1 POUZDANOST KOMPONENTI

1.1 Zrnatost sustava

Sustav je skup komponenata, podskupova ili dijelova dizajniranih kako bi se ostvarila željena funkcija s odgovarajućim stupnjem pouzdanosti. [WEIBULL]

Sustav je dio svijeta koji promatramo. Većina elektrotehničkih i računalnih sustava sastoji se od podsustava, koji mogu imati i svoje podsustave. Kada opisujemo sustav, opisujemo ga do onog stupnja zrnatosti koja je potrebna za razumijevanje rada sustava.

Zrnatost ovisi o dostupnosti podataka i najnižem operativnom članu.

1.1.1 Primjer

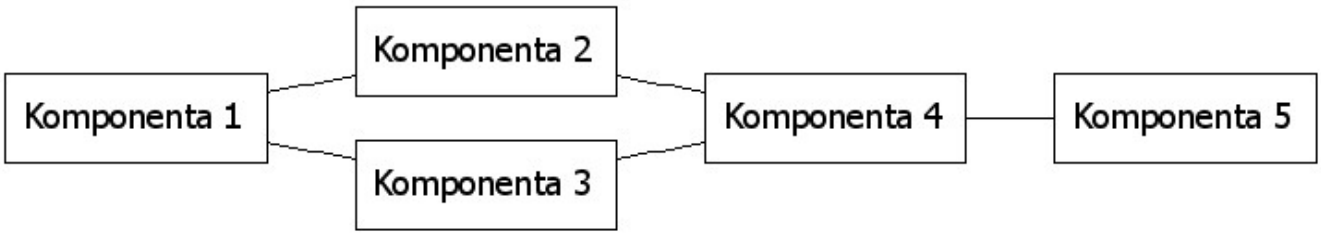
Za opisivanje rada osobnog računala ne mora biti potrebno opisivati dijelove do razine tranzistora ili čak niže. Ako je potrebno opisati prijenos podataka između glavnih dijelova sustava, potebne komponente su CPU, memorija, vanjske jedinice i sabirnice. Ako želimo promatrati gdje se smještaju potrebni podaci, CPU možemo prikazati kao skup registara (posebnih registara i registara opće namjene) i internih sabirnica, a memoriju kao skup memorijskih lokacija, memorijskog adresnog registra MAR, memorijskog podatkovnog registra MDR i internih sabirnica. Niža podjela od toga nije potrebna za razumijevanje rada sustava.

1.2 Blok-dijagram pouzdanosti

Blok-dijagram pouzdanosti (engl. *Reliability Block Diagram* – RBD) opisuje povezanost komponenti unutar sustava obzirom na pouzdanost. [WEIBULL]

Blok-dijagram pouzdanosti nije fizička reprezentacija sustava. Pokazuje kako su međusobno povezane komponente i njihov utjecaj na rad sustava. Ako komponenta mora raditi da bi sustav radio, tada je komponenta spojena serijski, a ako jedna od više komponenti mora raditi da bi sustav radio, te komponente su spojene paralelno (komponente su redundantne). Primjer je prikazan na slici 1.

Nezavisne komponente su takve komponente čiji kvar ne utječe na pouzdanost ostalih komponenata sustava.

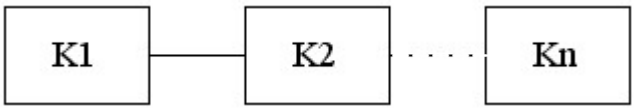


Slika 1: Blok dijagram pouzdanosti

U primjeru sa slike komponenta 2 i komponenta 3 su u paraleli, i njihov blok je u seriji s komponentama 1, 4 i 5. Pojednostavljeno, sustav radi ako postoji barem jedan put krenuvši od početka do kraja gdje sve komponente rade. U primjeru to mogu biti komponente 1, 2, 4, 5 ili 1, 3, 4, 5.

1.3 Sustav u seriji

Sustav u seriji (slika 2) takav je sustav čije komponente obavljaju specifičnu zadaću i svaka od komponenti mora raditi kako bi sustav radio. U slučaju nezavisnih komponenti pouzdanost takvog sustava iznosi [WEIBULL]:



Slika 2: Sustav u seriji

$$R_S = {}_{i=1}\Pi^n R_i = \\ = R_1 \times R_2 \times \ldots \times R_n,$$

pri čemu je:
R_S – pouzdanost sustava
R_i – pouzdanost *i*-te komponente

Pouzdanost sustava u seriji uvijek je manja od najmanje pouzdanosti komponenata, jer sustav neće raditi i kad otkaže komponenta s najmanjom pouzdanosti ali i kada otkaže bilo koja druga serijska komponenta.

Primjer sustava u seriji je osobno računalo gdje svaka komponenta izvršava svoju funkciju.

1.3.1 Primjer

Neka se sustav sastoji od tri nezavisne serijski spojene komponente čije su pouzdanosti nakon t = 1000 h redom $R_1(t)$ = 95%, $R_2(t)$ = 90%, $R_3(t)$ = 85%. Kolika je pouzdanost sustava nakon vremena t?

$$R_S = {}_{i=1}\Pi^3 R_i = \\ = R_1 \times R_2 \times R_3 = \\ = 0.95 \times 0.90 \times 0.85 = \\ = 0.72675 =$$

= 72.675%

Pouzdanost sustava nakon 1000 sati iznosi 72.675% i manja je od najmanje pouzdanosti bilo koje komponente. Iz toga je moguće izraziti i ostale podatke koje opisuju pouzdanost: *MTBF* i λ, kao i odrediti graf pouzdanosti i graf raspodjele kvarova.

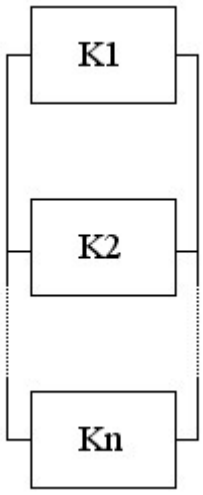
1.4 Sustav u paraleli

Sustav u paraleli (slika 3) takav je sustav čije funkcioniranje osiguravaju redundantne komponente kako bi pouzdanost sustava bila veća. Barem jedna komponenta mora raditi kako bi sustav radio. Pouzdanost sustava veća je od najveće pouzdanosti komponenata.

Kod sustava u paraleli uvodi se pojam *nepouzdanosti*, pojma komplementarnog pouzdanosti, a predstavlja vjerojatnost pojave kvara u određenom trenutku.

$$F_S = {}_{i=1}\Pi^n F_i =$$
$$= F_1 \times F_2 \times \ldots \times F_n,$$

pri čemu je:
F_S – nepouzdanost sustava
F_i – nepouzdanost *i*-te komponente



Slika 3: Sustav u paraleli

Pouzdanost i nepouzdanost međusobno su komplementarne funkcije, dakle:

$$R = 1 - F$$

pa je pouzdanost sustava u paraleli [WEIBULL]:

$$R_S = 1 - F_S =$$
$$= 1 - {}_{i=1}\Pi^n F_i =$$
$$= 1 - F_1 \times F_2 \times \ldots \times F_n =$$
$$= 1 - [(1 - R_1) \times (1 - R_2) \times \ldots \times (1 - R_n)]$$

Sustavi u paraleli koriste se u biomedicini, avioindustriji, računarstvu, u vojne svrhe...

1.4.1 Primjer

Neka se sustav sastoji od tri nezavisne paralelno spojene komponente čije su pouzdanosti nakon t = 1000 h redom $R_1(t)$ = 95%, $R_2(t)$ = 90%, $R_3(t)$ = 85%. Kolika je pouzdanost sustava nakon vremena t?

$R_S = 1 - F_S =$
$= 1 - {}_{i=1}\Pi^n F_i =$
$= 1 - F_1 \times F_2 \times ... \times F_n =$
$= 1 - [(1 - R1) \times (1 - R2) \times (1 - R3)] =$
$= 1 - 0.05 \times 0.10 \times 0.15 =$
$= 1 - 0.00075 =$
$= 0.99925 =$
$= 99.925\%$

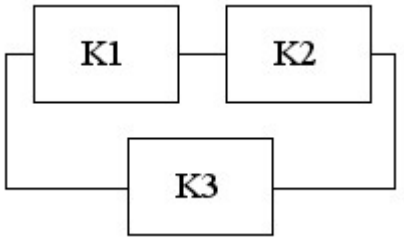
Pouzdanost sustava nakon 1000 sati iznosi 99.925% i veća je od najveće pouzdanosti bilo koje komponente. Iz toga je moguće izraziti i ostale podatke koje opisuju pouzdanost: *MTBF* i λ, kao i odrediti graf pouzdanosti i graf raspodjele kvarova.

Nepouzdanost, odnosno vjerojatnost pojave kvara nakon 1000 sati u ovom sustavu iznosi 0.00075, odnosno 0.075%.

1.5 Kombinacija serije i paralele

1.5.1 Primjer

Neka se sustav sastoji od tri nezavisne komponente kako je pokazano na slici 4. Njihove pouzdanosti nakon t = 1000 h iznose redom $R_{K1}(t)$ = 95%, $R_{K2}(t)$ = 90%, $R_{K3}(t)$ = 85%. Kolika je pouzdanost sustava nakon vremena t? Izračunaj *MTBF* i λ tog sustava.



Slika 4: Primjer

1.5.2 Rješenje

Komponente K_1 i K_2 su u seriji a komponenta K_3 je u paraleli s obje. Izračunajmo pouzdanost gornje grane:

$R_{12} = R_1 \times R_2$
$= 0.95 \times 0.90 = 0.855$

Sada možemo izračunati pouzdanost cijelog sustava:

$R_S = 1 - [(1 - R_{12}) \times (1 - R_3)] =$
$= 1 - 0.145 \times 0.15 = 0.97825$

Izračunali smo pouzdanost nakon 1000h. Iz toga možemo izračunati učestalost kvarova λ:

$R_S(t) = e^{-\lambda t}$, iz čega slijedi
$\lambda = - (\ln R_S) / t = 21.99 * 10^{-6}\ h^{-1}$

Slijedi *MTBF*:

$MTBF = 1 / \lambda = 45475.18$ h

2 SIGURNOST SUSTAVA

Sigurnost sustava može se promatrati dvojako. Ako sigurnost poistovjetimo s pouzdanošću, odnosno vjerojatnošću da sustav obavlja zamišljenu funkciju, potrebno je utvrditi pouzdanost sustava[1]. S druge strane, sigurnost se odnosi i na ljudsku sigurnost prilikom rada na sustavu.

2.1 Kvarovi, greške i načini rada u okviru sigurnosti

2.1.1 Kvarovi

Kvar je definiran na razini sustava, ako sustav ne izvršava zadanu funkciju. Greška je neispravnost komponente, koja može uzrokovati kvar i događa se na nižem nivou nego kvar, odnosno razina komponente niža je od razine sustava.

Kako bi se razlikovali uzroci kvarova, definiramo sljedeće vrste:

- Primarni kvar je otkaz nastao prilikom rada sustava u predviđenoj okolini.
- Sekundarni kvar je otkaz nastao prilikom rada sustava izvan predviđenih radnih uvjeta.

Sustav se smatra sigurnim ako se u 30 godina rada izgubi manje od jednog ljudskog života.

2.1.2 Greške

Greške se također mogu podijeliti u nekoliko vrsta: naredbena, kritična i katastrofalna greška. Naredbena greška ne isključuje mogućnost da je riječ istovremeno o kritičnoj ili katastrofalnoj greški.

Naredbena greška izaziva neispravnost u radu stroja iako radi kako je dizajnirana. Greška je posljedica nepredviđenog ulaza.

Kritična i katastrofalna greška opisuju posljedice koje neispravnost sustava može izazvati po ljude.

- Kritična greška ugrožava jednog ili nekoliko ljudi.
- Katastrofalna greška ugrožava znatan broj ljudi.

2.1.3 Načini sigurnog rada

Sustave je potrebno dizajnirati tako da osiguravaju takav način rada da ne utječu na ljudsku sigurnost.

1 Više o pouzdanosti sustava: http://www.srednja-dugoselo.hr/predmeti/diou/diou-1-uvod.pdf

Načini sigurnog rada su:

- Vjerojatnosno sigurni (engl. *probabilistically safe*). Sigurnost takvih sustava uzrokuje manje od jednog izgubljenog ljudskog života u milijardu sati (oko 11400 godina). Koristi se redundancija.
- Integralno sigurni (engl. *inherently safe*). Sustav je tako fizički napravljen da ne može uzrokovati štetu. Npr. stari semafori na željeznici imali su ručku koja je u horizontalnom položaju značila prolaz a u vertikalnom zaustavljanje. Ako se nešto semaforu dogodi i zbog utjecaja gravitacije ručka padne, vlakovi će se morati zaustaviti bez obzira na to što se ispred događa na pruzi. U suprotnom slučaju (da zbog greške pokazuje stalan prolaz) moglo bi doći do sudara.
- Sigurno gašenje (engl. *fail-safe*, jap. *poka-yoke*). Sustav se u slučaju pojedinačnog kvara mora automatski ugasiti na siguran način. Mora se izbjeći ili minimizirati šteta prema ljudima i drugim uređajima i/ili komponentama u sustavu. Npr. nuklearne rakete se neće moći lansirati u slučaju gubitka komunikacije.
- Otpornost na pogreške (engl. *fault-tolerant*). Sustav može nastaviti rad unatoč greškama, iako sama funkcionalnost može biti narušena.

2.2 Analiza sigurnosti

2.2.1 Analiza neispravnosti i učinaka

Analiza neispravnosti i učinaka (engl. *Failure mode and effects analysis* – FMEA) jedna je od tehnika kojom se pokušava utvrditi odgovara li sustav zadanim karakteristikama sigurnosti. Tokom dizajniranja, sustav je potrebno predstaviti blok-dijagramom i razmatra se što se događa kada pojedini blok nije dovoljno siguran. Dijelovi sustava se prilagođavaju tako dugo dok sustav u cijelosti ne zadovolji očekivanja sigurnosti.

Posljedice se procjenjuju prema sljedećim kriterijima sa skalom od 1 do 10:

- Ozbiljnost posljedica (engl. *Severity*) – S [1 .. 10]
- Vjerojatnost pojave (engl. *Likelihood of occurence*) – O [1 .. 10]
- Nemogućnost kontrolnih mehanizama da primjeti posljedicu (engl. *Detection*) – D [1 .. 10]

Rizik posljedice određuje se pomoću umnoška ovih vrijednosti:

RPN = S × O × D,

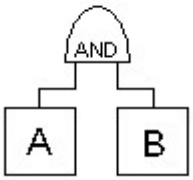
pri čemu je
RPN – Risk Priority Number (broj prioriteta rizika)

Rizik ima skalu od 1 do 1000, i što je veći rizik, to komponenta ima veći prioritet pri analizi i odlučivanju o mjerama.

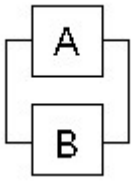
2.2.2 Analiza stablom grešaka

Analiza stablom grešaka (engl. *Fault Tree Analysis*) tehnika je izrade dijagrama (*Fault Tree Diagram* – FTD) kojom na vrhu predstavimo neželjeni učinak ili ponašanje sustava. Kreira se logička shema tako da se na svakoj razini opiše što može dovesti do učinka na višoj razini, korištenjem logičkih vrata.

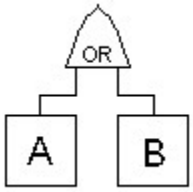
Neka postoje dva ulazna događaja A i B, koja na višoj razini mogu uzrokovati izlazni događaj. Ako je dovoljan jedan od ta dva događaja da iniciraju izlazni događaj, koriste se logička vrata OR, a ako su potrebna oba događaja, koriste se vrata AND. Ako usporedimo to s blok-dijagramom pouzdanosti RBD, tada logičkim AND predstavljamo paralelne komponente (*usp*. slike 5 i 6), a logičkim OR serijske (*usp*. slike 7 i 8).



Slika 5: FTD
A AND B



Slika 6: RBD
A || B



Slika 7: FTD
A OR B

Slika 8: RBD
A -> B

3 INTEGRIRANI SUSTAVI

Integrirani sustavi su sustavi s mikroprocesorom koji obavljaju predefinirane specifične funkcije a smješteni su u zatvoreno kućište.

3.1 Karakteristike integriranih sustava

Integrirani sustavi obuhvaćaju široki spektar uređaja koji se međusobno razlikuju po namjeni, veličini, načinu korištenja, načinu izvođenja zadatka... Karakteristike integriranih sustava ovdje su navedene općenito i ne moraju vrijediti za svaki pojedini takav sustav.

Integrirani sustavi sastoje se od mikroprocesora, često slabih karakteristika u usporedbi s današnjim procesorima korištenim u osobnim računalima ili serverima ali ipak dovoljnih karakteristika da obavljaju svoju zadaću. Cjelokupni sustav zatvoren je u kućište koje korisniku omogućava komunikaciju s procesom putem najčešće jednostavnih ulaza (tastatura s nekoliko tipki, *touch-screen*...).

Integrirani sustavi predviđeni su da rade bez otkaza godinama pa se izbjegavaju dijelovi koji bi se relativno brzo morali mijenjati kao što je hard disk. Softver koji koriste (*firmware*) obično je pohranjen u *flash* memoriji.

Procjenjuje se da je 98% proizvedenih u svijetu procesora namijenjeno integriranim uređajima, a svega 2% za osobna računala i servere.

3.2 Korisnička sučelja

Korisničko sučelje, za razliku od osobnih računala, izvedeno je tako da se najjednostavnije omogući komunikacija sa sustavom. Na primjer, digitalni radio alarm može imati tipke za podešavanje sati i minuta, podešavanje vremena alarma i tipku za gašenje alarma. Televizor (u jednostavnoj izvedbi) ima tipku za paljenje/gašenje, dvije tipke za odabir programa i dvije tipke za kontrolu jačine zvuka.

Mobiteli imaju kompliciranije sučelje s većim brojem tipki i najčešće ekranom za prikaz informacija. U posljednjih nekoliko godina trend je spajanja mogućnosti mobitela i osobnih računala pa se gubi stroga podjela na integrirane i modularne sustave.

3.3 Tipovi procesora

Procesor se odabire prema namjeni i kompleksnosti funkcija sustava, tako da može obraditi podatke u

predviđenom vremenu. Često je očekivano vrijeme obrade podataka stvarno vrijeme (engl. *real time*), pa je potrebno osigurati da se na obradu podataka ne čeka, pogotovo ako npr. o tome ovisi nastavak radnog procesa.

Arhitekture koje se koriste su, između ostalih: ARM, MIPS, PowerPC, X86, 8051... Moguće su i izvedbe korištenjem specifično izrađenih čipova koji mogu obavljati zadanu funkciju (engl. ASIC – *application specific integrated circuit*).

3.4 Samotestiranje i pouzdanost

Samotestiranje sustava provodi se kontinuirano ili periodički kako bi se ustanovila moguća pojava kvara u radu sustava izazvanog greškom u hardveru ili softveru. Testiraju se CPU, memorija, napajanje; provjera rezultata za generirane ulazne signale; provjera količine potrošnih materijala; provjera pouzdanosti sustava...

3.5 Primjeri integriranih sustava

- Kućanski aparati – TV, DVD, mikrovalne pećnice, klima uređaji, "inteligentni" uređaji – hladnjaci, štednjaci
- Mobiteli, kalkulatori, digitalni satovi, MP3 playeri, igraće konzole
- Fotokopirni uređaji
- Medicinska oprema
- ABS i GPS sustavi u automobilima...

4 MODULARNI SUSTAVI

4.1 Modularnost

Modul je potpuna jedinica koja u kombinaciji s drugim modulima čini veću potpunu jedinicu.

Modularnost se pojavljuje svugdje u prirodi – gotovo sve što možete zamisliti sastoji se od manjih jedinica koji u kombinaciji čine veću, bilo da govorimo o jednoj stanici, živom biću ili cijelom svemiru. Sustavi koje je dizajnirao čovjek također posjeduje to svojstvo. U računarstvu i elektrotehnici sustavi su izvedeni modularno kako bi se olakšala izvedba i razumijevanje samog sklopa.

Primjer modularnog sustava je računalo: sastoji se od komponenti koje imaju svoju funkciju a koje povezano daju sustav koji ima svoju funkciju (stroj za računanje odnosno obradu podataka).

4.2 Modularni sustavi

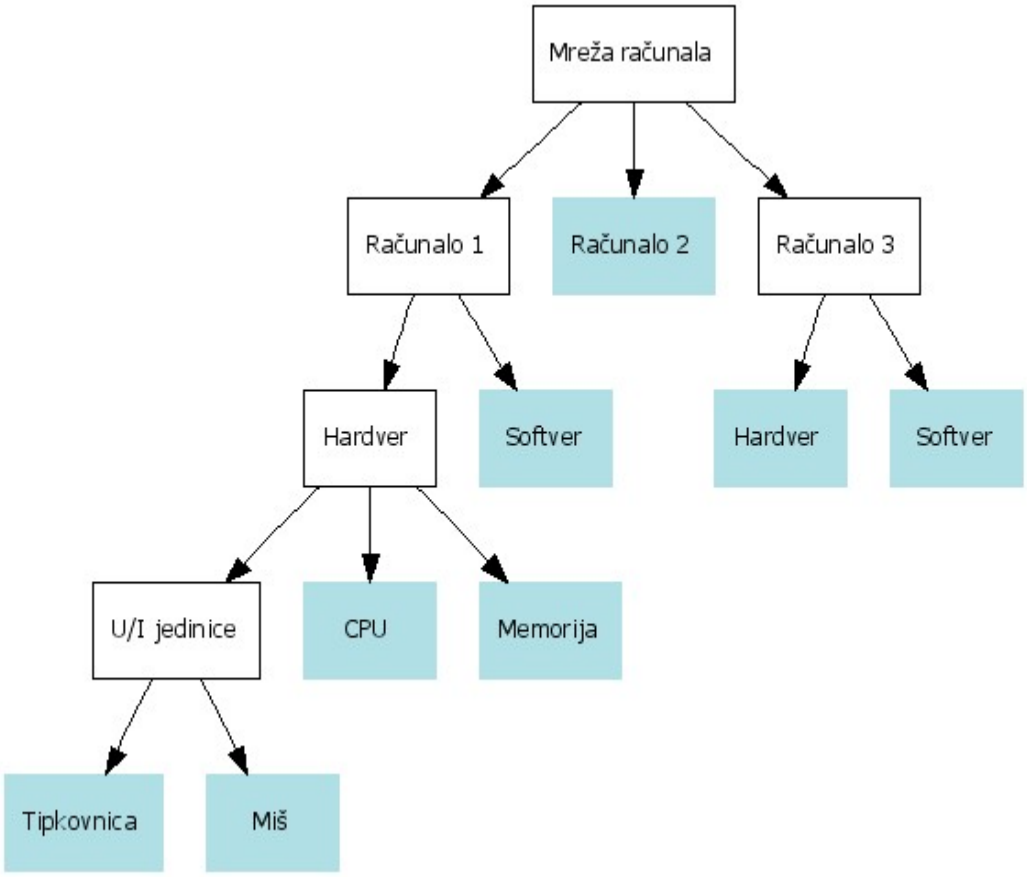
Najniži operativni član sustava (atomski sustav) takav je sustav ili dio sustava koji iz perspektive razmatranja rada nije potrebno razlagati u podsustave ili komponente jer se promatra samo funkcija takvog sustava, odnosno preslikavanje ulaza na izlaz.

Ideja modularnih sustava je u tome da se zadatak podijeli na dijelove koji rješavaju pojedini dijelovi sustava (podsustavi, komponente, moduli). Podsustav se može dalje dijeliti i time dalje dijeliti svoj zadatak na vlastite podsustave dok ne dođemo do najnižeg operativnog člana, onog koji ne možemo više dijeliti jer ne poznajemo njegovu unutrašnju strukturu, pa ga promatramo kao crnu kutiju.

4.3 Prednosti modularnih sustava

- Podjela funkcija na podsustave
- Lako sastavljanje i održavanje
- Izbor modula koji čine sustav omogućavaju specifikacije prema želji kupaca

4.4 Primjer



Slika 9: Modularni sustav - mreža računala

U primjeru na slici 9 neka je sustav koji pokušavamo složiti mreža računala sastavljena od tri računala. U prvom sloju računalo 2 (označen plavičastom pozadinom) ne dijeli se na podsustave – ono predstavlja atomski sustav, odnosno najniži operativni član i ne zanima nas kako je sastavljen, na primjer kupili smo ga gotovog od proizvođača.

Računalo 1 i računalo 3 (u prvom sloju) nismo kupili gotove. Računalo 1 sastoji se od hardvera i softvera. Hardver se sastoji od podsustava (U/I jedinice, CPU, memorija), pri čemu je podsustav U/I jedinica sastavljen od tipkovnice i miša. Tipkovnicu, miš, CPU i memoriju nabavljamo, pa nas ne zanima unutrašnja struktura tih dijelova. Softver je atomski sustav, npr. administrator sustava se pobrine da kada računalo bude složeno da sav potreban softver bude na njemu, dakle opet nas ne zanima što sve čini taj softverski sustav.

Slično je i s računalom 3, samo što smo tu primjerice nabavili hardver i softver i osposobili računalo da radi bez potrebe da znamo pojedinosti npr. tip matične ploče, veličinu hard diska ili operativni sustav.

5 RAZVOJ SOFTVERA

5.1 Što je softver

5.1.1 Pouzdanost softvera

Softver je ne-fizički dio računalnog sustava. Fizički je organiziran kao skup logičkih nula i jedinica u memoriji, organiziranih tako da računalo njihovom interpretacijom može izvesti predviđeni proces. Softver pišu ljudi, najčešće u višim programskim jezicima (malo se ljudi odlučuje programirati u asembleru ili strojnom kodu :-), a zatim prevodioc (*compiler*) prevodi napisani program u računalu razumljiv strojni kôd.

Unatoč težnji da se cjelokupno područje tehnologije formalizira, dakle da postoje predefinirani algoritmi koji će omogućiti računalu da, uz definiranje potrebnih parametara, sam oblikuje završni oblik, bilo uređaja ili logičkog sklopa ili bilo čega što čovjek inače proizvodi, još nije uspješno napravljen postupak koji će omogućiti računalu da sam sastavi smislen program. Pisanje softvera, odnosno softverski dizajn još se stoga ne može smatrati industrijom nego je u neku ruku umjetnost.

Ipak, softver ima velikog utjecaja na rad velikih (i malih, osobnih) računalnih sustava i stoga se postavljaju zahtjevi i potrebe mjerenja pouzdanosti softvera ili nekih karakteristika pomoću kojih se pouzdanost može odrediti.

5.1.2 Kontinuirano povećanje obujma softvera

Kontinuirano povećanje obujma softvera omogućuje pojavu većeg broja grešaka. Veliki softverski sustavi imaju milijune linija koda. S obzirom na nedovoljnu razvijenost automatizacije kreiranja koda, a s napretkom softverske industrije, u sve većim softverskim rješenjima koriste se i već korištena rješenja. Takva metoda sadrži u sebi dvostruku opasnost: osim što samo povećanje koda donosi veću vjerojatnost pojave greški, zbog nepostojanja formalne verifikacije (metode koje će za sve vjerojatnosti provjeriti ispravnost sustava), nepouzdanost iz tih korištenih dijelova softvera samo se, često nesvjesno, prenosi u novo rješenje.

5.1.3 Pojava neispravnosti

Neispravnost softvera u literaturi se navodi kao *failure*, dakle termin koji smo kod hardverskih sustava opisali kao *kvar*. S obzirom da pojam *kvar softvera* ne odgovara duhu hrvatskog jezika kako je pojam *software failure* u duhu engleskog jezika, tako je u nastavku taj pojam opisan kao *neispravnost*.

Neispravnosti se javljaju zbog netočnosti, dvostrukosti, previda ili pogrešnog razumijevanja zahtijeva koje

softver treba zadovoljiti, nemarnosti ili nesposobnosti pisanja koda, nedovoljnog testiranja, neispravnog ili nepredviđenog načina korištenja te drugih nepredviđenih problema. [PAN99].

Moguće je primijetiti da se neispravnosti javljaju zbog [LYU95]:

- Unutarnjih grešaka. Unutarnje greške su greške u dizajnu, odnosno pisanju koda, a aktiviraju se unosom određenih ulaza.
- Neispravnih ulaza. Izviru iz fizičke okoline ili rada korisnika.

5.2 Definicija pouzdanosti softvera

Pouzdanost softvera je vjerojatnost rada softvera bez greške u određenom vremenskom periodu i određenim radnim uvjetima. [Def. prema ANSI standardu]

Greške podrazumijevaju nedovoljno dobar pristup pisanju programa, čime je omogućeno da bilo kada u toku rada programa dođe do neispravnosti.

5.2.1 Primjer

Sljedeći programski odsječak napisan u programskom jeziku C uzima ulaznu vrijednost x i računa vrijednost y, pri čemu je $y = 1 / x$.

```
//deklaracija varijabli
int x;
int y;

//Kod koji obavlja neki pametan posao
...
...

// Unesi vrijednost x
printf (“Unesi vrijednost x: “);
scanf(“%d”, &x);

//izracunaj i ispisi y
y = 1 / x;
printf(“Vrijednost y = %d”, y);
```

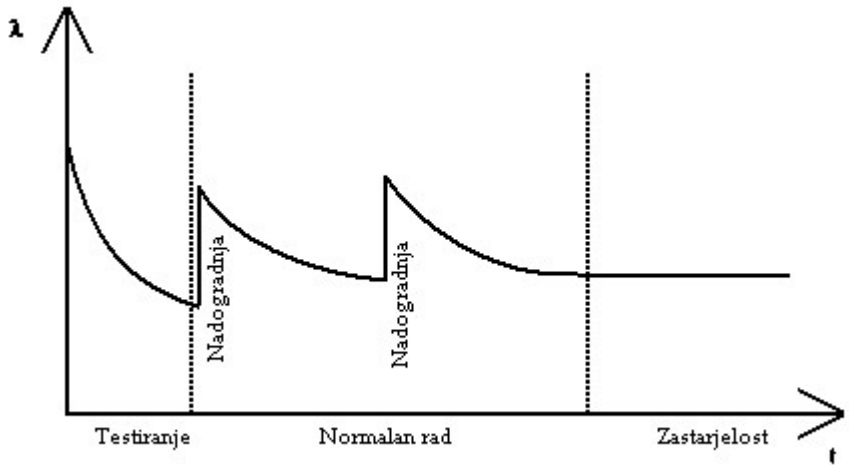
Napisani dio programa će biti ispravno preveden i može se pokrenuti, ali postoje barem dvije greške. Prva je ta što će ispisana vrijednost y biti uvijek cijeli broj. Što se druge greške tiče, što će se dogoditi ako korisnik programa za vrijednost x unese nulu?

Rezultat operacije bio bi beskonačno veliki broj koji nije moguće predstaviti ni jednim tipom podataka. Javit će se greška dijeljenja s nulom.

5.3 Nadogradnja (engl. upgrade)

5.3.1 Nadogradnja funkcionalnosti

Nadogradnja funkcionalnosti dodavanje je novih ili izmjena postojećih funkcija. Takvom nadogradnjom pouzdanost softvera pada, odnosno raste raspodjela kvarova, zbog samog novog koda koji povećava kompleksnost (uz pretpostavku da nove funkcije donose novi dio koda). Kroz testiranje dio grešaka moguće je ukloniti.



Slika 10: Izmijenjeni dijagram kade za pouzdanost softvera

Izmijenjeni dijagram kade za pouzdanost softvera na slici 10 opisuje ovu pojavu [PAN99].
Dijagram na slici u nazivu sadrži pojam "dijagram kade" zbog povezanosti s dijagramom kade hardvera koji pokazuje ovisnost učestalosti kvara o vremenu.

Dijagram kade softvera možemo podijeliti u tri faze: fazu testiranja, normalnog rada i zastarjelosti.

- Testiranje – Opadajuća učestalost kvarova zbog pronađenih i ispravljenih grešaka. Učestalost kvarova se približava nekoj konstantnoj raspodjeli uzrokovanoj neotkrivenim greškama u dizajnu.
- Normalan rad – Podrazumijeva korištenje softvera. Tokom normalnog rada proizvođač softvera može učiniti dostupnom nadogradnju funkcija. U tom slučaju, zbog povećane kompleksnosti softverskog sustava, raste vjerojatnost pojave kvara, koja s vremenom opada kako se otkrivaju nove greške, prema nekoj konstantnoj raspodjeli.
- Zastarjelost – Više ne postoji motivacija za razvojem novih nadogradnji, kako korisnici prelaze na nova rješenja koja odgovaraju njihovim potrebama. Sustav ima konstantnu raspodjelu kvarova (greške u dizajnu) - ne postoji degradacija.

5.3.2 Nadogradnja pouzdanosti

Nadogradnjom pouzdanosti želi se postići povećanje pouzdanosti i/ili sigurnosti sustava, odnosno otkloniti pronađene greške u dizajnu. Time obično vjerojatnost pojave neispravnosti pada, ali je moguće i određeno povećanje učestalosti kvara zbog novih sigurnosnih rupa ili propusta u funkcionalnosti.

5.4 Metode postizanja pouzdanosti softvera

Za postizanje pouzdanih softverskih sustava koriste se sljedeće metode [LYU95]:

- Prevencija greške. Pokušava se prilikom konstrukcije odnosno pisanja softvera izbjeći pojava greške. Neki od zahtjeva koji se postavljaju su pisanje čitkog i dobro strukturiranog koda
- Uklanjanje greške. Pokušava se testiranjem prilikom verifikacije i validacije prepoznati i ukloniti grešku.
- Tolerancija na grešku. Koristeći redunadanciju osigurati ispravan rad softvera neovisno o mogućoj pojavi grešaka. Koristi se tokom rada softvera, dakle nakon završetka njegovog razvoja.
- Predviđanje greške i kvara. Pokušava se procijeniti prisutnost greške te pojava i posljedice kvara.

5.5 Softverska redundancija

Softverska redundancija podrazumijeva korištenje više nezavisno razvijenih verzija istog softvera.

5.6 Testiranje/Verifikacija/Validacija

Mnoge tehnike razvijene su kako bi se osigurala sigurnost i pouzdanost sustava, među njima su na primjer: formalna verifikacija, Petrijeve mreže, deterministički konačni automati, analiza toka podataka, tablice istinitosti, simulacije, analiza stablom grešaka...

Verifikacija i validacija podrazumijevaju sustavnu analizu i testiranje softvera tokom razvoja radi povećanja pouzdanosti. Dok sam pojam "testiranje" često podrazumijeva provjeru ispravnosti gotovog proizvoda, verifikacija i validacija počinju istovremeno s razvojem sustava. Potrebno je utvrditi da softver obavlja opravo ono za što je sustav namijenjen i da ne ostvaruje nikakvu neželjenu funkciju, a proces podrazumijeva identificiranje i uklanjanje najrizičnijih problema u radu.

Verifikacija je provjera odgovaraju li rezultati etapa razvoja očekivanim rezultatima pojedine etape. Validacija je provjera daje li sustav očekivane rezultate kao funkciju ulaza.

Verifikacija i validacija često se smatraju jednim objedinjenim procesom u svrhu kontrole sustava.

Zadaci verifikacije i validacije [WALLACE96]:

- Procjena projektne dokumentacije – ocjena ispravnosti odabranih postupaka i alata.
- Procjena softverskih zahjeva – točnost i ispravnost zahjeva te odgovaraju li zahtjevima sustava.
- Provjera oblikovanja (dizajna) softvera – provjera zadovoljava li implementacija softverskim zahtjevima.

- Provjera koda – provjera ispravnosti prenošenja dizajna u kôd.
- Planiranje testiranja – provjera ponašanja sustava koje uključuje i neočekivane načine korištenja.
  - Testiranje jedinica – provjerava dizajn i implementaciju dijelova softvera.
  - Testiranje integracije – provjerava funkcionalnost i komunikaciju između dijelova softvera.
  - Testiranje sustava – provjerava komunikaciju sa sustavom i rad u radnom okruženju.
- Testiranje instalacije – provjera ispravnosti instalacije programa i verzija programa o kojem instalirani program ovisi.
- Verifikacija i validacija održavanja – u slučaju promjene softvera potrebno je razmotriti sve navedene zadatke radi ponovne verifikacije i validacije.

5.7 Ponovna iskoristivost softvera

Softver se ponekad može ponovno koristiti. Takve situacije se događaju prilikom korištenja komercijalnih rješenja, razvijenih funkcija i rutina, korištenje dijelova softvera primijenjenih u drugom softveru... Ponovna iskoristivost softvera nameće pitanje pouzdanosti takvog softverskog sustava, stoga je u takvim slučajevima potrebno provesti postupak verifikacije i validacije.

Dokumentacija softvera koji se ponovo koristi nužna je kako bi se kvalitetno moglo pristupiti testiranju.

Ako se sustav koji uključuje takav softver ne može adekvatno testirati, postoji velika vjerojatnost pojave neispravnosti, nesiguran je za korištenje ili može izazvati ozbiljne štete u slučaju pojave neispravnosti, takav softver ne bi trebalo koristiti.

Sastavio: Domagoj Maršić, 2006.
domagoj.marsic@gmail.com

Namijenjeno učenicima 4. razreda Srednje škole Dugo Selo, šk. god. 2006/07

6 LITERATURA

[LYU95] Michael R. Lyu, *Handbook of Software Reliability Engineering*, McGraw-Hill publishing, 1995, http://www.cse.cuhk.edu.hk/%7Elyu/book/reliability/

[PAN99] Jiantao Pan, *Software Reliability*, 1999, http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/

[WALLACE96] Dolores R. Wallace, Laura M. Ippolito, Barbara Cuthill, *Reference Information for the Software Verification and Validation Process*, NIST Special Publication 500-234, 1996, http://hissa.nist.gov/

[WEIBULL] *System Reliability Theory & Principles Reference from ReliaSoft*, http://www.weibull.com/systemrelwebcontents.htm

Članci koji mogu poslužiti za daljnje proučavanje problematike:

Safety Engineering, http://en.wikipedia.org/wiki/Safety_engineering

Fail-safe, http://en.wikipedia.org/wiki/Fail-safe

Poka-yoke, http://en.wikipedia.org/wiki/Poka-yoke

Inherent safety, http://en.wikipedia.org/wiki/Inherent_safety

Life-critical system, http://en.wikipedia.org/wiki/Life-critical_system

Failure mode and effects analysis, http://en.wikipedia.org/wiki/FMEA

Failure mode and effects analysis – FMEA, http://www.weibull.com/basics/fmea.htm

Fault Tree Analysis, Reliability Block Diagrams and the BlockSim FTI Edition, http://www.weibull.com/SystemRelWeb/fault_tree_analysis,_reliability_block_diagrams_and_the_blocksim_fti_edition.htm

Basic Gates, http://www.weibull.com/SystemRelWeb/basic_gates.htm

SADRŽAJ