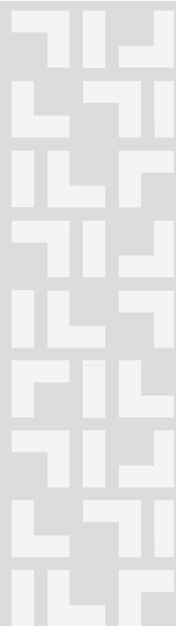
Univerza v Ljubljani
Fakulteta
za računalništvo
in informatiko



4. Zanesljivost programske opreme

Prosojnice za predavanja 4.UNI/RS

Pripravil: izr.prof.dr.Miha Mraz, štud.l.2010/11

4.1. Uvod

- Napake izvirajo iz faze nastanka (načrtovanje, programiranje, testiranje itd.), ne iz faze uporabe
- SW se ne obrabi, ne pokvari
- Metrike iz preteklosti: 80.leta, 8 napak/1000 LOC (angl. lines of code)
- Microsoft (1990-95): 12 program.ur za odpravo 1 napake
- 1995: le 5% proizvajalcev SW uporablja zanesljivostne metode in metrike pri razvoju SW

4.2. Razvojno življenski cikel programske opreme s klasifikacijo programskih produktov

Faze življenskega cikla programske opreme:

1.) Start projekta (časovna točka)
2.) Določitev osnovnih funkcij in zahtev
3.) Specifikacije
4.) Prototip – sistemski test ob prisotnosti stranke
5.) Revizija specifikacij
6.) Končni design (zmogljivost, zanesljivost, itd.)
7.) Realizacija kode
8.) Enotski test
9.) Integracijski test
10.) Sistemski test
11.) Sprejemni test (prisotnost naročnika)
12.) Eksploatacijska doba
13.) Upravljanje z SW v eksploatacijski dobi
14.) "Redesign"
15.) "Discard"

CMM (angl. *capability maturity model*): delitev programske opreme v 5 razredov, glede na njeno zasnovo, izdelavo, vzdrževanje itd.:

- 1. “Inital”: karakterističen stihijski razvoj
- 2. “Repeteable”: karakterisitčen razvoj na osnovi predhodnih izkušenj
- 3. “Defined”: karakterističen projektni pristop k razvoju
- 4. “Managed”: karakterističen dobro upravljan razvojni proces
- 5. “Optimizing”: karakterističen stalen razvoj v smislu optimizacije in nadgradnje

4.3. Redundanca programske opreme

- Izvajanje *n* kopij iste programske opreme praviloma ne doprinaša *k* zanesljivosti
- Možnost: *N*-verzijsko programiranje
  - Funkcionalna identičnost
  - Ločen razvoj več verzij programske opreme na osnovi istih specifikacij pri različni razvijalcih
  - Decizija: po glasovalnem pravilu
  - Problem: sinhronizacija delujočih verzij
  - Izhodišče: razvijalci ne smejo komunicirati brez prisotnosti naročnika (možnost napačnih interpretacij)
  - Razvojna okolja: po možnosti naj bi bila različna

4.4. “Roll back & Recovery” metode

- Postopki za detekcijo napak in ponovitev zagona
- Predpostavka: napake so večinoma minljive (angl. *transient errors*)
- “*Forward*” tehnike: zavedamo se napake, a procesiranje vršimo naprej, z upanjem, da v sekvenci odločitev napaka zaradi svoje minljivosti ne bo več prisotna
- “*Backward*” tehnike: ob napaki procesiranje “prevrtimo” za *n* korakov vzvratno
- Osnovni tipi “backward” tehnik:
  - “reboot” ali “restart” sistema (izguba časa in podatkov)
  - “recover” avtomatiziran poseg, ki vzpostavlja le del operacijskega sistema;
  - žurnalske tehnike: na osnovi beleženja procesnih korakov, lahko procesiranje “prevrtimo nazaj”
  - “retry” tehnike: ponavljanje procesnega koraka vse do njegove uspešne izvedbe
  - “check pointing” tehnike: beležimo le del procesnih korakov

4.5. Osnove testiranja programske opreme

- Običajno je nemogoče pretestirati vse možne scenarije (možne poti) izvajanja opazovane programske opreme
- Prva delitev testiranj:
  - Možen vpogled v izvorno kodo (“white box testing”)
  - Brez vpogleda v izvorno kodo (“black box testing”)
- Druga delitev testiranj:
  - Imamo možnost izvajanja programa: dinamično testiranje
  - Nimamo možnosti izvajanja programa: statično testiranje

- Statično “black box” testiranje: ne izvajamo programa, nimamo vpogleda v izvorno kodo -> gre za natančen pregled specifikacij
- Dinamično “black box” testiranje: možnost izvajanja programa brez vpogleda v izvorno kodo -> pomembno vlogo igrajo testni profili, ekvivalenčno particioniranje testnih podatkov; ločujemo med podatkovno usmerjenim testiranjem in testiranjem programskega toka;

- Statično “white box” testiranje: imamo vpogled v izvorno kodo, programa pa ne more poganjati -> gre za verifikacijo programske opreme (preverjanje skladnosti s specifikacijami, upoštevanja pravil dobre prakse kodiranja, upoštevanja ustreznih standardov, itd.); najdemo podatkovno referenčne napake, podatkovno deklaracijske napake, računske napake, primerjalne napake, I/O napake, itd.
- Dinamično “white box” testiranje: vpogled v izvorno kodo z možnostjo zaganjanja programa (te faze ne smemo enačiti s pojmom debugging-a)

4.6. Avtomatizacija postopkov testiranja

- Monitorji: prikazujejo prehojene poti v kodi (kodno pokrivni analizatorji)
- Gonilniki: krmilijo testirane programe
- Sprejemniki: sprejemajo izhode iz testiranih programov
- “Stress” & “Load” orodja: orodja za zmanjševanje količine resursov in zvečevanje bremen
- Generatorji šuma
- Snemalniki in predvajalniki (zmožnost beleženja in ponavljanja interakcij)
- Programabilna orodja z vsemi zgoraj naštetimi funkcionalnostmi vključno z zagonom in ustavitvijo ciljne aplikacije

“Monkey” koncept avtomatizacije testiranja - temelji na modeliranju manj ali bolj izkušenih uporabnikov:

- “Dumb monkey test”
- “Semi smart monkey test”
- “Smart monkey”

4.7. Metrike v programski opremi

Halsteadova metrika (1977) – izhaja iz strukturalno proceduralnih jezikov, kot sta bila v tisih časih Cobol in Fortran:

- Halsteadova metrika (mera) služi za oceno števila napak v programu
- Statičnost metrike: kode ne izvajamo
- n_1 ... št. različnih operacij (ukazov)
- n_2 ... št. Različnih operandov (spremenljivk)
- N_1 ... celotno število operacij
- N_2 ... celotno število operandov
- Predpostavka: na št. vnešenih napak vpliva le število operacij in operandov

- N ... Dolžina programa: $N=N_1+N_2$
- n ... Slovar programa: $n=n_1+n_2$
- V ... Volumen programa: $V=N*\log_2 n$
- D ... Kompleksnost programa (angl. *difficulty*): $D=(n_1/2)*(N_2/n_2)$
- E ... Vložek v kodo (angl. *effort*): $E=D*V$
- T ... Čas za razumevanje ali implementacijo programa (v sek.): $T=E/18$
- B ... Pričakovano število vnešenih napak: $B=E^{2/3}/3000$
- Statistično pridobljena ocena
- C in C++ koda prinašata po izkušnjah večje število vnešenih napak
- Vpliv LOC (angl. *Lines of code*) je posreden

LOC (angl. *Lines of code*) metrika:

- Statičnost metrike: kode ne izvajamo
- Odraz kompleksnosti programa naj bi bilo število vrstic izvorne kode (danes je pojem vrstice izvorne kode „zamegljen“ zaradi komplesnih razvojnih orodij)
- SLOC – source lines of code:
  - Fizično št. LOC: vključuje tudi komentarje, prazne vrstice, itd.
  - Logično št. LOC: samo vrstice z operacijami in operandi
- 1 KLOC – 1.000 LOC, 1 MLOC= 1.000 KLOC
- Obstajajo različni načini vrednotenja xLOC
- Zgledi metrik LOC:
  - Win 3.1.: 4.5 MLOC
  - Win 2000: > 29 MLOC
  - Win XP: > 45 MLOC
  - Red Hat Linux 7.1. (2001): > 30 MLOC

Mc Cabe-ova ciklomatična metrika:

- Statičnost metrike (kode ne izvajamo)
- Mc Cabe-ovo ciklomatično število G: izraža kompleksnost vejitev v programu, kar je neposredno določeno s številom alternativnih programskih poti

MI indeks (angl. *Maintainability index*):

- Vzdrževalski indeks
- Kombinacija predhodno navedenih metrik
- MI=MIwoc +Micw (brez komentarjev + uteži komentarjev)
- MIwoc=171-5,2*ln(ave(V))-0,23*ave(G)-16,2*ln(ave(LOC))
- MIcw=50*sin(sqrt(2,4*perCM))
- Statistično – eksperimentalno pridobljena ocena
- perCM ... Odstotek komentarjev na modul kode
- MI > 85 ... Dobra upravljivost kode
- MI < 65 ... Slaba upravljivost kode
- 85>MI>65 ... Povprečna upravljivost kode
- Metodo povzemajo mnoga programska orodja (www.verifysoft.com, CMT++, CMT Java, itd.)

Jelinski Moranda metrika:

- Stohastični model „preostalega“ števila napak v programski opremi
- V *n* časovnih intervalih (trajanja t_1, t_2, ..., t_n) smo s testiranjem našli *n* napak
- $\lambda(t_i) = \Phi*(N-(i-1))$:
  - i ... Št.že najdenih napak
  - N ... Neznano število napak ob začetku testiranja
  - Φ ... Doprinos posamezne napake
- Osnovna ideja: na osnovi dolžine časovnih intervalov in s pomočjo MLE (angl. *Maximum likelihood estimation*) metode t_1, t_2, ..., t_n izračunamo pričakovano vrednost N
- Na osnovi tega lahko naredimo tudi sklep, ali je po $t_1+t_2..+ t_n$ urinih periodah smiselno s testiranjem prekiniti, ali ne;
- Več o metodi naj si študent preberu v viru avtorice Bruderove na spletni strani predmeta

Literatura

Shooman, M.L.: Reliability of computer systems and networks, Wiley, 2002 (knjigo poseduje knjižnica FE in FRI)

Musa J.: Software reliability engineering, McGrawHill, 1998

Pham H.: Software Reliability, Springer, 2000

Lyu M.R.: Software reliability engineering, IEEE Press, 1995