©2017 Published in 5th International Symposium on Innovative Technologies in Engineering and Science 29-30 September 2017 (ISITES2017 Baku - Azerbaijan)

Yazılım Kalitesinde Kullanılan Zeki Sistemler Üzerine Bir Derleme Çalışması

[1]M. Fatih ADAK
[1]Department of Computer Engineering, Sakarya University, Sakarya, Turkey

Özet

Yazılımın her alana hakim olması ile kaliteli yazılımın sağlanmasının önemi artmıştır. Yazılım kalitesinin belirlenmesi biraz göreceli ve soyut kavramlara dayandığı için şuan halen temel bir kural getirilememiştir. Yapılan çalışmalarda belli başlı metrikler alınarak kalite hakkında yorum yapılmıştır. Bu çalışmada yazılım kalitesinin belirlenmesi ve sınıflandırılmasında yapılan çalışmalar incelenmiş ve özellikle zeki algoritmaların kullanıldığı çalışmalara yoğunlaşılmıştır. Çalışmalardan elde edilen genel kanı yazılım geliştirme sürecinde metrikler kullanılarak yazılım kusurları zeki yöntemler sayesinde tespit edilmiş ve çalışma anında oluşması muhtemel yazılım hatalarının önüne geçilmesi amaçlanmıştır.

Anahtar kelimeler: yazılım kalitesi, kalite metrikleri, zeki algoritmalar

A Survey on Software Quality Using Smart Systems

Abstract

The importance of creating quality software is start to increase as software are used everywhere in life. Since the determination of software quality is a bit abstract, there is still no primary rule. Comments are made about quality by taken into account some of the metrics. In this study, determination and classification studies on software quality are investigated and especially focused on the studies using smart algorithms. The general concept obtained from the studies is that software faults are detected by smart algorithms using software metrics in development process and it is aimed to prevent possible software errors that may occur during runtime.

Key words: software quality, quality metrics, smart algorithms

1. Giriş

Günümüzde bilişimin her alana dahil olması, her işlemin online yapılabilmesi, mobil cihazların yaygınlaşması ile kullanılan yazılımların performansı, tükettikleri enerji, hızı gündeme gelmiş ve önem kazanmıştır. Aynı işi gerçekleştiren birçok yazılımın bulunmasına rağmen bazılarının çok daha önde olması bazılarının hiç ilgi görmemesi aslında o yazılımın kalitesi ile doğrudan ilgilidir. Yazılım kalite denetlemesi, yazılım geliştirme sürecinde yapıldığı zaman geliştirme süreci açısından maliyet ve zaman kazancı anlamına gelecektir. Burada sorulması gereken soru yazılımın kalitesinin nasıl belirleneceğidir? Literatürde bu kısımda yazılım metrikleri ön plana çıkmaktadır ve araştırmacılar yazılım metrikleri kullanılarak, yazılım geliştirme sürecinde erken dönemde hata tahmininin yapılabileceğini belirtmektedirler [1], [2].

*Corresponding author: Address: Faculty of Computer and Information Sci., Department of Computer Engineering Sakarya University, 54187, Sakarya TURKEY. E-mail address: fatihadak@sakarya.edu.tr, Phone: +902642957049

Yazılım kalitesinin ölçümünde kullanılan yazılım metrikleri, kaynak kod üzerinde bazı parametrelerin belirlenmesi ve yazılım geliştirme sürecinin tümünde bu parametreler ile sisteme dahil olarak yazılımın kusuru hakkında bilgi vermesidir [3]. Yazılım metriklerinin standart hale getirilme çabaları olmuş ve yazılım kalitesinin nitelikleri şeklinde ISO/IEC 9126 tarafından tanıtılmıştır [4]. Bu çalışmada yazılım kalitesinin belirlenmesinde kullanılan zeki algoritmalara değinilecektir. Zeki algoritmalar, bir problemin çözümünde alternatif yollardan birini seçip elde edilen sonuca göre seçimi tekrarlayan ve makul bir sürede problemin tam veya kısmi bir çözümüne ulaşan fakat her zaman çözümün garanti edilmediği algoritmalardır [5]. Zeki algoritmalardan Genetik algoritma yazılım kalitesi alanında sıklıkla kullanılmıştır. Örneğin yapay sinir ağı ile birlikte kullanılan genetik algoritma, 6 metriğin bulunduğu CK metrik seti üzerine uygulanmış ve 10-fold ve 5-fold çapraz doğrulama ile başarılı sonuçlar elde edildiği görülmüştür [6]. Paralel Genetik algoritma kullanılarak yazılım metrikleri sınıflandırılmış ve sınıflandırma başarısı % 78,4 olarak ölçülmüştür [7]. Biraz farklı fakat aynı çerçevede değerlendirilebilecek bir çalışmada Genetik algoritma, Bulanık mantık ile birlikte kullanılmış ve daha kararlı bir yapı elde edilmiştir [8]. Yazılım modülleri üzerinde erken hata tahmininde kullanılan Genetik algoritma sayesinde RMSE değerini 0,0712 düzeyine düşürebilmiştir [9]. Yazılım kalitesinin belirlenmesinde istatistiksel metotların kullanıldığı ve başarılı sonuçlar elde edildiği de görülmüştür [1]. Yazılım kalitesinin ölçülmesinde hataya meyilli (fault prone) ya da hataya meyilli değil (non fault prone) şeklinde sınıflandırma yapılabilmektedir [10]. Bu sınıflandırmayı yaparken makine öğrenmesi kullanan birçok çalışmaya rastlamak mümkündür. Örneğin Kanmani ve arkadaşları nesne yönelimli yazılımlarda hata tahmini için yapay sinir ağlarını kullanmışlardır [11]. Yine nesne yönelimli yazılım üzerine bir başka hata tahmini çalışmasında % 91,53 oranında hatalı sınıf tespit edilebilmiştir [12]. Nasa yazılım verisi üzerinde yapılan yazılım efor tahmininde yapay sinir ağları kullanılmış ve başarılı sonuçlar elde edilmiştir [13].

2. Yazılım Metrikleri

Yazılım kalitesi alanındaki çalışmalar incelendiğinde etkisinin olabileceği düşünülen metrikler listelenmiştir. Bunun yanında nesne yönelimli yazılımlarda ayrıca birçok metrikten söz etmek mümkündür. Sıklıkla kullanılan yazılım metriklerinin kısaltmaları ve açıklamaları Tablo 1.'de verilmiştir [1], [7], [14], [15].

Table 1. Sıklıkla kullanılan yazılım metrikleri ve açıklamaları

Kısaltma	Açıklama
WMC	Sınıf başına düşen ağırlıklı metot
DIT	Kalıtım ağacının derinliği
NOC	Çocuk sayısı
CBO	Nesne sınıfları arasındaki bağlantı
RFC	Bir sınıfın verdiği cevap
LCOM	Metotlarda uyum eksikliği
NPM	Public metot sayısı
LOC	Kod satır sayısı
MFA	Fonksiyonel soyutlama ölçüsü
IC	Kalıtım bağlantısı
ALOC	Metot başına düşen satır sayısı ortalaması

Table 1 devam. Sıklıkla kullanılan yazılım metrikleri ve açıklamaları

RCC1	Yorum satır sayısının toplam satıra oranı
TOK	Simge sayısı
ATOK	Metot başına düşen simge ortalaması
MTOK	Metot başına düşen simge orta sayısı
DEC	for, while, if, switch,gibi karar mekanizmaları
CC	Döngüsel Karmaşıklık
ADEC	Metot başına düşen ortalama DEC sayısı
MDEC	Metot başına düşen orta DEC değeri
INCL	İç sınıf sayısı
MNL1	Maksimum metot ismi uzunluğu
MNL2	Minimum metot ismi uzunluğu
MNL3	Ortalama metot ismi uzunluğu
MNL4	Metot ismi uzunluğu orta değeri
MAXP	Maksimum parameter sayısı
CLAS	Sınıf sayısı
CONS	Yapıcı metot sayısı
OVRM	Override edilmiş metot sayısı
PRVM	Private üye yüzdeliği
PROM	Protected üye yüzdeliği
PUBM	Public üye yüzdeliği

Sıklıkla bu metriklerin kullanıldığı çalışmalardan da görülmektedir. Zhang ve arkadaşlarının yaptıkları bir çalışmada bu yazılım metriklerinden LOC ve CC'nin kullanılmasının kısıtlı optimizasyon tekniklerine göre daha verimli sonuçlar verdiği görülmüştür [16]. Yine yazılım kusur tespitinden bu metriklerden faydalanılıp başarılı sonuçlar elde edilmiştir [17]. Benzer başka çalışmalarda aynı amaç için Bayes ağları kullanılmıştır [18], [19]. Yazılım metriklerinden public, private gibi erişim niteleyicilerinin sayılarının da yazılım kalitesinde önemli olduğu Lei Ma'nın çalışmasında belirtilmiştir [20].

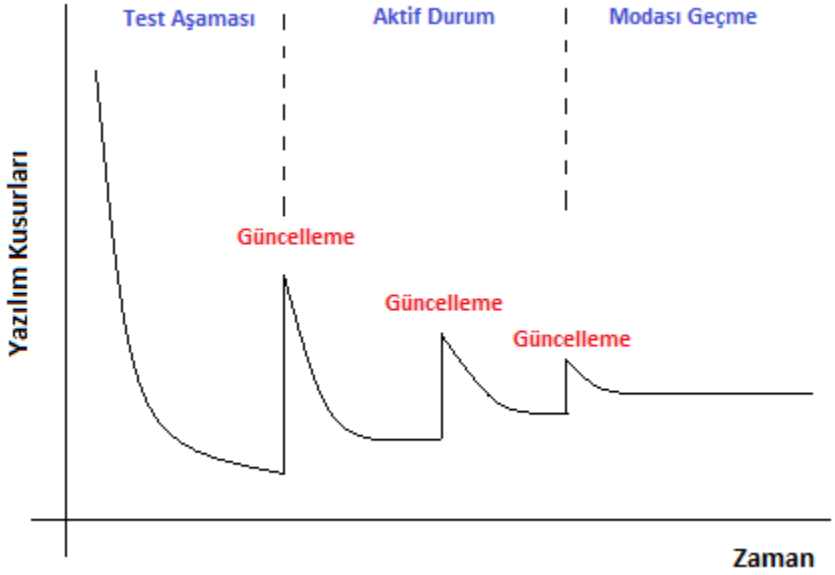
2.1. Metrikler Kullanılarak Yazılım Kusur Tespiti

Bir sistemde veya yazılımda eksiklik tespiti veya konusu geçtiği zaman terimler karıştırılabilmektedir. IEEE standardına göre kusur ve hata birbirinden farklı terimlerdir. Hata insan kaynaklı olup istenmeyen bir sonuç üreten bir eylemdir fakat kusur ise programdaki hatalı işlem, veri veya adımdır [21]. Bir yazılımda kusur tahmin etme performansı yazılım ölçüm metriklerinin kalitesine bağlıdır [22].

2.1.1. Yazılım Hatası

Yazılım geliştirme sürecinde tespit edilemeyen kusurlar neticesinde çalışma anında beliren olaylardır. Tespit edilmeleri zordur. Minimum hata için dikkatli ve ölçülü bir yazılım geliştirme süreci gerekir. Yazılım güvenirliğini doğrudan etkileyen bu faktör geliştirme sürecinde en önem verilmesi gereken faktördür. Şekil 1'e bakıldığında yazılım kusurlarının geliştirme süreci

boyunca düştüğü fakat yeni bir güncelleme ile kusurların tekrar arttığı güncellemelerin yazılımın modasının geçmesiyle birlikte son bulmasıyla kusur oranının sabit değerde kaldığı görülmektedir [23]. Aynı zamanda kod yazmanın kusur açısından çok hızlı bir artış yaptığı da görülmektedir.

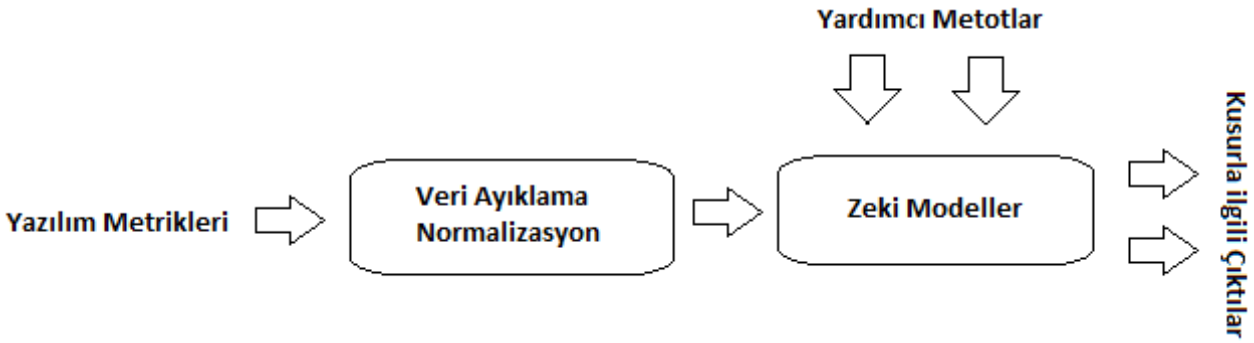


Şekil 1. Yazılım kusur grafiği

2.1.1. Yazılım Kusuru

Çalışma anında oluşan hatalar geliştirilen yazılımın kusurlu olduğunu gösterir. Normal şartlar altında tamamen hatadan arındırılmış bir yazılımın geliştirilmesi mümkün değildir. Burada amaçlanan kusurları olabildiğince minimum seviyeye indirmektir. Bunun için standart haline gelmiş ve gelmeye devam eden yazılım metrikleri kullanılır. Tabi burada belirtilmesi gereken yazılımcı temelli oluşan hatalar olabildiği gibi kullanıcı temelli hatalarda oluşabilmektedir. Fakat daha çok yazılımcı temelli oluşan hatanın önüne geçmek için çalışmalar yapılmaktadır. Bu önüne geçme çalışmaları da yazılımın kalitesini artırmaktadır.

Yazılım ile ilgilenen kişiler için yazılım kalitesini belirleyebilmek adına ellerinde yeterince araç bulunmamaktadır. Bu iş genelde metriklerden yola çıkarak yapılmaktadır. Çalışmalar incelendiğinde Şekil 2'ye benzer yapılar ortaya çıkmaktadır.



Şekil 2. Kusur tespitinde kullanılan genel yapı

Örneğin Shatnawi yazılım kusur tespiti için ROC analizini kullanmıştır [24]. Yazılım geliştirme sürecinde bu yazılım kusur tespiti ne kadar etkili çalıştırılırsa yazılım geliştirmenin erken evrelerinde kusur tespiti kolaylıkla yapılır ve bu geliştiriciye maliyetten ve zamandan kazanç olarak geri dönecektir. Hatalı sınıfların tespiti için yapılan çalışmaların incelendiği detaylı bir çalışmada Kumar ve arkadaşları, regresyon, makine öğrenmesi, bulanık mantık gibi birçok yöntemin kullanıldığı ve birçok farklı metriğin temel alındığını belirtmişlerdir [25]. Erturk ve Sezer yazılım kusur tahmininde, destek vektör makinesi, yapay sinir ağı ve ANFIS'i karşılaştırmış ANN ve ANFIS daha başarılı olmuştur [26]. Nesne yönelimli yazılımda kusur tespitinde destek vektör temelli bulanık sınıflandırma modeli kullanılmış ve 76,5 mean değeri elde edilmiştir [27].

Sonuçlar

Yazılım dünyasındaki gelişmeler beraberinde çalışma anında büyük sıkıntılar oluşturabilecek hataları getirmiştir. Binlerce veya daha fazla kod satırından oluşan yazılımlardaki kusurların bulunması belli bir metot geliştirmeden çok zor olmaktadır. Bundan dolayı yazılım metrikleri tanımlanmış ve bu metriklere bağlı kalarak zeki metotlar yardımıyla kusur tespiti yapılmıştır. Geliştirilecek yeni zeki metotlar ve anlamlı metrikler ile çalışma zamanında oluşacak hataların ve kod yazım aşamasındaki maliyetlerin düşürülmesi sağlanacaktır.

Referanslar

[1] R. Malhotra and A. Jain, "Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality," *J. Inf. Process. Syst.*, vol. 8, no. 2, pp. 241–262, Jun. 2012.

[2] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *J. Syst. Softw.*, vol. 82, no. 6, pp. 981–992, Jun. 2009.

[3] A. S. Nuñez-Varela, H. G. Pérez-Gonzalez, F. E. Martínez-Perez, and C. Soubervielle-Montalvo, "Source code metrics: A systematic mapping study," *J. Syst. Softw.*, vol. 128, pp. 164–197, Jun. 2017.

[4] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung, "Measuring Software Product Quality: A Survey of ISO/IEC 9126," *IEEE Softw.*, vol. 21, no. 5, pp. 88–92, Sep. 2004.

[5] S. Edelkamp and S. Schrödl, *Heuristic search : theory and applications*. Elsevier/Morgan Kaufmann, 2012.

[6] L. Kumar and S. K. Rath, "Neuro – Genetic Approach for Predicting Maintainability Using Chidamber and Kemerer Software Metrics Suite," vol. 361, H. Unger, P. Meesad, and S. Boonkrong, Eds. Cham: Springer International Publishing, 2015, pp. 31–40.

[7] R. Vivanco and N. Pizzi, "Finding Effective Software Metrics to Classify Maintainability Using a Parallel Genetic Algorithm," in *Genetic and Evolutionary Computation – GECCO 2004*, 2004, pp. 1388–1399.

[8] S. S. Dahiya, J. K. Chhabra, and S. Kumar, "Use of genetic algorithm for software maintainability metrics' conditioning," in *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, 2007, pp. 87–92.

[9] P. S. S, P. S. S, S. Khullar, S. Singh, S. K. Bains, M. Kaur, and G. Singh, "A Study on Early Prediction of Fault Proneness in Software Modules using Genetic Algorithm," *World Acad. Sci. Eng. Technol.*, vol. 48, pp. 648–653, 2010.

[10] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Appl. Soft Comput.*, vol. 27, pp. 504–518, Feb. 2015.

[11] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-oriented software fault prediction using neural networks," *Inf. Softw. Technol.*, vol. 49, no. 5, pp. 483–492, May 2007.

[12] A. Mahaweerawat, P. Sophatsathit, C. Lursinsap, and P. Musilek, "Fault Prediction in Object-Oriented Software Using Neural Network Techniques," *Proc. InTech Conf*, pp. 2–4, 2004.

[13] J. Kaur, S. Singh, K. S. Kahlon, and P. Bassi, "Neural Network-A Novel Technique for Software Effort Estimation," *Int. J. Comput. Theory Eng.*, pp. 17–19, 2010.
[14] S. H. Kan and S. H., *Metrics and models in software quality engineering*. Addison-Wesley, 2003.
[15] B. Henderson-Sellers and Brian, *Object-oriented metrics : measures of complexity*. Prentice Hall PTR, 1996.
[16] Z. Zhang, Z. Chen, R. Gao, E. Wong, and B. Xu, "An empirical study on constraint optimization techniques for test generation," *Sci. China Inf. Sci.*, vol. 60, no. 1, p. 12105, Jan. 2017.
[17] E. Rashid, "Improvisation of case-based reasoning and its application for software fault prediction," *Int. J. Serv. Technol. Manag.*, vol. 21, no. 4/5/6, p. 214, 2015.
[18] A. Okutan and O. T. Yıldız, "Software defect prediction using Bayesian networks," *Empir. Softw. Eng.*, vol. 19, no. 1, pp. 154–181, Feb. 2014.
[19] G. J. Pai and J. Bechta Dugan, "Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods," *IEEE Trans. Softw. Eng.*, vol. 33, no. 10, pp. 675–686, Oct. 2007.
[20] L. Ma, C. Zhang, B. Yu, and H. Sato, "An empirical study on the effects of code visibility on program testability," *Softw. Qual. J.*, vol. 25, no. 3, pp. 951–978, Sep. 2017.
[21] A. K. Pandey and N. K. Goyal, "Early Software Reliability Prediction," in *Early Software Reliability Prediction*, 2013, pp. 1–16.
[22] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Softw. Pract. Exp.*, vol. 41, no. 5, pp. 579–606, Apr. 2011.
[23] P. D. T. O'connor, "Software reliability: Measurement, prediction, application," *Qual. Reliab. Eng. Int.*, vol. 4, no. 3, pp. 296–296, Jul. 1988.
[24] R. Shatnawi, "The application of ROC analysis in threshold identification, data imbalance and metrics selection for software fault prediction," *Innov. Syst. Softw. Eng.*, Aug. 2017.
[25] L. Kumar, S. Misra, and S. K. Rath, "An empirical analysis of the effectiveness of software metrics and fault prediction model for identifying faulty classes," *Comput. Stand. Interfaces*, vol. 53, pp. 1–32, Aug. 2017.
[26] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Syst. Appl.*, vol. 42, no. 4, pp. 1872–1879, Mar. 2015.
[27] B. Mishra and K. K. Shukla, "Defect Prediction for Object Oriented Software using Support Vector based Fuzzy Classification Model," *Int. J. Comput. Appl.*, vol. 60, no. 15, pp. 975–8887, 2012.