Kritika na rad: AI-Based Fault-Proneness Metrics for Source Code Changes

Stefan Nožinić
Departman za matematiku i informatiku, Prirodno-matematički fakultet, Univerzitet u Novom Sadu.

25. januar 2024.

Sažetak

Ovaj dokument sadrži kritiku na rad koji ispituje metode procene sklonosti na greške tokom razvoja softvera upotrebom modela mašinskog učenja i primenom istih na promene u izvornom kodu.

1 Uvod i opis problema

Tokom razvoja softvera, veoma često dolazi do uvođenja grešaka u sam rad tog istog softvera. Greška je bilo kakva devijacija u odnosu na zahtevanu specifikaciju. Tokom vremena, praksa razvoja softvera je utvrdila procedure koje značajno mogu sprečiti uvođenje grešaka u sam softver [1].

Code Review je procedura koja ima za cilj da preventira unošenje grešaka u softver prilikom svake promene na izvornom kodu. Ova praksa se ogleda u tome da više inženjera pregleda i odobri svaku promenu izvornog koda pre nego što ona postane deo glavne grane razvoja. Pokazano je da ovakva praksa značajno može unaprediti kvalitet softvera [1].

Mana date metode jeste što ona zahteva vreme inženjera i dosta resursa (pre svega finansijskih). Zbog ovog ograničenja, dosta je istraživanja rađeno na automatskoj proceni sklonosti određenih delova koda na greške [2, 3, 4, 5].

Ovaj rad se bavi analizom i kritikom metoda pomenutih u [2]. U sekciji 2 se opisuje sam oblik i procesiranje ulaznih podataka. U sekciji 3 se opisuju i analiziraju metode korišćene u [2]. Unapređenja za pomenuti rad se detaljnije diskutuju u sekciji 4. Zaključak ovog rada je dat u sekciji 5.

2 Ulazni podaci

Ulazni podaci su promene u izvornom kodu koje se mogu dobiti kroz sistem za kontrolu verzija. U konkretnom slučaju su promene dobijene iz softverskih paketa otvorenog koda i upotrebom sistema za kontrolu verzija git [6].

Promene nad kojima su metode evaluirane su promene softverskih paketa napisanih u programskom jeziku Java i one uključuju promene u metodama klasa.

3 Metode

Prilikom procene sklonosti na greške za datu promenu, procenjuje se koliko je data promena velika odnosno koliko se prethodna verzija datog dela koda razlikuje od nove verzije. U [2] su analizirane metode koje se baziraju na merenju razlike stabala [7] i metode koje se baziraju na transformaciji datog koda u euklidski vektorski prostor [8].

Kvalitet datih metoda je meren tako što se njihova procena poredila sa procenom koju je dao profesionalni softverski inženjer. U radu autori su postavljali dva pitanja:

- Koliko je profesionalni softverski inženjer objektivan?
- Kolika je korelacija procena sklonosti na greške datih metoda i procene profesionalnog softverskog inženjera?

Na prvo pitanje, autori su sproveli dodatno istraživanje gde je jedan od autora, paralelno sa profesionalnim softverskim inženjerom procenjivao sklonost na greške nad datim promenama koda. Nakon ovog procesa, izračunata je korelacija između procena autora i profesionalnog softverskog inženjera koja je iznosila 0.85. Iz datih rezultata autori zaključuju da je profesionalni softverski inženjer objektivno procenjivao sklonost ka greškama.

Na drugo pitanje autori su pokušali da daju odgovor tako što su računali korelaciju između procena profesionalnog softverskog inženjera i analiziranih metoda. Četiri metode su bile analizirane i njihove korelacije sa ocenama profesionalnog softverskog inženjera su date u tabeli 1.

4 Unapređenja

Autori rada su u zaključku predložili nekoliko unapređenja. Jedan smer u kom može ići dalje istraživanje jeste implementacija datih metoda u realnim softverskim projektima. Pored toga, predlaže se da se metode mogu uporediti sa stvarnim brojem detektovanih grešaka koje su prijavljene naknadno, a koje su prouzrokovane datim promenama.

Dodatni predlozi za unapređenje bi bili i ispitivanje drugih metoda, kao što su [9, 10, 11]. Ove metode su se

Metoda	Koeficijent koeralcije	p-vrednost
STK	0.6	0.0000624
CodeBERT	0.52	0.0000169
PTK	0.40	0.00000518
SSTK	0.31	0.000443

Tabela 1: Koeficijenti korelacije

pokazale kao performantnije u primenama sumarizacije koda i detekcije duplikata u kodu.

Takođe, autori su ispitivali samo promene nad Java kodom i promene koje se dešavaju unutar metoda klasa. Ovde nedostaje šira pokrivenost jer često i same promene u strukturi koda mogu dovesti do grešaka kao i različit tretman nekih promena u zavisnosti od programskog jezika i njegove ekspresivnosti [12].

Dodatno, u radu nedostaje detaljnija analiza datih metoda na nekim veštačkim primerima kako bi se bolje razumelo ponašanje datih metoda. Takođe, rad navodi da je procenat promenjenih linija koda pozitivno koreliran sa ocenama koje su date od strane profesionalnog softverskog inženjera. Ovakvi rezultati mogu potencijalno značiti da su i same metode izuzetno naklonjene tome da visoko ocenjuju veće promene iako one mogu biti estetske prirode kao što je dodavanje komentara ili promena varijabli. Zbog navedenog, potrebno je ispitati prosečnu ocenu u zavisnosti od kategorije promene, gde bi se promene kategorizovale po semantičkom značenju.

5 Zaključak

Analizirani rad pruža pregled metoda koje mogu biti korišćene kako bi polu-automatizovale procenu uticaja određenih promena na sklonost greškama u softveru. Sam rad ne analizira stvarni uticaj analizom prijavljenih grešaka nego zapravo računa korelaciju između procene od strane modela i procene od strane čoveka.

Sam rad je moguće unaprediti uvođenjem implementacije u produkciji kao i proširivanjem analize na više metoda, analiza na više različitih programskih jezika kao i analiza svih mogućih promena koje se mogu desiti tokom razvoja softvera.

Literatura

[1] Denis Kozlov, Jussi Koskinen, et al. Fault-proneness of open source software: Exploring its relations to internal software quality and maintenance process. *The Open Software Engineering Journal*, 7(1), 2013.

[2] Francesco Altiero, Anna Corazza, Sergio Di Martino, Adriano Peron, and Luigi LL Starace. Ai-based fault-proneness metrics for source code changes. In *International Conference on Software Process and Product Measurement (IWSM/MENSURA)*, volume 23, 2023.

[3] Norman E Fenton and Martin Neil. A critique of software defect prediction models. *IEEE Transactions on software engineering*, 25(5):675–689, 1999.

[4] Iker Gondra. Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software*, 81(2):186–195, 2008.

[5] Alexandre Ouellet and Mourad Badri. Combining object-oriented metrics and centrality measures to predict faults in object-oriented software: An empirical validation. *Journal of Software: Evolution and Process*, page e2548, 2023.

[6] Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.

[7] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *11th conference of the European Chapter of the Association for Computational Linguistics*, pages 113–120, 2006.

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[9] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*, 2020.

[10] Jie Li, Hao Zhu, Lixuan Li, and Xiaofang Zhang. Graphplbart: Code summarization based on graph embedding and pre-trained model.

[11] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. Cc2vec: Distributed representations of code changes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 518–529, 2020.

[12] Gábor Törley. Expressiveness of programming languages and environments: a comparative study, in. *Teaching mathematics and computer science*, 6:111–141, 2008.