

UPOTREBA ALATA ZA STATIČKU ANALIZU KODA U OBRAZOVANJU

Danilo Nikolić¹, Sara Havzić², Miroslav Stefanović³, Dušan Krstić⁴, Sara Kijanović⁵

^{1,2,3,4,5}Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Novi Sad, Srbija

¹nikolic.danilo@uns.ac.rs

Kratak sadržaj: Alati za statičku analizu koda se sve više koriste za poboljšanje kvaliteta koda. Kvalitet izvornog koda je ključni faktor u svakom softverskom proizvodu i zahteva stalnu inspekciju i nadzor. Statička analiza koda je validan način da se zaključi ponašanje programa bez njegovog izvršavanja. Mnogi alati omogućavaju statičku analizu u različitim okvirima, različitim programskim jezicima i otkrivanje različitih nedostataka u izvornom kodu. Ovi alati sve više pronalaze primenu i u obrazovanju softver inženjera. U ovom radu opisani su načini i važnost primene alata za statičku analizu koda u visokoškolskom obrazovanju na usmerenjima koja obrađuju oblast softverskog inženjerstva.

Ključne reči: Statička analiza koda, kvalitet koda, alati, edukacija

USE OF STATIC CODE ANALYSIS TOOLS IN EDUCATION

Abstract: Static code analysis tools are increasingly used to improve code quality. Source code quality is a key factor in any software product and requires constant inspection and monitoring. Static code analysis is a valid way to infer the behavior of a program without actually executing it. Many tools allow static analysis in different frameworks, different programming languages and detection of different defects in the source code. These tools are increasingly being used in the education of software engineers. This paper describes the ways and importance of applying tools for static analysis of code in higher education in the field of software engineering.

Key Words: Static code analysis, code quality, tools, education

1. UVOD

Od početka razvoja softvera kao naučne discipline, programerima je nedostajala metoda procene kvaliteta izvornog koda [1].

Kvalitet izvornog koda je ključni faktor u svakom softverskom proizvodu i zahteva stalnu proveru i praćenje. Statička analiza se koristi za održavanje i poboljšanje kvaliteta izvornog koda. Statička analiza programskog koda se koristi od ranih 1960-ih za optimizaciju rada kompajlera [2]. Kasnije se pokazala korisnim za alate za otklanjanje grešaka, kao i za razvojna okruženja. Sve veći broj alata omogućava statičku analizu koda, od kojih mnogi omogućavaju analizu različitih programskih jezika [3].

Alati za statičku analizu koriste se za generisanje izveštaja i ukazivanje na određena odstupanja od propisanih standarda kvaliteta koda. Međutim, ovi alati ne dozvoljavaju automatske izmene izvornog koda.

Odluka o promeni strukture prethodno napisanog koda je na programerima softvera. Alat za statičku analizu koda pomaže programerima softvera tako što, pored generisanja izveštaja, navodi uzrok određenog defekta, kao i kako se ovaj nedostatak može ispraviti.

Kao prvi korak u ovom istraživanju sproveden je sistematski pregled literature iz oblasti statičke analize koda [4], i predstavljeni su različiti alati za statičku analizu koda, programski jezici za koje imaju podršku i vrste defekata koje otkrivaju, i analizirali.

Alati za statičku analizu koda primenjuju se direktno na izvorni kod programa [1]. Programeri softvera i drugi korisnici ovih alata koriste različite načine za primenu ovih alata preko izvornog koda programa. Ovaj rad opisuje rad i važnost upotrebe ovih alata i ima za cilj da identifikuje na koje sve načine je moguće ovakve alate uključiti u proces obrazovanja.

2. TEORIJSKE OSNOVE

Proces statičke analize koda je koristan ne samo za optimizaciju rada kompajlera (što je bila prvobitna svrha) već i za otkrivanje nedostataka i mogućih nepravilnosti. Na ovaj način je moguće kreirati alate koji će programerima pomoći da razumeju ponašanje programa i identifikuju različite nedostatke programa bez njegovog izvršavanja. Alati koji se koriste za statičku analizu koda su programi koji objašnjavaju ponašanje drugih programa [2].

Na početku je neophodno razmotriti vrste defekata koje statička analiza može otkriti. Problemi mogu nastati i zbog nerazumevanja šta podrazumeva sigurno programiranje. Nije neuobičajeno da programeri nisu svesni kako će napadači pokušati da iskoriste deo koda. Imajući ovo na umu, statička analiza je pogodna za identifikaciju problema

iz nekoliko razloga [5]: Alati za statičku analizu primenjuju provere temeljno i dosledno, bez ikakvih predrasuda koje programer može imati o tome koji delovi koda su „interesantni“ iz bezbednosne perspektive ili koji delovi koda se lako mogu izvršiti dinamičkim testiranjem. Pregled koda se mora obaviti bez subjektivnog osećaja programera. Stoga, nepristrasna analiza može biti dragocena, što alati za analizu statičkog koda pružaju.

Ispitivanjem samog koda, alati za statičku analizu često mogu ukazati na osnovni uzrok bezbednosnog problema, a ne samo na jedan od njegovih simptoma. Ovo je posebno važno kako bi se osiguralo da su nedostaci ispravljeni. Statička analiza može da pronađe nedostatke u ranom razvoju, čak i pre nego što se program pokrene po prvi put.

Rano pronalaženje nepravilnosti smanjuje troškove otklanjanja nepravilnosti, ali ciklus brzih povratnih informacija može pomoći u vođenju rada programera: Programer ima priliku da ispravi nedostatke za koje ranije nije bio svestan da se mogu pojaviti.

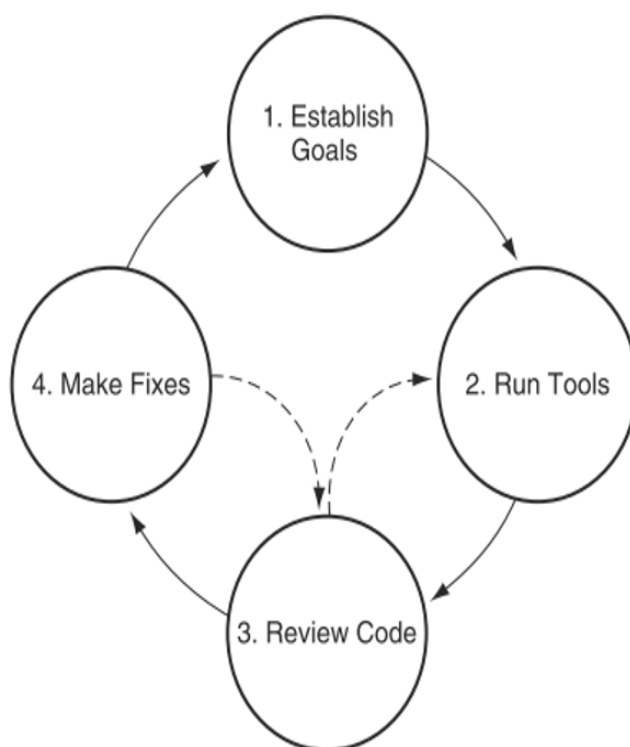
Kada analiza otkrije novi tip nepravilnosti, alati za statičku analizu olakšavaju ponovnu proveru velikog dela koda kako bi se otkrilo gde se nova nepravilnost može pojaviti. Neke bezbednosne greške su bile u softveru godinama pre nego što su otkrivene, zbog čega je procena zastarelog koda novootkrivenih tipova nepravilnosti neprocenjiva.

Međutim, uprkos svemu navedenom, alati za statičku analizu koda se ne koriste često. Programeri softvera ne odlučuju često da poboljšaju kvalitet svog izvornog koda koristeći ove alate [1].

Standardni ciklus pregleda koda uključuje četiri glavne faze [5]:

1. definisanje cilja,
2. pokretanje alata za statičku analizu koda,
3. pregled izvornog koda (na osnovu dobijenog rezultata statičke analize),
4. reinženjering izvornog koda.

Pored standardnog ciklusa, Slika 1 prikazuje nekoliko potencijalnih povratnih informacija ili manjih iteracija između koraka standardnog ciklusa, čineći ciklus složenijim od standardnog. Broj iteracija ovog ciklusa i potreba za povratnom informacijom unutar ciklusa zavise od primene izveštaja statičke analize koda.

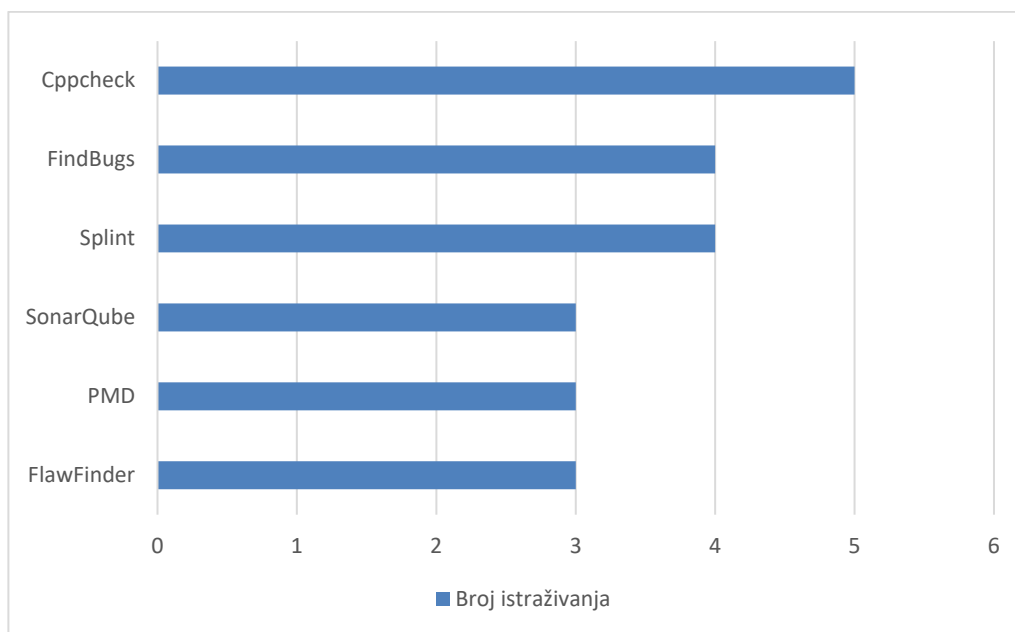


Slika 1. Standardni ciklus pregleda koda [5]

3. PRIMERI ALATA I PREDLOZI PRIMENE U NASTAVNOM PROCESU

Postoji veliki broj alata za statičku analizu koda [4], svakodnevno nastaju novi alati koji obezbeđuju proces statičke analize i postojeći alati se razvijaju tako da omoguće pronalazak većeg opsega nepravilnosti, kao i brži proces izvođenja statičke analize.

U istraživanjima najviše se koriste sledeći alati (Grafikon 1).



Grafikon 1- zastupljenost alata u naučnim istraživanjima [4]

U nastavku biće kratko opisani najzastupljeniji alati u istraživanjima:

1. Cppcheck – alat za statičku analizu otvorenog koda za C/C++ kod. Pruža jedinstvenu analizu koda za otkrivanje grešaka i fokusira se na otkrivanje nedefinisanog ponašanja i opasnih konstrukcija kodiranja [6].
2. FindBugs – program koji koristi statičku analizu da traži greške u Java kodu. To je softver otvorenog koda, distribuiran pod uslovima manje GNU javne licence [7].
3. SonarQube – platforma otvorenog koda koju je razvio SonarSource za kontinuiranu inspekciju kvaliteta koda. Sonar radi statičku analizu koda, koja daje detaljan izveštaj o greškama, mirisima koda, ranjivosti, dupliranju koda [8].

Postoji više načina za implementaciju alata za analizu statičkog koda u obrazovni proces na fakultetu, u ovom radu biće predloženi neki od njih:

1. Uključivanje alata u nastavni plan i program: Zadavanje zadatka iz programiranja koji zahtevaju od studenata da koriste alat i podnose rezultate kao deo svog rada.
2. Korišćenje alata prilikom pregleda izvornog koda na projektima: Studenti predaju projekat i izvorni kod na pregled i koriste alat da identifikuju potencijalne probleme pre nego što budu ocenjeni.
3. Održavanje radionica ili sesije obuke: Radionice ili sesije obuke za studente da nauče kako da koriste alat i njegove karakteristike
4. Korišćenje u svrhu alat za učenje: Studenti mogu da koriste neki od alata za pronalaženje i ispravljanje grešaka u uzorku koda, kako bi im pomogao da shvate kako da napišu bolji kod.
5. Korišćenje u procesu evaluacije: Kao dodatni komentar koji se dodaje svakom predatom radu sa ciljem isticanja određenih povećanja kvaliteta izvornog koda

Vredi napomenuti da je važno uskladiti upotrebu alata sa specifičnim ciljevima učenja kursa i pružiti smernice i podršku studentima dok uče da ga koriste.

Tri glavne prednosti upotrebe ovih alata u procesu edukacije su:

1. Poboljšanje kvaliteta koda: Statička analiza koda pomaže studentima da identifikuju potencijalne greške i nepravilnosti u svom kodu, što dovodi do poboljšanja ukupnog kvaliteta njihovog rada.
2. Povećano razumevanje najboljih praksi: Korišćenjem alata za statičku analizu koda, studenti razumeju i pridržavaju se najboljih praksi za kodiranje, kao što je poštovanje pravila pravilnog imenovanja i izbegavanje uobičajenih zamki.
3. Poboljšanje skupa veština: Korišćenjem alata i identifikovanjem problema, studenti će naučiti da pišu bolji kod i bolje razumeju principe i prakse razvoja softvera.

4. ZAKLJUČAK

Alati za statičku analizu koda sve više nalaze svoju primenu i sve više se koriste u svakodnevnom radu softver developera na različitim pozicijama. Takođe, ovi alati, još uvek ne u potpunosti, pronalaze svoju primenu i u edukaciji budućih softver developera o potencijalnim nepravilnostima koje mogu nastati u kodu, kao i o njihovom otklanjanju reinženjeringom izvornog koda.

Svakodnevno se razvijaju novi i unapređuju postojeći alati, nudeći svojim korisnicima sve kvalitetniju uslugu

stvaranjem poverenja u rezultate statičke analize i omogućavajući im da na kvalitetan način izbegnu određene nepravilnosti u kodu i učine svoje softversko rešenje kvalitetnijim. Ovo je posebno važno, iz razloga što je još uvek manja upotreba alata za statičku analizu koda uzrokovana upravo nepoverenjem korisnika u ove alate i vreme koje je potrebno utrošiti za sprovođenje ovakvih analiza nad izvornim kodom programa. Kada se govori o nepoverenju u rezultate statičke analize, oni se prvenstveno odnose na generisanje lažno pozitivnih i lažno negativnih rezultata. Razvijanjem i unapređenjem ovih alata, procenat grešaka se umanjuje i ovi alati daju bolje rezultate, samim tim se i više koriste.

U ovom radu predložena je i istaknuta važnost implementacije ovakvih alata u edukaciji inženjera softvera, kako bi se već u procesu obrazovanja budući softver inženjeri obučavali da pišu što kvalitetniji, odnosno kod sa što manjim brojem nepravilnosti.

Zaključno, korišćenje alata za statičku analizu koda i kontinuirani rad na unapređenju i održavanju kvaliteta izvornog koda obezbeđuju kvalitetnija softverska rešenja i predstavljaju korak napred u razvoju softvera, kao i budućih softver inženjera.

5. LITERATURA

- [1] E. Penttila, Improving C++ Software Quality With Static Code Analysis, Aalto: Aalto University, 2014.
- [2] I. S. A. Moller, Static program analysis, 2019.
- [3] R. B. a. M. S. M. Beller, Analyzing the state of static analysis: A large-scale of evaluation in open source software, 2016.
- [4] D. N. D. D. I. S. a. S. R. D. Stefanović, "Static code analysis tools: A systematic Literature Review," in Proceedings of 31st DAAAM International Symposium , Vienna, 2020.
- [5] J. a. B. C. West, Secure programming with static code analysis, Pearson education, 2007.
- [6] C. S. V. a. L. D. M. C Chandrapal, "Code Analysis for Software and System Security Using Open Source Tools," Information Security Journal, vol. 21, pp. 346-352, 2012.
- [7] S. S. R. O. G. C. a. M. D. P. F. Zampetti, "How Open Source Projects use Static Code," 2017.
- [8] G.-V. M. a. E.-B. J. G.-M. Javier, "Improved Metrics Handling in SonarQube for Software Quality Monitoring," Advances in Intelligent Systems and Computing, 2016.