Iskanje zakonov oblikovanja programske opreme z metodami znanosti o kompleksnosti

Peter Kokol[1,2], Milan Zorman[1,2], Mitja Lenič[1], Alojz Tapajner[1]

[1] Laboratorij za načrtovanje sistemov, FERI, Univerza v Mariboru, Smetanova 17, Maribor

[2] Center za interdisciplinarne in multidisciplinarne raziskave in študije Univerze v Mariboru, Krekova 2, Maribor

Povzetek

Z dramatičnim povečanjem kompleksnosti in velikosti programske opreme se veča tudi verjetnost pojava napak in kriznih situacij pri njeni uporabi. Žal konvencionalne metode za merjenje in kontrolo kvalitete niso dovolj uspešne, zato je potrebno uporabiti nekonvencionalne pristope, kot so teorija kompleksnosti in adaptivnih sistemov, fizikalno podprte metrike, evolucijske ekonomsko/tržne teorije ter inteligentno analizo podatkov, da bi dosegli temeljno razumevanje in večjo kakovost programske opreme in samega procesa njenega oblikovanja. V članku bomo predstavili prve rezultate evropskega projekta SQUFOL, v katerem smo za merjenje kvalitete programske opreme uporabili in nadgradili tehnike, ki so se uveljavile v zgoraj navedenih področjih.

Abstract

SOFTWARE QUALITY FOUNDED ON DESIGN LAWS AND SCIENCE OF COMPLEXITY

As the size and complexity of software systems is growing dramatically the possibility of crises from failure increases. Conventional methods for measuring and controlling quality are not successful enough, therefore it is our objective to use the science of complexity, the theory of chaos, fractals, traditionally 'physically based' methodologies, theory of complex adaptive systems, biologically and evolutionary inspired economic/market theories, intelligent data analysis and data mining to gain a fundamental understanding of the software process and the software products and to improve them. The aim of the recent paper is to present the first results of the European project SQUFOL.

1 Uvod

Razvoj in oblikovanje programske opreme je kompleksen proces [1, 4, 9, 14]. Še nedavno tega smo nanju gledali kot na umetnost, pa tudi danes še nista dosegla vseh zahtev popolnoma inženirske discipline. V zadnjih desetletjih, še posebej v zadnjih nekaj letih sta velikost in kompleksnost programske opreme dramatično narastli. Sistemi z več sto milijoni vrstic programskega koda niso več nobena posebnost, poleg tega zahteve po kompleksnih programskih sistemih naraščajo hitreje kot naše zmožnosti, da takšne sisteme načrtujemo, gradimo in vzdržujemo. S porastom naših zahtev in s porastom naše odvisnosti od računalnikov naraščajo možnosti, da bo zaradi izpada računalniškega sistema prišlo do različnih kriznih situacij, ki lahko obsegajo manjše neprijetnosti, finančne nepravilnosti in izgube, v skrajnih primerih pa celo izgube človeških življenj. Torej je jasno, da zanesljivost programske opreme ne zadeva več samo načrtovalcev programske opreme in računalniških strokovnjakov, ampak tudi širšo družbo kot celoto. Zato je konsistentno in ekonomično doseganje najvišjega nivoja kvalitete programske opreme ključnega pomena, vendar zato potrebujemo zakone razvoja programske opreme, ki jih žal še nismo odkrili. Popolnoma jasno je, da lahko zakone oblikovanja odkrijemo samo z uporabo empiričnega znanstveno/sistemskega pristopa, za kar pa potrebujemo uspešne metrike programske opreme. Žal konvencionalne metode za merjenje programske opreme niso dovolj uspešne, zato je potrebno uporabiti nekonvencionalne pristope.

1.1 Oblikovanje programske opreme je kompleksen adaptiven sistem

Tako kot kompleksni sistemi v ekonomiji in menedžmentu ima tudi razvoj programske opreme (razvoj tu zajema celoten življenjski cikel, od idej in opisov potreb do uporabe in vzdrževanja) podobne sistemske lastnosti [3, 11]: je kompleksen, dinamičen, nelinearen in adaptiven. Posledično je cilj evropskega projekta SQUFOL [5] (Software Quality Founded on Design Laws) odkriti temeljne zakone programske opreme,

njene evolucije, kvalitete in razvojnega procesa z uporabo pristopa, ki združuje teorijo kompleksnosti, razne sistemske teorije, teorijo kaosa ter fraktalov, tradicionalne fizikalno podprte metodologije, teorijo kompleksnih adaptivnih sistemov, evolucijske tržnoekonomske teorije, inteligentno analizo podatkov, podatkovno rudarjenje ter druge metode, ki še niso priznane s strani raziskovalcev oblikovanja programske opreme. Poznavanje temeljnih zakonov bo omogočalo povečanje znanja o kvaliteti programske opreme in njenem razvojnem procesu, oblikovanje enostavnega in razumljivega procesa oblikovanja in kot posledico razvoj visoko kvalitetne programske opreme, ki ne bo povzročala prej omenjenih težav. V članku bomo predstavili enega izmed prvih rezultatov projekta SQUFOL – zakonitost, da se oblikovanje programske opreme obnaša podobno kot dinamični kaotični sistemi, generirani s t. i. logistično enačbo.

2 Težave pri oblikovanju programske opreme

Porast zahtev po kvalitetni programski opremi in nezmožnost izpolnjevanja le-te sta pripeljala do dobro znane 'krize' programske opreme. Njeni vzroki in posledice so predmet mnogih razprav v literaturi [10], naše mnenje pa je, da je glavni vzrok te krize nezmožnost poiskati in razumeti temeljne zakone programske opreme, njihovo uporabo, kvaliteto in njihov proces razvoja. Menimo, da je osnova za reševanje krize odkrivanje temeljev in zakonov oblikovanja, ki v katerikoli znanstveni disciplini temelji na zmožnosti merjenja. Žal na področju razvoja programske opreme ni dovolj uspešnih programskih metrik, za kar so krivi naslednji vzroki [7]:

- Metrike so odvisne od jezika in oblike: za različne programske jezike (izvorna koda), objektne kode itd., je potrebno uporabiti različne metrike. Nadalje je programska oprema lahko predstavljena v različnih oblikah, kot so sistemski koncepti, zahteve, specifikacije, razvojna dokumentacija, uporabniški vmesniki, zbirke pomoči itd. Še dodatno so lahko vse te predstavitve predstavljene v različnih oblikah, kot so tekst v naravnem jeziku, grafična predstavitev, simbolni zapis, tekst, zapisan s formalnimi jeziki ipd. Za vsako od teh različnih predstavitev moramo uporabiti svojo metriko, kar pomeni, da nismo zmožni primerjati enakih izdelkov v različnih oblikah ali predstavitvah ter tako ne moremo slediti spremembam kompleksnosti skozi razvojne faze.
- Izhod tradicionalne metrike je številka, ki nima nikakršnega 'fizikalnega' pomena in enote (ne vemo, kaj merimo), niti nima znanih kritičnih vrednosti, ki bi povedale, kdaj je izmerjena vrednost majhna ali velika.
- Relacije metrike – programska oprema (za izdelke in procese) so redko znane, zato so tudi takšne metrike slaba podlaga za postavitev temeljnih zakonitosti. To je posledica predvsem dveh dejstev:
  - Metrika ni bila izpeljana na podlagi teorije, ampak na podlagi pragmatičnih kratkoročnih ciljev.
  - Implementacija učinkovitih metrik je pogosto izvedena brez popolnega razumevanja metrike same ali je le-ta osnovana na minimalni množici »poceni« postopkov za zbiranje podatkov.

3 Projekt SQUFOL

Da bi se izognili gornjim problemom, smo v projektu SQUFOL uporabili in nadgradili tehnike, ki so se uveljavile v okviru vse bolj priljubljene znanosti o kompleksnih sistemih:

1. Na program gledamo kot na zbirko simbolov [6, 12, 13] in tako nanj apliciramo tehnike, ki ta program pretvorijo v 'signale'. Na teh signalih uporabimo tehnike analize, ki izhajajo s področja fizike. Primeri teh analiz so izračun informacijske vsebnosti, daljnosežnih korelacij, entropije ipd.
2. Faze razvoja programske opreme lahko primerjamo s fazami iterativnih map [2].
3. Na razvoj programske opreme lahko gledamo kot na trženje idej. Vsaka ideja ima svoje gene/meme in tekmuje za vire. Najboljše ideje preživijo, se med seboj križajo in mutirajo. Zato lahko procese analiziramo z uporabo evolucijsko-bioloških ekonomsko/tržnih teorij.
4. S celovitim pristopom in sekvenčnimi študijami lahko izmerimo vsa dejstva o programski opremi in njenem razvoju.
5. Z uporabo inteligentne analize podatkov, podatkovnega rudarjenja in ekstrakcije znanja lahko odkrijemo relacije, skrito znanje in zakone oblikovanja programske opreme.

V nadaljevanju članka bomo predstavili prvi dve ideji.

4 Inverzne bifurkacije, logistična funkcija in oblikovanje programske opreme

Programsko opremo običajno oblikujemo z iterativnim procesom [2] – računalniški program se razvija

skozi različne verzije (slika 1). Na začetku procesa imamo veliko različnih idej, kako pravilno ali nepravilno predstaviti rešitev. Kreativnost je na vrhuncu, entropija je velika, vsebnost informacije je nizka. V naslednjih korakih se razvijejo pravilne in optimalne rešitve. V zadnji fazi se po navadi ukvarjamo z eno samo idejo (verjetno zelo kompleksno). Entropija in kreativnost sta nizki (imamo eno rešitev), medtem ko se informacijska vsebnost poveča.

Za modeliranje gornjega procesa lahko uporabimo zelo popularen in zanimiv koncept iz teorije kompleksnih adaptivnih sistemov – to je logistično funkcijo, ki ponazarja dinamiko biološke populacije. Zanjo predpostavljamo, da generira podobno obnašanje, kot je obnašanje procesa oblikovanja programske opreme, vendar v obratnem vrstnem redu. Logistična funkcija je definirana kot

$$X_{n+1} = \pi X_n (1 - X_n)$$

kjer X v originalu predstavlja normirano število osebkov populacije, v primeru oblikovanja programske opreme pa informacijsko vsebnost ideje.

Dinamika logistične funkcije vsebuje tri faze: kaos, bifurkacijo in normalno fazo. Bifurkacija predstavlja podvojitev števila atraktorjev, v našem inverznem procesu pa združevanje idej. Če naredimo preslikavo med logistično funkcijo in procesom razvoja programske opreme, lahko faze logistične funkcije preprosto identificiramo s kontrolnim parametrom p in tako ugotovimo, v kateri fazi oblikovanja smo. Seveda za to potrebujemo metriko informacijske vsebnosti idej (v našem primeru je ideja predstavljena kot računalniški program, psevdokod, algoritem ipd.) in jo opisujemo v naslednjih razdelkih.

5 Fraktalna programska metrika α

Fraktalna metrika α [6] temelji na izračunu daljnosežnih korelacij. Metriko smo razširili iz originalne Schenklove metode za pisana besedila na metodo znakov. Podobno kot na pisano besedilo lahko gledamo na računalniški program ali algoritem kot na niz znakov: črk, števk, ločil. Z uporabo kodne tabele pretvorimo vsak znak v pripadajoče binarno zaporedje; program tako preslikamo v model dvodimenzionalnega Brownovega gibanja, ki je osnova za izračun regresijske funkcije $F(l)$:

$$F^2(l) \equiv \overline{[\Delta y(l, l_0)]^2} - \left[\overline{\Delta y(l, l_0)}\right]^2$$

Prvi del zgornje enačbe predstavlja povprečje kvadratov razlik med dvema točkama v modelu Brownovega gibanja in drugi del kvadrat povprečij razlik, kjer razliko izračunamo s spodnjo enačbo:

$$\Delta y(l, l_0) \equiv y(l_0 + l) - y(l_0)$$

Regresijska funkcija $F(l)$ loči med dvema tipoma obnašanja:

- če je podano zaporedje znakov nekorelirano ali so prisotne lokalne korelacije, ki segajo do nekega karakterističnega ranga (npr. Markovske verige ali simbolično zaporedje, tovorjeno iz regularnih gramatik), potem je

$$F(l) \approx l^{0.5}$$

- če ne obstaja karakteristična dolžina in so korelacije "neskončne", potem je skalirna lastnost $F(l)$ opisana s potenčnim zakonom

$$F(l) \approx l^{\alpha} \text{ in } \alpha \neq 0.5$$

Koeficient α izračunamo po metodi najmanjših kvadratov kot linearno predstavitev točk na dvojni logaritemski skali $F(l)$, l in predstavlja kompleksnost računalniškega programa.

Metrika a meri vsebnost informacije programa. Večje kot je odstopanje α od 0.5, večja je informacijska vsebnost, manjša entropija in manjša kreativnost. Kot posledica vrednosti α blizu 0.5 pomenijo faze blizu kaotičnega obnašanja. To domnevo smo podkrepili s primerjavo realnih in naključnih programov z enostavnimi statističnimi metodami. Naleteli smo na razlike v vrednostih α med realnimi in naključnimi programi. Hipotezo smo preverili na podlagi dejstva, da so naključni programi (vsaj v našem poskusu) sintaktično pravilni, vendar brez pomena, kar pomeni, da imajo le sintaktično strukturo, ne pa tudi semantične; njihova struktura se torej bistveno razlikuje od strukture običajnih programov.

Da bi še bolj izpostavili lastnost metrike α, smo uvedli normalizirano obliko α:

$$\alpha_{nor} = 2|\alpha - 0.5|$$

V tej obliki ležijo vrednosti α_{nor} med 0 in 1, kjer 0 predstavlja maksimalno entropijo. Za izračun faze procesa razvoja programske opreme moramo izračunati vrednost π po naslednji enačbi:

$$\pi = \frac{X_{n+1}}{X_n(1-X_n)}$$

Bifurkacije v logistični funkciji predstavljajo število idej, medtem ko α_{nor} predstavlja inverzno relacijo števila idej (0 – veliko idej, 1 – zelo malo idej). Za izračun π potrebujemo tako inverzno relacijo normaliziranih idej

$$I = 1 - \alpha_{nor}$$

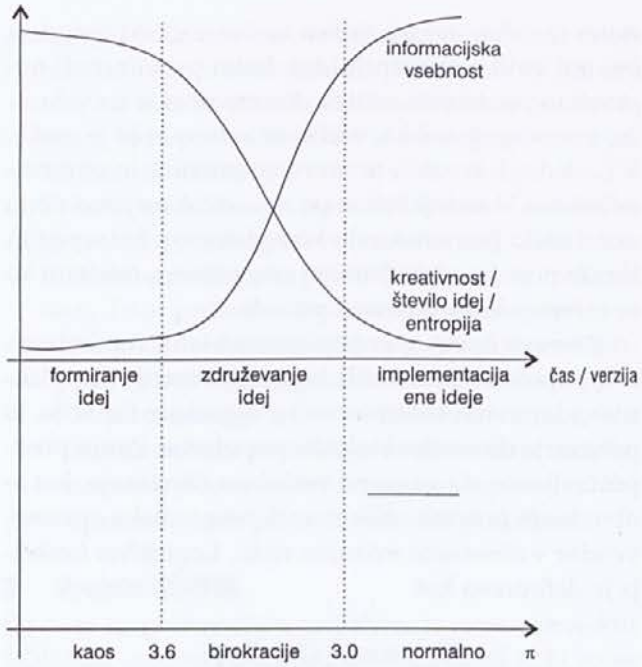
Glede na trend α vrednosti ločimo med sedmimi različnimi fazami, prikazanimi v tabeli 1.

5 Primer

Predstavljeno teorijo smo preizkusili na realnem projektu razvoja programske opreme. Prvi pogoj za izbiro projekta je bilo čim večje število verzij projekta skozi čim daljše obdobje. Pri iskanju primernega projekta smo se omejili na projekte z odprto kodo, saj za izvajanje opisane analize potrebujemo dostop do vseh verzij izvorne kode. V ta namen smo izbrali projekt Pure-FTPd 0 [15], ki se še aktivno razvija in je po navedbah avtorjev v stabilni/produkcijski fazi. Projekt je sestavljen iz več modulov. Projekt se ukvarja z razvojem storitve za prenos datotek prek FTP protokola (file transfer protocol).

V članku podrobneje prikazujemo analizo dveh modulov, ki sta se tekom oblikovanja najbolj spreminjala in obenem predstavljata jedro projekta. Prvi modul je razpoznavalnik (45 verzij), ki je vmesnik med samo storitvijo in uporabnikom/aplikacijo, ki to storitev uporablja. Drugi modul je storitev sama (FTPd) (310 verzij).

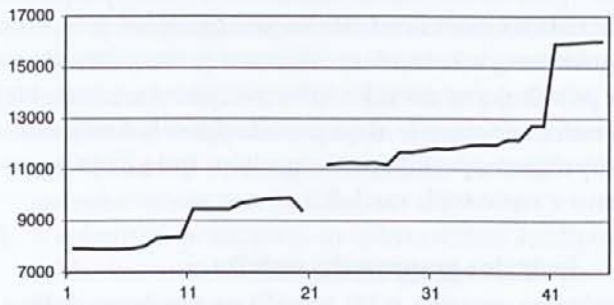
Tipična "napaka" konvencionalnih metrik kompleksnosti je tudi, da po navadi z večjo velikostjo izvornih datotek izmerijo tudi večjo kompleksnost/oceno, četudi se kompleksnost programa ni povečala. Če pogledamo sliki 2 in 3 vidimo, da α metrika nima te



Slika 1: **Razvoj programske opreme in logistična funkcija**

napake, saj se kompleksnost spreminja neodvisno od velikosti.

Da bi se izognili neprimernosti α metrike za majhne module (saj je α metrika načrtovana za izračun daljnosežnih korelacij), smo uvedli modificirano



Slika 2: **Rast velikosti razpoznavalnika skozi verzije**

α trend	Faza logistične funkcije/ Fazni trend	Faza razvojnega procesa/ Fazni trend	Trenutna faza
Stalen	Normalna	Ena ideja	Normalno/Ena ideja
Padajoč	Proti kaosu	Proti formiranju idej	π manjše kot 3 normalno/ Ena ideja
Rastoč	Proti normalnemu	Proti eni ideji	π večji od 3.6 kaos/ formiranje idej
Oscilira	Bifurkacija ali kaos	Združevanje idej ali kaos	π večji od 3.6 kaos/ formiranje idej

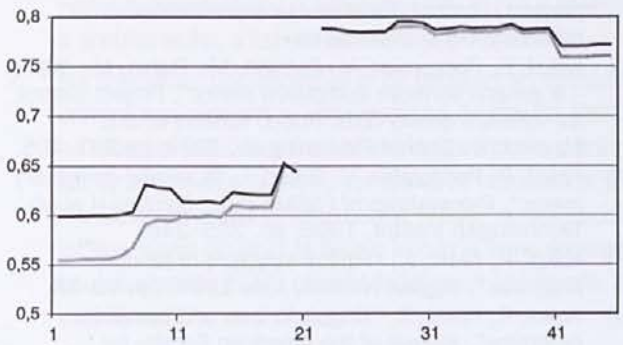
Tabela 1: **Faze in stanje programske opreme**

metriko α^*, ki po zgledu drugih matematičnih orodij za izračun korelacij upošteva, da je signal periodičen. Tako smo v izračunu α metrike spremenili:

$$\Delta y(l, l_0) \equiv y((l_0 + l) \bmod s) - y(l_0)$$

kjer je s dolžina Brownovega gibanja.

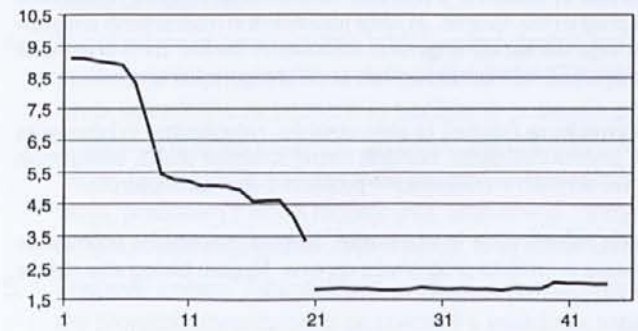
Na sliki 3 sta prikazani metriki α s sivo in α^* s črno barvo. Večje odstopanje metrik, kot je bilo pričakovano, opazimo samo v začetni fazi, tako da smo za vse nadaljnje preizkuse uporabili kar originalno α metriko.



Slika 3: **α in α^* metriki za razpoznavalnik**

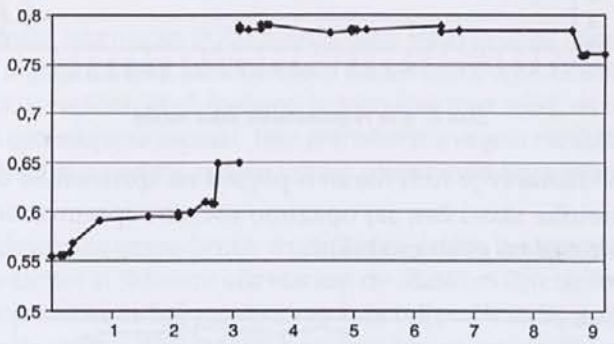
Na sliki 3 vidimo, da je med verzijo 21 (1.18) in 22 (1.19) prišlo do kvalitativnega preskoka, ki ni bil posledica bistvene spremembe velikosti datoteke – v konkretnem primeru je bil prepisan pregledovalnik, kar je doprineslo k večji informacijski vsebnosti programa/ideje.

Na sliki 4 lahko sledimo spremembi kontrolnega parametra π. Na začetku opazimo kaotičnost in počasno formiranje ideje ($\pi > 3.6$), kako razviti razpoznavalnik. Bistven preskok se ponovno zgodi med verzijama 21 in 22, kjer se očitno razjasni ideja rešitve.



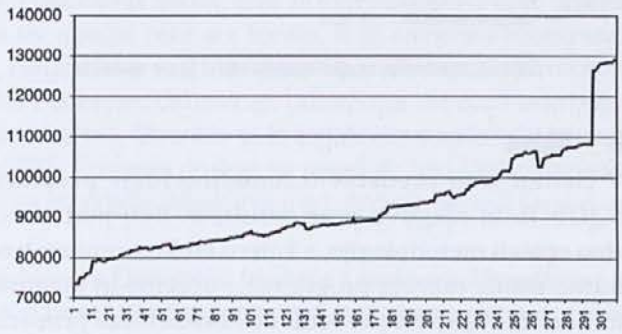
Slika 4: **π za razpoznavalnik skozi verzije**

Prikaz spreminjanja α metrike glede na verzije je lahko zavajajoč. Če pogledamo s časovno komponento (slika 5), nam osciliranje α prikaže v popolnoma drugačni luči. Opazimo, da se ideje in spremembe dogajajo časovno zgoščeno, nato pa ena rešitev začasno prevlada.

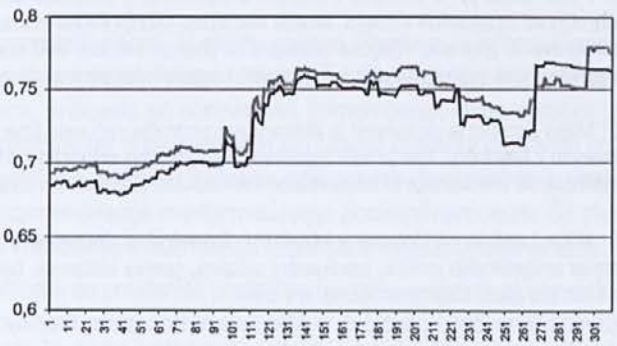


Slika 5: **α in α^* metriki za razpoznavalnik skozi čas v tednih**

Na slikah 6, 7 in 8 je prikazana analiza za modul FTPD. Opazimo lahko, da je ideja rešitve že od začetka relativno dobro znana in se skozi verzije samo kristalizira ($\pi < 3$).



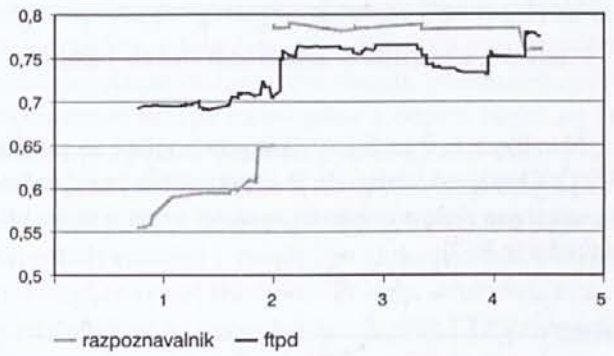
Slika 6: **Rast velikosti ftpd skozi verzije**



Slika 7: **α in α^* metriki za ftpd skozi verzije**

Slika 8: **π za razpoznavalnik skozi verzije**

Zanimiv je tudi hkraten pogled na spremembe α metrike skozi čas, saj opazimo sočasne spremembe metrike na obeh modulih.



Slika 9: **α metrika za oba modula skozi čas v mesecih**

6 Sklep

V članku smo predstavili temeljne ideje projekta SQUFOL in njegove prve rezultate. Bolj podrobno smo opisali metodologijo, s katero lahko ocenimo trenutno stanje razvoja programske opreme in njegove trende. Teorijo smo preverili na konkretnem primeru in pokazali njeno empirično veljavnost.

7 Literatura

[1] Bar Yam, Y., "Dynamics of Complex Systems", Addison Wesley, 1997.

[2] Cardoso, A. I., Crespo, G., "Is the software process a cahotic one?", Working paper of Mathematical Science Center, Madeira University, 1999.

[3] Gell-Mann, M., "What is complexity", Complexity 1(1), 1995, pp. 16–19.

[4] Kitchenham, B., "The certainty of uncertainty", Proceedings of FESMA (Eds: Combes H. et al), Technologish Institut, 1998, pp. 17–25.

[5] Kokol P. et al. Software quality founded on design laws (SQUFOL): final report [about EU project] 5th Framework Programme, reporting period under review 1/10/2002–30/9/2003, (5th European framework project). Maribor: Fakulteta za elektrotehniko, računalništvo in informatiko.

[6] Kokol, P., Podgorelec, V., Zorman, M., Pighin, M., "Alpha – a generic software complexity metric", Project control for software quality (Eds: Rob J. Kusters et al.), Maastricht : Shaker Publishing BV, 1999, pp 397–405.

[7] Kokol, P., Podgorelec, V., Brest, J., "A wishful complexity metric", Proceedings of FESMA (Eds: Combes H et al), Technologish Institut, 1998, pp. 235–246.

[8] Kokol, P., Brest, J., "Fractal structure of random programs", Sigplan Notices, June 1998, pp. 33–38.

[9] Kokol, P., Kokol, T., "Linguistic laws and computer programs", Journal of the American Society for Information Science, 47(10), 1996, pp. 781–785.

[10] Kokol, P., Stiglic, B. and Zumer, V.: Metaparadigm: a soft and situation oriented MIS design approach. International Journal of Bio-Medical Computing 39, 1995, pp. 243–256.

[11] Morowitz, H., "The Emergence of Complexity", Complexity 1(1), 1995, pp. 4.

[12] Podgorelec V., Kokol, P.: Uporaba fraktalne metrike kompleksnosti pri avtomatskem programiranju, Zbornik devete Elektrotehniške in računalniške konference ERK, 2000, str. 133–136.

[13] Schenkel, A., Zhang, J., Zhang, Y.: Long range correlations in human writings, Fractals 1(1), 1993, pp. 47–55.

[14] Tucker, A.B. (ed.), "The Computer Science and Engineering Handbook", CRC Press, 1997.

[15] http://sourceforge.net/projects/pureftpd/

Dr. Peter Kokol je na Univerzi v Mariboru diplomiral s področja elektrotehnike in doktoriral s področja računalništva. Njegova raziskovalna področja so inteligentni sistemi, teorija sistemov, teorija kaosa in kvaliteta programske opreme. Je vodja nacionalnih in mednarodnih projektov iz imenovanih področij. Njegova bibliografija obsega več kot 500 enot, od tega več kot 60 originalnih znanstvenih člankov. Bil je predsednik organizacijskih in programskih odborov več svetovnih konferenc. Je predsednik IEEE tehničnega komiteja za računsko medicino.

Dr. Milan Zorman je diplomiral in doktoriral s področja računalništva in informatike na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru, kjer je tudi zaposlen. Raziskovalno deluje na področjih umetne inteligence, hibridnih metod strojnega učenja, inteligentnih sistemov ter medicinske in zdravstvene informatike. Sodeluje pri izvajanju več domačih in mednarodnih projektov z omenjenih področij.

Dr. Mitja Lenič je na Univerzi v Mariboru diplomiral in doktoriral s področja računalništva in informatike. Njegova raziskovalna področja so teorija programskih jezikov, inteligentni sistemi, teorija sistemov, teorija kaosa in kvaliteta programske opreme. Njegova bibliografija obsega več kot sto enot, objavljenih doma in v tujini.

Alojz Tapajner je diplomiral s področja računalništva in informatike na Univerzi v Mariboru. Njegova raziskovalna področja so inteligentni sistemi in posebno njihova uporaba na finančnih področjih. Sodeluje pri raziskovalnih projektih Laboratorija za načrtovanje sistemov.