UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

DIPLOMSKO DELO

TESTIRANJE PROGRAMSKE OPREME IN IZDELAVA NAČRTA ZA TESTIRANJE TIME&SPACE SISTEMA

Ljubljana, oktober 2006 UROŠ KOČEVAR

IZJAVA

Študent Uroš Kočevar izjavljam, da sem avtor tega diplomskega dela, ki sem ga napisal pod mentorstvom prof. dr. Mojce Indihar Štemberger, in dovolim objavo diplomskega dela na fakultetnih spletnih straneh.

V Ljubljani, dne 20.10.2006 Podpis: _____

KAZALO

1. UVOD

V zadnjih dveh desetletjih so računalniški sistemi in programska oprema, ki omogoča njihovo delovanje, prodrli na trg in postali del našega vsakdanjika. Programsko opremo najdemo v avtu, pečici, mobilnem telefonu, video igricah in na delovnem mestu. Poganja bančne in komunikacijske sisteme ter skrbi za internetne povezave. Zaradi tako pomembne vloge programske opreme v našem življenju, tako z ekonomskega kot socialnega vidika, se povečuje pritisk na hitrejši razvoj novih produktov, ki pa morajo biti tudi dovolj kakovostni. Manj kakovostna programska oprema, ki lahko povzroči izgubo življenj ali podatkov, ni več splošno sprejemljiva. Poleg izgube uporabnikov povzroči tudi visoke stroške za podjetje, ki produkt ponuja na trgu.

Poglejmo si nekaj znanih programskih napak, ki so imele razsežne posledice (Wikipedia, 2006):

- »Millennium bug« ali problem leta 2000 – Napaka, o kateri smo na prehodu v novo tisočletje slišali mnogo, je bila posledica načrtovanja programske opreme v preteklosti. S 1. 1. 2000 naj bi računalniki napačno obračunavali datume in ure, kar je povzročilo paniko po vsem svetu. Ta se je kazala v množičnem posodabljanju sistemov, kar naj bi odpravilo ta problem. Čeprav se je izkazalo, da je bil alarm lažen, se stroški priprave na 1. 1. 2000 ocenjujejo na 300 milijard $.
- »North America blackout« – 14. 8. 2003 je prišlo do masovnega izpada električne energije po delih severovzhodne Amerike in Kanade zaradi napake v programski opremi, ki nadzira omrežje. Po ocenah naj bi vsega skupaj brez elektrike ostalo 50 milijonov ljudi, finančno izgubo pa ocenjujejo na 6 milijard $.
- Raketa Ariane 5, Polet 501 – 4. 7. 1996, 37 sekund po vzletu rakete Ariane 5, je prišlo do eksplozije zaradi napake v delovanju krmilnega sistema. Omenjena napaka je bila ena od najdražjih programskih napak v zgodovini.
- Deljenje s plavajočo vejico pri procesorjih Intel Pentium – 30. 10. 1994 je profesor Thomas Nicely z univerze v Lynchburgu odkril napako, ki se je pojavila v Pentium procesorjih, ko so le-ti izvajali deljenje s plavajočo vejico. Po zamenjavi procesorjev so stroški dosegli 475 milijonov $.

Delati napake je človeško, še posebej, če je naša naloga razviti kompleksno programsko opremo. Zaradi tega ima proces testiranja, v razvoju programske opreme, pomembno vlogo. V procesu testiranja je ključnega pomena to, da v razvojnem procesu programske opreme odkrijemo napako kar se da hitro, saj s kasnejšim časom odkritja eksponentno naraščajo stroški. Raziskava, ki jo je leta 2002 izvedel Research Triangle Institute (RTI) v Severni Karolini, je pokazala, da programske napake ameriškemu gospodarstvu povzročijo 59,9 milijarde $ škode, kar pomeni 0,9 % BDP USA. Ugotovili so tudi, da bi lahko tretjino teh stroškov odpravili z izboljšano infrastrukturo, ki jo uporabljajo pri testiranju (Software Errors Cost, 2006).

V realnosti se proces testiranja velikokrat odvija čisto drugače, kot je bilo prvotno načrtovano. Ker projekti pogosto časovno zamujajo, stroški pa presežejo načrtovane, je testiranje prva stvar, pri kateri vodje projektov naredijo spremembe. Posledica tega so zmanjšana sredstva in skrajšan čas testiranja, kar se manifestira v slabo kakovostnem produktu in še večjih stroških vzdrževanja.

2. TESTIRANJE PROGRAMSKE OPREME

2.1 Definicija testiranja

Pri testiranju programske opreme želimo le-tej dodati vrednost. Dodajanje vrednosti skozi testiranje pomeni povišanje kvalitete in zanesljivosti aplikacije. Višanje zanesljivosti pomeni iskanje in odpravljanje napak oz. »bugov«. Kadar se lotimo testiranja, moramo predpostaviti, da programska oprema ali sistem vsebuje napake (veljavna predpostavka za skoraj vsako aplikacijo). Teh skušamo, skozi proces testiranja, najti v čim večjem številu. Definicija testiranja programske opreme bi se lahko glasila takole (Myers, 2004, str. 6):

»Testiranje je proces izvajanja programa z namenom iskanja napak.«

Eden izmed primarnih vzrokov za slabo testiranje programske opreme je to, da večina programerjev napačno razume testiranje. Z njihove strani je velikokrat slišati (Myers, 2004, str. 5):

- *»Testiranje je proces dokazovanja, da napake niso prisotne.«*
- *»Namen testiranja je pokazati, da program pravilno izvaja pričakovane funkcije.«*
- *»Testiranje je proces vzpostavljanja zaupanja, da program dela, kar naj bi počel.«*

Takšno razmišljanje se posledično prenese na izvajalce testiranja, kar se pozna na rezultatih testa. Osnovni koncept testiranja sestavljata dve metodi, in sicer metoda črne in bele skrinjice. Pri metodah črne skrinjice ali funkcionalni analizi preverjamo izhodne podatke aplikacije pri danih vhodnih podatkih, ki morajo biti v skladu s funkcionalno specifikacijo aplikacije. Pri metodah bele skrinjice ali strukturni analizi preverjamo izhodne podatke aplikacije pri danih vhodnih podatkih, ki morajo biti v skladu s strukturalno specifikacijo aplikacije (Wikipedia).

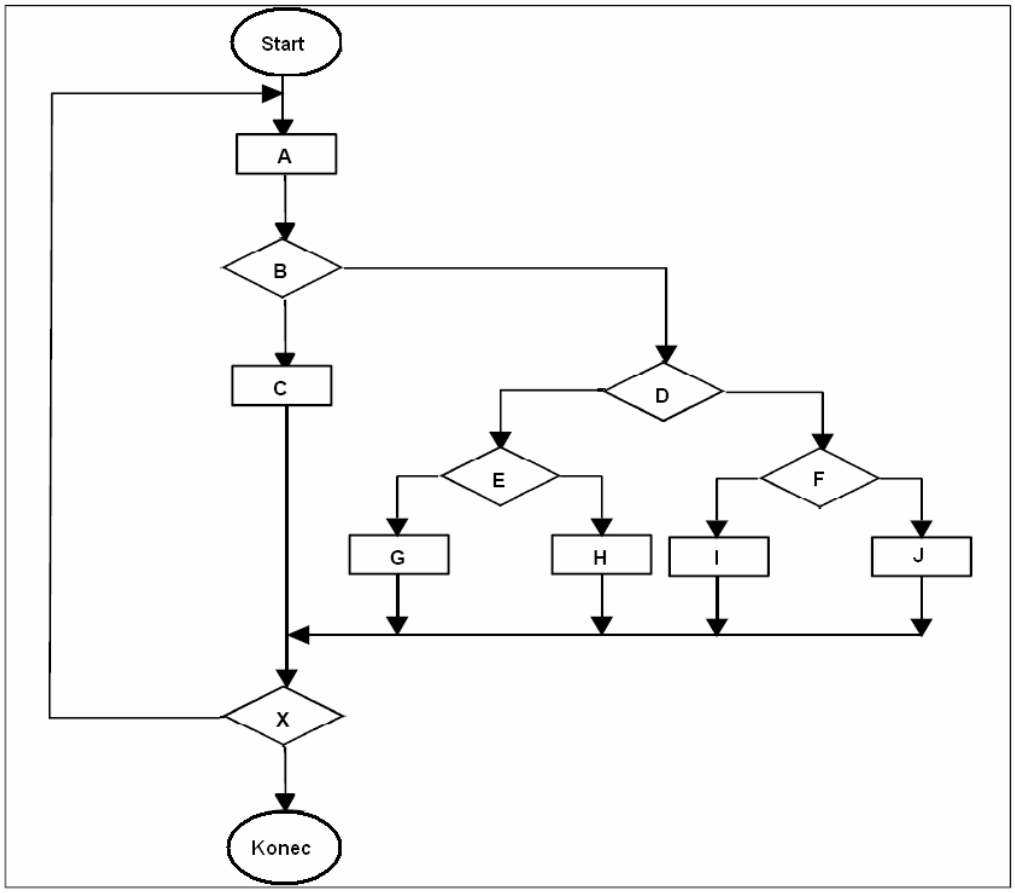
2.2 Zakaj ni mogoče izvesti popolnega testa?

Mnogi menijo, da je potrebno testirati vse kombinacije, da bi lahko dokazali, da programska oprema deluje. To seveda ni mogoče, saj ni možno preveriti vseh vhodnih podatkov, časovnih usklajevanj in možnih poti.

Slika 1 prikazuje strukturo preproste aplikacije, kjer so prikazane vse možne poti od začetka (START) do konca (KONEC). Pri zanki X se odločimo, ali gremo na KONEC ali se vrnemo na A, to lahko ponovimo 19-krat. Od A do X je možnih 5 poti:

- ABCX
- ABDEGX
- ABDEHX
- ABDFIX
- ABDFJX

Slika 1: Struktura preproste aplikacije



Vir: Practical Guide to Software System testing, 1998, str. 9.

Če izvedemo eno ponovitev, pri čemer se pri zanki X ne vračamo, imamo možnih 5 poti. Pri dveh ponovitvah odločitvene logike, pri čemer se pri zanki X vrnemo samo prvič, imamo 25 poti. V treh ponovitvah odločitvene logike število naraste na $25 = 5^3$. Število na koncu vseh možnih kombinacij doseže 100 bilijonov ($10^{14} = 5 + 5^2 + 5^3 + ….5^{20}$).

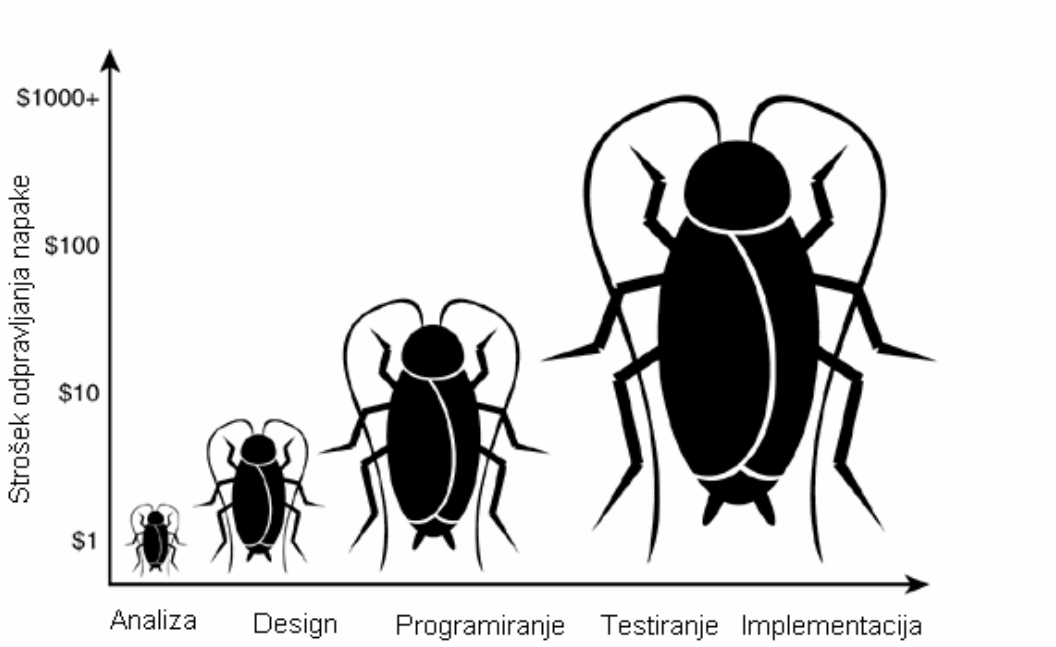
2.3. Programski hrošč oz. »bug«

Izraz izhaja iz časov, ko so bili računalniki sestavljeni iz elektronk. Hrošče je svetloba delujočih elektronk močno privlačila in zato so se ujemali v vezja, kjer so povzročali kratke stike in razpade sistemov. Danes pa se izraz hrošč pogosteje uporablja za programsko napako v računalniškem programu, ki je programer med svojim delom ni odkril. Te napake povzročajo napačno oz. nepredvidljivo delovanje, sesutje programa ali računalnika. Kaner, Falk in Nguyen so napake v programski opremi razdelili v 13 kategorij (Ross, 1998, str. 11):

- napake v uporabniškem vmesniku – sistem prikazuje napačno grafično obliko;
- ravnanje z napakami – način, kako so napake ugotovljene in obravnavane, je lahko tudi napaka;
- kontrola nad verzijami in kodo – zastarele aplikacije so lahko uporabljene v primeru, ko so na voljo prenovljene izdaje;
- napake pri rokovanju in razlaganju podatkov – prenos in preoblikovanje podatkov med programsko opremo lahko povzroči napake;
- napake v povezavi z mejno vrednostjo – obravnavanje mejnih vrednosti je lahko nepravilno;
- nadzor nad potekom logičnih poti – izbrano dejanje v nadaljevanju ni primerno za trenutno stanje;
- pogoji izbire – če je zahtevano, da se izvedeta dve nalogi, ima ena zmeraj prednost pred drugo; kakorkoli, lahko se zgodi, da se naloge ne izvedejo v pravilnem vrstnem redu in zaradi tega pride do nepravilnih rezultatov;
- obremenitvene zmožnosti – pri sistemu, ki deluje na robu svojih možnosti, se začnejo pojavljati napake;
- računske napake – računski in logični izračuni so lahko napačni;
- začetne napake – funkcija se napačno izjalovi le ob prvi izvedbi;
- strojne napake – komunikacija s strojno opremo ne deluje pravilno pod določenimi pogoji;
- dokumentacija – uporabniku ni na voljo v priročniku opisan postopek;
- napake pri testiranju – testerji delajo napake pri procesu testiranja, saj mislijo, da je za nepravilno delovanje kriv sistem.

Pri odkrivanju programskih napak je bistvenega pomena čas odkritja oz. faza razvoja programske opreme, v kateri je napaka odkrita. Če je napaka odkrita v zgodnjih fazah analize in oblikovanja specifikacije, je strošek odkrite napake zanemarljiv, kar prikazuje slika (Slika 2). Napaka, odkrita v fazi programiranja ali testiranja, lahko povzroči strošek, ki je za desetkrat ali stokrat večji od začetnega. Če napako odkrijejo končni kupci, se strošek lahko povzpne na tisoče ali milijone dolarjev.

Slika 2: Strošek odpravljanja napake glede na časovno periodo



Vir: Software Testing, 2005, str. 23.

2.4. Metode črne skrinjice

Pri teh metodah gledamo na aplikacijo kot na črno skrinjico. Naš cilj je, da smo popolnoma indiferentni do notranjega delovanja in strukture programa. Koncentriramo se na iskanje okoliščin, v katerih se program ne obnaša v skladu s specifikacijo. Pri omenjenih metodah so informacije za testne scenarije pridobljene izključno iz specifikacije (Myers, 2004, str. 9). Metode črne skrinjice sestojijo iz različnih metod, kot so npr. določanje ekvivalentnih vhodnih skupin, analiza mejne vrednosti, analiza načrta, odločitvene tabele, diagrami stanj prehodov itd. Nekatere od teh metod so bolj uporabne pri določeni stopnji, medtem ko so druge neodvisne od stopnje testa, kar je prikazano v Tabeli 1.

Tabela 1: Metode črne skrinjice glede na stopnjo testa

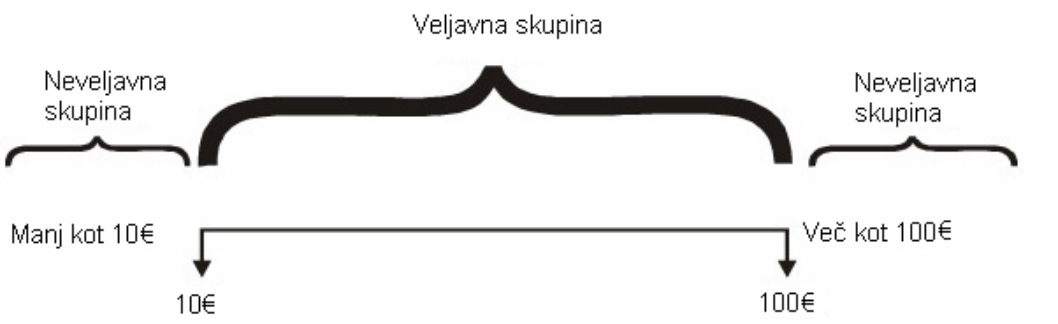
	Stopnja testiranja		
Metoda	Modul	Integracija	Sistem
Določanje ekvivalentnih vhodnih skupin	✓	✓	✓
Analiza mejne vrednosti	✓	✓	✓
Sledilna matrika			✓
Neveljavne kombinacije in procesi		✓	✓
Odločitvene tabele	✓	✓	✓
Diagrami stanj in prehodov			✓
Metoda naključnega iskanja napak	✓	✓	✓
Ortogonalne vrste	✓	✓	✓

Vir: Systematic Software Testing, 2002, str. 51.

3.4.1 Določanje ekvivalentnih vhodnih skupin

Določanje testne procedure je ena najbolj pomembnih nalog testiranja. Pri tej metodi določimo vhodne skupine, ki so enako obravnavane s strani sistema in ustvarijo enak rezultat. Vzemimo za primer bankomat, ki nam dovoli dvigovati gotovino v bankovcih po 10 €, v razponu od 10 € do 100 €. Imamo 3 ekvivalentne skupine: eno veljavno in dve neveljavni, kar je prikazano na Sliki 3.

Slika 3: Določanje vhodnih skupin



Vir: Systematic Software Testing, 2002, str. 58.

Če se tester odloči, da bo preizkusil vse možnosti, ker ima malo možnih kombinacij (dvigne 10 €, 20 €, 30 € itd.), bo tako samo tratil svoj čas. Če lahko dvigne 10 €, lahko dvigne tudi 20 €, 30 € … 100 €. Na računu seveda mora biti dovolj denarja, dnevni limit ne sme biti presežen in nenazadnje, v bankomatu mora biti dovolj denarja. V tem primeru potrebujemo samo dve neveljavni in eno veljavno vrednost, da je obseg delovanja pokrit.

2.4.2 Analiza mejne vrednosti

Izkušnje kažejo, da je metoda analize mejne vrednosti zelo pomembna, saj omejitve velikokrat niso veljavne. Filozofija tega testiranja je preprosta. Če lahko varno hodimo tik ob robu prepada, potem lahko skoraj z gotovostjo trdimo, da bomo varno hodili tudi po sredini polja. V našem primeru bankomata bi izbrali naslednje vrednosti: 0 € – vrednost izpod spodnje meje, 10 € – spodnja meja, 100 € – zgornja meja in 110 € – vrednost nad zgornjo mejo.

2.4.3 Odločitvene tabele

Odločitvene tabele nam prikazujejo vse možne vhodne pogoje in rezultate. V uporabi so že več let, vendar večina današnjih testnih inženirjev te metode ne bi uporabila za testiranje celotnih sistemov, predvsem zaradi obsežnosti le-teh. Odločitvene tabele so uporabne za test kritičnih enot sistema, ki so definirane s skupino pravil. Tabela 2 prikazuje odločitveno tabelo za računanje davkov na osnovi plače. Za vsako kombinacijo pogojev obstaja pravilo (Da, Ne, N – nepomembno).

Tabela 2: Odločitvena tabela za primer računanja davkov

	Pravila			
Pogoji	Pravilo 1	Pravilo 2	Pravilo 3	Pravilo 4
Zaslužena plača	Ne	Da	Da	Da
Konec plačilnega obdobja	N	Ne	Da	Da
Prekoračitev meje za socialno zavarovanje	N	N	Ne	Da
Akcije				
Zadrži davek za socialno zavarovanje	Ne	Ne	Da	Ne
Zadrži davek za zdravstveno zavarovanje	Ne	Ne	Da	Da
Zadrži davek na izplačano plačo	Ne	Ne	Da	Da

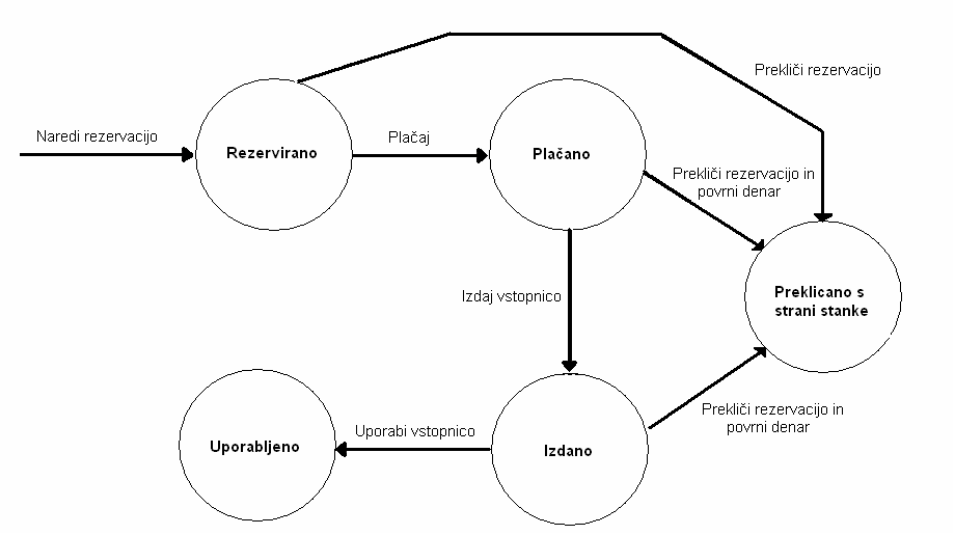
Vir: Systematic Software Testing, 2002, str. 65.

Branje tabele začnemo pri prvem pogoju in sledimo po vrstici proti desni, dokler pravilo ne izpolni pogoja. Potem nadaljujemo pri naslednjem zadovoljujočem pravilu in se pomikamo proti desni. V drugo vrstico se premaknemo, ko je prikladen pogoj izpolnjen. Ko je zadnji pogoj izpolnjen, izvedemo akcije, ki so pogojene s stolpcem, v katerem smo končali.

2.4.4 Diagrami stanj in prehodov

Diagram stanj in prehodov je zelo stara metoda, ki pa je še zmeraj učinkovita pri opisovanju modela sistema, ter je vodnik pri testiranju. Diagram je sistem ali komponenta, čigar funkcionalnost in rezultat nista izključno odvisna od trenutnega vhoda, temveč tudi od prejšnjih vhodov. Rezultat prejšnjega vhoda se imenuje stanje in prehodi so naloge, ki povzročijo premike iz enega v drugo stanje. Slika 4 prikazuje diagram stanj in prehodov na primeru rezervacije letalskih vozovnic.

Slika 4: Diagram stanj in prehodov



Vir: The art of software testing, 2003, str. 134.

2.4.5 Metoda naključnega iskanja napak

Velikokrat se zgodi, da so nekateri ljudje že po naravi vešči pri testiranju programske opreme. Brez posebnih metod jim uspe izslediti napake v aplikaciji. Ena izmed razlag tega pojava je, da ti ljudje podzavestno vadijo tehniko, ki se imenuje naključno iskanje napak. Ko se lotijo testiranja, sledijo svoji intuiciji in izkušnjam. Tako ustvarijo testen scenarij, ki je uspešen pri izsleditvi določenih vrst napak. Za to metodo je zelo težko določiti proceduro izvajanja, saj je v veliki meri odvisna od intuicije in ad-hoc[1] testiranja. Osnovna ideja je ta, da sestavimo listo možnih napak ali situacij, v katerih se bodo napake pokazale, in za te oblikujemo testne scenarije.

2.5 Metode bele skrinjice

Metode bele skrinjice nam dovoljujejo raziskati notranjo strukturo programa. Pri teh metodah se testne informacije pridobijo iz pregleda logike programa. Brez pogleda v skrinjico ni možno preveriti določene načine delovanja aplikacije. Raje, kot da poskušamo testirati vsako možno pot, uporabimo različne metode bele skrinjice, pri katerih so na voljo mehanizmi za izbiro posameznih poti testa. Pogosto so metode bele skrinjice povezane z merami za pokritost testiranja, ki merijo odstotek izbranih poti, izvršenih v določenem testnem scenariju. Projekti imajo ponavadi določene stopnje pokritosti testiranja, ki so razvijalcem in testerjem vodilo pri temu, kako temeljito je potrebno testirati aplikacijo (Ross, 1998, str. 94).

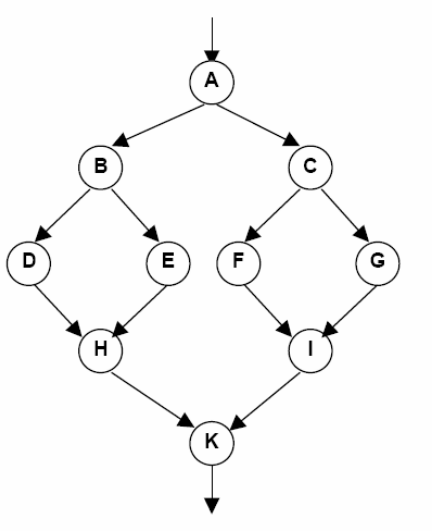
Pokritost trditev, vej in poti so metode bele skrinjice, ki izkoristijo logični tok aplikacije za izdelavo testnih scenarijev. Logični tok je način, kako se lahko deli aplikacije izvajajo, ko le-to poženemo. Logični tok aplikacije lahko predstavimo z grafom toka, kar vidimo na Sliki 5.

Graf je sestavljen iz:

- vozlišč – predstavljajo trditve, ki so lahko uporabljene pri izvajanju programa;
- robov – predstavljajo pot, po kateri logika omogoča aplikaciji prehod od ene trditve k drugi;
- vozlišč vej – to so vozlišča, ki imajo več kot en izhod;
- robov vej – to so robovi, ki zapuščajo vejo;
- poti – možne poti, po katerih se lahko premikamo od enega vozlišča do drugega po robovih.

[1] Ad-hoc testiranje – vrsta testiranja, ko se le-tega lotimo brez vnaprej pripravljenega testnega plana.

Slika 5: Graf toka



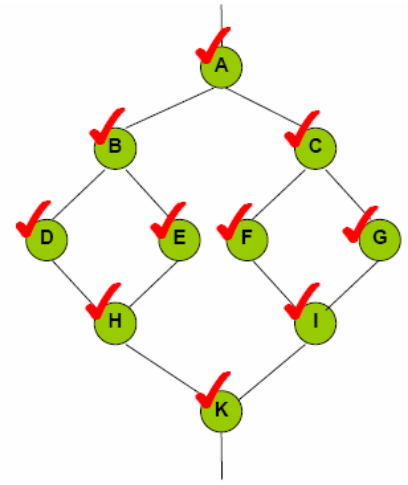
Vir: Practical Guide to Software System testing, 1998, str. 94.

Izvajanje vsakega testnega scenarija povzroči, da aplikacija izvede določen ukaz, ki se odraža v izbiri specifične poti v grafu toka. Različne poti lahko izbiramo s spreminjanjem pogojev, ki povzročijo drugačno izbiro robov vej na izhodu iz vozlišča vej.

2.5.1 Pokritost trditev

Pokritost trditev je določena z oceno deleža trditev, ki je pokrit z določenim testom. 100-odstotno pokritost trditev dosežemo tako, da je vsaka trditev v aplikaciji vsebovana vsaj v enem testnem scenariju. Pokritost trditev se ujema z obiskom vozlišč na grafu. 100-odstotna pokritost je tam, kjer so vsa vozlišča bila obiskana s strani poti v testnem scenariju. V grafu (Slika 6) imamo 10 vozlišč. Če izberemo pot A, B, D, H, K, pokrijemo 5 od 10 vozlišč, kar pomeni 50-odstotno pokritost.

Slika 6: Pokritost trditev

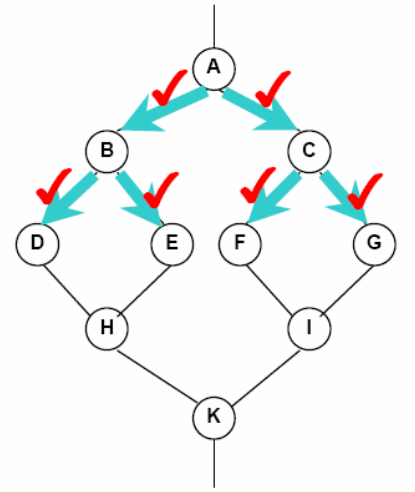


Vir: Practical Guide to Software System testing, 1998, str. 95.

2.5.2 Pokritost vej

Pokritost vej je določena z oceno deleža odločitvenih vej, ki so pokrite v testu. 100-odstotno pokritost dosežemo tako, da je vsaka odločitvena veja vsebovana v vsaj enem testnem scenariju. Pokritost vej se ujema z obiskom robov vej v grafu. 100-odstotna pokritost je tam, kjer so vsi robovi vej obiskani s strani poti v testnem scenariju. Slika 7 prikazuje 6 robov vej, če izberemo pot A, B, D, H, K, pokrijemo 2 od 6 robov vej, kar pomeni 33-odstotno pokritost.

Slika 7: Pokritost vej

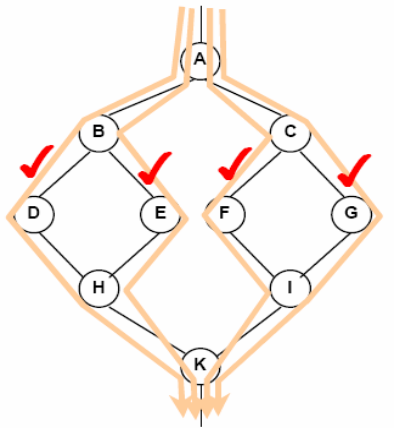


Vir: Practical Guide to Software System testing, 1998, str. 95.

2.5.3 Pokritost poti

Pokritost poti je določena z oceno deleža poti, ki je pokrit z določenim testom. 100-odstotno pokritost poti dosežemo tako, da je vsaka pot v aplikaciji vsebovana vsaj v enem testnem scenariju. Pokritost poti se ujema z obiskom poti v grafu. 100-odstotna pokritost je tam, kjer so vse poti obiskane v testnem scenariju. Na grafu, ki ga prikazuje Slika 8, so 4 poti. Če izberemo pot A, B, D, H, K, pokrijemo 1 od 4 poti, kar pomeni 25-odstotno pokritost.

Slika 8: Pokritost poti

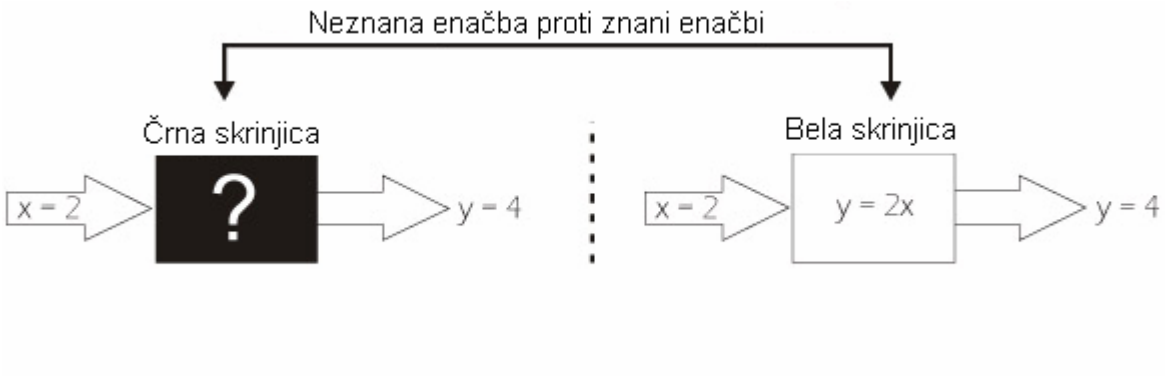


Vir: Practical Guide to Software System testing, 1998, str. 96

2.6 Primerjava metod

Medtem, ko sta obe metodi uspešni pri zagotavljanju pravilnega delovanja sistema, lahko samo metoda bele skrinjice potrdi, da je bil proces izvedbe pravilen (Slika 9). Običajno menimo, da je ob pravilnem rezultatu testa pravilen tudi proces obdelave. Temu pa seveda ni tako. V nekaterih primerih je možno, da dobimo pravilen rezultat testa z napačnim procesom. Ta fenomen je poznan kot naključna pravilnost, ki ga z uporabo strategije črne skrinjice ne odkrijemo zmeraj.

Slika 9: Primerjava metod testiranja



Vir: Effective software testing, 2003, str. 88.

Uporaba metode črne ali bele skrinjice izboljša kakovost testiranja za 40 %. Obe strategiji skupaj izboljšata kakovost za 60 % (Craig, 2002, str. 159).

Za lažje predstavljanje naključne pravilnosti si oglejmo preprost primer. Imamo aplikacijo, ki računa potreben čas za izvedbo določene naloge. Le-ta vsebuje formulo $y=2x$, kjer y pomeni potreben čas in x težavnost naloge. Če imamo nalogo, ki ima težavnost 5, potrebujemo 10 ur za dokončanje naloge. Ali je delovanje aplikacije pravilno, če tester vnese vrednost 2 in dobi rezultat 4? Domnevamo, da je programer narobe razumel algoritem in vnesel formulo $y=x^2$ namesto $y=2x$. Tako je tester dobil pravilen rezultat, čeprav je formula nepravilna. Če bi naredili še en test in vnesli x=3, bi dobili rezultat 9 namesto 6.

Za odkritje takšnih napak moramo pogledati v notranjost bele skrinjice. Metoda bele skrinjice omogoča testerjem, da uporabijo svoje znanje o sistemu in naredijo takšen testni scenarij, ki bo temeljil na modelu ali strukturi sistema. Če želijo izvesti teste po metodi bele skrinjice, morajo znati programski jezik in dokumentne načrte aplikacije.

3. PROCES TESTIRANJA

»Obstajajo tri stvari, katerih procesa izdelave ne bi smeli nikoli videti: zakoni, klobase in programska oprema.« To je šala, ki jo lahko slišimo v računalniški industriji. Njihovi procesi izdelave so velikokrat »neurejeni«, tako da je bolje počakati na končni rezultat. Nekatere programske opreme nastajajo pod strogim nadzorom in z disciplino programerjev, spet druge v kontroliranem kaosu in nenazadnje še tiste, ki so »zlepljene z lepilnim trakom«. Uporabniku je na koncu jasno, katera vrsta procesa je bila uporabljena. Proces izdelave programske opreme, od prvotnega koncepta do javne objave, se imenuje življenjski razvojni cikel programske opreme (Patton, 2005, str. 53). Obstaja več različic življenjskih razvojnih ciklov. Odločitev o izbiri določenega cikla je v celoti prepuščena razvojni ekipi.

Proces testiranja je nepogrešljivi del razvojnega cikla programske opreme. »Hroščati« programski opremi, ki je splošno nezaželena, so dnevi šteti. S procesom testiranja dvigujemo stopnjo zanesljivosti in kakovosti končnega produkta ter skušamo minimizirati število napak, ki jih odkrijejo končni uporabniki. Obstajajo različne stopnje testiranja, ki so odvisne od vsebine, ki jo testiramo. Najnižja stopnja predstavlja testiranje modula, najvišja pa sistemsko testiranje. Preden se lotimo testiranja, moramo opredeliti, kaj, kako in kdo bo testiral. Vse podatke zapišemo v testni plan, kot je razvidno iz priloge.

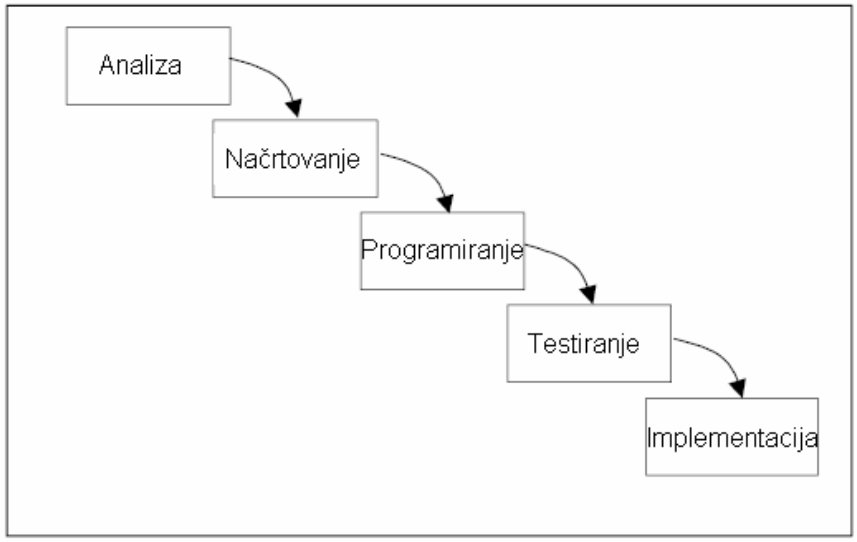
3.1 Življenjski razvojni cikli programske opreme

3.1.1 Tradicionalni življenjski cikel

Pri tradicionalnem življenjskem ciklu razvoja programske opreme testiranje nastopi po procesu programiranja aplikacije (Slika 10). Ker proces testiranja nastopi proti koncu omenjenega cikla, to predstavlja slabosti za ta model razvoja programske opreme. Te slabosti so:

- Pri oblikovanju testnih scenarijev moramo imeti vpogled v aplikacijo, ki jo testiramo. Tako se napaka odkrije, preden izvršimo testne scenarije.
- Običajno programiranje preseže časovni plan, zato nastane pritisk, da se produkt po fazi programiranja kar se da hitro izda. Ta časovni pritisk se prenese na fazo testiranja, kar se odraža v površnem in nedokončanem testiranju.

Slika 10: Tradicionalni življenjski cikel

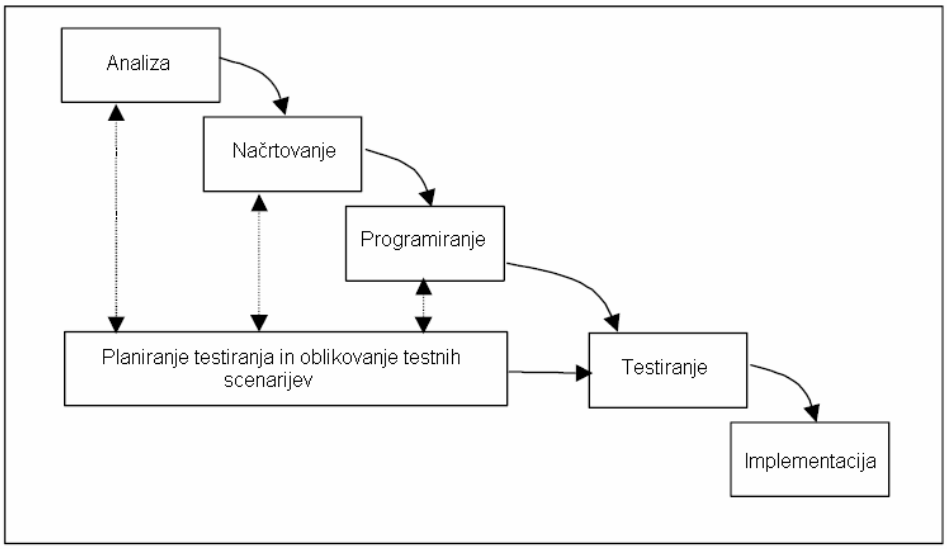


Vir: Practical Guide to Software System testing, 1998, str. 24.

3.1.2 Vzporeden življenjski cikel

Pri vzporednem življenjskem ciklu se planiranje testiranja in oblikovanje testnih scenarijev izvaja vzporedno z razvojem programske opreme (Slika 11). Vsakokrat, ko se spremeni specifikacija aplikacije, ki jo razvijamo, se naredi primerjava z že oblikovanim testnim planom in se ga popravi v skladu s spremembami. Slabost tega modela se kaže v tem, da je dražji kot tradicionalni razvojni model, saj moramo velikokrat popravljati testni plan. Prednost modela pa je v tem, da lahko takoj po končani fazi programiranja začnemo izvajati testiranje, saj je vsa potrebna dokumentacija za proces testiranja že pripravljena (Parallel Life Cycle).

Slika 11: Vzporeden življenjski cikel testiranja

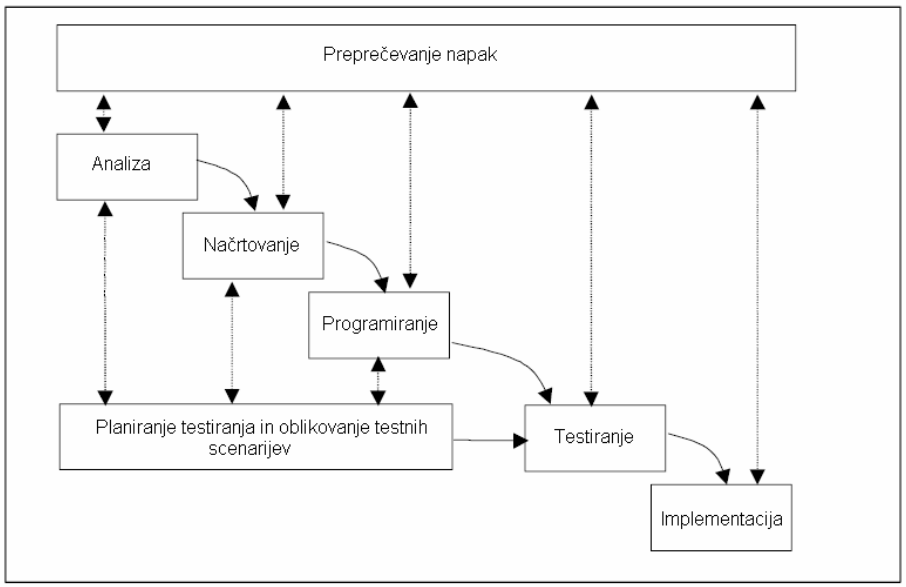


Vir: Practical Guide to Software System testing, 1998, str. 25.

3.1.3 Življenjski cikel preprečevanja napak

Metoda življenjskega cikla preprečevanja napak je nadgradnja vzporednega življenjskega cikla, kar se odraža v uporabi tehnik za preprečevanje napak v vseh fazah življenjskega cikla (Slika 12). Z uporabo tehnik za preprečevanje napak skušamo doseči, da že v začetku razvojnega cikla naredimo čim manj napak. Prednosti te metode so optimiziranje stroškov razvoja programske opreme in skrajšan razvojni cikel (Mukesh).

Slika 12: Življenjski cikel preprečevanja napak

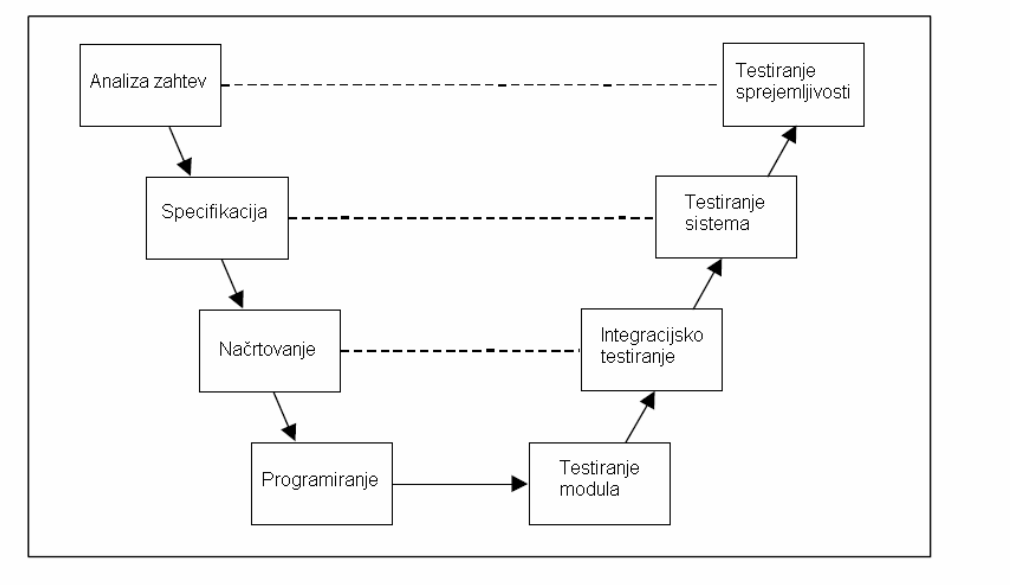


Vir: Practical Guide to Software System testing, 1998, str. 25.

3.1.4 Razvojni model – V

Razvojni model V sestavljata procesa verifikacije in validacije. Verifikacija je proces, pri katerem preverjamo, ali produkt tekoče faze ustreza zahtevam, postavljenim v predhodni fazi. Validacija je proces vrednotenja programske opreme na koncu njenega razvoja z namenom, da se ugotovi skladnost z zahtevami (Slika 13). Leva stran modela predstavlja proces razvoja programske opreme, desna pa proces testiranja. Za vsako točko na levi strani modela obstaja ustrezen test, ki preveri, če razvojna faza poteka pravilno. Ko pridemo na vrh desne strani, je programska oprema pripravljena za izdajo (Solina, 1998, str. 143).

Slika 13: Razvojni model –V



Vir: Practical Guide to Software System testing, 1998, str. 26.

3.2 Oblikovanje strategije testiranja

Metode oblikovanja testnih scenarijev so lahko povezane v celotno strategijo. Glavni vzrok za to je, da vsaka metoda prispeva specifičen nabor testnih scenarijev, vendar nobena ne pokriva celotnega področja. Myers predlaga uporabo naslednje strategije (Myers, 2004, str. 90):

1. Če specifikacija vsebuje kombinacije vhodnih stanj, začnemo z vzročno-posledičnimi diagrami oz. diagrami ribje kosti. S temi diagrami identificiramo in uredimo možne vzroke problema.
2. V vsakem primeru uporabimo analizo mejnih vrednosti. To je analiza vhodnih in izhodnih mejnih vrednosti.
3. Identificiramo ustrezne in neustrezne skupine vhodov in izhodov.
4. Uporabimo metodo naključnega iskanja napak, s katero lahko odkrijemo specifične napake.

Uporaba te strategije ne garantira, da bomo odkrili vse napake. Osnovana je bila zato, da nakazuje razumno pot, po kateri se bomo lotili testiranja.

3.3 Testiranje enot

Testiranje enot se izvaja posamezno na vsaki stopnji modula sistema. Testiranje običajno izvedejo razvijalci sami, preden se enote predajo v integracijski test z ostalimi enotami. Za testiranje na tej stopnji se uporablja metoda bele skrinjice. Testiranju enot lahko rečemo tudi

testiranje modulov, saj predstavlja najnižjo možno komponento za testiranje (Dustin, 2003, str. 85).

Ločimo tri tipe enot (Dogša, 1993, str. 18):

- navadna enota oz. enota,
- krmilna enota – ima nalogo, da prenese izbrane testne vzorce v enoto, ki jo testiramo, ter nato izpiše izhodne podatke;
- nadomestna enota, ki simulira delovanje originalne enote – sprejme in vrne podatke, ki jih zahteva klicna enota.

Pri izvedbi testiranja enot pa se lahko odločimo za dva pristopa (Dogša, 1993, str. 18):

- izolirano testiranje enot – namen je ločeno preizkusiti posamezne enote, preden jih integriramo v kompletni program;
- inkrementalno (postopno) testiranje enot – najprej testiramo enoto in nato temu dodamo novo, še ne testirano enoto.

3.4 Integracijsko testiranje

Integracijsko testiranje je vmesna faza med testiranjem modulov in sistemskim testiranjem, s katerim se testira medsebojno delovanje in složnost sestavljenega podsistema. Integracijsko testiranje se izvaja postopoma, med sestavljanjem posameznih modulov v podsistem. Podsistem lahko sestavimo na več načinov (Beizer, 1990, str. 176):

- od spodaj navzgor – najprej podsistem sestavimo z moduli iz najnižje stopnje, ki jih potem zamenjamo z moduli višje stopnje;
- od zgoraj navzdol – podsistem sestavimo z moduli najvišje stopnje, ki jih potem zamenjujemo z moduli nižje stopnje;
- mešanica pristopa od spodaj navzgor in pristopa od zgoraj navzdol.

Z integracijskim testiranjem opravimo delni test sistema, pri katerem ni potrebno čakati na dokončanje vseh komponent. Tako lahko odpravimo napako, ne da bi vplivali na proces razvoja ostalih komponent. Pri testiranju na tej stopnji se uporablja mešanica metod črne in bele skrinjice.

3.5 Sistemsko testiranje

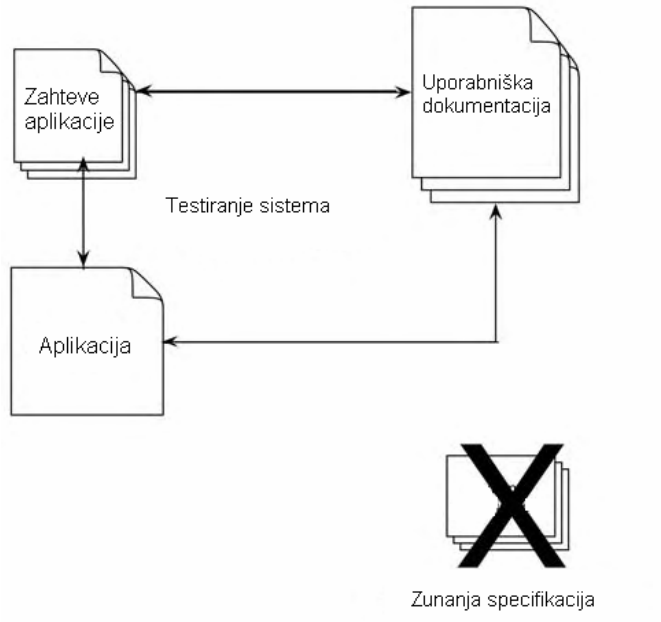
Sistemsko testiranje je najtežavnejši proces testiranja in velikokrat tudi napačno razumljen. Sistemsko testiranje še ne pomeni testiranje delovanja celotnega sistema, saj bi bilo potem funkcionalno testiranje odveč. Osnovna naloga je dokazati, da program v popolnosti ne ustreza vsem postavljenim kriterijem (Myers, 2004, str. 130):

1. Sistemsko testiranje ni omejeno na sistem. Če je produkt aplikacija, testiranje sistema poskuša prikazati, kako aplikacija kot celota ne ustreza zahtevam.
2. Sistemsko testiranje ni možno, če ni zapisanih in merljivih zahtev produkta.

Testne primere za sistemsko testiranje načrtujemo s pomočjo analize zahtev, izoblikujemo pa jih z analizo uporabniške dokumentacije. Slika 14 prikazuje, zakaj je testiranje sistema najtežji proces testiranja. Leva puščica na sliki ponazarja primerjanje programa z njegovimi zahtevami, kar je glavni namen sistemskega testiranja. Zaradi odsotnosti metod za sistemsko testiranje ta proces zahteva dodatno mero ustvarjalnosti. V nadaljevanju navajam različne kategorije, ki jih lahko uporabimo pri oblikovanju testnih scenarijev in nekatere izmed teh sem uporabil v planu testiranja Time&Space sistema:

- testiranje obsega podatkov,
- testiranje stabilnosti,
- testiranje uporabnosti,
- testiranje varnosti sistema,
- testiranje delovanja,
- testiranje skladiščenja podatkov,
- testiranje konfiguracije,
- testiranje kompatibilnosti,
- testiranje nameščanja,
- testiranje zanesljivosti,
- testiranje obnovitve delovanja,
- testiranje dokumentacije,
- testiranje postopkov.

Slika 14: Sistemsko testiranje



Vir: The art of software testing, 2004, str. 131.

3.5.1 Testiranje obsega podatkov

Pri tem procesu sistem ali program izpostavimo velikim količinam podatkov. Cilj tega testa je dokazati, da sistem ne prenese obsega podatkov, navedenih v njegovih zmožnostih.

3.5.2 Testiranje stabilnosti

Pri testiranju stabilnosti sistema ali programske opreme le-te izpostavimo velikemu »bremenu« oz. »naporu«. Tega ne smemo zamenjati s testiranjem obsega podatkov. Največja možna količina podatkov ali aktivnosti v kratkem časovnem razponu predstavlja velik napor. Npr. testiranje obsega podatkov pri stenografu bi pomenilo, da stenograf uspešno uredi dolga poročila. Testiranje stabilnosti pa bi pomenilo, da lahko stenograf tipka 70 besed na minuto. Eden od najbolj pogostih uporabnikov testiranja stabilnosti so internetne aplikacije.

3.5.3 Testiranje uporabnosti

Naslednja pomembna kategorija pri sistemskem testu je iskanje problemov pri uporabi programa. Velik del testiranja uporabnosti zajema testiranje uporabniškega vmesnika (UI[2]), preko katerega uporabnik komunicira z aplikacijo. Vsaka aplikacija ima neke vrste UI. Sodobni računalniški programi pa imajo grafični uporabniški vmesnik (GUI[3]). Dober uporabniški vmesnik naj bi imel naslednje lastnosti (Patton, 2005, str. 211):

- sledi standardom in trendom,
- intuitiven,
- skladen,
- prilagodljiv,
- udoben,
- pravilen,
- uporaben.

Pri testu uporabnosti se je treba posvetiti sledečemu (Myers, 2004, str. 135):

- Ali je bilo vsako uporabniško okno narejeno v skladu z inteligenco, stopnjo izobrazbe in dejavniki okolja končnega uporabnika?
- Ali so izhodni podatki sistema vsebinsko ustrezni in primerni za nadaljnjo uporabo?
- Ali so sporočila napak v sistemu razumljiva, tako da jih brez posebnega računalniškega znanja razumemo? Npr. če sistem javi naslednje sporočilo »AEK36955 ERROR IN FILE COMPACT.DLL CODE=6698«. Takšna sporočila so bila pogosta v sistemih v 1970-ih in 1980-ih. V današnjih sistemih, ki so namenjeni

[2] UI – user interface (angl.).
[3] GUI – graphical user interface (angl.).

masovni uporabi, še vedno zasledimo sporočila, kot so: »Prišlo je do nepričakovane napake.« ali »Prišlo je do napake, zato morate zapreti program.«

- Ali uporabniški vmesniki odražajo pomembno pojmovno celoto, složnost, enotnost sintakse, formata, sloga in kratic?
- V sistemih, kot so npr. bančni sistemi, kjer se preverjajo različni vhodni podatki, je točnost ključnega pomena. Te vrste bančni sistem bi moral zahtevati številko računa, ime lastnika in osebno identifikacijsko kodo (PIN[4]) kot zagotovilo, da do računa dostopa prava oseba.
- Ali sistem vsebuje prekomerno število možnosti ali opcij, ki se bodo uporabljale zelo redko? Eden od trendov v moderni programski opremi je ta, da uporabniku ponudimo samo tiste možnosti, ki jih najpogosteje uporablja.
- Ali se sistem na vse vhodne akcije odziva na enak način? Npr. z miško kliknemo na polje in posledica je takoj vidna. V drugem primeru, ko gre za akcijo, ki zahteva več časa za izvedbo, se mora pokazati sporočilo, ki uporabnika obvesti, kaj se dogaja.
- Ali je sistem enostaven za uporabo? Npr. ali je geslo za vstop občutljivo na velike in male črke brez vednosti uporabnika? Ali je jasno prikazano, kako se vrniti na osnovni meni, če je na razpolago cela vrsta menijev?

3.5.4 Testiranje varnosti sistema

Ker se v današnji družbi povečuje skrb za varovanje podatkov vseh vrst, imajo nekateri sistemi posebne varnostne naloge. Testiranje varnosti je proces, v katerem skušamo z izmišljenimi vrednostmi pretentati varnostno kontrolo sistema. Še posebej spletne aplikacije zahtevajo večji poudarek na testiranju varnosti, če imamo na primer opravka s spletno trgovino in bančništvom.

3.5.5 Testiranje delovanja

Mnogi sistemi imajo specifično določene zmogljivosti, kot so odzivni čas, čas obdelave in nastavitvene možnosti. S testom hočemo dokazati, da sistem ni narejen tako, da zadovolji te specifične zahteve.

3.5.6 Testiranje skladiščenja podatkov

Sistemi občasno vsebujejo zahtevo, ki določa količino primarnega in sekundarnega pomnilnika in velikost začasnih datotek. Pri oblikovanju testnega scenarija skušamo oblikovati takšen test, s katerim želimo dokazati, da sistem ne izpolnjuje te zahteve.

[4] PIN – personal identification number (angl.).

3.5.7 Testiranje konfiguracije

Kompleksni operacijski sistemi, SUBP[5] ali komunikacijska programska oprema podpirajo zelo široko paleto strojne opreme, veliko vhodnih in izhodnih naprav ter komunikacijskih poti. Pogosto je število možnih konfiguracij preveliko, da bi testirali vse kombinacije. Zato je dobro, da sistem testiramo vsaj s strojno opremo, kjer uporabimo minimalno in maksimalno število teh naprav. Današnje aplikacije so narejene tako, da delujejo na več operacijskih sistemih, zato morajo v test biti vključeni vsi podprti sistemi. Aplikacije, ki se izvajajo v spletnih brskalnikih, zahtevajo posebno previdnost, predvsem zato, ker se soočamo s poplavo spletnih brskalnikov, ki ne delujejo na enak način v istem in različnih operacijskih sistemih.

3.5.8 Testiranje kompatibilnosti

Večina programske opreme, ki se razvija, ni povsem nova. Običajno gre za zamenjavo nezadostne opreme. Takšne aplikacije imajo specifične zahteve, ki se nanašajo na kompatibilnost in migracije iz obstoječih sistemov. Pri tej vrsti testiranja skušamo dokazati, da zahteve glede kompatibilnosti in migracije ne držijo, npr. naredimo prenos podatkov iz ene baze v drugo.

3.5.9 Testiranje nameščanja

Nekatera programska oprema ima zapleten namestitveni proces. Test instalacijskega procesa je pomemben del testiranja sistema. To še posebej drži, če je ta proces avtomatiziran in je del programskega paketa. Napaka v instalacijskem procesu lahko povzroči negativno mnenje o celotni aplikaciji.

3.5.10 Testiranje zanesljivosti

Cilj vseh vrst testiranja je izboljšanje zanesljivosti programske opreme. Če aplikacija vsebuje specifične zahteve glede zanesljivosti, se specifični testi zanesljivosti lahko izmislijo. Testiranje zahtev zanesljivosti je lahko težavno. Pri on-line sistemu, kot je npr. WAN[6], je lahko zahteva po dostopnosti 99,97 % celotnega časa delovanja sistema. Z nobeno znano metodo ne moremo testirati te zahteve v testnem obdobju več mesecev ali celo let. Aplikacije ali sisteme s skromnimi časovnimi razponi med pojavljanjem napak (MTBF[7]) je možno testirati.

3.5.11 Testiranje obnovitve delovanja

Operacijski sistemi, podatkovne baze in komunikacijske aplikacije imajo mnogokrat zahtevo po samodejni ponovni vzpostavitvi delovanja, če pri temu pride do prekinitev. Eden izmed

[5] SUBP – sistemi za upravljanje baz podatkov.
[6] WAN – wide area network (angl.).
[7] MTBF – mean time between failures (angl.).

ciljev takih sistemov je minimiziranje povprečnega časa ponovne vzpostavitve delovanja (MTTR[8]). Čas nedelovanja sistema povzroči izpad prihodkov podjetja, ker sistem ne deluje (Tabela 3). Eden izmed namenov testa je dokazati, da sistem ne ustreza zahtevam MTTR. Običajno imajo vrednosti MTTR zgornjo in spodnjo vrednost, kar moramo v testu tudi upoštevati.

Tabela 3: Letno število ur delovanja sistema glede na različne zahteve

Zahteve delovanja v %	Število ur delovanja na leto
100	8760
99,9	8751,2
98	8584,8
97	8497,2
96	8409,6
95	8322

Vir: Practical software testing, 2003, str. 234.

3.5.12 Testiranje dokumentacije

Testiranje uporabniške dokumentacije je prav tako področje, s katerim se ukvarja sistemsko testiranje. Osnovni princip izvedbe testa dokumentacije je uporaba dokumentacije pri oblikovanju testov. Uporabniška dokumentacija naj bi bila tudi predmet inšpekcije, ki preverja točnost in jasnost le-te. Vsi navedeni primeri v dokumentaciji morajo biti vključeni v testne scenarije in testirani.

3.5.13 Testiranje postopkov

Mnoge aplikacije so sestavni deli večjih, ne v celoti avtomatiziranih sistemov. Vsaka ročno napisana procedura za sistemske operaterje, administratorje podatkovnih baz in končne uporabnike mora biti testirana v sklopu testa sistema. Administrator podatkovnih baz mora opisati postopek za arhiviranje in restavriranje podatkovne baze programske opreme ali sistema.

3.6 Izvedba testa

Pri izvedbi testa programske opreme je pomembna tudi določitev oseb, ki ga bodo izvedle. Treba se je zavedati, da morajo osebe, ki bodo izvajale sistemski test, biti sposobne razmišljati kot končni uporabniki, kar je pri razvijalcih sistema zelo malo verjetno. Povprečen končni uporabnik nima sposobnosti ali strokovnega znanja za izvedbo vseh zgoraj omenjenih testov, zato bi bila idealna testna ekipa sestavljena iz (Burnstein, 2002, str. 124):

[8] MTTR – mean to recovery (angl.).

- vodje testiranja – upravlja projekt testiranja;
- končnih uporabnikov sistema ali aplikacije – prispevajo izkušnje iz realnega okolja;
- razvijalci programske opreme – izdelava simulatorjev, gonilnikov itd.;
- eksperti za testna orodja in okoliščine – specifična orodja in podpora pri oblikovanju okoliščin za izvedbo testa;
- testnih svetovalcev – svetujejo glede strategije in metodologije.

Osebnostne lastnosti, ki jih naj bi krasile testerja, so naslednje (Patton, 2005, str. 63):

- So raziskovalci – testerji programske opreme se ne bojijo tvegati v novih situacijah. Uživajo v raziskovanju novih stvari.
- So iskalci in odpravljalci težav – testerji so dobri v iskanju vzrokov za težave. Radi se ukvarjajo z reševanjem ugank.
- So nepopustljivi – testerji nikoli ne odnehajo. Če se napaka pojavi samo enkrat ali jo je težko ponoviti, bodo poskušali na vse možne načine ugotoviti vzrok.
- So kreativni – testiranje razumljivega ne zadovolji testerja. Njihova naloga je razmišljati kreativno in uporabiti nekonvencionalne pristope.
- So perfekcionisti – stremijo za popolnostjo, vendar vejo, kdaj zadeva postane nedosegljiva, in se s tem zadovoljijo.
- So obzirni in diplomatski – testerji navadno prinašajo slabe novice, saj morajo obvestiti programerje, da je njihov »otročiček bolan«. Dobri testerji znajo biti obzirni in profesionalni.
- So prepričljivi – napake, ki jih testerji odkrijejo, ne bodo vedno obravnavane kot resne. Zato morajo biti sposobni podati dobre argumente, zakaj je določeno napako potrebno popraviti in tudi slediti izvajanju.
- Sprejemajo razumne odločitve – testerji morajo določiti, kaj in kako dolgo bodo testirali, in ali je problem, s katerim se ukvarjajo, v resnici napaka.

3.7. Planiranje in kontroliranje testiranja

Če nameravamo testirati zelo velik sistem, moramo računati na pisanje, izvrševanje in preverjanje na stotine testnih scenarijev, obvladovanje velikega števila modulov, popravljanje »sto in ene« napake, zaposlovanje »ducata« ljudi. Da bi uspešno izvedli planiranje, nadzor in kontroliranje testiranja sistema, potrebujemo znanje in izkušnje na področju projektnega managementa.

Ena izmed pogostih napak, ki se pojavijo pri planiranju testiranja, je tiho domnevanje, da ne bomo odkrili preveč napak. Logična posledica te napake je podcenjevanje planiranih sredstev (ljudi, časovne periode testa), kar je razširjen problem v računalniški industriji. Dejstvo je tudi, da proces testiranja velikokrat nastopi na koncu razvojnega cikla, kar pomeni, da je obseg sredstev težko spremeniti. Naslednji problem, s še večjo težo, je, da izvajalci testa nepravilno razumejo testiranje (Burnstein, 2003, str. 263).

Planiranje testiranja je ključno poglavje upravljanja procesa testiranja. Če hočemo narediti dober načrt testiranja, moramo upoštevati naslednje elemente, uporabljene tudi v mojem načrtu za testiranje Time&Space sistema, ki so povzeti po standardu IEEE Std[9] 829-1998:

1. Naloge – Definirati moramo naloge za vsako fazo testiranja.
2. Merilo zaključka – Oblikovati moramo kriterij, s katerim določimo, kdaj končati posamezno fazo testiranja.
3. Urniki – Za vsako fazo testiranja moramo določiti časovna obdobja, v katerih moramo navesti, kdaj bomo oblikovali, napisali in izvajali testne scenarije.
4. Obveznosti – Za vsako fazo testiranja moramo določiti ljudi, ki bodo oblikovali, izdelovali in izvajali testne scenarije, in ljudi, ki bodo popravljali odkrite napake.
5. Knjižnice in standardi testnih scenarijev – V velikih projektih je sistematiziranje metod identificiranja, pisanja in shranjevanja testnih scenarijev nujno.
6. Orodja – Potrebna orodja za izvajanje testa morajo biti specificirana. To vključuje načrt o razvoju in nakupu orodij ter kdo in kako jih bo uporabljal.
7. Konfiguracija strojne opreme – Če v testu potrebujemo specifične dele strojne opreme, moramo imeti plan, v katerem opišemo zahteve, nastavitve in časovno uporabo.
8. Integracija – V tem delu plana opredelimo, kako bomo sestavili sistem. Če sistem vsebuje večje podsisteme, ga lahko sestavimo inkrementalno ter pri tem uporabimo »top-down« ali »botton-up« pristop. Plan sistemske integracije definira red integracije in funkcionalno sposobnost vsake verzije sistema.
9. Postopki iskanja in odpravljanja napak – Postopki morajo biti definirani tako, da sporočajo zaznane napake, sledijo procesu popravkov in prispevajo k pravilnemu delovanju sistema.
10. Ponovno testiranje – Ponovno testiranje se izvede, ko so napake v sistemu odpravljene ali je popravljena funkcionalnost sistema. Namen ponovnega testiranja je ugotoviti, če so spremembe v sistemu povzročile neželene stranske učinke. Ponavadi ponovno izvedemo že prej kreirane testne scenarije, s katerimi smo odkrili napako.

3.7 Merilo za končanje testiranja

Precej velik problem nastane pri odločitvi, kdaj končati testiranje aplikacije ali sistema, saj nikakor ne moremo ugotoviti, ali nam je uspelo odpraviti vse napake. Dejstvo je, da z izjemo majhnih aplikacij ne moremo pričakovati, da bomo odkrili vse napake. Tako se velikokrat soočamo z dilemo, kdaj končati testiranje in kakšne kriterije pri tem upoštevati.

Kriteriji, ki so uporabljeni v praksi, so nepomembni in neproduktivni. Najpogosteje srečamo naslednja dva kriterija (Craig, 2002, str. 181):

- končamo, ko se izteče načrtovano časovno obdobje za testiranje;

[9] IEEE Std – Institute of Electrical and Electronics Engineers, Standard for Software Test Documentation (angl.)

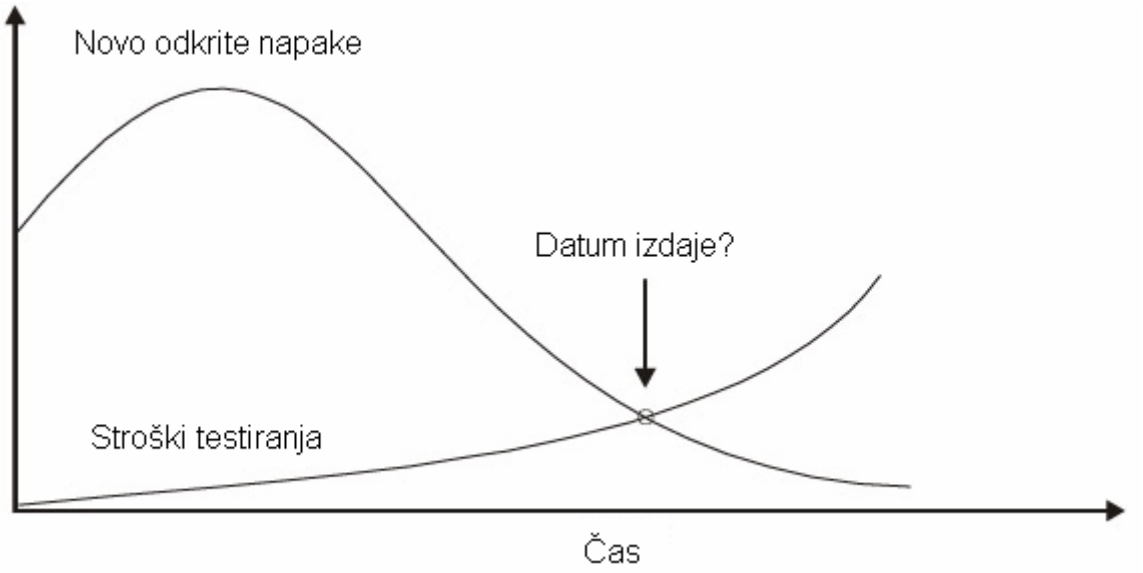
- končamo, ko pri izvajanju vseh testnih scenarijev ne odkrijemo novih napak.

Prvi kriterij je neuporaben, saj ga lahko izpolnimo tudi z brezdeljem in ne zagotavlja kakovosti testiranja. Drugi kriterij je tudi neuporaben, saj je neodvisen od kvalitete testnih scenarijev. Še več, je tudi neproduktiven, saj nas podzavestno vzpodbuja k pisanju testnih scenarijev, ki imajo majhne možnosti za izsleditev napak. V nadaljevanju si bomo pogledali kriterije, s katerimi si pomagamo pri odločitvi, kdaj končati proces testiranja.

3.7.1 Stopnja odkritih napak

Mnoga podjetja si pomagajo s stopnjo odkritih napak pri odločanju, kdaj končati s testiranjem. Ko stopnja odkritih napak pade pod določeno mejo, lahko predvidevamo, da je produkt pripravljen za izdajo. Padajoča stopnja je dober znak, vendar moramo tukaj upoštevati tudi druge dejavnike, kot so npr. manj vloženega truda, malo novih testnih scenarijev. Dobro je, da tako pomembno odločitev baziramo na več odločitvenih kriterijih. Kot je prikazano na Sliki 15, se število odkritih napak dnevno zmanjšuje, pri tem predvidevamo, da je vloženi trud na enaki ravni, stroški vsake nove napake pa se zvišujejo.

Slika 15: Stopnja odkritih napak



Vir: Software Quality Engineering, 1990, str. 211.

3.7.2 Merilo preostalih napak

Ena izmed metod, s katero si lahko pomagamo pri odločitvi, kdaj poslati izdelek »na police«, je ocenjeno število pričakovanih napak. To naredimo tako, da primerjamo trend odkritih napak pri našem projektu s kakšnim podobnim projektom. Da pa to lahko izvedemo, moramo zbrati in urediti obširno količino podatkov iz preteklosti. Podatke lahko med seboj primerjamo le, če je bila opravljena normalizacija podatkov (Burnstein, 2003, str. 293).

3.7.3 Pomanjkanje sredstev

V realnosti se pogostokrat zgodi, da je odločilen kriterij za končanje testiranja pomanjkanje časa in denarja. Večkrat se zgodi, da se podjetje odloči za izdajo produkta, čeprav ve, da ima produkt težave na določenih področjih. Za to se odloči, ker je ta različica boljša od tiste, ki jo ima uporabnik. Preden se odločimo za ta korak, se splača dobro premisliti o tem, kaj je bolj tvegano – izdati slab produkt ali ne izdati produkta ob predvidenem roku.

3.8 Merske enote v procesu testiranja

3.8.1 Stopnja odpravljenih napak

V računalniški industriji obstaja mit, ki govori o tem, da so vse najdene napake v procesu testiranja odpravljene, preden izdamo produkt. Študije, ki so bile izvedene leta 1998 in 2000, so pokazale, da je dejanska stopnja odpravljenih napak med 80 in 85 odstotkov v posameznih komercialnih aplikacijah (Tian, 2003, str. 324). Stopnja odpravljenih napak nam pove, koliko odkritih napak je bilo dejansko tudi popravljenih. Izračunamo jo po naslednji enačbi (Hutcheson, 2003, str. 188):

$$\text{Stopnja odpravljenih napak} = \frac{\text{Število odpravljenih napak v procesu testiranja}}{\text{Število najdenih napak v procesu testiranja}} x100$$

3.8.2 Testna pokritost sistema

Testna pokritost sistema nam pove, koliko testnih scenarijev je bilo izvedenih na sistemu. Gre za relativno mero, ker ne moremo narediti in izvesti toliko testnih scenarijev, ki bi pokrili vse primere iz realnosti. Izračunamo jo po naslednji enačbi (Hutcheson, 2003, str. 190):

$$\text{Testna pokritost sistema} = \frac{\text{Število izvedenih testov}}{\text{Število vseh testov}} x100$$

3.8.3 Učinkovitost testiranja

Učinkovitost testiranja je mera, ki nam pove, kako učinkoviti so testni scenariji pri iskanju napak. Izračunamo jo po naslednji enačbi (Hutcheson, 2003, str. 193):

$$\text{Učinkovitost testiranja} = \frac{\text{Število odkritih napak v procesu testiranja}}{\text{Celotno število odkritih napak}} x100$$

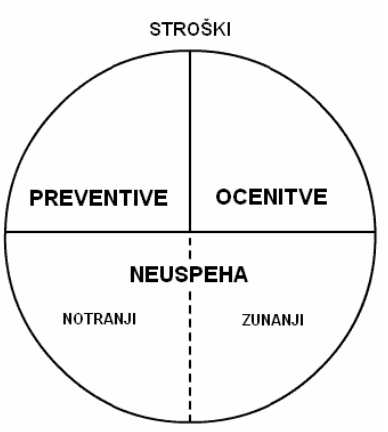
Celotno število odkritih napak izračunamo kot seštevek števila odkritih napak s strani testerjev in števila napak, odkritega s strani uporabnikov. Odstotek napak, odkritih s strani testerjev, naj bi znašal vsaj 85 % vseh odkritih napak (Rainsberger, 2005, str. 519).

3.9 Stroški zagotavljanja kakovosti

Potreba po kakovostnem produktu je s strani uporabnikov neizbežna. Na drugi strani se ponudniki srečujejo z vprašanjem, koliko stroškov bo prineslo zagotavljanje določene stopnje kakovosti programske opreme. Philip Crosby, avtor knjige iz leta 1979 »Quality is Free: The Art of Making Quality Certain«, je prepričan, da se strošek izdelave visoko kvalitetne aplikacije ne razlikuje ali je celo nižji od nizko kvalitetne aplikacije. Kako je to mogoče?

Najprej se moramo vrniti na graf s Slike 2, ki nam prikazuje, da se stroški odprave napake eksponentno povečujejo glede na čas razvoja programske opreme. Stroški zagotavljanja kakovosti so vsota stroškov, ki nastanejo v podjetju z namenom preprečitve slabe kvalitete produkta. Slabo kvaliteto povezujemo z aktivnostmi brez dodane vrednosti, izgubo, napakami ali neuspehi pri zagotavljanju potreb in zahtev kupca. Stroške zagotavljanja kakovosti lahko razdelimo v tri kategorije: preventivni, ocenitve in stroški neuspeha. Model stroškov zagotavljanja kakovosti pogosto imenujemo tudi PAF[10] model, po imenih kategorij (Slika 16).

Slika :16 Model PAF



Vir: Beecroft, 2000, str. 3.

Preventivni stroški: So načrtovani stroški, ki nastanejo v podjetju z zagotovilom, da nismo naredili napak v procesu izdelave programske opreme. Primeri preventivnih stroškov so stroški izobraževanja in šolanja, nenehnega izboljševanja, nadzora procesa, raziskav trga, testiranja in preventivnega vzdrževanja. Stroški ocenitve**:** So stroški preverjanja, dokazovanja ali vrednotenja produkta ter storitve v posameznih razvojnih stopnjah. Primeri stroškov

[10] PAF – prevention, appraisal and failure costs (angl.).

ocenitve so stroški pregleda, notranje revizije, aktivnosti, povezanih s pregledom, izdelavo ocenitvenih in revizijskih poročil. Stroški neuspeha**:** So stroški, ki nastanejo, ker produkt ali storitev ni zadovoljila zahtev. To je povzročilo popravke ali zamenjavo produkta ter ponovno izvedbo storitve. Stroške neuspeha naprej delimo na notranje in zunanje.

Notranji stroški neuspeha vsebujejo vse stroške, ki so posledica napak, odkritih, preden je bil produkt ali storitev dobavljen stranki. Primeri notranjih stroškov neuspeha so stroški predelav, dodatnega inventarja, ponovnega načrtovanja, reševanja, popravka poročil izvedenih akcij in nadur. Če gledamo graf s Slike 2, ti stroški padejo večinoma na levo stran. Zunanji stroški neuspeha vsebujejo vse stroške, ki nastanejo, ko končni uporabnik najde napako. Ti stroški ne vključujejo stroškov, ki nastanejo na strani uporabnika. Primeri zunanjih stroškov neuspeha so stroški garancij, oddelka podpore, zamenjave produkta, dostave, analize garancijskih primerov. Če gledamo graf s Slike 2, ti stroški padejo na desno stran.

Crosby zagovarja enačbo, po kateri je vsota preventivnih stroškov, stroškov ocenitve in notranjih stroškov neuspeha nižja od zunanjih stroškov neuspeha. To razloži tako, da če se v začetnih fazah razvoja znebimo napak ali jih celo ne kreiramo, bo naš produkt cenejši, kot bi bil v nasprotnem primeru. In smo ponovno na začetku pri trditvi: »Quality is free!« (Crosby, 1985).

4. TIME & SPACE SISTEM

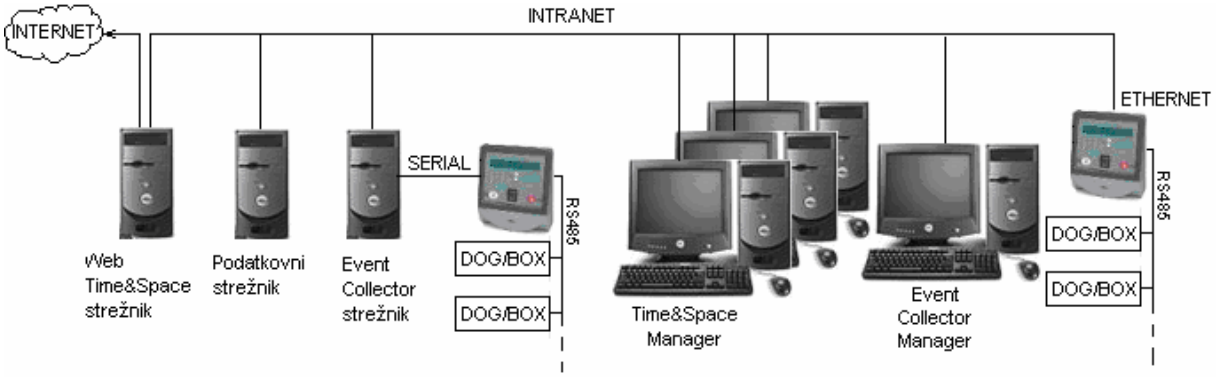
Time&Space (T&S, TS) je odprt in modularen informacijski sistem za kontrolo pristopa, beleženje prisotnosti ter registriranje in evidenco delovnega časa (T&S uporabniški priročnik). Sestavljajo ga programski moduli serije Time&Space, registracijski terminali DOG20, terminali za kontrolo pristopa BOX20 ter identifikacijske kartice, ki so lahko magnetne, brezkontaktne, RFID[11] ali brezkontaktne pametne kartice (smartcard). Na voljo je tudi biometrična identifikacija z uporabo čitalnikov prstnih odtisov. Programska oprema je zasnovana na najbolj razširjeni, standardni strojni arhitekturi osebnega računalnika in operacijskem sistemu Microsoft Windows (sodobne verzije, kot so Windows 2000 ali Windows XP). Sistem Time&Space je zasnovan s poudarkom na fleksibilnosti, skalabilnosti in enostavni uporabi, tako da lahko uspešno pokriva potrebe zelo različnih podjetij in organizacij. Med uporabnike sistema so vključene vladne in finančne ustanove, bolnišnice, maloprodajne verige, obrati, rudniki in tovarne. Na drugi strani je poskrbljeno tudi za pokrivanje različnih konfiguracij, ne glede na velikost organizacije; od najmanjših z nekaj zaposlenimi do največjih z več tisoč zaposlenimi in geografsko razpršeno IT[12] infrastrukturo.

Programska oprema je zasnovana na tehnologiji odjemalec/strežnik in relacijski bazi podatkov. Podprta sta dva najbolj uveljavljena SUBP, Oracle in Microsoft SQL Server. Za

[11] RFID – Radio Frequency Indentification (angl.).
[12] IT - Information Technology (angl.).

osnovno konfiguracijo zadostuje le en računalnik, na katerem teče MS Office Professional. Za manjše sisteme z nekaj deset uporabniki in enim ali dvema administratorjema je primeren tudi MSDE[13]. Registracijski terminal lahko priključimo kar na serijska vrata (COM Port) ali preko omrežja IP[14]. Kontrolo pristopa izvaja registracijski terminal. Večje konfiguracije lahko vsebujejo več registrirnih terminalov, povezanih skozi omrežje IP, mrežo terminalov RS485[15] ali kombinacijo obeh. Oddaljene terminale lahko priključimo v centralni sistem skozi omrežje IP ali preko modemske povezave (Slika 17).

Slika 17: Mrežna arhitektura T&S sistema



Vir: Interna gradiva. 2006.

Registrirni terminali in terminali za kontrolo pristopa (DOG in BOX) so lahko priključeni neposredno skozi omrežje IP (Ethernet) ali preko serijskih vmesnikov oz. modemov.

Sistem skrbi za zbiranje registracij z registracijskih terminalov v realnem času in za sočasni dostop več uporabnikov z delovnih postaj v omrežju. Poskrbljeno je za medprocesno signalizacijo in avtomatično usklajevanje celega sistema pri kritičnih operacijah. Popolna funkcionalnost sistema, vključno s sistemsko administracijo in upravljanjem kontrole pristopa, je dostopna preko odjemalske aplikacije TSM (Time&Space Manager), ki je nameščena na vsakem odjemalcu. Dostop do najpogosteje uporabljanih funkcij za evidenco delovnega časa je omogočen preko interneta in ekstraneta preko standardnega spletnega pregledovalnika. Spletni dostop omogoča komponenta Web Time&Space Server. Web Time&Space (WebTS) občutno zmanjša potrebo po administraciji sistema in omogoča administracijo delovnega časa od koderkoli, brez lokalne instalacije. Kapaciteta sistema je odvisna le od fizične zmogljivosti izbrane konfiguracije (hitrost procesorjev, velikost pomnilnika, velikost in hitrost diskov, hitrost in obremenjenost mreže, hitrost podatkovnega strežnika). Razširljivost sistema je praktično neomejena.

[13] MSDE – Microsoft SQL Server Desktop Engine (angl.).
[14] IP – Internet Protocol (angl.).
[15] RS485 – serijski komunikacijski vmesnik.

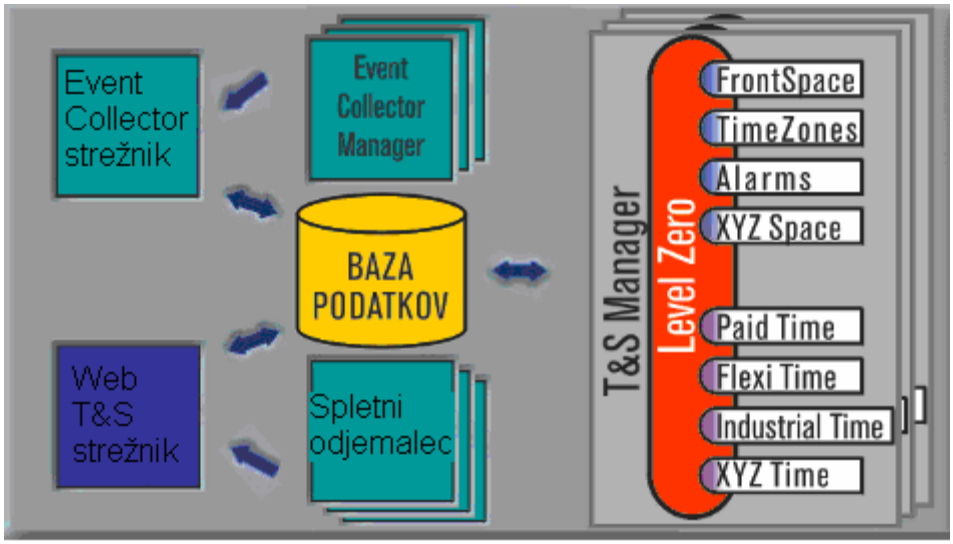
4.1 Programska oprema Time&Space

Programsko opremo sistema Time&Space odlikuje modularnost, ki uporabniku omogoča izbiro tistih komponent, ki jih v določenem trenutku potrebuje, ter morebitno poznejšo nadgradnjo. Dokumentirani relacijski podatkovni model in standardni podatkovni format omogočata integracijo rešitev drugih podjetij, kar omogoča visoko stopnjo povezljivosti z drugimi sistemi in programi. Uporabniški vmesnik vseh modulov, posebej tistih, ki so namenjeni delovnemu času, je grafično usmerjen, tako da lahko operater podatke ureja kar se da hitro in pregledno. Dogodki in urniki so ponazorjeni grafično na časovni osi, rezultati obračuna pa so podani v obliki grafov. V osnovni program je vgrajen generator izpisov, ki uporabniku omogoča krojenje izpisov po lastnih željah. Na voljo je v vseh tabelaričnih pogledih programa. Razen tega je na voljo tudi zaokrožen nabor prednastavljenih poročil. V aplikacijo je vgrajen modul za dodeljevanje operaterskih in uporabniških pooblastil, s katerim je poskrbljeno za zaščito podatkov pred neavtoriziranimi posegi.

4.2 Arhitektura sistema Time&Space

Programsko opremo sestavljata dve osnovni komponenti: Event Collector (EC) in T&S Manager. Dodatni modul Web Time&Space omogoča dostop do sistema preko interneta z uporabo spletnega brskalnika (Slika 18).

Slika 18: Arhitektura sistema T&S

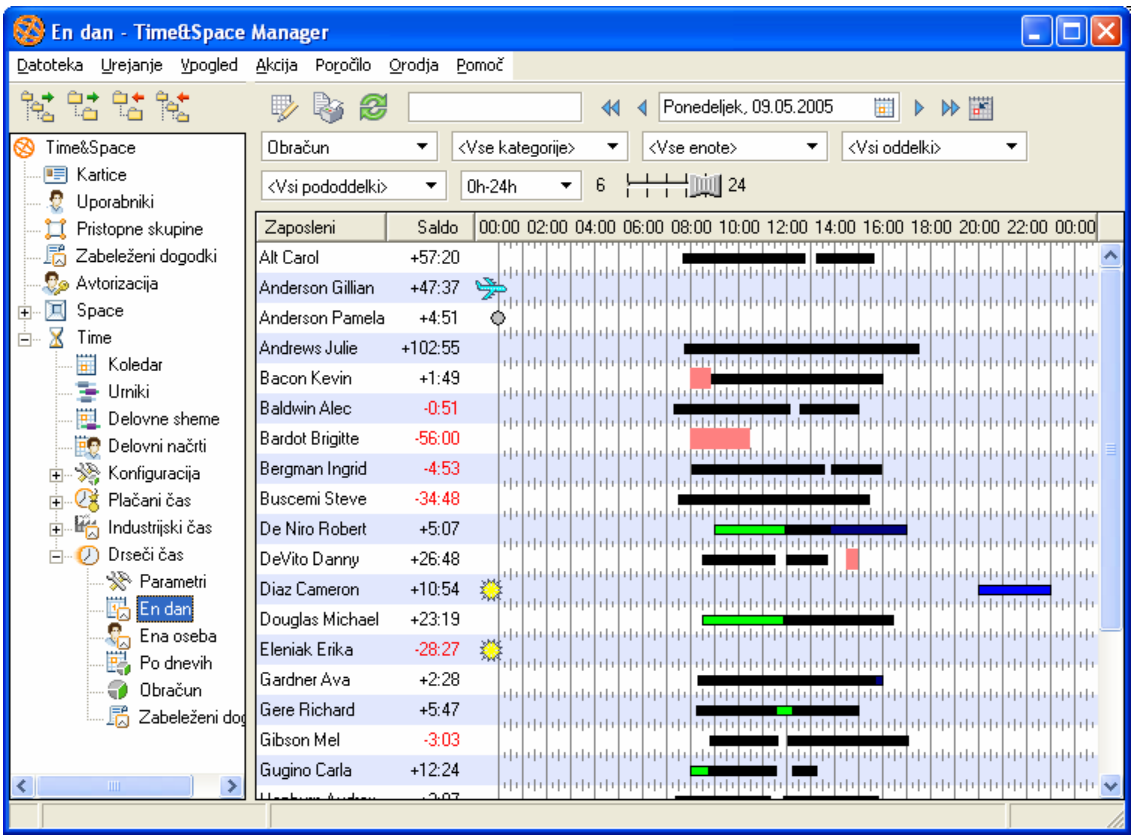


Vir: Interna gradiva. 2006.

EC je komunikacijski program, ki skrbi za izmenjavo podatkov s terminali DOG09 (BOX20). Zadolžen je za vrsto nastavitev v zvezi z zbiranjem dogodkov in nadzorom komunikacije. Med drugim nudi tudi možnost spremljanja registriranja v realnem času. Ko je nastavljen, deluje praktično samodejno, brez operaterskega nadzora.

TSM je glavni uporabniški program, ki vsebuje posamezne aplikativne module glede na izbrano konfiguracijo (Recepcija, Drseči čas itd.). Z njim operaterji administrirajo delovni čas in upravljajo kontrolo pristopa. Najmanj, kar lahko vsebuje, je funkcionalni modul Level Zero, saj ta vzdržuje osnovne podatke v sistemu: bazo uporabnikov, kartic, registrirnih točk, pristopnih pravic in dogodkov (Slika 19).

Slika 19: Time&Space Manager



Vir: Interna gradiva. 2006.

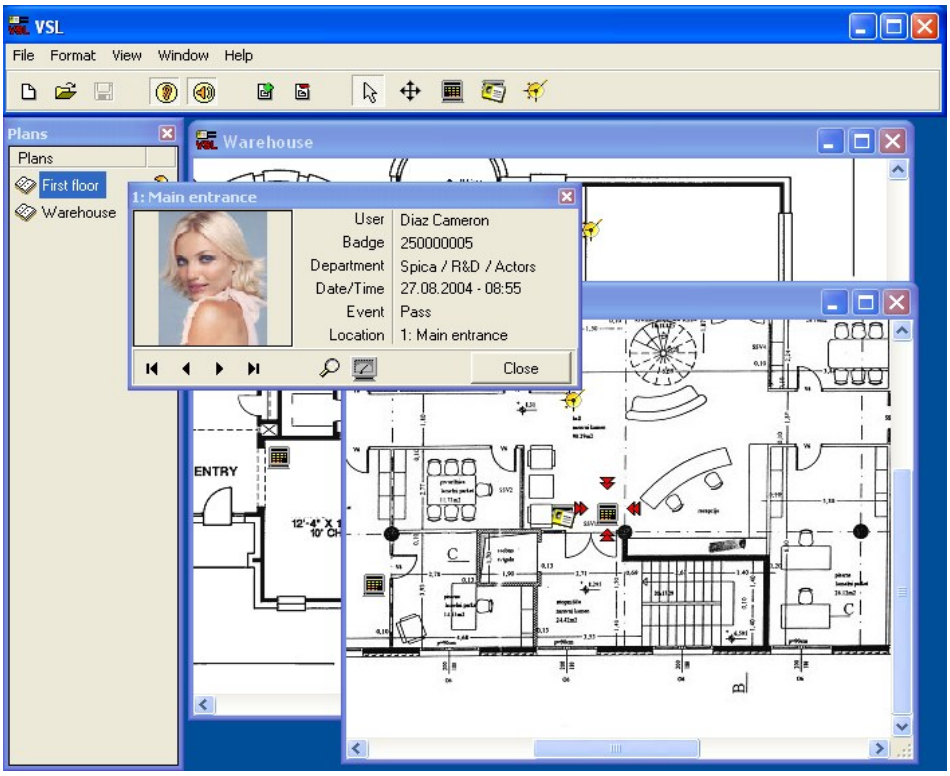
WebTS je preprost odjemalec, ki prek spleta (intranet ali internet) omogoča dostop do najpogostejših funkcij pri beleženju časa in prisotnosti. Omogoča tudi spletni dostop do osebnih podatkov in njihovo urejanje, dostop do podatkov za izbrano skupino uporabnikov, njihovo urejanje in registracijo dogodkov prek spleta.

4.3 Kontrola pristopa

Kontrola pristopa je osnova vsakega sistema Time&Space. Ta modul je zadolžen za vzdrževanje podatkovne baze uporabnikov, kartic, pristopnih točk in dogodkov. Predstavlja programsko platformo za dodajanje ostalih modulov sistema. Nudi osnovni nivo kontrole pristopa na osnovi osebnih pristopnih profilov oz. informacije »kdo sme kam in kdaj«. Omogoča pregledovanje in urejanje vseh podatkov iz skupnega skladišča, vključno z zabeleženimi dogodki. Moduli za kontrolo pristopa omogočajo tudi nadzor alarmnih dogodkov, ki jih lahko prožijo različne alarmne naprave, ki so povezane na pristopni ali registracijski terminal. Omogočajo tudi sledenje alarmnih dogodkov skozi čas, arhiviranje in pregled zgodovine dogodkov.

V kontrolo pristopa je vključen tudi modul za vizualni nadzor, VSL[16] (Slika 20). Omogoča grafično prostorsko spremljanje Time&Space dogodkov z uporabo zemljevidov, načrtov nadstropij ali celo zračnih posnetkov območja. Dogodki so prikazani z animiranimi ikonami na zemljevidu. Za posamezne tipe dogodkov (npr. požarne alarme) se lahko sprožijo tudi zvočni signali in prikažejo osnovni podatki o uporabniku, ki ga je sprožil, vključno z njegovo fotografijo.

Slika 20: Modul VSL



Vir: Interna gradiva. 2006.

[16] VSL – Visual Surveillance (angl.)

Kontroli obiskovalcev je namenjen nadgraditveni modul Recepcija**.** Kontrola pristopa obiskovalcev in njihovo sledenje temeljita na osnovi dodeljevanja začasnih identifikacijskih kartic. Recepcija nudi pregled nad trenutno aktivnimi in zaključenimi obiski, s pripadajočimi podatki o obiskovalcih in gostiteljih.

Nadgraditveni modul za digitalni video nadzor, imenovan DVS[17], omogoča snemanje in arhiviranje video posnetkov dogodkov v sistemu Time&Space. Tako lahko snemamo različne dogodke, npr. odpiranje vrat, registracije zaposlenih in alarmne dogodke. DVS uporablja spletne kamere (IP kamere), kar eliminira potrebo po pretvornikih analognega v digitalni signal in podobni strojni opremi. Video posnetki so vezani na dogodke v sistemu Time&Space, kar bistveno olajša iskanje in pregledovanje. Omogočeno je tudi neprekinjeno snemanje z možnostjo premikanja kamere in snemanje ob zaznavi gibanja.

4.4 Plan testiranj T&S sistema

Osnova pri izdelavi mojega testnega plana za T&S sistem je standard IEEE Std 829-1998. Omenjeni standard predpisuje elemente, ki naj bi jih testni plan vseboval. Dobrega testnega plana ne moreš izdelati brez izkušenj in natančnega poznavanja obravnavanega sistema ali aplikacije. Pri tem nisem imel težav, saj že slabi dve leti testiram T&S sistem. Če poenostavljeno predstavim vsebino testnega načrta, lahko rečem, da mora vsebovati odgovore na naslednja tri vprašanja: kaj, kdo in kako? V nadaljevanju tega poglavja bom predstavil odgovore na ta vprašanja. Celoten testni plan T&S sistema je v prilogi.

Vsak dober plan mora vsebovati tudi cilje in pričakovane rezultate, ki služijo kot vodilo in motivacija pri izvedbi testiranja. Cilji in pričakovani rezultati testnega plana, kot sledi, so:

Cilj testiranja je:

- preveriti pravilnost delovanja sistema glede na uporabniške zahteve;
- zagotoviti, da bo v sistemu ob predaji kar najmanj napak;
- preveriti prisotnost in pravilnost delovanja kontrol;
- preveriti delovanje sistema v povezavi s ostalimi moduli;
- preveriti delovanje sistema po nenačrtovanih prekinitvah.

Pričakovani rezultati testiranja so:

- pravilnost obračuna delovnega časa;
- izdelava in izpis različnih vrst poročil;
- načrtovanje in odpiranje obiskov;
- izvoz podatkov;
- spreminjanje obračunskih pravil.

[17] DVS - Digital Video Surveillance (angl.)

4.4.1 Vrste testiranja

Pri testiranju T&S sistema se bo izvajalo testiranje funkcionalnosti, uporabnosti, združljivosti, varnosti, zmogljivosti, okrevanja, uporabnosti in dokumentacije.

4.4.1.1 Testiranje funkcionalnosti

Cilj te vrste testiranja je priti do zaključka, da program v celoti ne ustreza zahtevani specifikaciji. Pozornost je treba usmeriti na protokol testiranja, ki mora zagotavljati ponovljivost testnega postopka. Takšno testiranje temelji na metodi črne skrinje.

4.4.1.2 Testiranje nameščanja

Pri testu nameščanja T&S sistema se skuša dokazati, da se le-ta ne namesti pravilno. Preveriti je potrebno, če se izvede nadgradnja obstoječe podatkovne baze že obstoječih modulov in instalacijo na novi delovni postaji.

4.4.1.3 Testiranje združljivosti

S testom združljivosti se skuša dokazati, da T&S 6.60 ni združljiv s predhodnimi verzijami sistema. Pri tem je poudarek predvsem na ohranitvi stanja salda pred in po nadgraditvi sistema in delovanju obračunskih pravil po ponovitvi obračuna.

4.4.1.4 Testiranje varnosti

Skuša se ugotoviti, ali T&S sistem izpolni zahteve glede varovanja podatkov. Predvsem se preverja prijavljanje v sistem z različnimi nivoji avtorizacije, prikazovanje in upravljanje s podatki glede na pravice.

4.4.1.5 Testiranje zmogljivosti programske opreme

S testom prekoračitve se bo T&S Manager izpostavil daljšim časovnim intervalom neobičajno velikih količin vhodnih podatkov. Skuša se ga pripraviti do odpovedi ali nepravilnega delovanja ter stresa, pri katerem T&S izpostavimo zelo velikim trenutnim prekoračitvam. Dokazati se želi, da delovanje T&S sistema v takšnih pogojih ni pravilno. Pri generiranju vhodnih podatkov si bomo pomagali z aplikacijo ECstress[18], ki simulira delovanje terminalov in pošilja poljubno število dogodkov v T&S sistem.

[18] ECstress – aplikacija za simuliranje delovanja registracijskih terminalov.

4.4.1.6 Testiranje okrevanja po napakah oz. ponovnega zagona

Namen je dokazati, da se ponovni zagon programa ne izvede v skladu z zahtevami oziroma da pride pri odpovedi programa do izgube podatkov. Osredotočiti se je potrebno na izgubo povezave s SQL strežnikom, izpadom električne energije in mrežne povezave.

4.4.1.7 Testiranje uporabnosti

Za razliko od ostalih tipov testiranja ta oblika ne išče napak v izvajanju algoritmov, temveč v zasnovi uporabniškega vmesnika. Pri testu uporabnosti v T&S Managerju se pozornost namenja sledečemu:

- Ali so izhodni podatki sistema vsebinsko ustrezni in primerni za nadaljnjo uporabo?
- Ali so sporočila napak v sistemu razumljiva, tako da jih brez posebnega računalniškega znanja razumemo?
- Ali uporabniški vmesniki odražajo pomembno pojmovno celoto, složnost, enotnost sintakse, formata, sloga in kratic?
- Ali sistem vsebuje prekomerno število možnosti ali opcij, ki se bodo uporabljale zelo redko?
- Ali se sistem na vse vhodne akcije odziva na enak način?

4.4.1.8 Testiranje dokumentacije

S tem testom se preverja natančnost uporabniške dokumentacije. Tako se zagotovi, da so vse funkcionalnosti opisane in da je gradivo prijazno uporabniku.

4.4.2 Testno okolje

V procesu testiranja se bo uporabljalo 10 osebnih računalnikov HP Pentium 4 (min 1.5 GHz, min 1 Gb RAM, trdi disk 80 Gb) z operacijskim sistemom Windows XP. Računalniki so povezani v lokalno omrežje. Na vsakem računalniku bo nameščen program VMware, ki bo imel slike sistemov: Windows 2000, Windows 2003 server, Windows XP.

4.4.2.1 Testna ekipa in odgovornosti

Testno ekipo sestavljajo:

- vodja testiranja,
- tester 1 – Spica int., d. o. o.,
- tester 2 – Spica int., d. o. o.,
- tester 3 – oddelek za podporo,
- tester 4 – oddelek za podporo,
- tester 5 – vodja prodaje,

- tester 6 – komercialist T&S sistema,
- tester 7 – komercialist T&S sistema,
- tester 8 – serviser T&S,
- tester 9 – serviser T&S,
- SQL server administrator,
- manager T&S projekta.

Pri formuliranju testne skupine se vodja testiranja seznani z njihovim znanjem na področju T&S sistema in metodami testiranja ter poskrbi za njihovo izobraževanje. Vodja testiranja in projektni manager T&S odločata o začetku in koncu testiranja sistema v okviru terminskega plana. Vodja testiranja je odgovoren za koordinacijo terminov, opremo in orodja testerjev, kot tudi za posodabljanje testnega plana, pisanje tedenskih testnih poročil in končnega testnega poročila. Testerji so odgovorni za pisanje testnih scenarijev in izvršitev le-teh. Administrator SQL strežnika skrbi za administracijo testnih baz.

4.4.2.2 Pravila, postopki in dogovori

Plan testiranja se nanaša na aplikacijo T&S Manager. Ostali moduli iz T&S sistema niso predmet tega plana. Uporablja se jih samo kot podporo pri testiranju T&S Manager aplikacije. Demo bazo za testiranje pripravi SQL server administrator. Testni scenariji morajo biti pripravljeni v skladu s priporočili za pripravo testnih podatkov. Testirajo se vse mejne vrednosti: najnižja in najvišja vrednost, vrednosti zunaj določenega intervala in vrednosti znotraj intervala. Pri kreiranju testnih primerov ne smemo pozabiti na napačne vnose v okviru ekvivalentnega razreda.
Testni primer vsebuje naslednje podatke:

- datum testiranja,
- ime in priimek izvajalca testiranja,
- ime in verzijo programskega proizvoda, ki se testira,
- opis testnega primera,
- vhodne podatke,
- pričakovane rezultate,
- dejanske rezultate.

Vsak tester ima na testni platformi v Outlooku odprt svoj test, za katerega poskrbi vodja testiranja. Izvajalci testiranja objavijo ugotovljene napake na testni platformi. Pri opisu napake morajo biti navedeni naslednji podatki:

- pri katerem postopku je prišlo do napake,
- vhodni podatki,
- dobljeni rezultati,
- pričakovani rezultati.

5. SKLEP

Pomena kakovosti storitve se zelo dobro zavedajo storitvena podjetja z dolgo tradicijo. Zavedajo se, da je njihov položaj na trgu odvisen predvsem od celovite kakovosti njihovih produktov in storitev. Podjetja, ki se ukvarjajo z razvojem programskih rešitev, pa morajo skrbeti, da njihove aplikacije delujejo čimbolj brezhibno in izpolnjujejo specifikacije, ki so jih upoštevali pri razvoju.

Glede na kompleksnost in velikost današnje programske opreme je pisanje programske kode, ki ne vsebuje napak, zelo težko izvedljivo tudi za najbolj izkušene programerje. Zato testiranje programske opreme spada med kritične naloge pri razvoju programske opreme, ki se mora izvesti profesionalno in učinkovito. Z nenehnim večanjem zaupanja v programsko opremo pri vsakodnevnih opravilih in prodornostjo le-te v medicino, telekomunikacije, proizvodno in finančno industrijo lahko programska napaka povzroči katastrofo. Kakovostne programske opreme ne moremo narediti z ad–hoc, delnim ali naključnim testiranjem. Za to je potreben načrten in discipliniran pristop pri preprečevanju, iskanju in prijavljanju napak.

Izdelava testnega plana, tudi za majhen projekt, predstavlja obsežno nalogo, ki je ne smemo podceniti. Čeprav bi bilo najlažje izpolniti prazne obrazce in natisniti kopije testnega plana, s tem ne bi zajeli bistva te naloge. Planiranje testiranja je naloga, ki naj bi vključevala celotno testno ekipo in vse ključne osebe iz razvojnega tima. Čas, ki je potreben za temeljito izdelavo testnega načrta, lahko obsega več tednov. Izdelava izčrpnega in razumljivega testnega načrta, kjer je določeno kaj in kako bomo testirali, bo pripomogla k boljši izvedbi v primeru, da v osnovnih načrtih razvoja produkta pride do spremembe.

Pri vsakem projektu se soočamo s tremi ključnimi komponentami: s časom, kakovostjo in denarjem. Pri razvoju programske opreme se velik del testiranja običajno izvaja na koncu razvoja programske opreme. Takrat nastane velik pritisk, zaradi bližajočega se dobavnega roka, preseženih načrtovanih stroškov in zahteve po dobrem produktu. Brez kakovostnega testnega načrta lahko še prehitro izgubimo bitko. Pritisk lahko zmanjšamo tako, da začnemo testirati že v začetku razvoja programske opreme.

V drugem delu diplomskega dela sem izdelal testni načrt za Time&Space sistem, ki ga testiram že slabi dve leti. Gre za kompleksen on-line sistem, ki služi za kontrolo pristopa in obračun delovnega časa. Glede na obsežnost in zapletenost sistema je sistem potrebno dobro testirati, preden ga implementiramo pri stranki. Obračunani podatki v sistemu se uporabljajo pri izračunu plač zaposlenih. To je področje, pri katerem je večina ljudi lahko zelo občutljiva, zato je tukaj potrebna še posebna natančnost. Rezultatov, ki jih bomo dosegli s tem testnim planom, še ne morem navesti, saj ga še nismo začeli izvajati.

Literatura

1. Beecroft G. D.: Cost of Quality, Quality Planning and The Bottom Line. 6 str. [URL: http://www.bisrg.uwaterloo.ca/archive/RR-01-08.pdf], 10. 8. 2006.
2. Beizer B.: Software Testing Techniques. New York : The Coriolis Group, 1990. 176 str.
3. Burnstein I.: Practical software testing. New York : Springer-Verlag, 2003. 583 str.
4. Craig D. R., Jaskiel S. P.: Systematic Software Testing. Boston : Artech House, 2002. 536 str.
5. Crosby P.: Quality Improvement Through Defect Prevention. Boston : Philip Crosby Associates, 1985. 286 str.
6. Dustin E.: Effective software testing: 50 specific ways to improve your testing. Boston : Pearson Education, 2003. 203 str.
7. Falk J., Kaner C., Nguyen H. Q.: Testing Computer Software. 2nd Ed. New York : Int. Thomson Computer Press, 1993. 541 str.
8. Hutcheson M. L.: Software Testing Fundamentals: Methods and Metrics. Indianapolis : Wiley Publishing, 2003. 377 str.
9. Myers G. J.: The art of software testing. 2nd ed. New Jersey : John Wiley & Sons, Inc., 2004. 212 str.
10. Patton R.: Software Testing. Indianapolis : Sams Publishing, 2005. 352 str.
11. Rainsberger J. B.: JUnit Recipes Practical Methods for Programmer Testing. New York : Manning Publications, 2005. 625 str.
12. Ross K.: Practical Guide to Software System testing. West Burleigh : K. J. Ross & Associates, 1998. 196 str.
13. Solina F.: Projektno vodenje razvoja programske opreme. Ljubljana : Založba FE in FRI, 1998. 222 str.
14. Tian J.: Software Quality Engineering. New Jersey : John Wiley & Sons, Inc. 2005. 403 str.

Viri

1. Dogša T.: Verifikacija in validacija programske opreme. [URL: www.kal.si/4gb_gkr/rso/vsebina/TESTIRANJE.doc], 3.8.2006.
2. Famous computer bugs. [URL: http://en.wikipedia.org/wiki/Software_bugs], 15. 7. 2006.
3. Faught D. R.: Introduction to Software Testing & Quality Assurance. [URL: http://tejasconsulting.com/courses], 15. 5. 2006.
4. Mukesh Soni: Defect Prevention: Reducing Costs and Enhancing Quality. [URL: http://software.isixsigma.com/library/content/c060719b.asp], 22. 8. 2006.
5. Načrt testiranja programske opreme. 7 str. [URL: http://lisa.uni-mb.si/%7Ebregar/Vaje/Nacrt%20testiranja%20PO.pdf], 11. 6. 2006.

6. Parallel Life Cycle.
   [URL: http://portal.etsi.org/mbs/Testing/DevCycle/paraCycle.asp], 22. 8. 2006.
7. Software Errors Cost.
   [URL: http://www.nist.gov/public_affairs/releases/n02-10.htm], 28. 7. 2006.
8. Špica International d.o.o., Interna gradiva.

Projekt:	Time&Space 6.60	Verzija:	1.0
Dokument:	Tesni plan	Datum:	20.9.2006
Datoteka:	Testni_plan.doc		

PRILOGA

PLAN TESTIRANJA

Time & Space 6.60

Ver. 1.0

Projekt:	Time&Space 6.60	Verzija:	1.0
Dokument:	Tesni plan	Datum:	20.9.2006
Datoteka:	Testni_plan.doc		

Zgodovina revizije

Datum	Verzija	Opis	Avtor
20.9.2006	1.0	Prvotna verzija	Uroš Kočevar

Kazalo

1 .Opis produkta

T&S Manager je glavni uporabniški program, ki vsebuje posamezne aplikativne module, glede na izbrano konfiguracijo (Level Zero, Space, Time). Z njim operaterji administrirajo delovni čas in upravljajo kontrolo pristopa. Najmanj, kar lahko vsebuje, je modul Level Zero, saj ta vzdržuje osnovne podatke o sistemu: bazo uporabnikov, kartic, registrirnih točk, pristopnih pravic in dogodkov. Z modulom Space dobimo možnost administracije obiskov in alarmov. Najbolj kompleksen je modul Time, ki vsebuje tri podmodule (plačani, industrijski in drseči čas). V time modulu razporedimo zaposlene glede na naravo njihovega dela in jim določimo delavna pravila.

Zbrani podatki v TSM služijo kot osnova, na kateri se obračunajo plače in izdelajo posamezna poročila o delovni prisotnosti zaposlenih v poljubnem časovnem obdobju; receptorjem ali vratarjem, ki skrbijo za obiske in pristopne pravice obiskovalcev; vodstvenemu timu, ki želi kontrolirati in administrirati prisotnost zaposlenih in načrtovati delavne plane zaposlenih.

1.1 Sistem avtorizacije

Sistem avtorizacije v T&S omogoča različne stopnje dostopa do vsebin in orodij, ki jih ponuja posamezen modul. V aplikacijo so vgrajene naslednje stopnje pooblastil:

- administrator – skrbnik T&S sistema, ki skrbi za celotno delovanje sistema in podatkovno bazo;
- operater – administrira celotno ali omejeno skupino zaposlenih, izdeluje poročila, izvoz v plače itd.;
- receptor – dostopa do Level Zero modula in recepcije, odpira nove obiske, dodeljuje dostopne pravice gostov in ima pregled nad alarmi;
- osebni WebTS uporabnik – zaposlena oseba, ki ima pravico pregleda/administracije svojih podatkov glede registracije delovnega časa, preko spletne aplikacije;
- skupinski WebTS uporabnik – zaposlena oseba, ki ima pravico pregleda/administracije obračunskih pravil in dogodkov delovnega časa preko spletne aplikacije.

1.2 Cilji in pričakovani rezultati testiranja

Cilj testiranja je:

- preveriti pravilnost delovanja sistema glede na uporabniške zahteve;
- zagotoviti, da bo v sistemu ob predaji kar najmanj napak;
- preveriti prisotnost in pravilnost delovanja kontrol;
- preveriti delovanje sistema v povezavi s ostalimi moduli;
- preveriti delovanje sistema po nenačrtovanih prekinitvah.

Pričakovani rezultati testiranja so:

- pravilnost obračuna delovnega časa;
- izdelava in izpis različnih vrst poročil;
- načrtovanje in odpiranje obiskov;
- izvoz podatkov;
- spreminjanje obračunskih pravil.

2. Zahteve za testiranja

2.1 Funkcionalnosti sistema

2.1.1 Administracija zaposlenih

- preveriti, ali lahko vnašamo, brišemo, spreminjamo podatke o zaposlenih v T&S Managerju;
- preveriti, ali lahko zaposlenim dodamo poljubne pravice, ki so opisane v T&S priročniku, s katerimi se bodo lahko prijavili v različne module v T&S sistemu;
- preveriti, ali lahko zaposlenemu dodajamo, brišemo kartice in pristopne pravice;
- preveriti, ali se upoštevajo poljubno dodane pristopne pravice na pristopnih točkah.

2.1.2 Administracija obiskov

- preveriti, ali lahko vnašamo, brišemo in spreminjamo podatke o obiskovalcih;
- preveriti, ali lahko za poljubnega obiskovalca najavimo obisk;
- preveriti, ali lahko odpremo oz. zapremo obisk za poljubnega obiskovalca pri poljubnem zaposlenem;
- preveriti, ali je grafičen prikaz odprtih obiskov ustrezen;
- preveriti, ali se pravilno upoštevajo pristopne pravice obiskovalcev na pristopnih točkah.

2.1.3 Administracija delavnih shem

- preveri, ali lahko dodajamo, brišemo in spreminjamo delovne sheme vseh treh vrst (plačane, industrijske, drseče);
- preveri, ali lahko naredimo kopijo obstoječe delovne sheme;
- preveri, ali lahko dodajamo, brišemo in spreminjamo obračunska pravila na delovni shemi;
- preveri, ali se obračunska pravila na delovni shemi upoštevajo pri obračunavanju delovnega časa;
- preveri, ali lahko dodajamo in odstranjujemo delovne sheme pri poljubnem zaposlenem;
- preveri, ali lahko spreminjamo tip izbire urnikov v delovni shemi.

2.1.4 Administracija konfiguracije Time parametrov

- preveri, ali lahko dodajamo, brišemo in spreminjamo praznike, plačilne dneve in meje lanskega dopusta v koledarju ter ali se le-ti upoštevajo pri obračunavanju delovnega časa;
- preveri, ali lahko dodajamo, brišemo, kopiramo in spreminjamo urnike za poljubni delovni čas;
- preveri, ali je grafičen prikaz urnikov pravilen;
- preveri, ali se pravilno upošteva konfiguracija urnikov;
- preveri, ali lahko dodajamo, brišemo in spreminjamo poljubne kategorije;
- preveri, ali lahko dodajamo, brišemo in spreminjamo poljubne dogodke;
- preveri, ali lahko dodajamo, brišemo in spreminjamo globalne parametre in ali se le-ti upoštevajo pri obračunavanju delovnega časa;
- preveri, ali lahko dodajamo, brišemo in spreminjamo poljubne števce poljubnih kategorij.

2.1.5 Izdelava in izpis poljubnih poročil v T&S sistemu

- preveri, ali deluje kreiranje poljubnih poročil v T&S sistemu;
- preveri, ali lahko poročila izvozimo v poljubne formate (xls, pdf, txt, csv);
- preveri vsebinsko ustreznost izdelanih poročil;
- preveri, ali deluje tiskanje poljubnih poročil v T&S sistemu;
- preveri, ali deluje izvoz podatkov v Oracle, CSV in Paradox.

2.1.6 Time moduli plačani, industrijski, drseči čas

- preveri, ali deluje vnašanje, brisanje, spreminjanje poljubnih dogodkov v vseh treh modulih;
- preveri, ali je grafičen prikaz obstoječih dogodkov pravilen v vseh treh modulih;
- preveri delovanje filtrov, s katerimi prikazujemo zaposlene na pogledih en dan, ena oseba, po dnevih, obračun in zabeleženi dogodki, v vseh treh modulih;
- preveri pravilen obračun poljubno vnesenih dogodkov.

3. Strategija testiranja

Strategija testiranja vsebuje različne tipe testov, s katerimi se bo, kar se da podrobno, pretestiral T&S Manager. Glavni namen teh testov je odkriti omejitve in pravilnost delovanja T&S sistema.

3.1 Vrste testiranja

Pri testiranju aplikacije T&S Manager se bo izvajalo testiranje funkcionalnosti, uporabnosti, združljivosti, varnosti, zmogljivosti, okrevanja, uporabnosti in dokumentacije.

3.1.1 Testiranje funkcionalnosti

Cilj te vrste testiranja je priti do zaključka, da program v celoti ne ustreza zahtevani specifikaciji. Pozornost je treba usmeriti na protokol testiranja, ki mora zagotavljati ponovljivost testnega postopka. Takšno testiranje temelji na metodi črne skrinje.

3.1.2 Testiranje nameščanja

Pri testu nameščanja T&S sistema se skuša dokazati, da se le-ta ne namesti pravilno. Preveriti je potrebno, če se izvede nadgradnja obstoječe podatkovne baze že obstoječih modulov in instalacijo na novi delovni postaji.

3.1.3 Testiranje združljivosti

S testom združljivosti se skuša dokazati, da T&S 6.60 ni združljiv s predhodnimi verzijami sistema. Pri tem je poudarek predvsem na ohranitvi stanja salda pred in po nadgraditvi sistema in delovanju obračunskih pravil po ponovitvi obračuna.

3.1.4 Testiranje varnosti

Skuša se ugotoviti, ali T&S sistem izpolni zahteve glede varovanja podatkov. Predvsem se preverja prijavljanje v sistem z različnimi nivoji avtorizacije, prikazovanje in upravljanje s podatki glede na pravice.

3.1.5 Testiranje zmogljivosti programske opreme

S testom prekoračitve se bo T&S Manager izpostavil daljšim časovnim intervalom neobičajno velikih količin vhodnih podatkov. Skuša se ga pripraviti do odpovedi ali nepravilnega delovanja ter stresa, pri katerem T&S izpostavimo zelo velikim trenutnim prekoračitvam. Dokazati se želi, da delovanje T&S sistema v takšnih pogojih ni pravilno. Pri generiranju vhodnih podatkov si bomo pomagali z aplikacijo ECstress, ki simulira delovanje terminalov in pošilja poljubno število dogodkov v T&S sistem.

3.1.6 Testiranje okrevanja po napakah oz. ponovnega zagona

Namen je dokazati, da se ponovni zagon programa ne izvede v skladu z zahtevami oziroma da pride pri odpovedi programa do izgube podatkov. Osredotočiti se je potrebno na izgubo povezave s SQL strežnikom, izpadom električne energije in mrežne povezave.

3.1.7 Testiranje uporabnosti

Za razliko od ostalih tipov testiranja ne išče napak v izvajanju algoritmov, temveč v zasnovi uporabniškega vmesnika. Pri testu uporabnosti v T&S Managerju se pozornost namenja sledečemu:

- Ali so izhodni podatki sistema vsebinsko ustrezni in primerni za nadaljnjo uporabo?
- Ali so sporočila napak v sistemu razumljiva, tako da jih brez posebnega računalniškega znanja razumemo?
- Ali uporabniški vmesniki odražajo pomembno pojmovno celoto, složnost, enotnost sintakse, formata, sloga in kratic?
- Ali sistem vsebuje prekomerno število možnosti ali opcij, ki se bodo uporabljale zelo redko?
- Ali se sistem na vse vhodne akcije odziva na enak način?

3.1.8 Testiranje dokumentacije

S tem testom se preverja natančnost uporabniške dokumentacije. Tako se zagotovi, da so vse funkcionalnosti opisane in da je gradivo prijazno uporabniku.

3.2 Orodja

Seznam orodij, ki se bodo uporabljala pri procesu testiranja:

- ECstress ver. 1.2 – in-house aplikacija,
- CondenseCalcCompare 1.0 – in-house aplikacija,
- VMware Workstation 5.0 – VMware inc.

4. Testno okolje

V procesu testiranja se bo uporabljalo 10 osebnih računalnikov HP Pentium 4 (min 1.5 ghz, min 1 Gb RAM, 80gb trdi disk) z operacijskim sistemom Windows XP, ki so povezani v lokalno mrežo. Na vsakem računalniku bo nameščen program VMware, ki bo imel slike sistemov: Windows 2000, Windows 2003 server, Windows XP.

4.1 Testna ekipa in odgovornosti

Testno ekipo sestavljajo:

- vodja testiranja,
- tester 1 – Spica int., d. o. o.,
- tester 2 – Spica int., d. o. o.,
- tester 3 – Oddelek za podporo,
- tester 4 – Oddelek za podporo,
- tester 5 – vodja prodaje,
- tester 6 – komercialist T&S sistema,
- tester 7 – komercialist T&S sistema,
- tester 8 – serviser T&S,

- tester 9 – serviser T&S,
- SQL server administrator,
- manager T&S projekta.

Pri formuliranju testne skupine se vodja testiranja seznani z njihovim znanjem na področju T&S sistema in metodami testiranja ter poskrbi za njihovo izobraževanje. Vodja testiranja in projektni manager T&S odločata o začetku in koncu testiranja sistema, v okviru terminskega plana. Vodja testiranja je odgovoren za koordinacijo terminov, opremo in orodja testerjev, kot tudi za posodabljanje testnega plana, pisanje tedenskih testnih poročil in končnega testnega poročila. Testerji so odgovorni za pisanje testnih scenarijev in izvršitev le-teh. SQL server administrator skrbi za administracijo testnih baz.

4.2 Pravila, postopki in dogovori

Plan testiranja se nanaša na aplikacijo T&S Manager. Ostali moduli iz T&S sistema niso predmet tega plana. Uporablja se jih samo kot podporo pri testiranju T&S Manager aplikacije. Demo bazo za testiranje pripravi SQL server administrator. Testni scenariji morajo biti pripravljeni v skladu s priporočili za pripravo testnih podatkov. Testirajo se vse mejne vrednosti: najnižja in najvišja vrednost, vrednosti zunaj določenega intervala in vrednosti znotraj intervala. Pri kreiranju testnih primerov ne smemo pozabiti na napačne vnose v okviru ekvivalentnega razreda.

Testni primer vsebuje naslednje podatke:

- datum testiranja;
- ime in priimek izvajalca testiranja;
- ime in verzijo programskega proizvoda, ki se testira;
- opis testnega primera;
- vhodne podatke;
- pričakovane rezultate;
- dejanske rezultate.

Vsak tester ima na testni platformi v Outlooku odprt svoj test, za katerega poskrbi vodja testiranja. Izvajalci testiranja objavijo ugotovljene napake na testni platformi. Pri opisu napake morajo biti navedeni naslednji podatki:

- pri katerem postopku je prišlo do napake;
- vhodni podatki;
- dobljeni rezultati;
- pričakovani rezultati.

4.3 Prekinitev in nadaljevanje testiranja

V primeru, da ena izmed odkritih napak resno vpliva na proces testa, se lahko vodja testerjev odloči, da izvajanje procesa začasno odloži. Kriteriji, ki opravičujejo začasno prekinitev testiranja, so naslednji:

- strojna ali programska oprema ni razpoložljiva glede na urnik testiranja;
- testna aplikacija vsebuje eno ali več kritičnih napak, ki resno ogrožajo ali omejujejo izvedbo testa;
- več kot 50-odstotna nepopolnost testnega tima.

Če je bilo testiranje začasno ustavljeno, nadaljujemo s testom, ko je odpravljen problem, ki je povzročil prekinitev. Če je bil vzrok za prekinitev kritična napaka, mora biti popravek, preden se nadaljuje s testom, preverjen s strani oddelka za testiranje.

5. Tveganja in posledice

5.1 Urnik testiranja

Urnik je zelo zaseden, zato lahko vpliva na proces testiranja. Spodrsljaj pri kakšnem urniku na določeni fazi se lahko odrazi v zamiku naslednjih faz testiranja.

5.2 Osebje

Zaradi agresivnega urnika je pomembno, da imamo izkušene testerje na tem projektu. Lahko pride do nenačrtovanih sprememb, ki vplivajo na proces testiranja. Posledica tega bo večji pritisk na testerje in izčrpanost.

5.3 Tehnična

Ker bo za izvedbo testa izposojenih 5 zmogljivih računalnikov z drugih oddelkov, se lahko zgodi, da bo potrebno katerega tudi vrniti. Zato sta v rezervi še 2 povprečno zmogljiva računalnika, s katerima se bo nadomestila izguba.

6. Mejniki projekta in urniki testiranja

Naloge mejnikov	Začetek	Konec
Build T&S 6.60.A Planiranje testa Oblikovanje testov Izvedba testa Ovrednotenje testa	**2.10.06**	**13.10.06**
Build T&S 6.60.B Planiranje testa Oblikovanje testov Izvedba testa Ovrednotenje testa	**16.10.06**	**27.10.06**
Build T&S 6.60.C Planiranje testa Oblikovanje testov Izvedba testa Ovrednotenje testa	**30.10.06**	**10.11.06**
Build T&S 6.60.D – preizkusna verzija Oblikovanje testov Izvedba testa Ovrednotenje testa	**13.11.06**	**23.11.06**
Build T&S 6.60.E – preizkusna verzija Oblikovanje testov Izvedba testa Ovrednotenje testa	**27.11.06**	**6.12.06**

7. Referenčni dokumenti

Izvajalcem testiranja so na voljo naslednji dokumenti:

- Whats new,
- Release notes,
- Priročnik namestitve in delovanja T&S sistema,
- Uporabniški priročnik T&S Manager,
- Navodila za nadgradnjo,
- Nezdružljivost T&S,
- Vnos kartic.

8. Potrditev testirnega načrta

Ime in priimek	Podpis	Datum
1.		
2.		
3.		
4.		
5.		