de kullanılabilir [8].

Hata sınıflandırmaları değişik amaçlar için yapılabilir. Hata sınıflandırılması yazılım mühendislerine, testçilere ve yöneticilere önemli bilgiler verir. Ayrıca şirketlerdeki yazılım süreçlerini geliştirmek için kullanılacak çeşitli yazılım metrikleri hata sınıflandırma sonucu elde edilebilir.

Hata sınıflandırmaları bazen elle yapılmaktadır. Bu çaba ve zaman gerektiren bir süreçtir. Elle sınıflandırma yazılım testçilerin ve/veya yazılım mühendislerinin kişisel tercihlerine bağlı olacağı için bu süreçleri standartlaştırmak zordur.

2 İlgili Çalışmalar

Literatürde yazılım hatalarının sınıflandırılması için önerilen pek çok yöntem vardır. Yazılım hataları üzerinde farklı analizler yapabilmek için farklı yöntemler önerilmektedir. Proje ve hata sınıflandırma amacına göre kullanılan hata şemaları da farklılık göstermektedir. Hata önceliği, hata temel sebebi, hata önem derecesi gibi metrikler için de farklı hata sınıflandırma yöntemleri sunulmuştur.

En eski çalışmalardan biri Beizer [9] ve arkadaşları tarafından gerçekleştirilmiştir. Bu sınıflandırma sürecinde geniş kapsamlı yazılım hata şemaları kullanılmıştır. Beizer bu hata sınıflandırma şemalarının amacını açıkça belirtmemektedir. Beizer'e göre hata sınıflandırmayı etkileyen çeşitli etkenler vardır ve süreç programcının önerilen sınıfları kullanmasına dayanmaktadır. Önerilen 10 ana kategori altında 100'den fazla hata sınıfı bulunmaktadır. Bu sınıfların anlamsal ilişkilerinin açık bir şekilde belirtilmemesi hata sınıflarının anlaşılmasında ve kullanılmasında karışıklığa sebep olmaktadır.

Gray [10] makalesinde hata sınıflarının şeması yerine hataların sebeplerini vermiştir.

Yazılım hatalarının sayısının azaltılması için Cillarege ve arkadaşları [11] "Birbirine Dik Hata Sınıflandırması" kavramını önermiştir. Yazılım mühendislerinden geribildirim almayı amaçlayan bu yöntem IBM'de 50'den fazla projede kullanılmıştır.

Plosky ve arkadaşları [5] literatür araştırması yaparak yazılım hataları üzerine sayısal veriler elde etmişlerdir. Yazılım hatalarının dağılımının projelere özgü faktörlere sıkı bir şekilde bağlı olduğunu söyleyen çalışma yazılım hata sınıflandırma çabalarının yakın bir gelecekte bitmeyecek gibi göründüğünü belirtir.

Seamon ve arkadaşlarının çalışması [6] geçmiş projelerdeki verilerin gelecek projeler için temel alınması esasına dayanır. Yazılım hataları için 3 ana inceleme alanı önerilmiştir:

1. Gereksinimle ilgili hata tipleri,
2. Yazılım ve kaynak kodla ilgili hata tipleri ve
3. Test ile ilgili hata tipleri

Yazılım ve kaynak kodla ilgili hata tipleri ana başlığı altında 10 adet hata tipi Seamon'ın çalışmasında belirtilmiştir:

1. Algoritma / metot (algorithm / method)
2. Atama / ilklendirme (assignment / initialization)
3. Kontrol (checking)
4. Veri (data)
5. Harici arayüz (external interface)
6. Dâhili arayüz (internal interface)
7. Mantık (logic)
8. Fonksiyonel olmayan (non-functional)
9. Zamanlama / optimizasyon (timing / optimization)
10. Diğer (other)

Seamon ve arkadaşlarının çalışmasının kolay anlaşılır ve uygulanabilir olduğu değerlendirilmiştir.

3 Yöntem

3.1 Yazılım Hata Sınıfları

Önceki bölümlerde belirtildiği gibi, yazılım hata sınıfları için genel olarak kabul görmüş bir hata şeması bulunmamaktadır. Yazılım hata tipleri proje tipine sıkı bir şekilde bağlı olduğu için [7] yazılım mühendisleri ve/veya testçiler analiz amacına uygun yazılım hata şemalarını seçmelidirler.

Bu makalede, Bölüm-2'de verilen Seamon ve arkadaşlarının önerdiği 10 adet hata tipi kullanılmıştır. Ancak, sonraki kısımlarda açıklandığı gibi, proje ihtiyaçlarına ve analiz amacına göre farklı hata şemaları da geliştirilen yazılım ile kolay bir şekilde kullanılabilmektedir.

3.2 Örnek Çalışma: VATOZ® Deniz Görev Yönetim Yazılımı

Bu makalede, ASELSAN tarafından geliştirilmiş olan VATOZ® yazılımının Yeni Tip Karakol Botu (YTKB) sürümü örnek çalışma olarak kullanılmıştır (VATOZ®-YTKB). YTKB Projesi kapsamında geliştirilmiş olan VATOZ®-YTKB Deniz Görev Yönetim Yazılımının teslimatı yapılmış olup Deniz Kuvvetleri envanterindeki Yeni Tip Karakol Botlarında kullanılmaktadır.

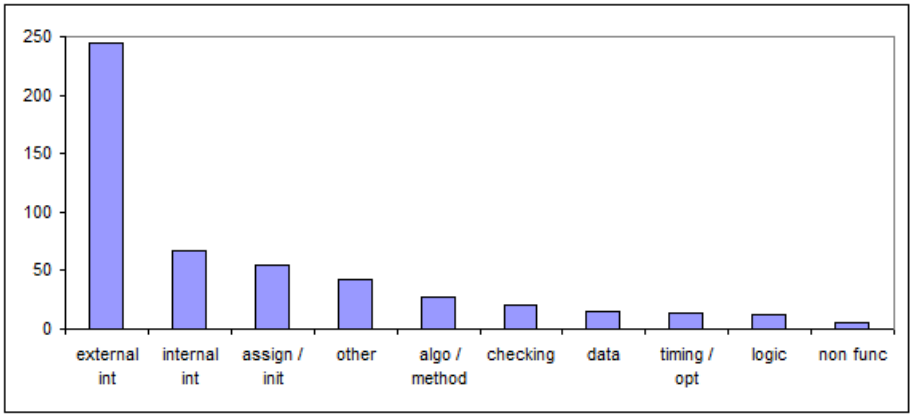
VATOZ®-YTKB 4 senelik proje süresince 10 kişilik bir proje ekibi ile geliştirilmiştir. Kaynak kod büyüklüğü yaklaşık 175000 satırdır. Yazılım testleri ayrı bir test ekibi tarafından gerçekleştirilmiş ve ilk yazılım entegrasyon testleri sonucu 504 hata bulunmuştur. Yazılım test hataları ve açıklamaları TRAC hata takip aracına Türkçe olarak girilmiştir. İlk entegrasyon testleri sonucunda bulunan 504 hata sonraki

bölümlerde anlatıldığı gibi önce elle sonra otomatik olarak sınıflandırılarak analiz edilmiştir. Hataların çözümlenmesinin ardından gerçekleştirilen 2. Yazılım entegrasyon testlerinde tüm hataların çözüldüğü doğrulanmıştır.

3.3 Elle Hata Sınıflandırma

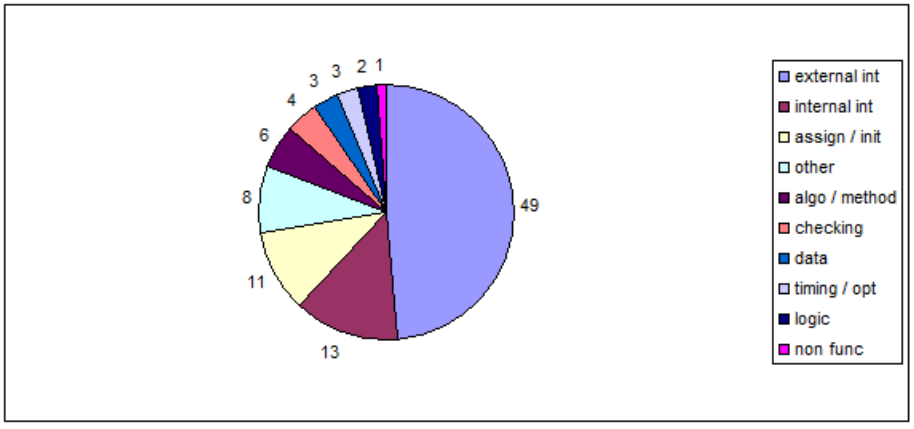
Bulunan 504 hata öncelikle bir yazılım mühendisi tarafından elle sınıflandırılmıştır. Sınıflandırma şeması yukarıda belirtildiği gibi Seamon ve arkadaşları tarafından önerilen yazılım hata şeması olarak seçilmiştir.

Elle yapılan sınıflandırma sonucu yazılım hatalarının sınıflara göre dağılımı Şekil-1'de verilmiştir.



Şekil 1. Yazılım Hata Dağılımı

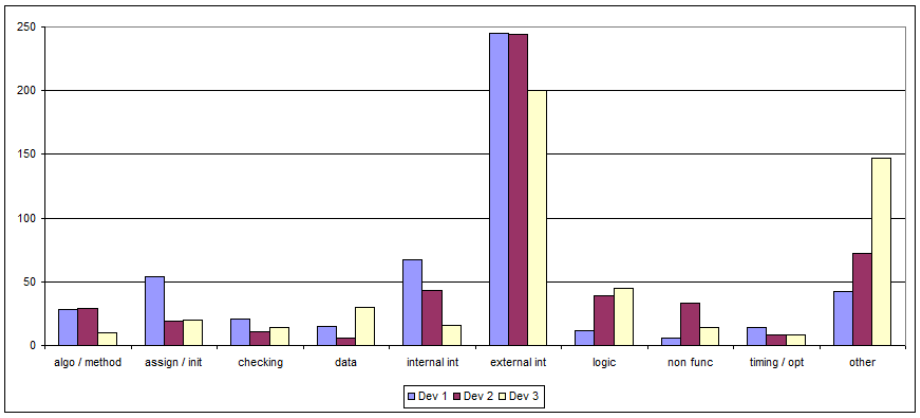
Hata dağılımlarının Seamon şemasına göre dağılım yüzdeleri Şekil-2'de verilmiştir.



Şekil 2. Yazılım Hata Dağılım Yüzdeleri

Şekil 1 ve 2'de sunulan veriler sonraki bölümlerde açıklandığı gibi otomatik sınıflandırma için "1. Veri Seti" olarak kullanılmıştır.

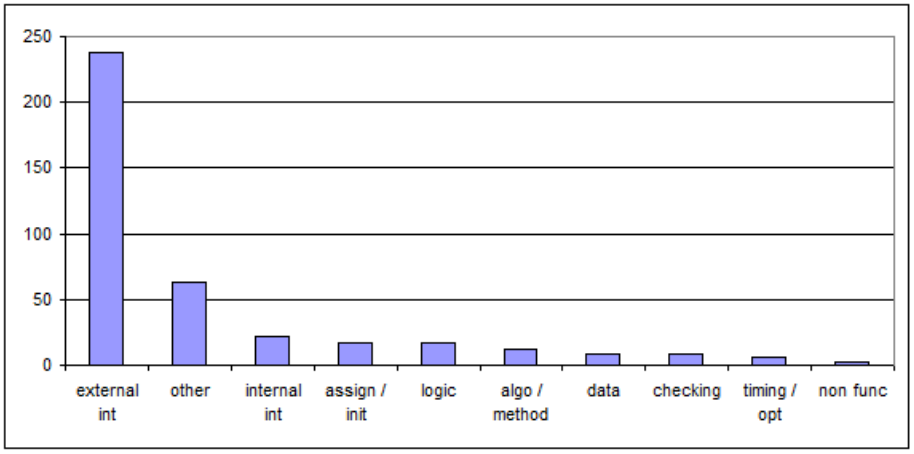
Yukarıdaki dağılımların 1 yazılım mühendisi tarafından oluşturulmasından sonra, bir referans teşkil etmesi açısından aynı 504 hata 3 ayrı yazılım mühendisi tarafından sınıflandırılmıştır. Hataların gerçek kaynağı henüz bulunmadan sadece yazılım hata tanımları üzerinden sınıflandırma yapıldığı için yazılım mühendislerinin hata sınıflandırmaları da farklılık göstermiştir. Örneğin "cihazdan gelen 'hayattayım' sinyali yazılımda görünmüyor" şeklinde yazılan bir hatanın gerçek sebebi cihazın çalışmaması/hatalı çalışması, cihaz ile haberleşen harici arayüz biriminin hatalı olması, harici arayüz bileşeninden bilgi alan dâhili arayüz biriminin hatalı olması, bilginin ekrana gönderildiği grafik arayüz kontrol birimindeki zamanlama problemi ya da diğer bir sebep olabilir. Gerçek sebep ancak hata çözüldükten sonra anlaşılacağı için farklı yazılım mühendislerinin sadece hata tanımına bakarak hatayı farklı şekilde sınıflandırmaları beklenen bir sonuçtur. Üç yazılım mühendisinin 504 adet hatayı ayrı ayrı sınıflandırmaları ile ortaya çıkan hata dağılımı Şekil 3'te verilmiştir.



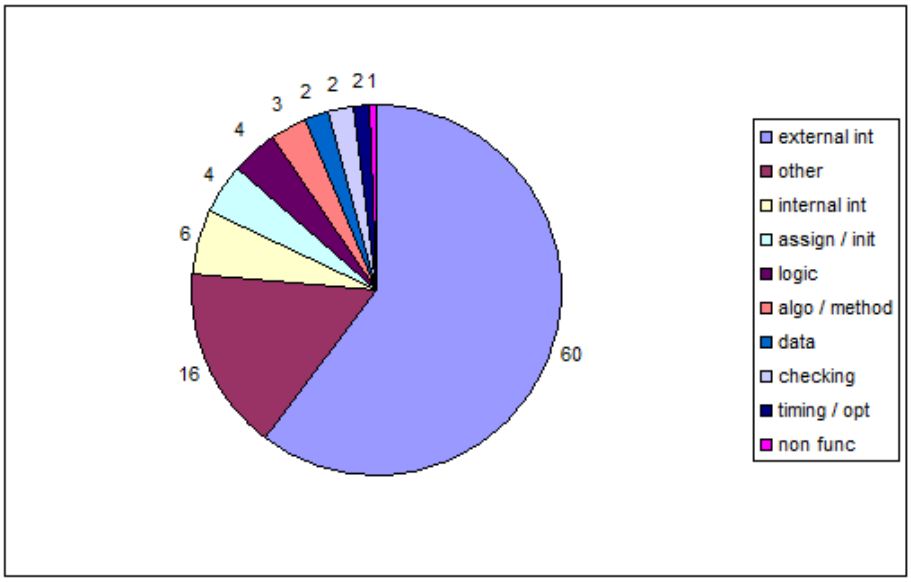
Şekil 3. Üç Yazılım Mühendisinin Yazılım Hata Sınıflandırması

Şekil 3'te verilen hata sınıflandırmalarının incelenmesi ile yazılım mühendislerinin farklı sınıflandırdıkları hataların ancak hatalar çözüldükten sonra sınıflandırılmaları durumunda doğru bir çalışmanın gerçekleştirilebileceği anlaşılmıştır. Yazılım hata tanımlarının ve açıklamalarının kısa ve detay verilmeden yazılması sınıflandırmanın doğru bir şekilde yapılmasını zorlaştıran bir unsurdur. Ayrıca sisteme girilen bazı hata tanımlarının ve/veya açıklamalarının da hatalı olabileceği unutulmamalıdır.

Referans çalışmasının daha doğru sonuca ulaşması için bir sonraki adım en az iki yazılım mühendisinin aynı sınıflandırmayı yaptığı hataların kullanılması olmuştur. Bu duruma uyan 394 hatanın sınıflara göre dağılımı ve bu hataların dağılım yüzdeleri Şekil 4 ve 5'te sırasıyla verilmiştir.



Şekil 4. Ortak sınıflanan hataların dağılımı



Şekil 5. Ortak sınıflanan hataların dağılım yüzdeleri

Şekil 4 ve 5'te sunulan veriler sonraki bölümlerde açıklandığı gibi otomatik sınıflandırma için "2. Veri Seti" olarak kullanılmıştır.

3.4 Otomatik Sınıflandırma Yazılımı

Bu çalışmada Türkçe olarak yazılmış yazılım hatalarının otomatik sınıflandırılmasında kullanılmak üzere Bilgi Erişimi (Information Retrieval) tekniklerini kullanan bir yazılım geliştirilmiştir.

Yazılım Java dilinde geliştirilmiştir. Türkçe tanımları işleyebilmek için Zemberek Kütüphanesi [13] kullanılmıştır.

Sınıflandırma algoritması olarak "Support Vector Machine" (SVM) ve "Naïve Bayes" (NB) kullanılmıştır. SVM'in "Polynomial Kernel" ve "Radial Bases Kernel" varyasyonları ile çalışılmıştır. Sınıflandırma algoritmaları WEKA kütüphaneleri [12] kullanılarak gerçekleştirilmiştir.

Yazılım, öncelikle veriler (yazılım hata tanımları) üzerinde temizleme ve kök bulma (stemming) operasyonları uygulayarak verileri bilgi erişimi için uygun hale getirmektedir. Sonra yukarıda belirtilen sınıflandırma algoritmaları koşularak hata sınıflandırmaları yapılmıştır.

4 Deneysel Sonuçlar

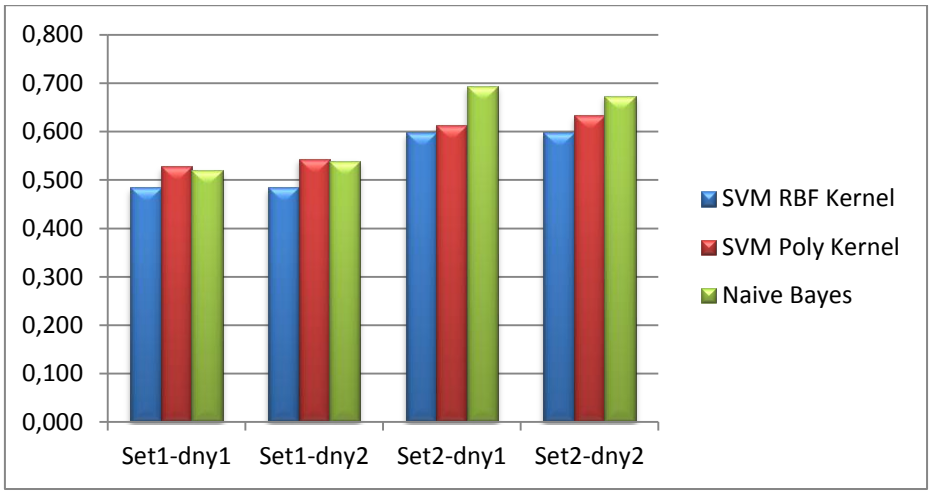
Deneyler 2 adet veri seti ile gerçekleştirilmiştir. 1. Veri setinde bir yazılım mühendisinin 504 yazılım hatasını elle sınıflandırması ile elde edilen veriler kullanılmıştır. Bu veriler Şekil 1 ve 2'de sunulmuştur. 2. Veri setinde ise Şekil 4 ve 5'te sunulan 3 yazılım mühendisinin en az 2'sinin ortak sınıflandırdığı 394 yazılım hatası kullanılmıştır.

Her iki veri setinde verilerin %50'si eğitim veri seti olarak programa girilmiş ve diğer %50'si deney seti olarak kullanılmıştır.

Her iki veri setinde de SVM-Polynomial Kernel, SVM-Radial Bases Kernel ve Naïve Bayes algoritmaları ile sınıflandırmalar gerçekleştirilmiştir. Veri setlerinde duraklama kelimeleri (Stop Words) kullanılarak yapılan deneyler "dny–1" duraklama kelimeleri kullanılmadan yapılan deneyler "dny–2" olarak adlandırılmıştır.

SVM-Polynomial Kernel algoritmasında en iyi sonuçlar SMO 1,0 değerinde elde edilmiştir. Diğer taraftan SVM-Radial Bases Kernel algoritmasında en iyi sonuca SMO 0,034 değerinde ulaşılmıştır.

Deney sonuçları Şekil-6'da verilmiştir. Elde edilen sonuçlar ışığında, geliştirilen otomatik sınıflama yazılımının %70'e varan oranlarda doğru sınıflandırma yaptığını göstermiştir. En iyi sonuç Naïve Bayes algoritmasının duraklama kelimeleri olmadan kullanıldığı durumda elde edilmiştir. Ayrıca 3 mühendis ile yapılan sınıflandırma sonucu elde edilen verilerin tek kişilik sınıflama verilerine göre otomatik sınıflandırmada daha doğru sonuçlar ürettiği gözlemlenmiştir.



Şekil 6. Deney Sonuçları

5 Sonuç ve Tartışma

Bu makalede, Türkçe yazılmış yazılım hata tanımlarının otomatik olarak sınıflandırılması için geliştirilmiş bir yöntem sunulmuştur. Yazılım hatalarının sınıflandırılarak analiz edilmesi ile elde edilen geribildirimler yazılım geliştirme, tasarım ve test süreçlerinin iyileştirilmesinde doğrudan katkı verecek sonuçlar üretmektedir.

İngilizce olarak yazılmış yazılım hataların sınıflandırılması için kullanılan bazı yöntemler bulunmakla beraber Türkçe hata tanımlarını otomatik sınıflayabilen bir çalışma literatürde bulunamamıştır. Bu nedenle bu çalışmanın Türkçe yazılım sınıflandırması yapabilen ilk örnek/prototip olduğu düşünülmektedir. Bu konuyu vurgulayabilmek için makalede, iyi bilinen sınıflandırma algoritmaları ve araçları hakkındaki detaylara fazla yer verilmemiştir. Ancak, kullanılan Türkçe "Zemberek" kütüphanesinin bu çalışmayı mümkün kıldığının altı çizilmelidir.

Kullanılan Seamon hata şeması kolay anlaşılır ve kolay kullanılabilir olması ile örnek olarak kullanılan yazılım projesi için uygun olduğu gibi pek çok yazılım projesi için gerekli hata geri bildirimini verebilecek bir şema olduğu düşünülmektedir. Ancak ihtiyaç olması durumunda diğer yazılım hata şemaları da geliştirilen yazılım üzerinden kolayca kullanılabilmektedir.

Yazılım hata sınıflandırmasının otomatik olarak yapılması yazılım mühendisi ve/veya testçinin kişisel tercihlerini temel almadan sınıflandırma yapılmasına dolayısı ile yöntemin standartlaştırılmasına olanak vermektedir.

Çalışmanın devamı olarak SVM ve NB sınıflandırma algoritmalarının yanı sıra diğer sınıflandırma algoritmalarının da geliştirilen yazılım üzerinden kullanılarak daha hassas sonuçlar elde edilmesi önerilmektedir. Ayrıca testlerde bulunan hata tanımlarının ve açıklamaların daha detaylı ve doğru bir şekilde hata takip sistemlerine girilmesinin elde edilecek sonuçları doğrudan etkilediği unutulmamalıdır.

Kaynaklar

1. Antoniol G., Ayari K., Penta M., Khomh F., Gueheneue Y.: Is it a bug or an Enhancement? A Text-based Approach to Classify Change Requests. Montreal Üniversitesi, Quebec, Canada (2008)
2. Bettenburg N., Just S. and Schröter A.: What Makes a Good Bug Report, SIGSOFT 2008/FSE-16 (2008)
3. Kim S., Whitehead E.J. and Bevan J.: Analysis of Signature Change Patterns, MSR'05 (2005)
4. Li Z., Tan L., Wang X., Lu S., Zhou Y., Zhai C.: Have Things Changed Now ?, ASID'06 (2006)
5. Plosky J., Rohr M., Schwenkenberg P. and Hasselbring W.: Research Issues in Software Fault Categorization, ACM SIGSOFT Software Engineering Notes, V:32, No:6 (2007)
6. Seaman C., Shull F., Regardie M., Elbert D., Feldmann R, Guo Y. and Godfrey S.: Defect Categorization: Making Use of a Decade of Widely Varying Historical Data. ACM ESEM'08 (2008)
7. Shivaji S., Whitehead E.J., Akelle Jr. R.: Reducing Features to Improve Bug Prediction. 2009 IEEE/ACM Conference on Automated Software Engineering (2009)
8. Tan L., Yuan D., Krishna G. and Zhou Y.: iComment: Bugs or Bad Comments? SOSP'2007 (2007)
9. Beizer B. and Vinter O.: Software Testing Techniques. Van Nostrand Reinhold Co., New York, USA, 2nd Edition (1990)
10. Gray J.: Why do computers stop and what can be done about it? IEEE CS Press (1986)
11. Chillarege R., Bhandari I.S., Chaar J.K., Halliday M.J., Moebus D.S., Ray B.K. and Wong M.Y.: Orthogonal defect classification: A concept for in-process measurements. IEEE Transactions on Software Engineering, 18(11): 943-956 (1992)
12. [Online] http://www.cs.waikato.ac.nz/ml/weka/ (2013, July)
13. [Online] https://code.google.com/p/zemberek/ (2013, July)