XXVII JUGOSLOVENSKA KONFERENCIJA ETAN-a, STRUGA, 6 — 10. JUNA 1983. GODINE

Armin PAVIĆ
Emil RIFATI
Milan ZORIĆ
ELEKTROTEHNIČKI FAKULTET
ZAGREB, Unska 3

PROGRAMSKA PODRŠKA U TELEMATICI KAO TEHNOLOŠKI PROIZVOD

SOFTWARE PRODUCTION TECHNOLOGY IN TELEMATICS

SADRŽAJ - Programska podrška postaje veoma značajna komponenta telekomunikacijskih, a time i budućih telematičkih sistema. Dosadašnja istraživanja procesa proizvodnje programske podrške dovela su do niza saznanja o tom procesu, koja ga približavaju nivou znanja i metodologiji rada u tradicionalnim inženjerskim disciplinama, te možemo govoriti o tehnologiji proizvodnje programske podrške. U ovom radu analizirani su odredjeni aspekti definicije sistema, osiguranja ispravnosti te vrednovanja programske podrške, posebno imajući u vidu pomak ka standardizaciji postupaka i funkcijskih komponenata u telematici.

ABSTRACT - Software becomes a very important component of telecommunication, and thus future telematic systems. Research in software engineering has led to an accumulation of knowledge about software production process that begins to resemble the knowledge and methodologies of more traditional engineering disciplines allowing us to speak of software production technology. In this paper certain aspects of system definition, validation and evaluation methods are analysed, bearing in mind a trend towards standardization of procedures and functional components in telematics.

1. UVOD

Razvoj telematičkih sistema treba danas promatrati u sklopu razvoja ka digitalnoj mreži integriranih usluga. Pri tome integrirana mreža ne daje samo komunikacijsku osnovu, već može dati niz složenijih usluga, obzirom na prisutne kapacitete procesora, baze podataka, informacijske sistem itd.

Većina funkcija ovakvog sistema realizirana je u odgovarajućoj programskoj podršci, čime ona dobija na značaju i na odredjeni način postaje dio naše svakodnevnice. Pri tome susrećemo niz zahtjeva na tu programsku podršku, koje je teško istovremeno postići. S jedne strane traži se vrhunska ispravnost (posebno u funkcijama mreže), s druge strane prikladnost i prilagodjenost različitim potrebama neračunarski orijentiranih korisnika (u aplikacijskim funk-

cijama). Polazeći od dosadašnjih iskustava pokušati ćemo analizirati neke značajne karakteristike sadašnjeg stanja te naznačiti moguće pravce razvoja.

Današnje tehnike razvoja programske podrške često dozvoljavaju individualnim programerima znatnu slobodu u odlučivanju kako dizajnirati komponente programske podrške. Modul mora zadovoljiti specifikaciju - nema zahtjeva da se traže postojeća rješenja ili standardne komponente, koje ako se dodaju ili oduzmu neki parametri mogu biti upotrebljene umjesto novih. Pored umnažanja truda u razvoju taj proces proizvodi sisteme koji s vremenom mogu gubiti svoja početna svojstva, stoga što ispravljanje i održavanje može zasjeniti strukturu koju je sistem u početku posjedovao. Ovaj fenomen posebno dolazi do izražaja ako se pri održavanju ne obraća pažnja na dokumentaciju, ponovno dizajniranje i slične administrativne nužnosti. Ako tome dodamo rast sistema, uz rastuću kompleksnost, što je osnovna karakteristika telekomunikacijskih sistema, dolazimo u situaciju kada promjene i dodavanje funkcija postaju sve teže, a ponašanje sistema postaje nepredvidivo.

Očito je da neophodno postići bolju ravnotežu izmedju slobode inovacije i discipliniranog razvoja programskih komponenata. Za očekivati je da će novi programski sistemi biti izgradjivani od verificiranih komponenata koje će biti pogodne za modificiranje, rekonfiguraciju i ponovnu upotrebu. Uloga programera biti će stoga pomaknuta prema primjeni metoda dizajniranja i strukturiranja pri izgradnji sistema od standardnih funkcijskih komponenata.

Na putu do takvog načina izgradnje programske podrške predstoje opsežna istraživanja koja moraju obuhvatiti istraživanja optimalne veličine modula, spektra funkcija, alternativnih implementacija, metoda zadavanja, testiranja odnosno verifikacije itd. U ovom radu obradjeni su slijedeći aspekti problema:

- uključivanje korisnika u fazu definicije i razvoja sistema sa ciljem osiguranja prikladnosti sistema svim zahtjevima korisnika,
- osiguranje i kontrola ispravnosti, da bi se postigla odgovarajuća pouzdanost rada sistema, i
- vrednovanje sistema u svim fazama proizvodnje (životnog ciklusa) sa ciljem osiguranja minimalnih troškova i maksimalne efektivnosti sistema.

2. DEFINICIJA SISTEMA

Problemi koji se pojavljuju u realizaciji programske podrške su mnogobrojni ali se ističu dva oblika: problem efikasnog i točnog specificiranja i problem stabilnosti programa. Prvi oblik je odraz nemogućnosti da se ostvari jednostavan sistem transformacije aplikacijskog područja u specifikacije (definiciju sistema) na potpun, točan i efikasan način. Put kojim se pokušava riješiti taj prob-

lem je formalno i poluformalno specificiranje različitim postupcima i metodama. Stabilnost programa vezana je za pojavu da se aplikacijski program proizvede za odredjenu okolinu ali dolazi do promjene okoline (objekata, ponašanja, procesa pa prema tome i specifikacija), bilo pod utjecajem samog programa ili nekih drugih razloga.

Tri osnovna područja sačinjavaju strukturu problema. To su područja aplikacijske okoline, specifikacija i programa (programskog sistema). Aplikacijsko područje čini stvarna okolina i primjena, svi procesi, podaci i akteri realnog sistema. Specifikacijsko područje je prvi korak konkretizacije problema, to je aplikacijsko područje, zahtjevi i opisi procesa i podataka zadani na neki formalni ili neformalni način (jezično, simbolički, grafički, pomoćnim metodama i sistemima, visokim jezicima itd.). Programsko područje je skup finaliziranog proizvoda, to je program ili sistem programa namjenjen za rad u realnoj okolini.

Proces proizvodnje programa je u osnovi sadržan u postupku opisa, izražavanja i transformiranja aplikacijskog područja u specifikacije i daljnja pretvorba specifikacija u programsko područje. Riješiti probleme koji nastaju u tim procesima, a kasnije i u radu i održavanju aplikacijskih programa, značilo bi riješiti tehnološki dio problema proizvodnje programske podrške.

Analiza problema započinje zaokruživanjem specifičnosti i elemenata koje ulaze u sistem i njihovo odvajanje od akcija i elemenata koje se odvijaju i nalaze u totalnoj okolini. Prije svega jasno je da se sve akcije i dogadjaji, tj. cijelokupna aktivnost u korisničkom području, ne može riješiti odjednom i kroz jedan programski sistem. Dio koji smo ograničili kao problem (na osnovu zahtjeva) mora biti rješiv u okviru tehnoloških mogućnosti i trenutačnih znanstvenih dostignuća. Zaključno, potrebno je odmjeriti veličinu sistema i opsežnost zahtjeva kako bi se proces razvoja i proizvodnje smjestio u razumne vremenske rokove. Zahvaćanje odnosa iz ovog dijela pristupa problemu može imati različite aspekte:

- u vezi s razlikama izmedju krajnjeg korisnika i operatera treba sagledati razliku izmedju okoline, u kojoj će djelovati sistem u jednom i drugom slučaju, i ograničenja i definicija funkcija na tom planu.
- oblik zatvorenosti i samostalnosti programskog sistema zavisi i od složenosti problema. Relativno jednostavni sistemi mogu se isporučiti u obliku "ključ u ruke" (paketa) uz detaljna objašnjenja i školovanje korisnika. Kompleksni problemi uključuju aktivno vrednovanje od strane korisnika i sudjelovanje u svim fazama razvoja. To predstavlja jedan oblik trajnog definiranja i većeg utjecaja korisnika u životnom ciklusu proizvoda.
- izbor, selekcija i mjera atributa kvalitete koje sistem mora zadovoljiti posebno je važni element analize. Uz definiciju sistema osim troškova i rasporeda vezan je pojam korisnosti kao složeni oblik opisa pouzdanosti, održavanja, razumljivosti, prilagodljivosti i efikasnosti.

- za kompleksne i nedefinirane sisteme po kriterijima definicije, dizajna i implementacije važnu mogućnost predstavljaju automatski aplikacijski sistemi, koji omogućuju da se iz jednostavnih ali posebno pripremljenih modula i struktura izgradjuju sistemi u toku upotrebe od samog korisnika. Ovaj koncept posebno je upotrebljiv u telematičkim sistemima u budućnosti.

Automatski aplikacijski sistemi samo su jedan oblik procesa definicije sistema i sadrže navedene elemente i transformacije. Metode razvoja mogu se uglavnom svrstati u tri grupe prema stupnju prebacivanja aktivnosti na računalo:
- u prvoj grupi želi se realizirati oslobadjanje programera od kompliciranih i teških poslova, optimizacija i lokalnog opterećenja, tj. od rješavanja problema koji usporavaju dinamiku programiranja. Ova grupa očito spada u onaj dio aktivnosti razvoja programske podrške gdje se pojedinačno realiziraju programi a ostali dio definiranja i specificiranja je već riješen.
- u drugoj grupi automatiziraju se postupci koji bi trebali služiti kao pomoćno sredstvo u razvoju i dizajnu programske podrške. Ako znamo da je put od specifikacije do programa sastavljen upravo od postupnih transformacija cjeline i pojedinih dijelova jasno je da će ovaj pristup osim u pisanju programa poslužiti i u razvojnom procesu.
- u trećoj grupi nalaze se automatizirani procesi programske konstrukcije kao naziv za podizanje automatskih aktivnosti na viši nivo i put k formiranju sistema za automatsku sintezu programa. Analizirajući mjesto ovakvih pristupa, uz uključivanje korisnika u definiranje vlastitih specifikacija i realizaciju programa, vidimo da se nalaze i u transformacijama aplikacijskog područja prema specifikacijskom i specifikacijskog područja prema programima.

Aktivnosti definiranja vezane su uz tehniku opisa, strukturiranja i formaliziranja specifikacija i zahtijeva. Opis i strukturiranje mogu se provesti na različite načine: analizom na bazi apstraktnih automata, semantičkom i strukturalnom analizom. Semantička analiza temelji se na mogućnosti koceptualizacije jezičnim putem. Identificiraju se i klasificiraju izrazi koji se upotrebljavaju u opisu sistema. Iz klasifikacije izraza koji opisuju objekte izvodi se struktura. Strukturnom analizom identificiraju se procesi na temelju spajanja cjelokupnog sistema u vremenu.

Definicija elemenata korisničkih zahtjeva i opisa mora biti precizna. Strogom sintaksom opisuju se: objekti (nazivi stvari), dogadjaji, veze (kako se odnose objekti medjusobno i u relaciji s dogadjajima), atributi (opisuju dogadjaje i objekte).

U nastavku se opisuje okolina izvan područja definiranja: identificiraju se objekti koji okružuju sistem, identificiraju se dogadjaji u okolini a definirani podaci i dogadjaji strukturiraju se u obliku dijagrama toka podataka u okolini sistema koji se razvija.

Analizom rezultata, zahtjeva i mogućnosti realizacije završava se ova faza definiranja sistema. Kao popratni rezultat odredjuju se atributi kvalitete proizvoda i njihova mjera, što uslovljava mnoge aktivnosti daljnjeg razvoja, planiranja i procjena pa i troškovnu analizu.

Formalizacijom specifikacija faza definiranja sistema ulazi u završni dio. Logički povezane i opisane specifikacije služe kao ulaz u fazu razvoja programskog sistema i omogućuju korisniku kontrolu. Zbog toga oblik formalnog prikaza ovisi o načinu daljnjeg razvoja i zahtjeva korisnika. Neki oblici formalizacije mogu biti dijagrami toka podataka, dijagrami kontrolnih struktura, sistem definicije podataka i sl.. Formaliziranje može biti provedeno i sa posebnim ili standardnim programskim jezicima koji kroz apstraktne podatke i procedure opisuju funkcije i karakteristike definiranog sistema. Osim olakšanja i automatizacije postupaka formaliziranja ovakvi pristupi, koji su dio novih napora i istraživanja na planu definicija sistema, osigurava u bolju i točniju komunikaciju s korisnikom i razumljivost, te efikasniju verifikaciju ispravnosti i logike specifikacija i zahtjeva.

3. OSIGURANJE ISPRAVNOSTI PROGRAMSKE PODRŠKE

Dok se razvoj teorije verifikacije i testiranja programske podrške nastavlja, osiguranje ispravnosti se u najvećem broju slučajeva postiže iscrpnim testiranjem. Pri tome je za očekivati da će najprije biti iscrpljena intuicija programera, koja mu govori koje puteve u programu testirati.

Kod testiranja programske podrške možemo u osnovi razlikovati funkcionalno testiranje, te strukturalno testiranje. Funkcionalno testiranje zanemaruje internu strukturu programa a test podaci se konstruiraju iz funkcionalnih svojstava programa specificiranih u zahtjevima. Kako bi funkcionalno testiranje bilo dobro obavljeno tester mora biti dobro upoznat sa specifikacijama, te imati duboko razumijevanje kako program radi. Howden /1o/ ističe prednosti ovakvog pristupa u odnosu na strukturalno testiranje, no treba reći da je istraživanje vršeno samo na jednom tipu programske podrške. Nedostaci ovakvog testiranja su u tome da se ignoriraju važna svojstva programa koja nisu navedena u zahtjevima već proizlaze iz dizajna odnosno realizacije. Broj test slučajeva može preći mogućnosti testiranja. No i-pak, osnovni nedostatak je nemogućnost odredjivanja kompletnosti skupa testova.

Strukturno testiranje podrazumijeva analizu reprezentacije podataka te kontrolnih puteva u programu. Program može biti prikazan kao usmjereni graf u kojem čvorovi predstavljaju naredbe a grane predstavljaju mogući tok izvodjenja naredbi. Strukturno testiranje podrazumijeva izbor skupa podataka koji daju dobro prekrivanje grafa programa. Mjera prekrivenosti grafa koristi se kao pokazatelj kako daleko se otišlo u testiranju. Ovisno o primijenjenom načinu prekrivanja, rezultati su različiti i kreću se od 67 - 100% otkrivenih grešaka /5/, /6/, /7/.

Bez obzira što strukturno testiranje garantira izvjestan vid prekrivanja

programa, ono ne garantira korektnost testiranih komponenata. Naredba, ili skup naredbi mogu biti testirane a da zahvaljujući izboru test podataka prisustvo greške ostane neprimijećeno.

Postoje pristupi kojima se pokušava iskoristiti dobre strane oba ova načina testiranja. Osnovna ideja je da se za testiranje najprije odabire skup test podataka pri čemu se koriste metode funkcionalnog i strukturnog testiranja. Po izvršenju prvobitno odabranih testova moguće je uočiti dijelove programa koji su nedovoljno testirani, te generirati podatke koji će kontrolirati upravo te dijelove. Takav sistem opisan je detaljnije u /1/.

Novija istraživanja /7/ utvrdjuju, razradjuju, proširuju i eksperimentiraju sa idejama i teorijama testiranja programske podrške. Čini se da prevladava mišljenje da ne treba očekivati tako značajnu teoriju testiranja, koja bi vodila ka potpuno automatiziranim sistemima za testiranje. Umjesto toga, zahtijevati će se od osobe koja vrši testiranje da na sustavan način koristi svoju intuiciju i znanje o prirodi problema, kako bi u testiranju pokušao otkriti niz specificiranih tipova grešaka.

Bez obzira na značaj testiranja u osiguranju kvalitete programske podrške, kao i očekivanje novih rezultata u teoriji testiranja, mora se istaći da testiranje ni u kom slučaju ne može nadomjestiti nedovoljan napor uložen u fazi dizajniranja programa. U tom smislu može se reći da metode verifikacije programske podrške daju dublji uvid u proces izgradnje programske podrške. Niz autora ističe zahtjev da program i dokaz korektnosti moraju biti paralelno radjeni, pri čemu dokazi trebaju usmjeravati razvoj programa /12/, /9/, /8/. Iz toga proizlazi i zahtjev za boljim školovanjem kako bi se promjenio pristup i navike, onih koji učestvuju u razvoju programske podrške.

4. VREDNOVANJE KVALITETE

Proces vrednovanja kvalitete potrebno je ostvarivati tokom cijelog životnog ciklusa programske podrške, da bi se pribavili podaci neophodni za praćenje, organiziranje, te planiranje njezine proizvodnje i korištenja. Pri tome se postavlja pitanje mjerila kvalitete. S obzirom na dinamički karakter programske podrške, koja se potpuno realizira tek kroz svoje djelovanje, nameću se dva osnovna kriterija kvalitete. Prvi kriterij jest efektivnost djelovanja, u smislu stupanja izvršavanja postavljenih zahtjeva, a drugi kriterij su troškovi potrebni za ostvarenje takove efektivnosti.

Specifičnost programske podrške, kao proizvoda, jest smanjenje efektivnosti usljed neprikladnosti za pojedine aplikacije. Vrednovanje prikladnosti (uz sudjelovanje i potencijalnog korisnika) treba stoga da, u vidu kontrole kvalitete, postane sastavni dio razvoja programske podrške. Kao osnovno mjerilo prikladnosti može se uzeti kompletnost svih dijelova i funkcija programskog proizvoda.

Glavni uzrok neefektivnosti ipak su, kako je prikazano u /3/, greške programske podrške. Vrednovanje ispravnosti predstavlja stoga značajan vid kontrole kvalitete, koji mora pratiti sve faze proizvodnje programske podrške. Pouzdanost programskog proizvoda, definirana kao vjerojatnost pojave greške u toku odredjenog vremena (ili broja izvršenja), predstavlja vremensku projekciju njegove (ne) ispravnosti, a time osnovno mjerilc efektivnosti. Pouzdanost se kvantificira matema-matematičkim modelima, čiji parametri, kao što su učestalost grešaka te broj sadržanih neispravnosti, se procjenjuju uglavnom na temelju podataka o greškama registriranim tokom ispitivanja ispravnosti. Na ispitivanje ispravnosti otpada često najveći dio troškova proizvodnje, pa je pravilno dimenzioniranje ove faze od posebnog značaja. U tu svrhu moguće je, kako je pokazano u /4/, koristiti modele pouzdanosti, sa ciljem procjene količine (vremena) testiranja potrebne da se učestalost grešaka smanji na odredjenu, prihvatljivu razinu. Pouzdanost postaje ovako mjera gotovosti programskog proizvoda, koju treba postaviti već prilikom planiranja proizvodnje.

Ako neprikladnost i neispravnost degradiraju efektivnost programske podrške, njezinim ispravljanjem te izmjenama (tj. održavanjem), efektivnost se povećava. Pogodnost za održavanje treća je karakteristika efektivnosti, a može se izraziti i kao vjerojatnost da će sistem biti vraćen u operativno stanje unutar odredjenog vremena održavanja. Procjena ove karakteristike predstavlja najveći problem, a moguća je na osnovu podataka o održavanju prethodnih sistema, organiziranosti službe održavanja, te raspoloživih automatiziranih pomagala.

Procjena efektivnosti, u svrhu planiranja i praćenja kvalitete, moguća je primjenom teorije pouzdanosti, na osnovu procjene opisanih karakteristika prikladnosti, pouzdanosti, te pogodnosti za održavanje. Još neke mogućnosti procjene efektivnosti, na temelju podataka o korištenju, te pouzdanosti, opisane su u /3/.

Postignuta efektivnost ima i svoju cijenu. Tu se javlja i drugi element, koji treba uzeti u obzir prilikom planiranja i organizacije proizvodnje, a to su troškovi proizvodnje i korištenja programske podrške. S obzirom na to, možemo izdvojiti dva tipa pogrešnog pristupa proizvodnji programske podrške. Jedan tip pogreške jest inzistiranje na što višem nivou pouzdanosti, a to rezultira sve naglijim porastom troškova proizvodnje, iako operativni zahtjevi mogu biti ostvareni i uz niži nivo ugradjene pouzdanosti.
Drugi tip pogreške jest inzistiranje na "spretnim" kodnim kombinacijama, koje štede vrijeme i memoriju stroja (troškove), ali zato, svojom improviziranošću i nestandardiziranošću, predstavljaju mnogo značajniji izvor neispravnosti i nepogodnosti za održavanje (tj. neefektivnosti) programske podrške.

Ovdje se, kao pogodno i upotrebljivo mjerilo kvalitete (tj. korisne vrijednosti) programske podrške, nameće omjer željene efektivnosti i, za to potrebnih, troškova. Maksimiziranje ovog kvocijenta, uz zadovoljenje kriterija minimalne zah-

tijevane efektivnosti i maksimalno dozvoljenih troškova, daje mogućnost optimiziranja proizvodnje i korištenja programske podrške.

5. ZAKLJUČAK

Programska podrška koju susrećemo u telematici obuhvaća širok spektar od sistemske programske podrške, programske podrške za obradu prometa odnosno održavanje u komutacijskim čvorovima do aplikacijske programske podrške sistema oslonjenih na telekomunikacijsku mrežu. Da bi bio moguć pomak ka standardizaciji potrebno je identificirati funkcije koje su za to pogodne, istražiti optimalnu veličinu blokova, te dalje razviti metode izgradnje, dokumentiranja, testiranja i verifikacije, te ocjene efektivnosti imajući u vidu specifičnosti područja primjene. Potrebno je stoga analizirati postojeću programsku podršku, programska pomagala, te način prenosa tehnologije proizvodnje programske podrške do širokog kruga programera i korisnika.

LITERATURA:

/1/ Clarke L.A.: "Automatic Test Data Selection Techniques" in Infotech State-of-the-Art Report "Program Testing" 1979.

/2/ The Management of Software Engineering, IBM Systems Journal, Vol 19, No 4, 1980.

/3/ A.Pavić, "Greške i efektivnost programske podrške", JUREMA, Zagreb, 1981.

/4/ A.Pavić, "Teorija pouzdanosti i kvaliteta programskih sistema", Elektrotehnika, broj 2, str. 111-115, ožujak 1981.

/5/ R.E.FAIRLEY, "Static Analysis and Dinamic Testing of Computer Software", Computer April 1978.

/6/ E.F.Miller, Jr. "Program Testing: Art Meets Theory", Computer, July 1977.

/7/ Special collection an Program Testing IEEE Trans. an Software Engineering Vol. SE 6 No 3 May 1980.

/8/ R.C.Linger, H.D.Mills, B.I.Witt: "Structured Programming" Addison Wesley 1979.

/9/ R.T.Yeh, Ed.: "Current Trends in Programming Methodology", Vol. II Program Validation, Prentice-Hall, Inc. 1977.

/10/ W.E.Howden, "Functional Program Testing", IEEE Trans. on Software Engineering Vol. SE-6 No 2 Mr 1980 pp 162-169.

/11/ D.Gries: Current Ideas in Programming Methodology Lecture Notes in Comp.Sci., Springer-Verlag 1979.