Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike i računarstva

Želimir Piljić

Ciklomatska složenost

Seminarski rad

Osijek, 2017.

Sadržaj

1 Motivacija

Osiguravanje pouzdanosti i sigurnosti je proces provjeravanja zadovoljavaju li kritični dijelovi sustava zahtjeve pouzdanosti. Nužan preduvjet je verifikacija i validacija u svim fazama procesa (specifikacija, dizajn, implementacija) koji će detektirati pogreške koje mogu utjecati na dostupnost, sigurnost i pouzdanost sustava. Verifikacija i validacija kritičnih dijelova sustava je zajednička bilo kojem softverskom sustavu. Također, cilj je tim radnjama pokazati da ponašanje sustava i usluga kao i sami sustav zadovoljava zahtjeve klijenta. Zato se upravo u to vrijeme obično otkrivaju pogreške u zahtjevima, dizajnu i „bugovi“ koje treba otkloniti. Testiranje i analiziranje radi se zbog dva osnovna razloga:

1. **Posljedice kvara.** Posljedice kvara kritičnog dijela sustava su puno veće nego nekritičnog. Pojačanom verifikacijom i validacijom smanjuje se rizik kvara. Obično je puno jeftinije naći i otkloniti pogreške prije nego je sustav isporučen nego li plaćati posljedice uzrokovane nekim nedostatkom.

2. **Validacija pouzdanosti atributa.** Ponekad je potrebno napraviti službenu validaciju kako bi neko regulatorno tijelo odobrilo sustav. Tako u nekim slučajevima vanjsko regulatorno tijelo odobrava sustav (avijacija, medicina). Kako bi se dobio potreban certifikat potrebno je pokazati da je sustav primjereno validiran.

Tehnike statičke analize su tehnike verifikacije koje ne zahtijevaju pokretanje programa. Iz tog razloga naglasak je na izvornoj reprezentaciji softvera – bilo modela specifikacije ili dizajna, bilo izvornog koda programa. Statičkom analizom mogu se identificirati pogreške sustava prije nego je sam programski kod uopće napisan. Uobičajena tehnika statičke analize je inspekcija i recenzija grupe ljudi. Oni pregledavaju dizajn i kod u detalj tražeći potencijalne pogreške i omaške. Jedna od metoda je i korištenje alata za modeliranje dizajna koji traži anomalije u UML–u kao što je korištenje istog imena za različite objekte.
Za kritične sustave moguće je koristiti dodatne tehnike statičke analize:

1. **Formalna verifikacija.** Matematički argumenti da program ispunjava specifikacije.

2. **Provjera modela.** Osoba provjerava postoje li nekonzistentnosti u formalnom opisu.

3. **Automatizirana analiza programa.** Provjerava se postoje li uobičajene pogreške u izvornom kodu.

Ove tehnike su usko povezane. Provjera modela ovisi o formalnom modelu sustava koji može biti napravljen iz formalne specifikacije. Automatizirani analizatori mogu koristiti formalne tvrdnje uklopljene u program u obliku komentara kako bi provjerili da je pridruženi kod konzistentan.
Kako bi testiranje bilo što jednostavnije i potpunije, potrebno je paziti da softverske komponente budu što manje složenosti.

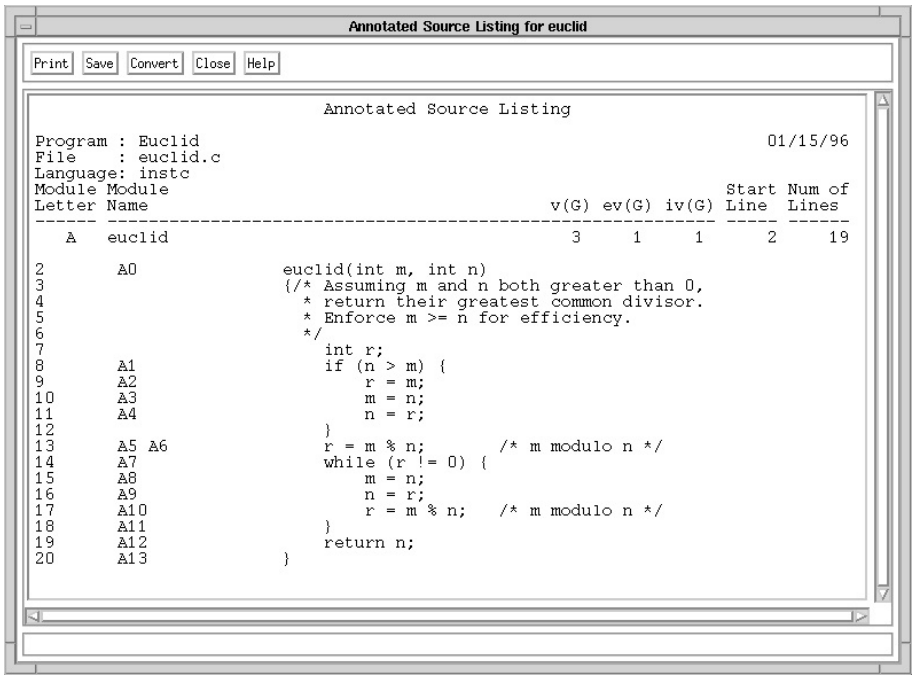
2 Ciklomatska složenost

2.1 Mjere složenosti softvera

Postoji puno mjera složenosti softvera od jednostavnih kao što je broj linija izvornog koda pa do broja definicija i korištenja varijabli. Kako bi mjera bila objektivna, trebala bi biti neovisna o formatiranju i programskom jeziku. Neka mjera složenosti trebala bi biti jednaka ako je dobivena iz izvornog koda pisanog u C–u, Javi, Pythonu ili nekom drugom programskom jeziku. Broj linija koda ne ispunjava ovo svojstvo. Štoviše, ova mjera uvelike ovisi o programskom jeziku, formatiranju i stilu kodiranja. Ciklomatska mjera zadovoljava ovaj kriterij.
Ciklomatska složenost mjeri broj logičkih odluka u jednom softverskom modulu. Koristi se iz dva razloga. Prvi je taj što daje preporučeni broj testova za softver. Drugi, koristi se tijekom svih faza softverskog razvoja, počevši od dizajna kako vi omogućio pouzdanost, testabilnost i održivost softvera. Ciklomatska složenost bazirana je u potpunosti na strukturi dijagrama tijeka softvera.

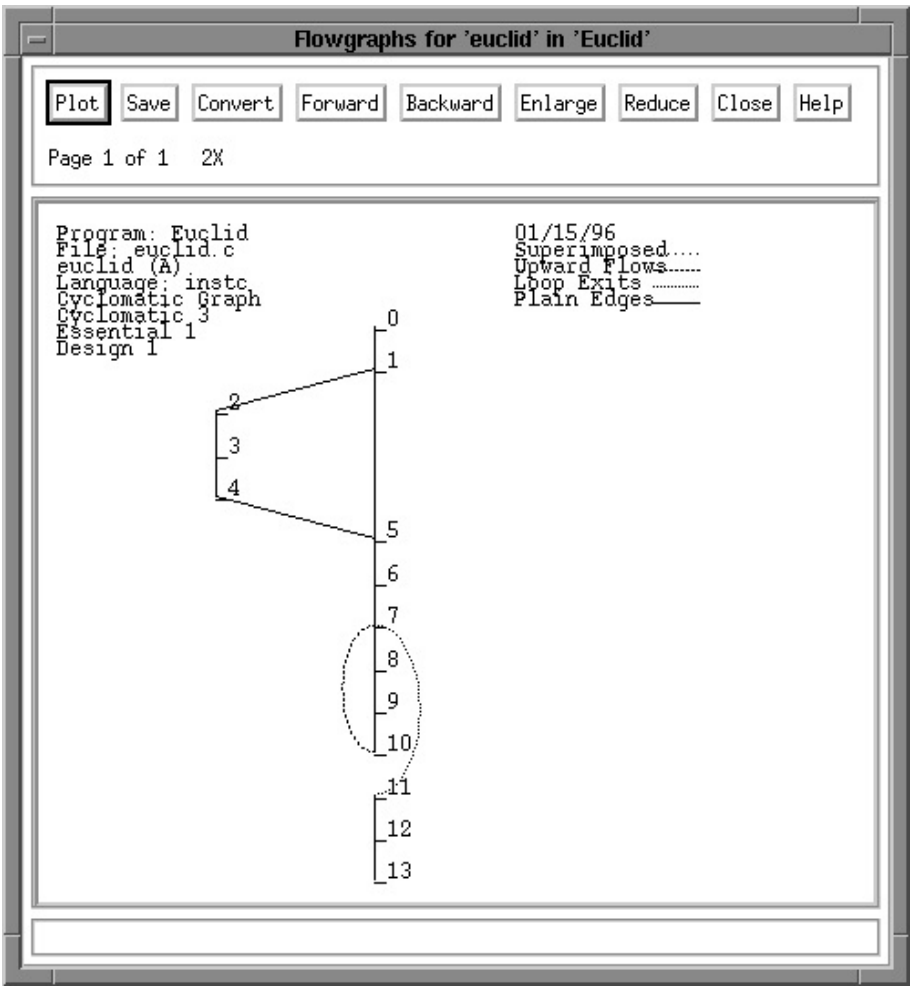
2.2 Graf tijeka

Graf tijeka opisuje logičku strukturu softverskog modula. Kad kažemo modul, mislimo na točno jednu funkciju ili rutinu u nekom programskom jeziku koja ima jednu početnu i jednu završnu točku i moguće ju je koristiti kao komponentu pomoću pozovi/vrati mehanizma. Svaki graf tijeka sastoji se od vrhova i bridova. Vrhovi predstavljaju izraze, a bridovi prijelaz kontrole između vrhova. Svaki mogući put izvršavanja softverskog modula ima pripadni put od početne do završne točke na grafu tijeka.
Uzmimo za primjer funkciju u C–u koja implementira Euklidov algoritam za traženje najvećeg zajedničkog djelitelja koju možemo vidjeti na Slici (1).



Slika 1: Euklidov algoritam u C programskom jeziku

Vrhovi su numerirani od $A0$ do $A13$. Graf tijeka je prikazan na Slici (2).



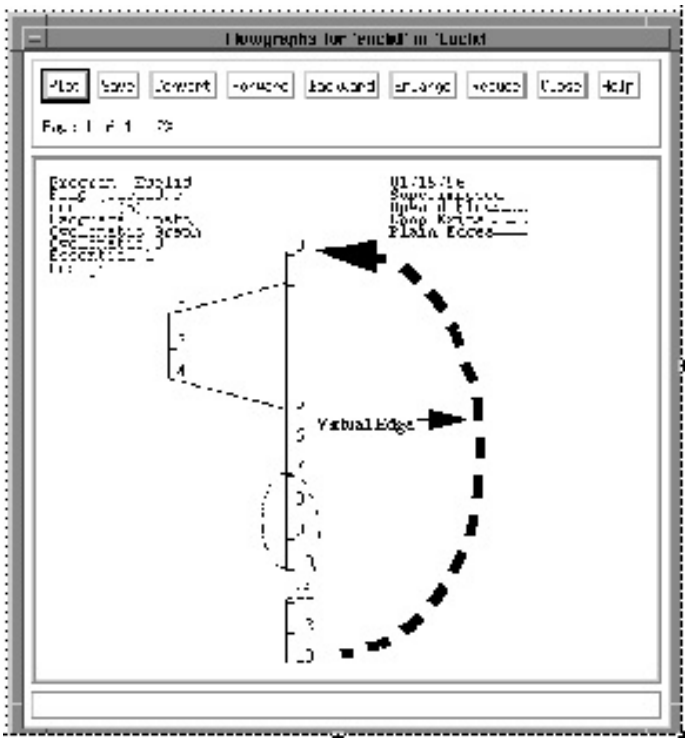
Slika 2: Graf tijeka modula „euclid“ modula

Vrh 1 predstavlja „if“ naredbu pri čemu je vrh 2 ukoliko je izraz „$n > m$“ „true“ i 5 ukoliko je isti izraz „false“. Odluka u „while“ petlji predstavljena je vrhom 7, a sljedeća iteracija isprekidanom linijom od čvora 10 do 7.

2.3 Definicija ciklomatske složenosti

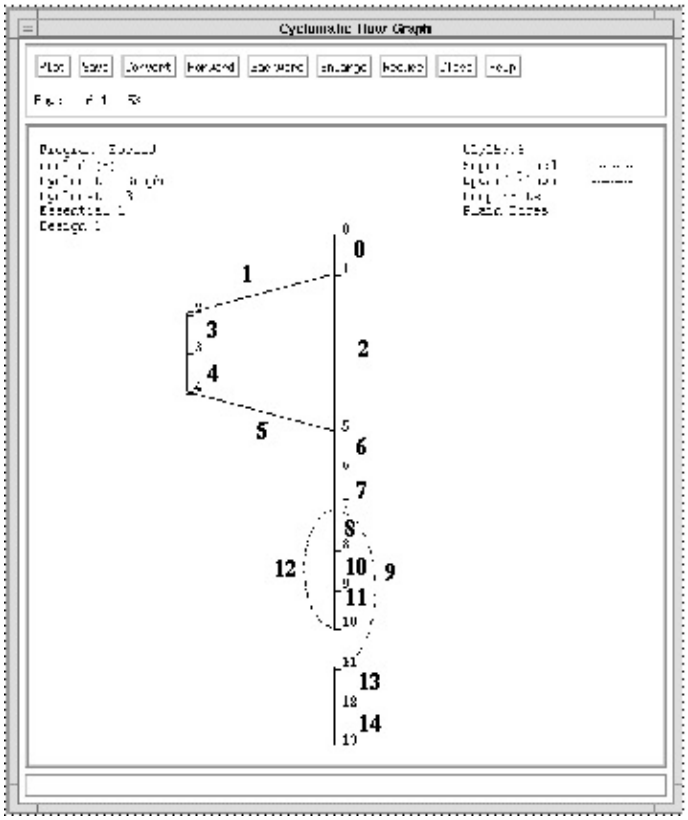
Ciklomatska složenost za svaki modul definirana je kao $e - n + 2$, gdje je e broj bridova i n broj vrhova u grafu tijeka. Dakle, za Euklidov algoritam u Poglavlju 2.2, ciklomatska složenost je 3 (15 bridova, manje 14 vrhova, više 2). Ciklomatsku složenost često označavamo s $v(G)$, pri čemu je v ciklomatski broj u teoriji grafova ovisan o grafu G. Ciklomatski broj u teoriji grafova je definiran kao $e - n + 1$. Graf tijeka nije čvrsto povezan[1], ali postaje kada mu se doda virtualni brid koji povezuje završnu i početnu točku. Ciklomatska složenost izvedena je iz definicije ciklomatskog broja dodajući 1 zbog virtualnog brida. Definicija čini ciklomatsku složenost jednaku broju neovisnih puteva u grafu tijeka.

[1]Ne postoji put između svaka dva vrha.



Slika 3: Graf tijeka „euclid“ modula s dodanim virtualnim bridom

Slika (3) prikazuje graf tijeka sa Slike (2) s dodanim virtualnim bridom. Intuitivno, on predstavlja tijek ostatka programa u kojem se koristi modul. Moguće je izračunati broj (virtualnih) tijekova koristeći jednadžbe očuvanja na početnom i završnom čvoru koji je jednak broju izvršavanja modula. Za svaki drugi pojedinačni put u modulu, ovaj broj jednak je 1. Virtualni brid neće biti potreban u nastavku rada, ali treba uočiti da iako se tijek može izračunati kao linearna kombinacija tijekova koji koriste stvarne bridove, njegovo postojanje ili nepostojanje ne pravi nikakvu razliku u računanju broja linearno nezavisnih putova u samom modulu.



Slika 4: Graf tijeka „euclid“ modula s numeriranim bridovima

Slika (4) prikazuje graf tijeka modula „euclid“ pri čemu je 15 bridova numerirano od 0 do 14 zajedno s 14 vrhova numeriranih od 0 do 13. Kako je ciklomatska složenost 3, postoji skup koji čini bazu od 3 puta. Jedan takav skup sadrži putove $B1$ do $B3$ koji se mogu vidjeti na Slici (5).

```
Module: euclid

                          Basis Test Paths: 3 Paths

Test Path B1: 0 1 5 6 7 11 12 13
        8(    1): n>m ==> FALSE
       14(    7): r!=0 ==> FALSE

Test Path B2: 0 1 2 3 4 5 6 7 11 12 13
        8(    1): n>m ==> TRUE
       14(    7): r!=0 ==> FALSE

Test Path B3: 0 1 5 6 7 8 9 10 7 11 12 13
        8(    1): n>m ==> FALSE
       14(    7): r!=0 ==> TRUE
       14(    7): r!=0 ==> FALSE
```

Slika 5: Baza koja sadrži putove $B1$ do $B3$

Svaki put se može prikazati kao linearna kombinacija puteva baze $B1$ do $B3$. Na primjer, put P prikazan na Slici (6) jednak je $B2–2 \cdot B1 + 2 \cdot B3$

```
Module: euclid

                    User Specified Path: 1 Path

Test Path P: 0 1 2 3 4 5 6 7 8 9 10 7 8 9 10 7 11 12 13
        8(    1): n>m ==> TRUE
       14(    7): r!=0 ==> TRUE
       14(    7): r!=0 ==> TRUE
       14(    7): r!=0 ==> FALSE
```

Slika 6: Put P

Ovo se vidi na Slici (7) koja prikazuje broj izvršavanja svakog brida u svakom putu.

Path/Edge	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
B1	1	0	1	0	0	0	1	1	0	1	0	0	0	1	1
B2	1	1	0	1	1	1	1	1	0	1	0	0	0	1	1
B3	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1
P	1	1	0	1	1	1	1	1	2	1	2	2	2	1	1

Slika 7: Matrica incidencije bridova za puteve $B1 - B3$ i put P

Zanimljivo svojstvo je da ako se izvrše putevi baze, oni će nužno pokriti svako kontrolno grananje. To znači da pokrivanje svih bridova ne zahtjeva puteva više od ciklomatske složenosti. Međutim, izvršavanje puteva baze nije uvijek optimalno. Pokrivanje svih bridova može se obično postići s manje puteva. U ovom primjeru, putevi $B2$ i $B3$ pokrivaju sve bridove bez puta $B1$.
Potrebno je uočiti da putevi $B1$ do $B3$ nisu ništa specijalno osim što čine bazu. Put P u kombinaciji s bilo koja dva puta od $B1$ do $B3$ također bi tvorio bazu. Činjenica da postoji puno baza putova je važno za testiranje jer to znači da postoji određena sloboda u biranju puteva za testiranje.

2.4 Ograničavanje cikomatske složenosti na 10

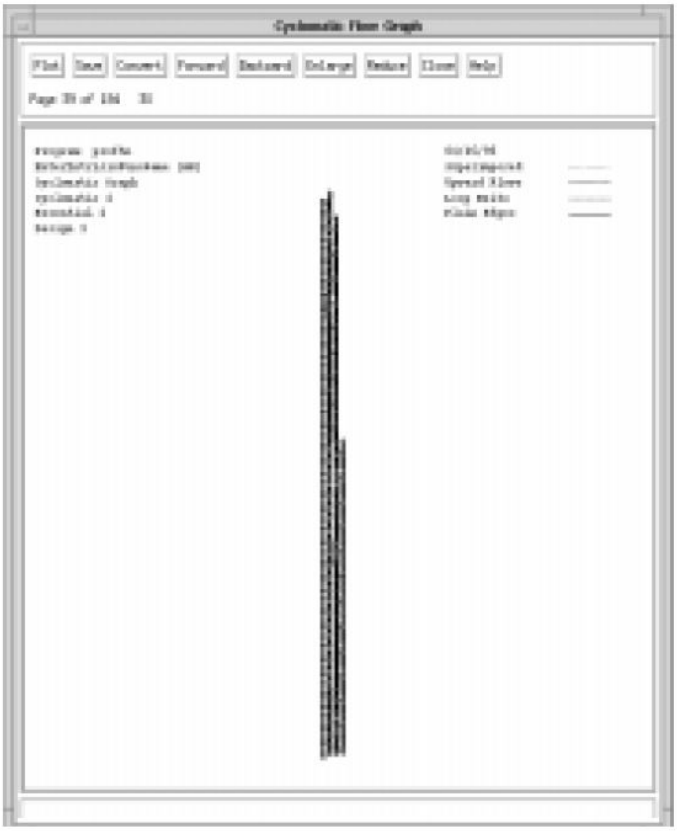
Postoji puno dobrih razloga zašto ograničiti ciklomatsku složenost. Presloženi moduli su podložniji greškama, teže ih je razumjeti, testirati i modificirati. Namjerno ograničavanje složenosti u svim fazama razvoja softvera kasnije rezultira poboljšanjem prethodno navedenih problema. Puno tvrtki uspješno provodi ograničavanje kompleksnosti njihovih programskih rješenja. Točan broj na koji je potrebno ograničiti kompleksnost je nemoguće jednoznačno odrediti te postaviti neki univerzalni standard. Originalno ograničenje je predložio McCabe[2] i ono iznosi 10. Ciklomatska složenost veća od 10 bi trebala biti rezervirana za iskusne programere, formalni dizajn, moderni programski jezik, strukturirano programiranje itd. Drugim riječima, tvrtka može koristiti ograničenje veće od 10 samo ako je sigurna da zna što radi i spremna je uložiti dodatan rad pri testiranju koje zahtijevaju kompleksniji moduli. Osim same složenosti, zanimljive su i njene iznimke. Na primjer, McCabe je prvotno predložio izuzimanje „switch“, tj. „case“ grananja iz ograničavanja složenosti. Problem takvog grananja je tijekom godina interpretirano na više načina pa su neki softverski inženjeri počeli složenost, definiranu na taj način, zloupotrebljavati. Konačan rezultat je bio da su moduli s nekoliko grananja bili identificirani kao programi male složenosti.

[2]Thomas J. McCabe (1941. –), američki matematičar.

Ciklomatska složenost daje broj potrebnih testova koji je za višestruko grananje jednako broju grana odluka. Svaki pokušaj modificiranja mjerenja složenosti rezultira smanjenim brojem testova koji neće testirati sve grane odlučivanja i samim time biti manjkav za potrebe testiranja.
Ograničavanje složenosti odražava se na količinu koda u individualnim modulima što rezultira većim brojem modula u aplikaciji. Osim složenosti, treba uzeti u obzir i principe strukturiranog dizajna: modul obavlja jednu konceptualnu zadaću i to radi bez interakcije s drugim modulima osim što ih koristi kao subrutine.
Poštujući ograničavanje složenosti i principa strukturiranog dizajna možemo zaključiti: „Neka svaki modul bude ciklomatske složenosti 10. U suprotnom, potrebno je objasniti zašto je taj broj premašen.“

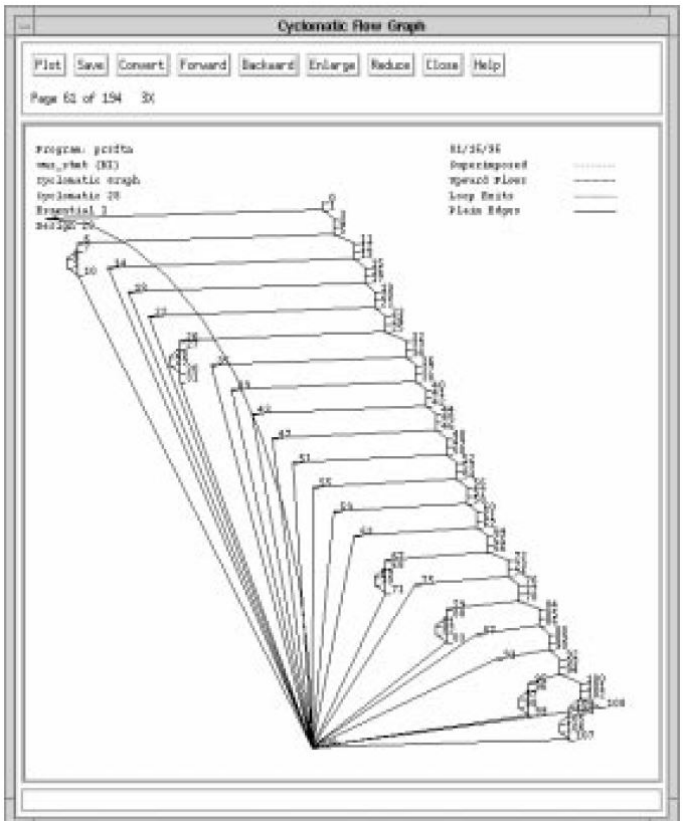
2.5 Nezavisnost ciklomatske složenosti i broja redaka koda

Kako je 10 uobičajeno ograničenje za ciklomatsku složenost, tako je 60 uobičajeno ograničenje za broj redaka koda (arhaično ograničenje kako bi svaki softverski modul može biti ispisan na jednoj stranici papira).



Slika 8: Graf tijeka ciklomatske složenosti 1

Iako je broj linija koda često korišten kao mjera složenosti, ne postoji nikakva konzistentna korelacija između njih. Moguće je konstruirati puno modula koji nemaju grananje (minimalne ciklomatske složenosti 1), a sadrže puno više od 60 linija. Također, vrijedi i obrnuto. Na primjer, graf tijeka na Slici (8) ima složenost 1 i 282 redaka koda, dok onaj na Slici (9) ima složenost 28 i 30 redaka koda. Iako je broj redaka važna mjera, ona je nezavisna o složenosti i ne treba imati istu svrhu.



Slika 9: Graf tijeka ciklomatske složenosti 28

Literatura

[1] I. Sommerville, Software engineering, Addison-Wesley, Boston, 2011.

[2] Arthur H. Watson, Thomas J. McCabe, Natl. Inst. Stand. Technol. Spec. Publ. 500-235, 123 pages (September 1996)