GOSTOTA NAPAK IN ODPOVEDI - PROBLEMATIČNO MERILO KAKOVOSTI

Tomaž Dogša
cV&Vs Center za verifikacijo in validacijo sistemov
tdogsa@uni-mb.si
Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru

Povzetek

V tem prispevku podrobneje obravnavamo problematiko merjenja in interpretiranja gostote napak in odpovedi. Obe merili se zelo pogosto uporabljata za ocenjevanje nekaterih atributov kakovosti programskega produkta (npr. zanesljivosti) in razvoja. Zaradi nedorečenosti metrik so lahko rezultati izrazito netočni, kar je potrebno pri interpretiranju raznih komparativnih analiz upoštevati. V prispevku so tudi opisani vzroki, ki povzročajo veliko nenatančnost metrike. Ker na tem področju še ni enotne terminologije, v prispevku predlagamo določeno nomenklaturo, ki jo potrebuje preverjevalec pri pisanju poročil.

Abstract

In the paper we investigate the problems of collecting and interpreting the fault and failure densities. Both measures are used as an indicator of product (reliability) and process quality. Metrics have so far not been sufficiently developed. Thus, results obtained from them can be very inaccurate. This drawback must be considered when we would like to interpret the results of comparative analyses. We also describe the reasons for inaccuracy of fault and failure densities. A suitable nomenclature needed for writing problem reports is also proposed.

■■■

1. Uvod

Pri sistematičnem preverjanju in tudi kasneje pri vzdrževanju se pojavljajo določene nepravilnosti v delovanju programske opreme. Sistematično beleženje teh podatkov in ustrezna klasifikacija predstavljata osnovni zahtevi vsakega sistema kakovosti. Zaradi tega lahko najdemo takšne zahteve v mnogih standardih, ki govorijo o kakovosti. Te podatke lahko zbiramo zaradi tega, ker to pač zahteva npr. standard, ali pa iz nekih drugih razlogov. V prispevku bomo podrobneje osvetlili problematiko štetja napak in odpovedi. Opozorili bomo na nekatere omejitve, katerih neupoštevanje lahko privede do popolnoma napačnih zaključkov.

Ocenjevanje oziroma merjenje kakovosti je še vedno aktualna tema mnogih raziskav. Pojavili so se tudi standardi (npr. [ISO/IEC 9126: 1991]), ki sicer govorijo o kakovosti, o njenih atributih, vendar brez konkretnih definicij metrik. Dokler kakovost ne bo natančneje definirana z merljivimi atributi, je tudi ne bo mogoče kvantificirati na objektiven način. Če bi imeli kvantitativno oceno kakovosti, bi lahko:

- primerjali produkte glede kakovosti,
- ocenjevali produktivnost,
- lažje iskali kompromise med kakovostjo, stroški, viri in roki,
- lažje usmerjali razvoj,
- natančneje ocenili razna tveganja,
- natančneje ocenili stroške vzdrževanja.

Kakovost programske opreme sestavlja več atributov, od katerih ima zanesljivost zelo pomembno vlogo. Direktna meritev zanesljivosti zahteva zelo veliko časa in se zaradi tega redko uporablja. V literaturi obstaja sicer niz zanesljivostnih modelov, ki pa zaenkrat še niso na takšnem nivoju, da bi bili splošno uporabni. Namesto njih se uporabljajo razni indikatorji oziroma merila, ki kažejo na večjo ali manjšo stopnjo zanesljivosti oziroma kakovosti. Najbolj pogosto uporabljeni indikatorji zanesljivosti so: gostota napak in gostota odpovedi. Štetje napak je na videz zelo enostavna metrika, ki ne potrebuje nobenih dragih orodij in je tudi razumljiva povprečnemu uporabniku oziroma managerju. Ker se tovrstna metrika pogosto pojavlja v raznih publikacijah, je za razvijalce zanimiva tudi zaradi komparativnih analiz.

1.1. Definicija napake

Pri beleženju podatkov moramo ločiti najmanj dve stvari: nepravilno obnašanje programa in vzroke za to obnašanje. V angleščini imajo za opisovanje rezultatov preverjanja na razpolago naslednje besede: *error, fault,*

failure, mistake, defect, bug, glitch, problem, issue, trouble, anomaly. Ker so različne panoge uporabljale svoje termine, prihaja do različnih interpretacij. Pogoste so tudi spremembe v samih standardih in literaturi ([CHILLAREGE,1995], [BEIZER,1990]). Kljub nekaterim standardom ([ANSI,1988a]), ki skušajo vpeljati red, zaenkrat ne obstaja konsistentna terminologija, ki bi bila splošno sprejeta. V nadaljevanju bomo prikazali nomenklaturo, ki večinoma temelji na standardu [ANSI,1988b].

Pri preverjanju in pri uporabi programa lahko opazujemo samo njegovo obnašanje ali njegova stanja. Notranja stanja so nam, razen izjem, nedostopna. Poznati moramo tudi pričakovano ali pravilno obnašanje programa, oziroma katera stanja so pravilna. Pri določenih vhodnih podatkih se včasih pojavi razlika med opaženim in pričakovanim obnašanjem oz. stanjem. Ta pojav bomo poimenovali **anomalija (anomaly)**. Nekatere anomalije lahko bistveno vplivajo na kakovost, nekatere pa zelo malo. Tiste, zaradi katerih se bomo odločili za popravilo, bomo poimenovali **odpovedi (failures)**. Če anomalijo toleriramo (začasno do nove verzije ali pa za vedno), postane **hiba**. Pri odpravljanju odpovedi moramo odstraniti vzroke, ki so privedli do njenega nastopa. Te vzroke bomo poimenovali **napake ali okvare (faults, defects)**. Relacija med anomalijo in napako je v splošnem *mnogo: mnogo*, saj je možno, da se samo ena napaka manifestira v več anomalijah oziroma, da je vzrok za eno anomalijo več napak. Napake v programski opremi so posledica človeških slabosti. Zato jih bomo poimenovali **zmote (error)**.

Če je v programu znak + namesto -, lahko takoj ugotovimo, da je to ena napaka. V splošnem problem ni tako preprost, saj je zelo težko opredeliti število oziroma področje napake. Zato Voas [VOAS,1998] loči med enojno napako (single-point fault) in skupkom napak (distributed faults). Slednje, se žal največkrat pojavljajo.

Pri preverjanju oziroma pri pisanju poročil moramo torej ločiti med zmoto, napako, hibo, odpovedjo in anomalijo. Kot bomo videli v kasnejšem konkretnem zgledu, je najteže šteti napake, saj jih je zelo težko definirati.

1.2. Dodatni atributi

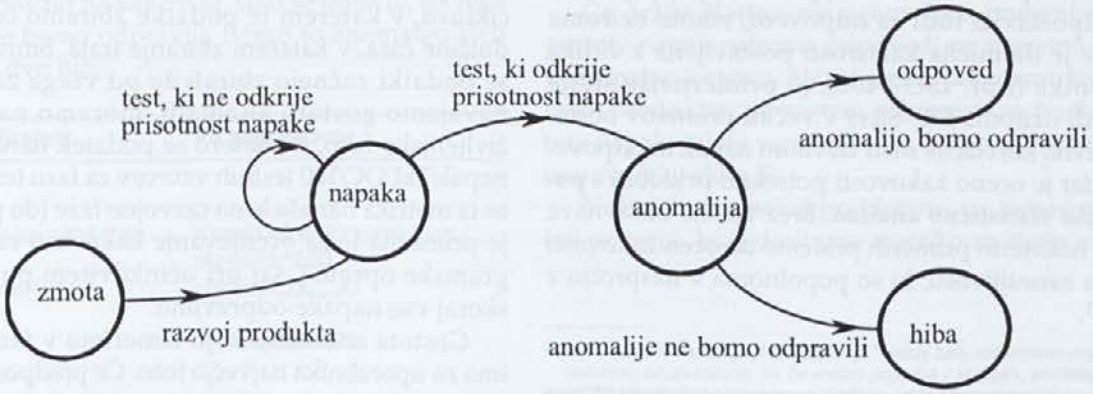
Odpovedi oziroma anomaliji lahko priredimo najmanj tri atribute: resnost, pogostost oziroma verjetnost nastopa ter čas nastopa. *Resnost odpovedi* je faktor, proporcionalen škodi, ki nastane pri njenem nastopu. Ker so ti podatki le redko na razpolago, si pomagamo s subjektivnim razvrščanjem v posamezne kategorije. Beizer [BEIZER,1990] je predlagal kar 10 kategorij. Najbolj pogosto so uporabljene naslednje tri kategorije:

1. zelo resna oziroma katastrofalna odpoved,
2. resna odpoved in
3. nepomembna odpoved.

Zelo resne so tiste, zaradi katerih je potrebno popravilo, saj zelo resno vplivajo na kakovost produkta. Pri resnih odpovedih se lahko izjemoma odločimo, da jih ne bomo odpravili. Nepomembna odpoved je tista, ki bistveno ne degradira kakovosti. Le če je povezana z majhnim naporom za odstranitev njenih vzrokov (napak), se bomo odločili za popravilo.

Nekatere odpovedi se pojavljajo pogosteje kot druge. Zato jim lahko priredimo tudi verjetnost oziroma pogostost nastopa. Kljub temu da pri dveh produktih opazimo enako število odpovedi, to ne pomeni, da se bodo z enako verjetnostjo pojavljale. Uporabnika moti predvsem **pogostost** odpovedi! Podobne atribute lahko priredimo tudi napakam, zmotam in hibam ter anomalijam.

Število odpovedi, hib oziroma odkritih napak je funkcija časa. Se ta podatek nanaša na posamezno fazo, na vse faze skupaj, ali pa samo na vzdrževanje? Podobno kot za odpoved lahko iste atribute priredimo tudi anomalijam, hibam in napakam.



Slika 1 Povezava med zmoto, napako, hibo, odpovedjo in anomalijo.

2. Štetje napak in gostota napak

Merjenje je v bistvu preslikava resničnih lastnosti nekega objekta v matematični prostor. Najbolj pogosta metoda je kvantizacija. Vsaka numerična vrednost, dobljena s kvantizacijo, spada v eno izmed kategorij, ki so prikazane v tabeli 1.

lestvica	dovoljene (smiselne) matematične operacije
nominalna	= ≠
ordinalna	= ≠ > <
intervalna	= ≠ > < + -
racionalna	= ≠ > < + - * /

Zaželeno bi bilo, da bi imeli takšno metriko, katere rezultat bi spadal v racionalno lestvico, ki omogoča vse aritmetične operacije. Z današnjimi (večinoma subjektivnimi) metodami ocenjevanja kakovosti dosegamo kvečjemu nominalno ali ordinalno lestvico. Tipičen zgled za oceno, ki spada v nominalno lestvico, je certifikat kakovosti. Certifikat nam pove samo to, ali produkt/proces ustreza določenim standardom. Če dobita dva produkta enak certifikat, nikakor ne moremo reči, kateri produkt je kakovostnejši. Taka primerjava bi bila možna šele, če bi rezultati spadali v ordinalno lestvico.

Namesto direktne meritve kakovosti, oziroma njenih atributov, lahko merimo nekatere lažje merljive lastnosti, ki so nedvoumno povezane s kakovostjo. Ena izmed njih je število napak. Meritev kakovosti v tem primeru temelji na naslednji predpostavki:

Večje število napak v programu pomeni nižjo kakovost. (1)

Na celotno podobo kakovosti vplivajo seveda tudi druge lastnosti (npr.: razumljivost, testabilnost itd.). Šele če imata dva programa druge atribute kakovosti približno enake, lahko trdimo, da je program z 10 napakami bolj kakovosten kot ta, ki jih ima 20. Ker so napake in odpovedi med seboj povezane, velja podobna predpostavka tudi za odpovedi, zmote oziroma hibe. Če je definicija kakovosti postavljena z vidika uporabnika (npr. ISO 8402), je primernejše štetje odpovedi oziroma hib. Sicer v večini primerov obstaja pozitivna korelacija med številom napak in odpovedi, vendar je oceno kakovosti potrebno pridobiti s **podrobnejšo statistično analizo**. Brez takšne obravnave lahko v nekaterih primerih pridemo do ocen kakovosti oziroma zanesljivosti, ki so popolnoma v nasprotju z dejstvi [1].

Na prvi pogled je videti, da smo s štetjem napak dobili objektivno metriko, ki bi jo lahko uvrstili v ordinalno lestvico. Ker se verjetnost, da smo napravili napako, veča s kompleksnostjo produkta, je potrebno število napak normirati glede na kompleksnost.

Predpostavka štev. 2:
Večja kompleksnost produkta pomeni večje število napak (2)

Z raziskovanjem povezave med kompleksnostjo in številom napak so začeli zelo zgodaj [CONTE,1986]. Najbolj enostavna tovrstna metrika je dolžina programa izražena z vrsticami izvirne kode (LOC[2]). Označimo z N_n število odkritih napak in s K vrednost kompleksnosti. Če želimo primerjati različno kompleksne produkte, moramo uvesti gostoto napak G_n:

$$G_n = \frac{N_n}{K} \quad (3)$$

Na podoben način lahko definiramo gostoto odpovedi, hib ali anomalij.

Število napak oziroma anomalij je odvisno tudi od načina uporabe programa. Pri testiranju izbiramo takšne testne vzorce, ki odkrijejo čim več anomalij. Žal nikoli ne vemo, koliko jih je ostalo neodkritih. Če želimo primerjati kakovost produktov v fazi testiranja, tega ne moremo storiti s primerjavo gostote napak oziroma odpovedi, saj je njihova vrednost odvisna tudi od učinkovitosti testiranja. Zaradi tega moramo gostoto dodatno normirati s faktorjem *Ut*, ki je proporcionalen učinkovitosti testiranja.

$$G_n = \frac{N_n}{K \cdot U_t} \quad (4)$$

Če si izberemo število uporabljenih testnih vzorcev za merilo temeljitosti testiranja, potem bi lahko npr. rekli, da ima program 5 napak/1kLOC/100 testnih vzorcev. Pri štetju moramo upoštevati tudi resnost odpovedi, na katero se te napake nanašajo.

Število napak je odvisno tudi od faze v življenskem ciklusu, v katerem te podatke zbiramo oziroma od dolžine časa, v katerem zbiranje traja. Smiselno je, da se podatki začnejo zbirati že od vsega začetka. Ko navajamo gostoto anomalij, moramo navesti tudi življenjsko fazo, na katero se podatek nanaša. Npr. 5 napak/1kLOC/100 testnih vzorcev za fazo testiranja. Če se ta metrika nanaša le na razvojne faze (do prevzema), je primerna le za ocenjevanje kakovosti razvoja programske opreme, saj pri učinkovitem popravljanju skoraj vse napake odpravimo.

Gostota anomalij, ki jo izmerimo v fazi uporabe, ima za uporabnika največjo težo. Če predpostavimo, da vsi uporabniki uporabljajo program približno na enak način, lahko Ut (učinkovitost testiranja) izpustimo.

[1] *V raziskavi operacijskega sistema IBM so ugotovili, da je samo 2% vseh napak povzročalo 80% odpovedi [PFLEEGER,1997].*

[2] *LOC = lines of code (število vrstic izvirne kode).*

2.1. Problematika štetja napak oziroma odpovedi

Začnimo s preprostim hipotetičnim zgledom, s katerim bomo ilustrirali problematiko štetja anomalij. Imamo program, ki naj izračuna stanje na bančnem računu. Pri preverjanju programa, dolgega 1000 vrstic, smo z nekim testnim vzorcem najprej ugotovili, da program napačno izračuna stanje na računu in da v izpisu manjkajo naši šumniki oziroma sičniki (glej sliko 2). Ko smo pozorneje pogledali izpis, smo videli, da je vzrok v napačno izračunanih obrestih. Ali smo opazili dve ali tri anomalije?

Z analizo programa je avtor ugotovil, da je v stavku IF OR pomotoma zamenjal z AND in pri dveh formulah za obresti je pozabil množiti s 100. Hkrati je tudi opazil, da je pri izpisu napačno postavil decimalno vejico (glej sliko 3). Kakšen je rezultat naših ugotovitev? Bomo šteli število spremenjenih vrstic, ali število sprememb v vrsticah? Ali smo našli 1 napako/1kLOC, 2 napaki/1kLOC, 3 napake/kLOC, ...? Ker natančna definicija napake zaenkrat še ni sprejeta, je potrebno rezultate predstaviti s tolerancami (npr. 2 ±1 napak/kLOC). Z internimi navodili lahko sicer sami postavimo svoje definicije in pravila in tako rešimo ta problem, vendar še vedno ostaja problematiča primerjava med konkurenčnimi produkti.

Stanje pri preverjanju		**Pravilno stanje**	
Stanje na bancnem racunu:		Stanje na bančnem računu:	
staro stanje	120,000,00	staro stanje	120,000,00
obresti	230,00	obresti	20,00
novo stanje	120.230,00	novo stanje	120.020,00

Slika 2. Opaženo stanje pri preverjanju

Slika 4 prikazuje stanje po popravilu. Koliko hib je ostalo? Takoj lahko opazimo, da sta dva sičnika napačno napisana. Ker smo presodili, da to ne bo bistveno vplivalo na kakovost, smo se odločili, da tega zaenkrat ne bomo odpravili. Bomo to anomalijo šteli kot eno ali dve hibi?

Pred popravljanjem	**Po popravljanju**
:	:
:	:
if vsota=0 and k>suma then	if vsota=0 **or** k>suma then ....
obr=k/bb	obr=k/bb***100**
print (obr,',',xx,k)	print (obr**,xx,',',k)**
obr=mm/bb	obr=mm/bb***100**
:	:

Slika 3. Spremembe v izvirni kodi

Pravilno stanje		**Popravljeno stanje**	
Stanje na bančnem računu:		Stanje na bancnem racunu:	
staro stanje	120,000,00	staro stanje	120,000,00
obresti	20,00	obresti	20,00
novo stanje	120.020,00	novo stanje	120.020,00

Slika 4. Stanje po popravilu

2.2. Problematika merjenja kompleksnosti

Danes obstaja cela vrsta modelov, ki merijo kompleksnost programskega produkta [CONTE,1986], [FENTON,1996]. Našteli bomo samo najpomembnejše: število vrstic izvirne kode (LOC), ciklomatično število [McCABE,79], funkcijske točke [JEFFERY,1993]. Mnenja o uporabnosti in natančnosti so še vedno deljena, saj se je izkazalo, da večina metrik ni v bistvu izrazito boljša od števila vrstic izvirne kode, med tem ko drugi avtorji[3] nekatere metrike odločno zavračajo. Ker se število vrstic izvirne kode najpogosteje uporablja, bomo v nadaljevanju pokazali, kje so vzroki za kritične poglede nekaterih strokovnjakov. Prvi vzrok je v veliki nenatančnosti, drugi pa v nemonotonem poteku kompleksnosti .

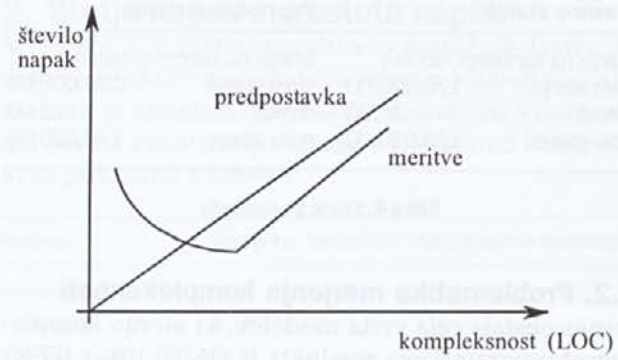
Ker ne obstaja dogovor, kaj je vrstica v izvirni kodi, se lahko meritve med seboj zelo razlikujejo (npr. razmerje med vrsticami zaključenimi z <ENTER> in tistimi, v katerih se nahajajo ukazi, je za C jezik v povprečju razmerje 1:2, za nekatere druge jezike pa celo 1:5 [JONES, 1994].

Najnovejše raziskave skušajo potrditi nekatere nepričakovane empirične ugotovitve, ki se ne skladajo s predpostavko štev. 2. Les Hatton [HATTON,1997] ugotavlja, da relacija med številom napak in kompleksnostjo **ni monotono** naraščajoča funkcija. Če manjšamo kompleksnost, se nekaj časa manjša število napak, nato pa začne zopet naraščati (glej sliko 5). Glede na njegovo raziskavo naj bi bila optimalna velikost programskega produkta nekje med 200 do 400 vrsticami. Za posplošitev te trditve bodo potrebne še nadaljnje raziskave.

Če držijo Hattonove ugotovitve, pomeni, da tudi gostote napak oziroma odpovedi ne moremo uvrstiti v ordinalno lestvico. Ali z drugimi besedami, kakovosti dveh produktov ne morem primerjati na podlagi gostote napak, saj ne vemo ali smo desno ali levo od minimuma (glej sliko 5).

Našteli bomo nekaj problemov, na katere moramo biti pozorni, ko definiramo metriko za štetje vrstic:

[3] *Npr. Casper Jones [JONES,1994]: "Nekaj zelo razširjenih metrik, ki so dokazno neuporabne, se še vedno pojavlja v knjigah, enciklopedijah in citatih. Dokler se bodo te metrike tako brezbrižno uporabljale, tako dolgo ne bo pravega programskega inženirstva, ampak neke vrste amatersko rokodelstvo..."*



Slika 5. Približna povezava med številom napak in kompleksnostjo.

1. Bomo v štetju upoštevali tudi vrstice s komentarji?
2. Je vrstica identična številu znakov, ki označujejo konec vrstice?
3. Se upoštevajo tudi prazne vrstice?
4. Se upoštevajo tudi deklaracije?

3. Kaj lahko sklepamo, če poznamo gostoto napak oziroma gostoto odpovedi?

Gostota napak je še vedno ena izmed zelo privlačnih metrik, s katerimi skušamo posredno ocenjevati kakovost produktov in procesov. Pogosto se na podlagi teh podatkov odločamo o zelo pomembnih odločitvah (npr. razporejanju resursov, o prenehanju testiranja, o prevzemu produkta itd.). Industrijsko povprečje kakovostne programske opreme v ZDA je 5 do 6 napak/1kLOC, zelo zanesljiva oprema ima približno 0,7 napak/kLOC, najbolj zanesljiva pa 0,5 do 1 napaka/kLOC [HATTON,1997a]. Poglejmo, s kakšnimi problemi se srečamo, če želimo te podatke uporabiti v komparativnih analizah:

1. Večinoma ne poznamo pravil, po katerih so drugi šteli vrstice in napake.
2. Pri podatkih, ki jih navajajo viri, pogosto ne vemo, na katero fazo se nanašajo. (Večinoma se nanašajo na fazo, v kateri uporabnik produkt uporablja.)
3. Kategorija resnosti je večinoma neznana.
4. Zelo pomembno je tudi, ali napake sproti odpravljamo, ali ne. Ker število odkritih hib in s tem tudi napak s časom uporabe narašča, potrebujemo tudi ta podatek.

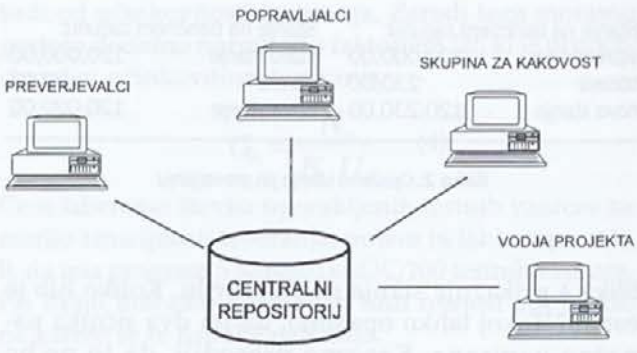
Kljub vsem tem pomanjkljivostim, je tovrsten podatek zanimiv za skupino, ki je zadolžena za kakovost. To še posebej velja, če se podatki nanašajo na fazo uporabe. Če imamo tudi podatke o tem, iz katere faze izvira največ napak, lahko te slabosti v sledečih projektih odpravimo.

Za uporabnika je seveda bolj zanimiva gostota odpovedi, saj ga ta najbolj prizadene. Ker še vedno nismo sposobni napovedovati zanesljivosti oziroma verjetnosti odpovedi, je gostota odpovedi edini indikator, na podlagi katerega lahko sklepamo o pogostosti težav oziroma o stroških popravljanja.

4. Orodja za podporo zbiranja podatkov

Zbiranje podatkov o najdenih napakah, opaženih hibah, neustreznostih in anomalijah zahteva od razvijalcev in preverjevalcev določeno dodatno delo. Zaradi tega se pogosto dogaja, da nevestno beležijo podatke. Ker jih sami ne potrebujejo, je vzrok tudi v pomanjkanju ustreznega motiva. Z relativno enostavnimi orodji za ravnanje z anomalijami (Problem Management Tools), lahko ta postopek delno poenostavimo. Dodatna težava so viri, od katerih dobivamo poročila o napakah oz. odpovedih, saj so zelo pogosto porazdeljeni (oddaljeni preverjevalci, beta preskuševalci, uporabniki itd.). Obstajajo tudi že standardi (npr. [ANSI,1988b]), ki podrobneje definirajo način beleženja in ustrezno klasifikacijo napak.

Preverjanje, popravljanje, informiranje in nadzor nad konfiguracijo so aktivnosti, ki so mnogokrat popolnoma ločene faze v življenskem ciklusu. V bistvu morajo biti te aktivnosti zaradi velikega števila informacij informacijsko tesno povezane v obliki pisnih ali ustnih sporočil. To povezavo omogočajo orodja za vodenje anomalij.



Slika 6 Centraliziran računalniško podprt proces za nadzor nad opaženimi hibami

5. Zaključek

Štetje napak oziroma gostoto napak lahko uporabimo za merjenje učinkovitosti raznih aktivnosti v razvojnem ciklusu (npr. testiranje, inšpekcije ipd.). Ta metrika je torej v nekem smislu zanimiva za ocenjevanje kakovosti procesa. Uporabnika ne zanima število napak, niti ne število hib, ampak **verjetnost pojavljanja odpovedi oziroma ocena tveganja, da bo nastopila določena škoda oziroma da bo potrebno popravilo.**

Uporabnika zanima metrika, ki se nanaša **na produkt** in ne na proces. Problematika merjenja gostote napak oziroma neustreznosti je tudi tesno povezana z metrikami, s katerimi merimo kompleksnost produkta in temeljitost testiranja. Dokler ne bodo sprejeti določeni standardi, ki bodo natančno definirali posamezne metrike, tako dolgo ne bo možno izvajati nobenih objektivnih komparativnih analiz med produkti različnih proizvajalcev.

Opozorili smo na nekatere pomanjkljivosti ali pasti, na katere naletimo pri uporabi gostote napak in odpovedi. Nedorečenost standardov lahko povzroča velike variacije pri meritvah. Temu se lahko izognemo tako, da začasno definiramo lastne standarde, po katerih se naj izvaja merjenje. Zelo težko je definirati metriko napak, saj imamo večinoma opravka s skupkom napak. Zaradi težavnosti definiranja prihaja do neponovljivosti oziroma do neobjektivnosti štetja. To pomeni, da kljub temu, da programa nismo spremenili (popravili), lahko zamenjava osebe, ki šteje napake, privede do popolnoma novih rezultatov, katere lahko pomotoma interpretiramo kot npr. izboljšanje kakovosti razvoja. V inženirstvu ta problem rešujemo z določitvijo ustreznih toleranc, ki definirajo interval, v katerem se nahaja pravilna vrednost. Zavedati se moramo, da je vsaka meritev (tudi štetje) podvržena določeni **netočnosti**. Ali poznamo tolerance naših meritev? Npr. ali vemo, da je 2 napaki/1kLOC v bistvu 1 .. 5 napake/1kLOC.

Opisana problematika še ne pomeni, da štetje anomalij nima nobenega smisla. S prispevkom smo želeli samo opozoriti na nekatere pomankljivosti, ki jih lahko delno omilimo, če se metrike lotimo na sistematičen način.

6. Reference

[ANSI,1988a]
ANSI/IEEE Std 982.1-1988, *"IEEE Standard Dictionary of Measures to Produce Reliability Software"*, The Institute of Electrical and Electronics Engineers, Inc.

[ANSI,1988b]
ANSI/IEEE Std 1044-1993, *"IEEE Standard Classification for Software Anomalies"*, The Institute of Electrical and Electronics Engineers, Inc.

[BEIZER,1990]
Boris Beizer : *"Software Testing Techniques"*, Van Nostrand Reinhold, New York, 1990, druga izdaja.

[CHILLAREGE,1995]
R. Chillarege: *"Orthogonal Defect Classification"*, 9. poglavje v knjigi: M. R. Lyu: *"Handbook of software reliability engineering"*, McGraw-Hill, 1995

[CONTE,1986]
S. D. Conte, H. E. Dunsmore, V. Y. Shen: *"Software engineering metrics and models"*, The Benjamin/Cummings Publ. Co., Inc., 1986.

[FENTON,1996]
N.E. Fenton, S.L. Pfleger: *"Software Metrics: A Rigorous Approach"*, 2 nd edition, Int'l Thomson Computer Press, London, 1996.

[HATTON,1997]
Les Haton: *"Reexamining the Fault Density-Component Size Connection"*, IEEE Software, maj/junij 1997, str. 89-97

[HATTON,1997a]
Les Haton: *"N-Version Design Versus One Good Version"*, IEEE Software, nov/dec 1997, str. 71-76

[JEFFERY,1993]
Jeffery D.R.,Low G.C., Barnes M.,*"A comparison of Function Point Counting Techniques"*, IEEE Transactions on Software Engineering, Vol.19, štev. 5, maj, 1993.

[JONES,1994]
C. Jones: *"Software Metrics: Good, bad, and missing"*, IEEE Computer, sept. 1994, str.98-100

[MCCABE,1976]
T. J. McCabe : *"A complexity measure"*, IEEE Trans. on Software, Vol. 2, št. 4, 1976, str. 308-320.

[PFLEEGER,1997]
S.L. Pfleeger et al.: *"Status Report on Software Measurement"*, IEEE Software, maj/april, 1997, str. 33- 43.

[VOAS,1998]
J.M.Voas, G. McGraw: *"Software Fault Injection: Inoculating Programs Against Errors"*, McGraw-Hill, 1998

Avtor je docent na Fakulteti za elektrotehniko, računalništvo in informatiko v Mariboru, kjer predava na dodiplomski in podiplomski stopnji in vodi Center za verifikacijo in validacijo sistemov. Na raziskovalnem področju se ukvarja predvsem z V&V tehnologijo oziroma testirnimi orodji.