Lehman-ovi zakoni evolucije softvera i *Open Source Software*

Miroslav Stefanović, Sonja Ristić, Đorđe Pržulj, Miloš Vukmanović
Departman za industrijsko inženjerstvo i menadžment
Fakultet tehničkih nauka
Novi Sad, Srbija
mstef@uns.ac.rs, sdristic@uns.ac.rs, przulj@uns.ac.rs, vukmanovic.milos@uns.ac.rs

Sažetak— **U radu je dat pregled nekoliko istraživanja na temu evolucije *open source software*-a, sa posebnim osvrtom na primenjivost *Lehman*-ovih zakona, ustanovljenih na osnovu podataka o evoluciji softvera koji se razvijao unutar velikih korporacija, na razvoj softvera od strane jedne velike zajednice programera. Dat je kratak pregled *Lehman*-ovih zakona, nekoliko bitnih razlika između softvera razvijenog unutar velikih kompanija i *open source software*-a, kao i prikaz rezultata nekoliko ranije sprovedenih istraživanja. Dodatno istraživanje sprovedeno je nad *Spring Framework*. Rezultati prikazanih istraživanja govore u prilog potrebe revizije *Lehman*-ovih zakona, posebno na polju njihove primenjivosti na *open source software*.**

Ključne riječi- kernel, Debian, održavanje, revizija Lehman-ovih zakona, Spring Framework

I. UVOD

U radovima koji se bave evolucijom softvera često se pored termina evolucija koristi i termin održavanje softvera. Određeni autori, poput *Pries-Heje* i *Lindwall*, ne prave razliku između ova dva termina i u svojim radovima evoluciju i održavanje koriste kao sinonime. *ISO/EIC 14764* pod održavanjem softvera podrazumeva aktivnosti vezane za softverski proizvod koji prolazi kroz modifikacije koda i povezane dokumentacije zbog problema ili potrebe za unapređenjem, dok *IEEE Standard 1219* pod održavanjem softvera podrazumeva aktivnosti vezane za softverski proizvod nakon njegove isporuke, kako bi se ispravile greške, unapredile performanse ili drugi atributi ili kako bi se proizvod prilagodio izmenjenim uslovima okoline [1].

Sa druge strane autori poput *Bennett* i *Xu* prave razliku definišući održavanje kao sve oblike podrške nakon isporuke softvera, dok se evolucija koristi za aktivnosti vezane za promene kako bi se softver usavršio, poput aktivnosti izazvanih promenama u zahtevima [2].

Iako je termin evolucije softvera vezan za 1965. godinu i *Mark Helpern*-a, koji ga je koristio kako bi opisao rast softvera, ono što je zajedničko većini radova na ovu temu je da otpočinju prikazom *Lehman*-ovih zakona evolucije softvera.

II. *LEHMAN*-OVI ZAKONI EVOLUCIJE SOFTVERA

Belady i *Lehman* razlikuju tri grupe softvera i to *P-type*, *S-type* i *E-type*. Za *P-type* softver, ne postoji održavanje, jer se radi o softveru koji se koristi jednokratno i potom odbacuje. Kod *S-type* softvera rast je manji od 10% godišnje, što se smatra održavanjem, dok u slučaju *E-type* softvera rast je veći od 10% što se smatra evolucijom [3].

Lehman definiše *E-type* softver kao softver čija prihvaćenost zavisi od percepcije, ocene i stepena zadovoljstva odgovarajućih zainteresovanih strana, kao softver koji se koristi za rešavanje problema i koji ima primenu u realnom svetu [5], te kao primere navodi računarske operativne sisteme, sisteme kontrole leta i berzanske sisteme [4].

Lehman definiše osam zakona evolucije softvera i to:

1. Kontinualne promene – Softver koji se koristi i koji kroz svoju implementaciju predstavlja neku drugu stvarnost prolazi kroz kontinualne promene ili progresivno postaje sve manje koristan. Promena ili proces degradacije nastavlja se dok se ne odluči da je ekonomičnije zameniti sistem novom verzijom.

2. Povećanje kompleksnosti – Kako se softver kontinualno menja, njegova kompleksnost, kao posledica degradacije strukture, raste ukoliko se ne preduzmu koraci da se ona održi na istom nivou ili umanji.

3. Fundamentalni zakon evolucije softvera – Softverski sistemi pokazuju pravilnosti u ponašanju i trendovima koje možemo da merimo i predviđamo.

4. Očuvanje organizacione stabilnosti – U nekom trenutku resursi i izlazi dostižu optimalan nivo i dodavanje resursa ne menja značajno izlaz.

5. Očuvanje upoznatosti – Tokom aktivnog životnog veka softvera, sadržaj verzije (broj izmena, dodaci, brisanja) je nepromenjiv ukoliko se posmatraju uzastopne verzije [4].

6. Kontinualni rast – Funkcionalni sadržaj softvera mora kontinualno rasti kako bi se zadržalo zadovoljstvo korisnika tokom sopstvenog životnog veka.

7. Opadanje kvaliteta – Smatraće se da kvalitet *E-type* softvera opada ukoliko se rigorozno ne održava i ne menja u skladu sa promenama u okruženju.

8. Sistem povratne sprege – Proces programiranja *E-type* softvera je višeslojni sa povratnom spregom i petljama i mora se tretirati kao takav da bi mogao uspešno da se modifikuje i unapređuje.

Studije koje su sproveli *Lehman* i kolege pružaju podatke zasnovane na posmatranju pet softverskih sistema: dva operativna sistema (*IBM OS 360*, *ICL VME Kernel*), jednog finansijskog sistema (*Logica FW*), jednog telekomunikacionog sistema (*Lucent*) i jednog odbrambenog sistema (*Matra BAE Dynamics*) [6]. Ono što je zajedničko za sve posmatrane sisteme je da se radi o velikim softverskim sistemima, da su razvijani i održavani u okruženju velikih korporacija, da su korisnici ovih sistema, vrlo verovatno, velike kompanije i i da nijedan nije kreiran kao softverski sistem namenjen maloprodaji.

III. *OPEN SOURCE SOFTWARE*

Postavlja se pitanje, koliko će se *Lehman*-ovi zakoni, za čiju je postavku, kao empirijska osnova, poslužio razvoj i održavanje softverskog sistema u okruženju velikih korporacija, biti primenjivi u slučaju *open source software*-a.

Kao ilustracija može da posluži nekoliko karakteristika razvoja *open source software*-a koje potencijalno nisu kompatibilne sa komercijalnim razvojem:

– Zahtevi – Komercijalni projekti obično posvećuju značajan trud prikupljanju i analizi zahteva, tokom procesa koji često obuhvata nekoliko disciplina, poput marketinga, upravljanja proizvodom i softverskog inženjerstva. *Open source* projekti se, sa druge strane, oslanjaju se na korisnike koji ujedno i razvijaju softver kako bi napravili funkcionalnosti koje su im potrebne i kako bi ispravili greške.

– Dodela zadataka – U kompanijama, programeri su raspoređeni od strane menadžmenta na projekte i zadatke, obično se teži da zadaci koji se daju programeru odgovaraju njegovim veštinama, ali lični izbor programera je obično ograničen. U *open source* projektima programeri obično biraju na čemu žele da rade. Generalno gledano, prave nešto što je potrebno njima kao korisnicima softvera.

– Arhitektura softvera – Arhitektura *open source software*-a zahteva modularniju arhitekturu u poređenju sa komercijalnim projektima. Opšte je prihvaćeno da je struktura organizacije jedna od kritičnih determinanti strukture programskog koda. Ono što nije dovoljno jasno je u kojoj meri bi arhitektura kreirana za razvoj u komercijalnom okruženju bila pogodna za saradnju koja je neophodna u *open source* projektima.

– Kompatibilnost alata – Većina *open source* projekta postoji nezavisno jedna od drugih ili koegzistiraju na *hosting* servisima drugih projekata koji su prihvatili isti skup alata. U komercijalnom okruženju, situacija je generalno nešto složenija. Koristi se širi opseg alata i nije jasno kako bi se podržala *open source* praksa u ovako heterogenom okruženju.

– Softverski procesi – U slučaju komercijalnih projekata često imamo prolazne tačke u kojima se projekat ocenjuje. Ova praksa se često smatra od kritične važnosti kako bi se osigurao kvalitet softvera. Sa druge strane, kada se govori o *open source* procesima, retko postoje formalni procesi, a odluku o tome šta će se naći u narednom izdanju donosi "blagonakloni diktator" ili mala grupa dokazanih osvedočenih stručnjaka. Ova dva pristupa mogu biti nekompatibilna.

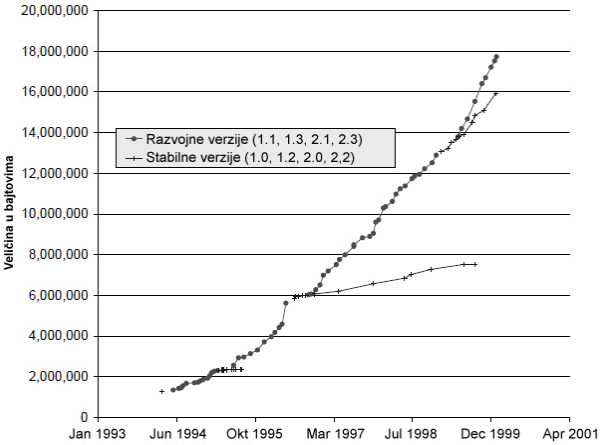
Motivacija – Komercijalni projekti su zasnovani na profitu, dok u slučaju *open source* projekata motivi mogu biti različiti npr. želja da se nauče nove veštine, želja da se programu dodaju mogućnosti koje su programeru potrebne, filozofsko verovanje u učestvovanju u kreiranju nečega za opšte dobro, sloboda da se kreira šta se želi, a nekada i politički stavovi komercijalnim projektima [8].

Imajući u vidu navedene razlike, nije iznenađujuće da je veliki broj autora posebnu pažnju posvetio evoluciji *open source software*-a i istraživanju u kolikoj meri ta evolucija zadovoljava *Lehman*-ove zakone. Prikaz nekoliko radova na ovu temu dat je u narednom poglavlju.

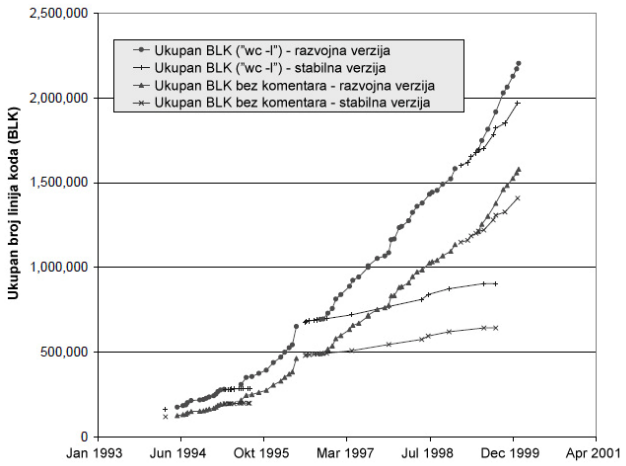
IV. EVOLUCIJA OPEN SOURCE SOFTWARE-A

Godfrey i *Qiang* su pošli od pretpostavke da će zbog činjenice da je razvoj *open source software*-a manje struktuiran i planiran sa manje detalja nego tradicionalni razvoj softvera unutar komercijalnih kompanije, razvoj *open source software* biti u skladu sa *Lehman*-ovim zakonima, odnosno da će sa rastom sistema, brzina rasta opadati.

Za potrebe istraživanja, autori su izabrali da posmatraju Linux, operativni sistem koji je razvio *Linus Torvalds*, ali na kojem je tokom godina radilo stotine programera. Istraživanje je sprovedeno nad 96 verzija kernela, od toga 34 stabilne i 62 razvojne verzije. Merili su veličinu distribucije kao veličinu *tar* datoteke kompresovane uz pomoću *gzip*-a, brojane su linije koda uz pomoć "*wc -l*" komande, a potom koristili *awk* skript kako bi se filtrirale prazne linije i komentari, a korišćen je program *exuberant ctags* kako bi se utvrdio broj globalnih funkcija, varijabli i makroa. Rezultati istraživanja *Godfrey* i *Qiang* vezanih za veličine verzija kernela u bajtovima i broju linija koda prikazani su na slikama 1. i 2.



Slika 1. Veličine verzija kernela u bajtovima



Slika 2. Veličine verzija kernela po broju linija koda

Zaključak do kog su autori došli, nakon praćenja šestogodišnjeg rasta Linux-a u nekoliko kategorija, bio je da je na sistemskom nivou, rast bio superlinearan i da taj rezultat nije bio očekivan imajući u vidu:

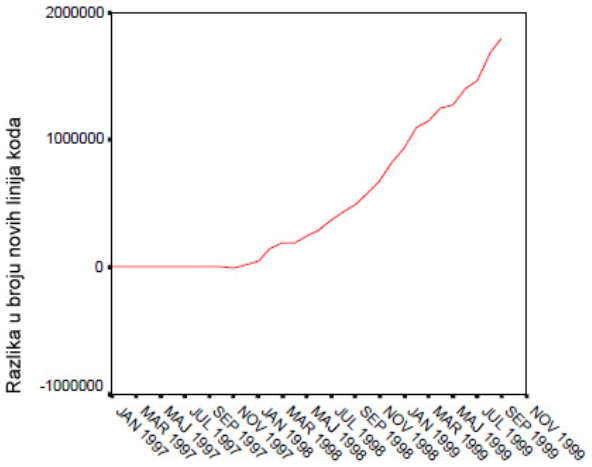
a) veličinu (preko dva miliona linija koda, uključujući prazne linije i komentare),

b) razvojni model (visok stepen saradnje, programeri na različitim geografskim lokacijama, programeri koji rade besplatno),

c) ranija istraživanja koja su sugerisala da se rast softvera usporava sa porastom sistema [9].

Koch i *Schneider* analiziraju razvoj *GNOME - GNU Network Object Model Environment*, koji predstavlja *open source* projekat razvoja desktop okruženja.

Mereći rast sistema kroz broj linija koda rezultati njihovog istraživanja takođe pokazuju da sa rastom sistema nije došlo do usporavanja rasta softvera, kao i da je broj linija koda između dve verzije rastao od verzije do verzije [10]. Na slici 3. prikazan je rezultat istraživanja *Koch* i *Schneider* i razlika u broju novih linija koda na mesečnom nivou.



Slika 3. Razlika u broju novih linija koda na mesečnom novou

West i *O'Mahony* bave se posebnim slučajem evolucije *open source software*-a koji nastaje na osnovu softvera razvijenog unutar neke organizacije i posmatraju *VistA*, zdravstveni informacioni sistem koji je razvio *United States Department of Veterans Affairs*, te zaključuju da zreli projekti kojima upravlja zajednica evoluiraju kroz proces sporog rasta, da bi potom došli do faze ubrzanog rasta [11].

Već pominjani *Godfrey* i *Qiang* iznose i rezultate rasta još nekoliko *open source* projekata, odnosno *VIM*, *Fetchmail* i *GCC* i postavljaju hipotezu da rast *open source software*-a ima razvojnu dinamiku koja se razlikuje od većine industrijskog softvera, koja mu omogućava da raste po superlinearnom modelu duži vremenski period i predlažu dodatna istraživanja na ovu temu [12].

Na osnovu ovih i drugih istraživanja, u kojima se veličina sistema merila kroz broj linija *source code*-a (*SLOC - Source lines of code*), *Scacchi* dovodi u pitanje svih osam *Lehman*-ovih zakona evolucije softvera ističući da u slučaju:

1. Kontinualne promene – postoje podaci koji podržavaju tezu da je neophodno da se program kontinualno menja inače će postati sve manje zadovoljavajući, ali postavlja se pitanje kako meriti "zadovoljstvo korisnika" na osnovu podataka o rastu veličine sistema.

2. Povećanje kompleksnosti – postoje podaci koji potvrđuju važenje ovog zakona, ali se postavlja pitanje kako meriti "kompleksnost" na osnovu podataka o rastu veličine sistema.

3. Fundamentalni zakon evolucije softvera – postavlja pitanje zašto i sa kojim ciljem softver mora biti samoregulišući, kako funkcioniše samoregulacija i koje su joj ulazne veličine i koja funkcija transformiše te ulazne veličine u proces samoregulacije.

4. Očuvanje organizacione stabilnosti – primeri razvoja GNOME-a i Debian-a koji pokazuju superlinearan rast sistema sa povećanjem osoblja su u suprotnosti sa ovim zakonom.

5. Očuvanje upoznatosti – u slučaju *open source software*-a postoje primeri u kojima je veličina sadržaja u uzastopnim verzijama nepromenjiva, ali i slučajeva u kojima veličina sadržaja raste ili opada.

6. Kontinualni rast – ne postoje empirijske osnove koje pružaju podatke kako se procenjuje zadovoljstvo korisnika sistemom ili kako se menja u zavisnosti od verzije do verzije u odnosu na čitav životni vek sistema.

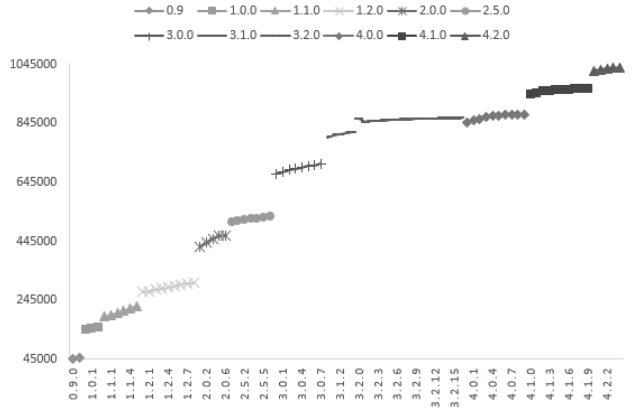
7. Opadanje kvaliteta – Kako će se procenjivati kvalitet softvera i koji podaci će se meriti kako bi se odredio trend? Da li je broj problema ili zahteva za modifikacijom indikator kvaliteta softvera ili je to nešto drugo?

8. Sistem povratne sprege – Od ranih dana softverskog inženjerstva, procesi su se dekomponovali na potprocese, razvijani su ciklično (npr. model vodopada) i uključivali su ljude i alate kako bi se sprovodile aktivnosti softverskog inženjerstva. Postavlja se pitanje da li je ovaj zakon jednostavno tautološki recept onoga što bi trebalo da predstavlja apstraktni model procesa evolucije softvera.

Analiza zakona evolucije softvera, zasnovana na značajnoj količini empirijskih podataka, pokazuje da zakoni možda ne mogu da objasne sve pojave koji su prikazane u ovim studijama. Stoga se čini potrebnim videti kako se zakoni evolucije softvera mogu korigovati kako bi adekvatnije povezali teoriju, praksu i empirijske podatke [6].

V. *SPRING FRAMEWORK* ISTRAŽIVANJE

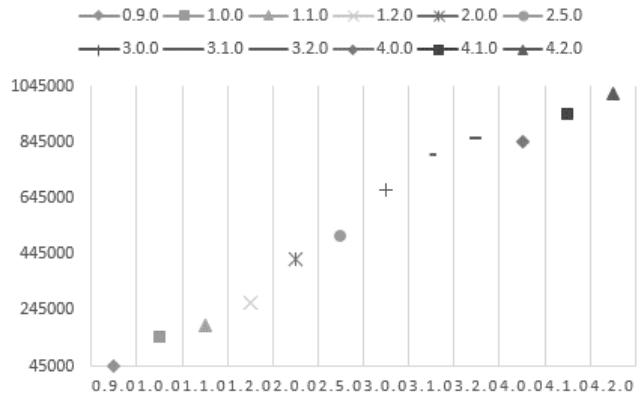
Polazeći od *Lehman*-ovih zakona, ali i već pominjanih istraživanja različitih *open source* programa obavljeno je istraživanje *Spring Framework*-a. Koristeći *cloc* program, izvršeno je prebrojavanje linija koda različitih verzija *Spring Framework*-a, od verzije 0.9 do aktuelne verzije 4.2.4. Rezultati istraživanja prikazani su na slici 4.



Slika 4. Veličine verzija *Spring Framework*-a po broju linija koda

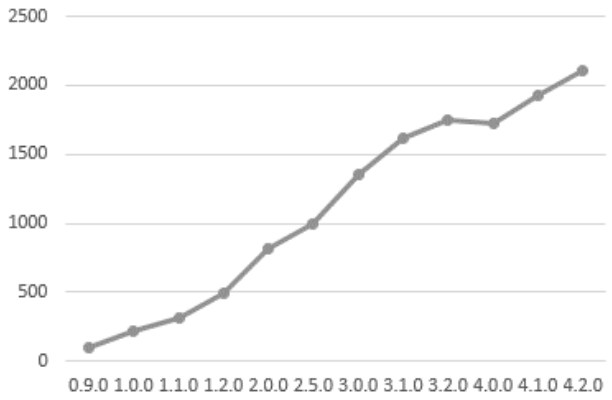
Na osnovu rezultata prikazanih na slici 4. može se zaključiti da nije došlo do usporavanja rasta sistema, koji bi se očekivao u skladu sa *Lehman*-ovim zakonima.

Takođe, posmatranjem slike 5 u kojoj su prikazane samo početne verzije svake grane Spring Framework-a, može se videti da od verzije 1.0.* zaključno sa verzijom 3.2.* rast bio superlinearan.



Slika 5. Veličine prvih verzija grana *Spring Framework*-a po broju linija koda

Pored toga ukoliko se koristi metoda koju su koristili i *Lehman* i kolege, odnosno kreira grafik koji će na X-osi prikazivati verzije, a na Y-osi procentualni rast broja linija koda, u slučaju *Spring Framework*-a dobijaju se rezultati prikazani na slici 6.



Slika 6. Procentualan rast prvih verzija grana *Spring Framework*-a po broju linija koda

Nasuprot očekivanjima na osnovu *Lehman*-ovih istraživanja da bi rast u ovom slučaju trebao da bude linearan ili inverzna kvadratna funkcija, ponovo se može primetiti blagi superlinearan trend.

Bitno je napomenuti da tokom razmatranja nisu uzimane u obzir verzije 0.9, kao i verzije 4.* iz razloga što je na razvoju verzije 0.9 radila samo jedna osoba. Dok su verzije 4.* donele značajne promene, te da 3 postojeće grane ne predstavljaju dovoljno veliki uzorak za izvođenje bilo kakvih zaključaka.

Analiza zakona evolucije softvera, zasnovana na značajnoj količini empirijskih podataka, pokazuje da zakoni možda ne mogu da objasne sve pojave koji su prikazane u ovim studijama. Stoga se čini potrebnim videti kako se zakoni evolucije softvera mogu korigovati kako bi adekvatnije povezali teoriju, praksu i empirijske podatke

ZAKLJUČAK

U radu su prikazani su rezultati nekoliko istraživanja koja su se bavila evolucijom *open source software*-a. Ono što je zajedničko za rezultate tih istraživanja je zaključak da je potrebno izvršiti dodatna istraživanja i reviziju *Lehman*-ovih zakona.

Zakoni i teorija evolucije softvera koju je predložio *Lehman* i kolege priznate su kao veliki doprinos polju softverskog inženjerstva i disciplini računarskih nauka. Ovi zakoni su generalno davali moguće objašnjenje kako softverski sistemi evoluiraju tokom svog životnog veka. Istraživalo se u periodu dužem od trideset godina, tako da je njihova istrajnost značajno dostignuće. Sa razvojem tehnologije, procesa i praksi razvoja i održavanja softvera, posebno od kraja devedesetih kada se pojavio veliki broj *open source* projekata, postalo je jasno da se zakoni i teorije polako degradiraju. Zakoni evolucije softvera ne pružaju dovoljno dobro objašnjenje evolucije *open source software*-a [6].

Na osnovu prikazanih rezultata istraživanja, zaključak je da su najznačajniji razlozi zbog kojih zakoni evolucije softvera ne pružaju dobro objašnjenje evolucije *open source software*-a oni koji su navedeni su u III poglavlju. Naime, početne tačke u razvoju softvera unutar jedne velike kompanije, u odnosu na *open source* projekat, se najčešće značajno razlikuju. Velika kompanije bi najčešće trebalo da otkrije priliku na tržištu i da

na osnovu toga razvije softver, uz poštovanje budžeta, planova, organizacione hijerarhije, vremenskog okvira, da bi se potom pojavio proizvod koji zadovoljava unapred definisane potrebe, iza koga kompanija može da stane sa svojim imenom i koji kompaniji treba da donese dobit.

Sa druge strane u razvoju *open source software*-a često se javlja pojedinac koji nužno ne otkriva tržišnu nišu, već ima ličnu potrebu, koji razvija softver bez želje da mu donese finansijsku dobit i koji neretko kao rešenje nudi softver koji nije u potpunosti funkcionalan, koji je alfa verzija, ali koji želi što ranije da podeli sa zajednicom. Samim tim početne tačke evolucije i održavanja su im različite.

Ono o čemu bi takođe trebalo razmišljati je i da li se evolucija jednog softvera može porediti sa evolucijom bilo kog drugog softvera ili bi bilo svrsishodnije porediti evolucije softvera koji rešavaju isti problem ili pripadaju istim kategorijama? Da li bi bilo bolje porediti evoluciju Windows-a sa Linux-om ili sa Nano editorom teksta?

Takođe, u ogromnoj većini prikazanih radova kao mera rasta sistema koriste se linije koda, ali kao što je u radu i pominjano, da li je to adekvatna mera evolucije softvera? Procesom refaktoringa može se smanjiti broj linija koda, a zadržati prethodna funkcionalnost, pa se razlika u broju linija koda može "iskoristiti" za dodavanje novih funkcionalnosti. Na taj način se može doći do proizvoda koji će imati manji broj linija koda, ali će evoluirati u procesu zadovoljenja potreba korisnika.

Iako metodologija studijsko istraživačkog rada podrazumeva da se pre novih istraživanja prouče ranije korišćene istraživačke metoda, stiče se utisak da bi u ovoj oblasti možda trebalo potražiti nove metode za istraživanje evolucije softvera kako bi one poslužile kao osnova za pokušaj ustanovljavanja nekih novih zakonitosti.

LITERATURA

[1] J. Pries-Heje and D. Lindwall, "Evolution or maintenance of quality software: an exploratory interview study in nine Swedish organisation", Proceedings of IRIS. 2006

[2] K. H. Bennett and X. Jie, "Software services and software maintenance" Software maintenance and reengineering, 2003. Proceedings of Seventh European Conference on IEEE, 2003.

[3] H. M. Sneed and P. Brossler, "Critical success factors in software maintenance: a case study". Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on. IEEE, 2003.

[4] M. M. Lehman, "Programs, life cycles, and laws of software evolution.", Proceedings of the IEEE 68.9 1980.

[5] M. M. Lehman, "Laws of software evolution revisited", Software process technology 108-124. Springer Berlin Heidelberg, 1996.

[6] W. Scacchi, "Understanding open source software rvolution: applying, breaking, and rethinking the laws of software evolution", Software Evolution and Feedback: Theory and Practice 181-206, 2006.

[7] E. Carmel, and B. Shirley, "A process model for packaged software development." Engineering Management, IEEE Transactions on 42.1, 50-61, 1995.

[8] V. K. Gurbani, A. Garvert and J. D. Herbsleb, "A case study of a corporate open source development model", Proceedings of the 28th international conference on Software engineering. ACM, 2006..

[9] M. W. Godfrey and T. Qiang, "Evolution in open source software: A case study" Software Maintenance, 2000. Proceedings. International Conference on. IEEE, 2000.

[10] S. Koch and G. Schneider, "Effort, co-operation and co- ordination in an open source software project: GNOME" Information Systems Journal 12.1 (2002): 27-42.

[11] J. West and S. O'Mahony, "Contrasting community building in sponsored and community founded open source projects", Research Policy 32.7 1217-1241, 2003.

[12] M. W. Godfrey and T. Qiang, "Growth, evolution, and structural change in open source software", Proceedings of the 4th international workshop on principles of software evolution. ACM, 2001.

ABSTRACT

The paper gives an overview of several studies on the evolution of open source software, with a special focus on the applicability of Lehman's laws of software evolution, established on the basis of data on the evolution of the software being developed within large corporations, on software developed by a large community of developers. Paper gives a brief review of Lehman's laws, several important differences between the software developed within large companies and open source software and presents results of several earlier studies on this subject. Additional research was conducted on Spring Framework. Presented results show that there's a need for auditing Lehman's laws, especially regarding their applicability to the open source software.

LEHMAN'S LAWS OF SOFTWARE EVOLUTION AND OPEN SOURCE SOFTWARE

Miroslav Stefanovic, Sonja Ristic, Djordje Przulj, Milos Vukmanovic