UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Mateja Baša

INTEGRACIJSKO TESTIRANJE V SISTEMIH SOA

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01591/2009
Datum: 15.10.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJA BAŠA**

Naslov: **INTEGRACIJSKO TESTIRANJE V SISTEMIH SOA**
SOA INTEGRATION TESTING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Opišite, kaj je testiranje in kakšne vrste testiranja programske opreme poznamo, kakšne so napake in vzroki za njihov nastanek. Nato analizirajte metode testiranja, ki so danes uveljavljene. Osredotočite se na sisteme s storitveno usmerjeno arhitekturo (SOA) in navedite posebnosti, ki jih je potrebno upoštevati pri integracijskem testiranju v sistemih SOA. Podajte metodologijo integracijskega testiranja in ga primerno uvrstite v celoten postopek testiranja. Na izbranem praktičnem primeru implementirajte integracijsko testiranje, opišite najdene napake in kako ste prišli do njih ter podajte prednosti in slabosti predlaganega postopka.

Mentor:
M. Ciglarič
doc. dr. Mojca Ciglarič

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Franc Solina

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Mateja Baša

INTEGRACIJSKO TESTIRANJE V SISTEMIH SOA

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Mojca Ciglarič

Ljubljana, 2010

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisana Mateja Baša, z vpisno številko 63030133, izjavljam, da sem avtorica diplomskega dela z naslovom **Integracijsko testiranje v sistemih SOA**. S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod vodstvom mentorice doc. dr. Mojce Ciglarič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

Mateja Baša

Ljubljana, april 2010

Zahvala

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za vodenje, pomoč in nasvete pri izdelavi diplomskega dela.

Zahvaljujem se tudi podjetju Telekom Slovenije, ki mi je omogočilo delo ob študiju ter sodelavcem v podjetju, od katerih sem pridobila obširna znanja, ki so mi pomagala pri izdelavi diplomskega dela.

KAZALO VSEBINE

SEZNAM UPORABLJENIH KRATIC

APEK - Agencija za pošto in elektronske komunikacije Republike Slovenije

BPEL (angl. Business Process Execution Language) - programski jezik za izvajanje poslovnih procesov

CBP - centralna baza prenesenih številk

HTTP (angl. HyperText Transport Protocol) - protokol za prenos hiperteksta

LBPŠ - lokalna baza prenesenih številk

QoS (angl. Ouality of service) – kakovost storitve

SAAS (angl. Software As A Service) - aplikacija kot storitev

SOA (angl. Service Oriented Architectur) - storitveno usmerjena arhitektura

SOAP (angl. Simple Object Access Protocol) - standard za spletne storitve, ki temelji na XML

STQE (angl. Software Testing and Quality Engineering) - testiranje programske opreme in zagotavljanje kvalitete

WS (angl. Web Service) - spletna storitev

WS-I (angl. Web Service Interoperability Organization) - organizacija za interoperabilnost spletnih storitev

WSDL (angl. Web Services Description Language) - opisni jezik spletnih storitev

XML (angl. eXtensible Markup Language) - razširljiv označevalni jezik

ZEKom - Zakon o elektronskih komunikacijah

POVZETEK

V diplomskem delu sem opisala kaj je testiranje in kakšne vrste testiranja programske opreme poznamo. Osredotočila sem se na pristope k testiranju sistemov s storitveno usmerjeno arhitekturo (SOA), s poudarkom na integracijskem testiranju. Preučila sem različne metode testiranje in podrobno opisala eno izmed priporočenih metodologij testiranja SOA.

V uvodnem delu sem predstavila celoten postopek testiranja skozi razvoj, najpogostejše pojme, s katerimi se srečujemo v fazi testiranja ter vrste napak in vzroki za njihov nastanek. Ker je uspešnost testiranja težko izmeriti, sem podala še nekaj preverjenih formul, s pomočjo katerih si lahko pomagamo pri izračunu uspešnosti testov.

V nadaljevanju sem predstavila storitveno usmerjeno arhitekturo, ki se v današnjem svetu pogosto pojavlja. Testiranje takšne arhitekture je drugačno od tradicionalnih postopkov testiranja, zato sem na tem mestu opozorila na prednosti in slabosti.

Jedrni del sem posvetila integracijskemu testiranju storitveno usmerjenih sistemov ter predstavitvi ene izmed priporočenih metodologij. Metodologijo sem predstavila v več fazah. Vsako fazo posebej sem opisala, predstavila vrste napak, ki jih odkrivamo tekom posamezne faze, ter opisala predlagan postopek izvedbe. Predlagala sem tudi ogrodje za zapis testne specifikacije.

Sledi praktičen del. Opisala sem postopek prenosa telefonskih številk, na katerem sem predstavila uporabo prej opisane metodologije. Tekom testiranj sem beležila napake in sproti ugotavljala pomanjkljivosti in prednosti te metodologije.

V zaključnem delu sem predstavila sklepne ugotovitve in podala lastno kritično mnenje k takšnemu pristopu testiranja.

Ključne besede:

integracijsko testiranje, storitveno usmerjena arhitektura (SOA), spletne storitve, …

ABSTRACT

In my bachelor's thesis I described what testing is and what kind of software testing is known. I focused on approaches to testing systems with service oriented architecture (SOA) with an emphasis on integration testing. I explored different testing methods and detailed one of the recommended SOA testing methodologies.

In the introduction I presented the entire testing process through development, the most common terms encountered in the testing phase and the types of errors and the reasons for their occurrence. Because it is difficult to measure the performance of testing, I included some proven formulas which can help in calculating the performance of tests.

I also presented a service-oriented architecture which often occurs in today's world. Testing such an architecture is different from traditional testing approaches. I pointed out the advantages and the disadvantages.

The core part is focused on the integration testing of service-oriented systems. I described one of the recommended methodologies. I presented the chosen methodology in several stages. For each phase, I have described the types of errors and the recommended implementation process. I also recommended a framework for recording the test's specification.

In practical work of my thesis I described the process of transferring phone numbers. On a concrete example I presented the use of methodology that I had described before. During the test I recorded the errors and promptly identified the weaknesses and the strengths of this methodology.

In the final part I presented the conclusions and made my own critical opinion of such an approach to testing.

Keywords:

integration testing, service-oriented architecture (SOA), Web services, ...

1 UVOD

Razvoj informacijskega sistema, ki bi nudil celovito podporo vsem poslovnim procesom organizacije, je že sam po sebi zapleten in dolgotrajen postopek. Če pa v organizaciji želimo, da so poslovni procesi agilni, da se hitro spreminjajo in prilagajajo vedno novim potrebam strank oziroma naročnikov, potem postane težavnost podpore toliko večja.

Storitveno usmerjena arhitektura (v nadaljevanju SOA) nudi dokaj nov arhitekturni pristop, s katerim lahko spremembe ali razširitve poslovnih procesov hitreje realiziramo. Pri realizaciji pa moramo paziti, da ohranimo agilnost poslovnih procesov.

Ker razvoj sistema po načelih SOA poteka drugače od prej poznanih gradenj in ker imajo storitve posebne karakteristike, katere moramo upoštevati, se postopki testiranja takšnih sistemov razlikujejo od tradicionalnih postopkov testiranja. Čeprav se SOA zadnje čase pogosto uporablja, opažamo očitno pomanjkanje testnih metodologij, ki bi bile specifične le za SOA sisteme. Novi pristopi in nove metodologije pa so nujne za verifikacijo in validacijo sistemov. Veliko organizacij razvije svojo metodologijo testiranja, ki je prilagojena lastnim poslovnim procesom.

V realnosti pa se testiranje programske opreme ali sistema žal prepogosto izvaja povsem drugače kot je prvotno načrtovano. Vzrok za to so najpogosteje kratki roki in presežene finančne zmožnosti. Posledica tega so zmanjšana sredstva za testiranje ter skrajšan čas testiranja, kar pa močno vpliva na kakovost produkta ter posledično tudi na višje stroške vzdrževanja. Aktivnosti testiranja torej nikoli ne bi smeli skrajšati, ali hujše, kar preskočiti, saj je v procesu razvoja zelo pomembna. Da je testiranje učinkovito, se ga je potrebno lotiti sistematično, z dobro definirano testno specifikacijo in zadostnimi resursi.

Cilji diplomskega dela:

- predstaviti prednosti in slabosti storitveno usmerjenih sistemov,
- predstaviti integracijsko testiranje storitveno usmerjenih sistemov,
- prikazati uporabo ene izmed metodologij na konkretnem primeru in
- preučiti možnosti izboljšave uporabljene metodologije.

2 TESTIRANJE

V uvodu sem opisala problematiko, kateri bo posvečen velik del diplomskega dela. Še prej pa bi rada predstavila postopek testiranja, razložila kaj je integracijsko testiranje in le-tega uvrstila v celoten proces testiranja.

Prav je, da na kratko razložim tudi kaj je SOA, zakaj je testiranje SOA sistemov drugačno in kateri so ključni dejavniki, ki omejujejo testiranje takšnih sistemov.

2.1 Osnovni pojmi testiranja

Navajam nekatere izmed najpogostejših terminov, s katerimi se srečujemo pri testiranju programske opreme [1].

Napaka (angl. error) je pomota, nepravilnost ali nerazumevanje s strani izvajalca.

Okvara (angl. fault) je posledica napake. To je anomalija v programski opremi, ki lahko povzroči, da se programska oprema obnaša nepravilno, oziroma ne tako kot je določeno v specifikacijah.

Odpoved (angl. failure) je nezmožnost programskega sistema ali njegove komponente, da opravi svojo funkcijo, ali doseže preformance, kot je bilo zahtevano v zahtevah.

Lahko se tudi zgodi, da okvara v programski kodi ne povzroči odpovedi. Okvarjena programska oprema lahko obratuje dolgo časa brez očitnih nepravilnosti, ko pa so izpolnjeni določeni pogoji, se napaka nepričakovano pojavi kot odpoved.

2.2 Definicija testiranja

Pogoste definicije testiranja so [2]:

- testiranje je proces, s katerim skušamo prikazati da v sistemu, ki ga gradimo, ni napak,
- testiranje je aktivnost ali proces, s katerim lahko prikažemo, da program ali sistem deluje v pričakovanih okvirih,
- testiranje je aktivnost, s katero dokažemo in dosežemo zaupanje, da sistem dela, tako kot bi moral delati – v skladu z zahtevami, ki so jih specificirali uporabniki.

Zgoraj navedene definicije so sicer vse pravilne, vendar zavzemajo preveč »pozitivno« stališče. V vseh hočemo dokazati, da sistem deluje. Testiranje lahko pokaže prisotnost napak, vendar ne more pa pokazati, da napak v sistemu ni [3].

Osnovni namen testiranja je torej dokazati prisotnost ene ali več napak. Zaradi tega bi se najbolj strinjala z definicijo [4]: »Testiranje je proces izvajanja programa z namenom odkrivanja napak«.

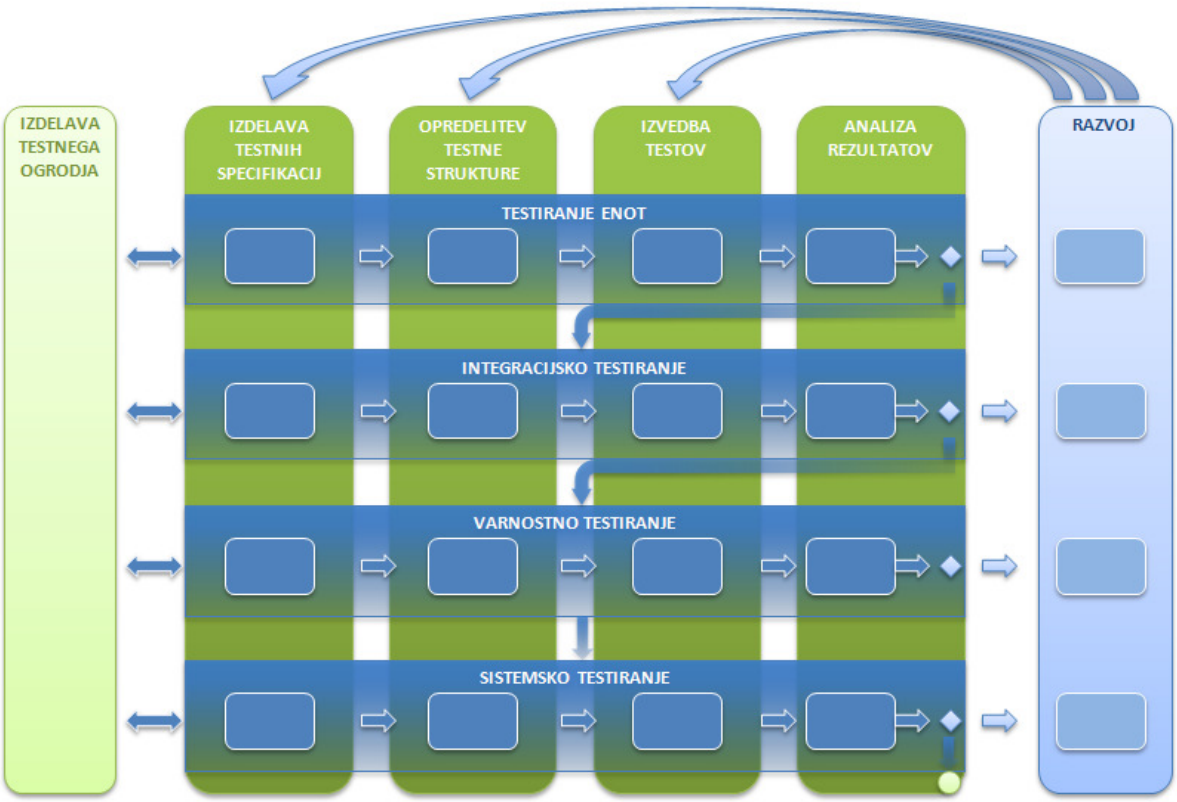
Bistvo testiranja je neskončno veliko množico kombinacij prevesti v končno minimalno število najverjetnejših vzorcev, ki bodo nastopili pri uporabniku, in le-to temeljito preveriti [5].

Testiranje je postopek, s katerim ugotavljamo pravilnost delovanja, popolnost, varnost in kakovost programske opreme. Testiranje pogosto enačijo s procesom zagotavljanja kakovosti programske opreme, kar je napačno, saj je proces zagotavljanja kakovosti širši pojem, ki poleg testiranja vključuje še druge organizacijske procese. Testiranje je kljub vsemu eden najpomembnejših postopkov znotraj razvoja ali nakupa programske opreme, saj uspešnost izvedenih testov zagotavlja, da programska oprema ustreza glede na zahteve naročnika in da deluje pravilno in v skladu s specifikacijo.

Testiranje programske opreme se izvaja v različnih fazah, ki so namenjeni odkrivanju različnih vrst napak in pomanjkljivosti. Te faze so [6]:

- testiranje enot,
- integracijsko testiranje,
- varnostno testiranje,
- sistemsko testiranje,
- regresijsko testiranje.

Slika 1 prikazuje postopek testiranja [6]. V posameznih fazah testiranja se aktivnosti izvajajo v enakih sklopih, in sicer: izdelava testnega ogrodja, izdelava testnih specifikacij, opredelitev testne strukture, izvedba testov in analiza rezultatov. Vidimo lahko, da je postopek iterativen in da najprej izvajamo fazo testiranja enot, nato fazo integracijskega testiranja, za tem pa fazi varnostnega in sistemskega testiranja. Varnostno in sistemsko testiranje lahko za velik del aktivnosti izvajamo sočasno. Na sliki lahko vidimo tudi, da je za izvedbo testiranja potrebno izdelati tudi testno ogrodje, ki omogoča izvajanje testiranja na različnih ravneh. Pri tem je pomembno poudariti, da naj bi bila izdelava testnega ogrodja v tesni povezavi z izdelavo testnih specifikacij, saj naj bi testno ogrodje omogočalo čim višjo raven avtomatizacije testiranja. Testno ogrodje je tako v vseh fazah izdelano tako, da v čim večji meri omogoča avtomatizacijo izdelave testnih specifikacij in avtomatizacijo izvajanja testov na podlagi testnih specifikacij.



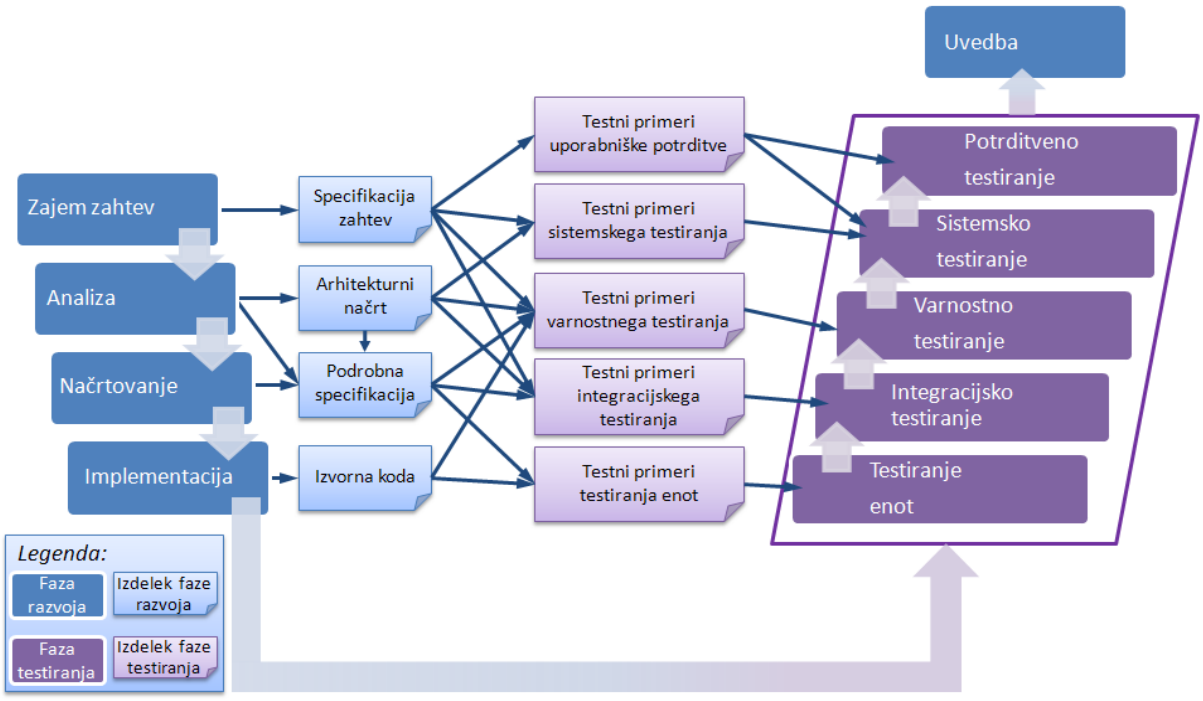
Slika 1: Prikaz postopka testiranja

V diplomskem delu se bom posvetila integracijskemu testiranju (ki je prisotno v največjem delu razvoja).

2.3 Pregled postopka testiranja

Slika 2 prikazuje postopek testiranja in njegovo umestitev v razvojni cikel programske opreme [6]. Slika ponazarja le sosledje posameznih faz in povezanost njihovih izdelkov, pri čemer gre za iterativen postopek, kot prikazuje Slika 1. Na primer, če testi pokažejo napačno delovanje, to lahko vodi nazaj v fazo implementacije, v kateri se izvedejo ustrezni popravki, na nato spet nastopi faza testiranja.

Pri testiranju enot in njegovi umestitvi v razvojni cikel programske opreme obstaja več različic in Slika 2 prikazuje zgolj eno izmed njih; pri njej se testiranje enot izvaja za fazo implementacije. Obstajajo tudi drugi zelo uveljavljeni pristopi, pri katerih se testiranje enot izvaja kot del faze implementacije, in sicer v okviru ekstremnega programiranja (Extreme Programming) in testno-usmerjenega razvoja programske opreme (Test-Driven Development). Pri tem ločimo dve različici, in sicer pristop, pri katerem se za vsakim delom (chunk) napisane kode izvede tudi test, ki preverja njeno pravilnost, in pristop, pri katerem se najprej določi in razvije test za posamezen del programske kode, nato pa šele dejanska programska koda, ki jo razvijalec izdeluje, dokler se test ne izvede uspešno (Test-First).



Slika 2: Prikaz umestitve testiranja v razvojni cikel

2.4 Izdelki

Poleg umestitve postopka testiranja v razvojni cikel programske opreme, Slika 2 ponazarja tudi povezanost izdelkov posameznih faz razvojnega cikla z vhodnimi izdelki faz postopka testiranja. Ti opredeljujejo testne primere, ki jih določimo na podlagi izdelkov posameznih razvojnih faz, in sicer:

- na podlagi specifikacije zahtev, ki so izdelane v fazi zajema zahtev, določimo testne primere za uporabniško potrditev,
- testne primere sistemskega testiranja določimo iz specifikacije zahtev, na kar jih dopolnimo glede na specifikacije arhitekturnega načrta, opredeljene v fazi analize;
- varnostno testiranje je potrebno izvajati glede na vse ravni programske opreme, zato osnovo za testne primere varnostnega testiranja predstavljajo izdelki vseh predhodnih razvojnih faz, se pravi specifikacije zahtev, arhitekturni načrt, podrobni načrt in izvorna koda;
- testne primere integracijskega testiranja določimo na podlagi specifikacij zahtev, arhitekturnega načrta in s podrobneje določenimi funkcionalnimi zahtevami podrobnega načrta, ki je izdelan v fazi načrtovanja;
- testne primere testiranja enot določimo iz podrobnega načrta ter jih nato v fazi načrtovanja in implementacije dopolnimo glede na izvorno kodo.

Aktivnost opredelitve testnih primerov se lahko izvaja v sami fazi testiranja ali že pred njo, v posameznih razvojnih fazah pred fazo testiranja.

Izhodni izdelki, ki nastajajo v fazah testiranja, so izpolnjene testne specifikacije z rezultati testa ter morebitne analize in statistike, ki jih testno orodje izdela na podlagi izvedenih testov.

2.5 Okolja

Ločimo tri različna okolja, v katerih izvajamo različne faze testiranja, in sicer:

- produkcijsko okolje: okolje, v katerem bo sistem deloval, ko bo v uporabi;
- testno okolje: okolje, ki čim bolj ponazarja okolje, v katerem se bo sistem dejansko uporabljal, se pravi produkcijsko okolje in tako vključuje enako strojno opremo, operacijski sistem, aplikacijski strežnik itd.; v testnem okolju spletne storitve, podatkovne baze itd. repliciramo, kar dopušča njihovo uporabo in spreminjanje vsebin;
- razvojno okolje: okolje, v katerem sistem razvijamo, in se lahko razlikuje od produkcijskega okolja.

Tabela 1 prikazuje, v katerih okoljih se izvajajo posamezne faze testiranja.

	Razvojno okolje	Testno okolje	Produkcijsko okolje
Testiranje enot	✓	✗	✗
Integracijsko testiranje	✓	✓	✗
Varnostno testiranje	✗	✓	✓
Sistemsko testiranje	✗	✓	✗

Tabela 1: Faze testiranja in okolja, v katerih se izvajajo

2.6 Napake in vzroki za njihov nastanek

Napaka je prisotna v vsakem primeru, ko sistem oziroma program ne deluje tako, kot pričakujejo končni uporabniki [4]. Ta definicija je sicer resnična, vendar pomanjkljiva, saj je v njej izključen človeški faktor. Le-ta pa je eden najpogostejših faktorjev, ki vpliva na napake.

O napakah govorimo, kadar [5]:

- program ne opravlja nečesa, kar navaja specifikacija,
- program izvaja nekaj, kar specifikacija pravi, da ne bi smel,
- program počne nekaj, česar specifikacija ne omenja,
- program je težko razumeti, težko ga je uporabljati, je počasen, v očeh uporabnika pa ne bo viden kot ustrezen.

Pri definiciji napake moramo vedno upoštevati specifikacijo izdelka, saj le-ta podrobno opredeli izdelek. Napaka je vsakršno odstopanje od zahtev navedenih v specifikaciji, vendar samo v primeru, če te specifikacije dejansko obstajajo in so pravilno napisane. Če specifikacija ni pravilno zapisana in ji pri razvoju dosledno sledimo, ne moremo pričakovati da bo program deloval po pričakovanjih.

Poglaviten razlog za napake naj bi bila specifikacija. Naj podkrepim to izjavo [5]:

- za veliko programsko opreme specifikacija enostavno ne obstaja,
- specifikacija ni dovolj konkretna in podrobna,
- specifikacija se neprestano spreminja.

Pogosto se zgodi, da zahteva ni zapisana v specifikaciji in je kljub temu vgrajena v sistem. Lahko predpostavim, da gre tudi tu za napako, saj je glede na zgornjo definicijo napaka vse, kar se razlikuje od zahtev, podanih v specifikaciji.

Ločimo dve vrsti odstopanj delovanja sistema [7]:

- od zahtev, ki so bile podane v specifikaciji (lahko pride tudi do napačnega razumevanja specifikacije),
- od pričakovanj ključnih uporabnikov (to lahko pomeni le to, da so bile že specifikacije nepravilno oz. pomanjkljivo napisane).

Vsaka napaka v sistemu lahko povzroči veliko škodo podjetju. Nekatere napake so sicer manjše in njihove posledice ne povzročijo veliko težav, druge pa so večje, ki lahko povzročijo ogromno škodo.

Da do te škode ne pride, je potrebno posvetiti čas tudi testiranju sistema, ne le ob koncu razvoja, temveč skozi celoten čas razvoja. Večji in kompleksnejši kot je sistem, toliko več časa bomo porabili za njegovo testiranje. Le tako bomo preprečili težave pri delovanju sistema.

2.7 Sledenje napak

Nad celotnim postopkom testiranja želimo imeti celovit pregled. Da lahko le tega omogočimo, je potrebno napake natančno evidentirati. Večina sistemov za spremljanje napak omogoča določitev:

- prioritete,
- statusa napake,
- in komponente oz. enote v sistemu, na katero se nanaša.

Prioriteta

Zavedeni napaki je potrebno določiti prioriteto na podlagi tega, kako kritična je za proces. Napaka je lahko samo lepotne narave, lahko pa onemogoča celotno nadaljnje testiranje. Pri reševanju napak je torej potrebno upoštevati prioriteto napake.

Status napake

Ko napako prvič zavedemo v sistem za spremljanje napak, dobi status *»odprta«*. Ko razvijalec odpravi napako ali njen vzrok, ter objavi novo verzijo procesov, mora napaki spremeniti tudi status na *»odpravljena«*. Popravek napake pa je vedno dobro ponovno preveriti na testu. Če je popravek ustrezen in se napaka ne pojavi več, potem ji lahko spremenimo status na *»zaprta«,* sicer moramo napako ponovno »odpreti«.

Komponente

Ob vsaki napaki je dobro označiti, pri katerem testnem scenariju smo naleteli nanjo, ter na katero komponento se je test nanašal. Po številu prijavljenih napak v zvezi s posamezno komponento lahko vidimo, katera komponenta je tista, ki nam povzroča največ preglavic in jo je oziroma jo bo potrebno še posebej dobro prestestirati.

2.8 Merjenje uspešnosti testiranja

V praksi najpogosteje domnevamo, da je testiranje uspešno končano, če pri izvedbi vseh testnih scenarijev ne odkrijemo nobenih napak. To domnevanje pa ni dobro, saj je neodvisno od kvalitete in kvantitete testnih scenarijev. Število odkritih napak ne more biti edini kriterij, na podlagi katerega bi lahko govorili o uspešnosti testiranja.

Ker je pojem uspešnosti testiranj težko definirati, si pogosto pomagamo z izračuni nekaterih karakteristik testiranja [8]:

a) **testna pokritost sistema**

Če je sistem večji, v vsakem testnem ciklu ne moremo izvesti vseh definiranih testov, zato je teste potrebno izbrati sistematsko. Testna pokritost sistema pove, kakšen odstotek testov je bil izveden v posameznem ciklu testiranja.

$$\text{testna pokritost sistema} = \frac{\text{število izvedenih testov}}{\text{število vseh testov}} \text{ x } 100$$

Enačba 1: Testna pokritost sitema

b) **delež odpravljenih napak**

Delež odpravljenih napak nam pove, koliko odkritih napak je bilo dejansko odpravljenih.

$$\text{delež odpravljenih napak} = \frac{\text{število odpravljenih napak}}{\text{število odkritih napak}} \text{ x } 100$$

Enačba 2: Delež odpravljenih napak

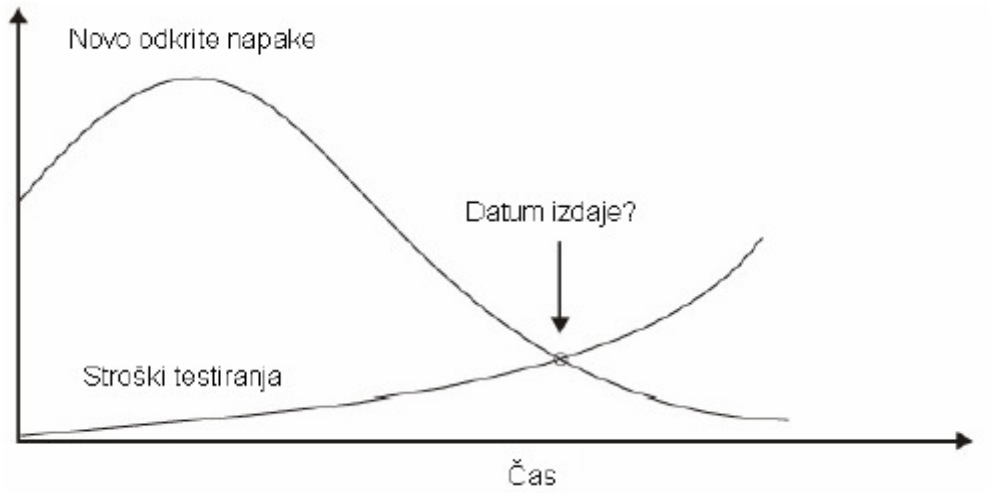
c) **učinkovitost testiranja**

Učinkovitost testiranja pa nam pove, koliko napak je bilo odkritih v enem ciklu testiranja, glede na celotno število odkritih napak tekom testiranja.

$$\text{učinkovitost testiranja} = \frac{\text{število odkritih napak v enem ciklu testiranja}}{\text{celotno število odkritih napak}} \text{ x } 100$$

Enačba 3: Učinkovitost testiranja

Bolj kot se omenjene karakteristike približujejo 100%, bolj uspešni smo. 100% uspešnost pa je v realnosti (žal) težko doseči, zato se pri odločanju, kdaj s testiranjem zaključiti, pomagamo še s stopnjo odkritih napak [9].



Slika 3: Stopnja odkritih napak

Graf prikazuje število novo odkritih napak v odvisnosti od časa in stroškov testiranja. Stroški testiranja s časom naraščajo, število novo odkritih napak pa se s časom zmanjšuje. Ko je stopnja odkritih napak pod določeno mejo, lahko predvidevamo, da je produkt pripravljen na izdajo.

Priprava testne dokumentacije mora biti končana hkrati z dokončanjem aplikacije. Takrat se začne testiranje in popravljanje napak v lastnem okolju, to je tako imenovano alfa testiranje (angl. alfa testing). Ko aplikacija zadošča zahtevam (kritičnih napak ni več), jo lahko inštaliramo v produkcijsko okolje (okolje stranke), kjer se aplikacija ponovno testira, začne se tako imenovano beta testiranje (angl. beta testing).

Takemu testiranju pravimo V-testiranje (angl.V-testing). Skupina testnih inženirjev in razvijalcev že od samega začetka tesno sodelujeta. To zmanjša tveganje projekta in poveča verjetnost pravočasnega končanja aplikacije.

Mogoče ni odveč omeniti, da morajo biti tako komponente projekta kot vsi predmeti testiranja (testni podatki, rezultati testov enot (angl. unit test), testni plani, testne procedure in testni podatki) pod kontrolo verzij. Shranjene morajo biti vse prejšnje verzije, da se lahko ob odkritju napak vzpostavi stara verzija.

3 METODE TESTIRANJA PROGRAMSKE OPREME

Testiranje, ki so ga uporabljali nekoč se pravzaprav uporabljajo še danes. Testiranje je drago, vendar je lahko »ne-testiranje« še dražje. Samo po sebi je destruktivno, vendar se moramo zavedati, da so rezultati testiranje zelo konstruktivni.

3.1 Klasifikacija metod testiranja, osnovana na načinu generiranja testov

Na način generiranja testov vplivajo različni faktorji: specifikacija, predhodno znanje, intuicija, struktura kode, predhodno odkriti problemi, … Metode glede na omenjene faktorje so [7]:

- **metoda »ad hoc«**

  Metoda se nanaša na intuicijo in sposobnost izvajalca testov ter na njegove izkušnje s podobnimi programi.

  Nekateri trdijo, da postopek testiranja preide v rutino, ki ne zahteva kreativnosti. Takšno mišljenje pa je napačno. V resnici testiranje ni vedno rutinska naloga, še posebej ne, če ga želimo opraviti kvalitetno. Zahteva znanje, izkušnje in tudi iznajdljivost. Tester z bogatimi izkušnjami in poznavanjem področja, ki ga podpira testna aplikacija, bo lahko odkril veliko ključnih pomanjkljivosti in pripravil boljše testne primere kot tester brez izkušenj, ki se opira le na teoretično znanje.

- **metoda pretehtavanja vseh možnosti**

  Testne primere generiramo na osnovi vseh možnih kombinacij logičnih relacij med pogoji in funkcijami.

- **metoda mejnih vrednosti**

  Testni primere generiramo na osnovi mejnih vrednosti podatkov.

- **naključno testiranje**

  Preverimo delovanje funkcij sistema z naključnimi vrednostmi. Tester za takšno testiranje ne potrebuje izkušenj in ni nujno, da pozna področje, ki ga podpira testna aplikacija.

Vse odkrite napake je potrebno ustrezno dokumentirati in jih v prihodnje vključiti v standardno testno množico.

3.2 Klasifikacija metod testiranja, osnovana na razumevanju implementacije projekta

Metode osnovane na razumevanju implementacije projekta se delijo na [10]:

- **testiranje po načelu črne skrinjice ali funkcionalno testiranje (black-box testing)**

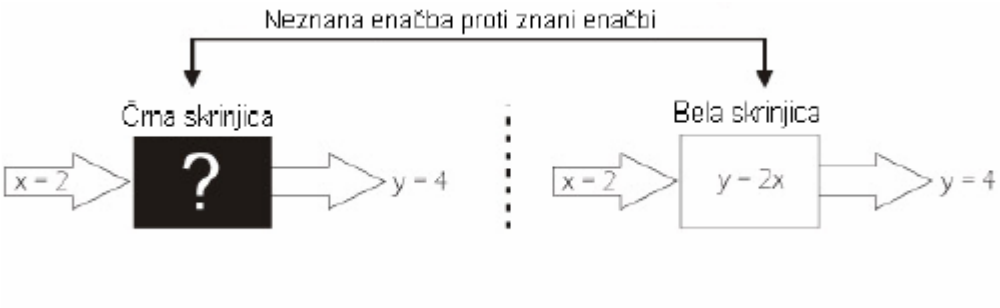
Pri testiranju po načelu črne skrinjice gledamo na program, ki ga testiramo, kot na črno škatlo. To pomeni, da izvajalec testa pri tem ne ve, kakšna je notranja struktura in kako program deluje.

- **testiranje po načelu bele skrinjice ali strukturno testiranje (white-box testing):**

V nasprotju s testiranjem po načelu črne skrinjice, je testiranje po načelu bele skrinjice. Preverja se pravilnost delovanja programa glede na njegovo notranjo strukturo, ki je v tem primeru izvajalcu testa znana.

- **kombinacija zgornjih dveh, torej načela bele in črne skrinjice**

Oba zgoraj omenjena procesa sta enakovredna in se dopolnjujeta. V praksi ju najpogosteje uporabljamo hkrati. Za kombinacijo obeh pristopov nekateri predlagajo nov termin »siva skrinjica« [11], ki naj bi poudaril nujnost uporabe obeh pristopov za uspešno testiranje.



Slika 4: Primerjava metod testiranja črne in bele skrinjice

4 STORITVENO USMERJENA ARHITEKTURA - SOA

Storitveno usmerjena arhitektura (SOA) omogoča informacijskim oddelkom podjetij, da spremenijo pogled na svoj sistem. Sistemi so bili predhodno zastavljeni in osredotočeni le na aplikacije znotraj sistema (»application-centric view«) sedaj pa se vse več sistemov gradi po načelih SOA, ki omogoča gradnjo sistema, ki je osredotočen predvsem na procese. Torej vse več je procesno usmerjenih sistemov in vse manj funkcijsko usmerjenih [12].

Ravno zaradi takšne gradnje sistemov ni več težav pri povezovanju poslovnih procesov iz različnih aplikacij in pri zagotavljanju kakovostnega end-to-end poslovnega procesa. SOA uporablja mehanizme, ki ohranjajo rahlo povezanost integracije sistema (največkrat so to spletne storitve (WS - Web Service)). In ta rahla povezanost razvijalcem omogoča spremembe in nadgradnje aplikacije, ne da bi pri tem povzročali škodo na povezanih aplikacijah.

Čeprav se SOA zadnje čase pogosto uporablja, opažamo očitno pomanjkanje testnih metodologij, ki bi bile specifične le za SOA sisteme. Novi pristopi in nove metodologije pa so nujne za verifikacijo in validacijo sistemov.

Testiranje SOA je drugačno zaradi agilnosti in prilagodljivosti sistema, zgrajenega po SOA. Zahteva testiranje vmesnikov in storitev, ki združujejo (ali pa bi lahko združevali) sisteme in platforme. Testna ekipa mora poleg obsežnih znanj o testiranem sistemu, razumeti še SOA pristop.

Implementacija SOA lahko združi dve ali več avtonomnih notranjih storitev. Razpoložljivost teh notranjih aplikacij pa je ključnega pomena, saj morajo biti dosegljive ves čas testiranja.

Na kratko bi rada opisala kaj je SOA, osnovne komponente, kdaj je primerna in kdaj ne, ter na kaj je potrebno paziti pri testiranju SOA sistemov.

4.1 Opis SOA

Poslovni informacijski sistemi so bili v preteklosti največkrat načrtovani funkcionalno. Takšni sistemi, včasih imenovani tudi silosi, vsebujejo velike količine podatkov in množico funkcionalnosti. Njihov osnovni problem je v tem, da zagotavljajo podporo določeni funkciji oz. aktivnosti, ne pa celotnemu poslovnemu procesu.

Celoviti poslovni procesi morajo torej delovati nad več takšnimi aplikacijami – silosi. S tehničnega vidika to pomeni integracijski problem, kjer je potrebno ustrezno povezati različne aplikacije, izdelane v različnih tehnologijah in poskrbeti za ustrezno zaporedje izvajanja. Pri tem ne gre pozabiti, da je tudi funkcionalnost takega poslovnega procesa razpršena med več obstoječih sistemov – z drugimi besedami, funkcionalnost je fragmentirana in tesno sklopljena.

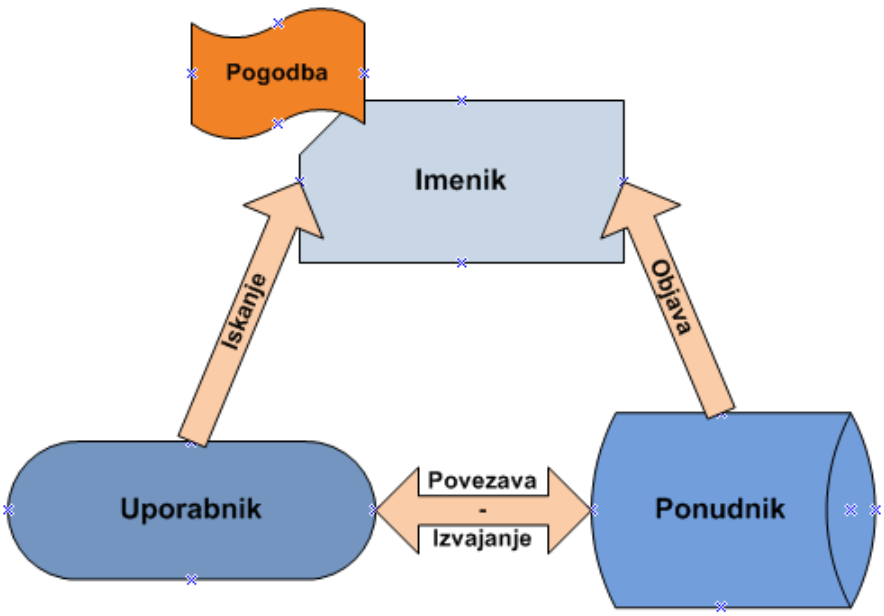
Izhajajoč iz tega spoznanja je očitno, da je že razvoj poslovnih procesov, ki morajo povezati množico obstoječih sistemov, zamuden in kompleksen. Enako velja za dopolnjevanje in spreminjanje. Več kot očitno je, da na tak način informatiki ne moremo zadostiti potrebam po fleksibilnih in hitro spreminjajočih se poslovnih procesih.

Storitvene arhitekture ponujajo odgovor na ta problem in ponujajo načine razvoja in integracije poslovnih aplikacij na osnovi modularnih, šibko sklopljenih storitev. Ključni poudarek storitvenih arhitektur je torej na definiciji načina integracije avtonomnih obstoječih ali novo razvitih aplikacij – imenovanih storitev - v celovit sistem s posebnim poudarkom na modelu komunikacije, podpori različnih transportnih mehanizmov, zagotavljanju varnosti in transakcijske identitete, zanesljivosti sporočanja ter koordinaciji in kompoziciji.

4.2 Osnovne komponente

Pri spletnih storitvah imamo tipično štiri arhitekturne gradnike:

- ponudnik storitve,
- uporabnik storitve,
- imenik storitev,
- pogodba.



Slika 5: Shema SOA arhitekture

Tipičen postopek [13] [14]:

- ponudnik storitve razvije storitev in opiše vmesnik v WSDL dokumentu,
- ponudnik lahko objavi storitev v imeniku storitev,
- uporabnik poišče želeno storitev v imeniku storitev,
- ko najdemo želeno storitev, preberemo WSDL dokument, ki opisuje vmesnik storitve,
- uporabimo želeno storitev na način, ki ga določa vmesnik.

Ponudnik storitve

Ponudnik storitve je medmrežno naslovljena entiteta, ki sprejema in izvaja zahteve s strani uporabnika storitve. Ponudnik je lahko glavni računalnik, komponenta, ali kateri drugi tip programskega sistema, ki obdeluje storitvene zahteve. Ponudnik storitve lahko zahteva uporabo pogodbe v imeniku storitev in s tem omeji dostop do posamezne storitve in v zameno zahteva plačilo.

Ponudnik storitve lahko ponudi vmesnik spletne storitve, neposredno uporabniku, v tem primeru je uporaba imenika storitev nepotrebna.

Uporabnik storitve

Uporabnik storitve je lahko aplikacija, fizična oseba kot končni uporabnik, druga storitev ali kak drug programski modul, ki za izvajanje potrebuje to storitev. Uporabnik storitve poišče lokacijo (naslov) storitve v imeniku, se poveže s storitvijo ter zahteva izvajanje storitve. Uporabnik izvede storitev, tako da pošlje zahtevo v obliki, kot je predpisana v pogodbi.

Imenik storitev

Imenik storitev je medmrežni imenik, ki vsebuje naslove spletnih storitev. Je enota, ki sprejema in shranjuje pogodbe s strani ponudnikov storitev ter na drugi strani ponuja pogodbe zainteresiranim uporabnikom storitev.

Pogodba

Pogodba je specifikacija, kako uporabnik storitve vzajemno komunicira s ponudnikom storitve. Pogodba predpisuje obliko zahteve in odgovora spletne storitve. Pogodba lahko vsebuje določene predpogoje in popogoje, ki opisujejo, kakšno mora biti stanje v sistemu, bodisi pred ali pa po izvajanju storitve. Pogodba lahko predpisuje tudi različne stopnje

kakovosti (QoS), ki opisujejo zgolj nefunkcionalne vidike kakovosti storitve. Čas dostopa ter čas izvajanja storitve opisujejo nefunkcionalno kakovost storitve.

4.3 Karakteristike storitveno usmerjene arhitekture

Ključne karakteristike storitveno usmerjene arhitekture [13]:

- Enkapsulacija – implementacija strežnika je skrita odjemalcu, saj strežnik objavi samo vmesnik. Uporabnik nima nikakršne ideje, kako je storitev realizirana.
- Strežnik ni vezan na fizično lokacijo. Strežnik se prijavi v imenik storitev, nato pa uporabnik lahko pridobi trenutni mrežni naslov storitve. Če strežnik zamenja mrežni naslov, se lahko ponovno prijavi v imenik storitev, na ta način razbremenimo uporabnika, saj mu ni potrebno pomniti mrežnega naslova storitve.
- Razpoložljivost strežnika je splošna, to pomeni, da je strežnik na voljo 24 ur na dan, 7 dni v tednu. Da zadovoljimo te potrebe, mora biti vzdrževanje opravljeno v času, ko prizadenemo najmanj uporabnikov. Ob izpadu je zaželena samodejna namestitev nadomestnega strežnika.
- Zasedenost strežnika mora biti skrbno načrtovana, strežnik lahko vzporedno obdeluje samo toliko zahtev, kot je določeno z zgornjo mejo obremenitve.
- Zanesljivost strežnika mora biti čim večja, zagotavljati mora podatkovno integriteto oziroma konsistenco v celotnem sklopu vseh aplikacij.
- Strežnik poskrbi za varnost na različnih nivojih storitvene arhitekture, saj so podatki pogosto tajni oziroma so poslovna skrivnost.
- Dokaz o identiteti je bistven del poslovnega sveta, saj v vsaki poslovni aktivnosti težimo k preglednosti sistema ter pravilom, ki opisujejo privilegije in obveznosti odjemalca.
- Upravljanje z napakami in izjemami je potrebno zato, da obvestimo odjemalca (uporabnika) in poslovne partnerje o ne-dokončanju poslovnega procesa, v katerem sodelujejo.
- Usklajevanje poslovnih procesov pa zagotovimo z jezikom BPEL. BPEL je standardiziran jezik za opisovanje poteka poslovnega procesa z uporabo spletnih storitev kot aktivnosti v poslovnem procesu.

4.4 Primernost storitveno usmerjene arhitekture

Storitveno usmerjena arhitektura ni primerna za vsakogar. Pred vpeljavo je potrebno dobro premisliti o funkcionalnosti sistema, ki ga imamo in katerega želimo.

Naj omenim nekaj ključnih primerov, kjer je uporaba storitvene arhitekture neprimerna [15]:

- V stabilnem nespremenljivem informacijskem okolju ni potrebe po vgraditvi storitvene arhitekture ali pa cena presega učinkovitost naložbe.
- V informacijskem okolju, ki ne ponuja aplikacij kot storitev zunanjim poslovnim partnerjem.
- V informacijskem okolju, ki ne uporablja zunanjih storitev, katere zahtevajo fleksibilnost in standardne povezovalne postopke.
- V primeru realno časovnega pristopa, saj SOA temelji na ohlapno vezani asinhroni komunikaciji, ki ne zagotavlja najhitrejšega odziva.

4.5 Prednosti in slabosti SOA testiranja

V diplomski nalogi bom predstavila postopek testiranja sistemov zgrajenih po načelih storitveno usmerjena arhitekture (SOA). Zakaj je testiranje SOA tako drugačno od tradicionalnih postopkov testiranja? Zato, ker imajo ti sistemi določene posebnosti kot so na primer:

- sistemi temeljijo na storitvah in so naravno porazdeljeni,
- storitve se v sistemu spreminjajo neodvisno ena od druge,
- lastništvo posameznih delov sistema je porazdeljeno med različne deležnike,
- sistemi implementirajo prilagodljivo obnašanje bodisi s spreminjanjem obstoječih storitev ali dodajanjem novih storitev,
- sistemi so naravno porazdeljeni, kar zahteva zagotavljanje kvalitete storitev (Quality of Servise - QoS) z drugačnimi namestitvenimi konfiguracijami.

Velik del uveljavljenih pristopov testiranja, ki so se v preteklosti uporabljali za testiranje tradicionalnih sistemov, se nanaša tudi na storitveno usmerjene sisteme. Za popolno testiranje se uporablja kombinacija testiranja enot, integracijskega, varnostnega, sistemskega in regresijskega testiranja. Zaradi dinamične in prilagodljive narave SOA večina metod testiranja ni neposredno primerna za testiranje storitev in storitveno usmerjenih sistemov.

Ključni dejavniki, ki omejujejo testiranje SOA sistemov, so [6]:

- nepoznavanje ali pomanjkljivo poznavanje kode in strukture storitev,
- dinamičnost in prilagodljivost storitev,
- pomanjkanje nadzora,
- cena testiranja.

Nepoznavanje ali pomanjkljivo poznavanje kode in strukture storitev

Uporabniki, ki uporabljajo storitev, ne poznajo strukture storitev. Zanje so storitve vidne le kot vmesniki, kar pomeni, da nam je v primeru, ko nismo ponudniki storitve temveč le njeni uporabniki, preprečeno testiranje po načelu bele škatle, ki zahteva poznavanje notranje strukture storitve.

Dinamičnost in prilagodljivost storitev

V tradicionalnih sistemih običajno določimo klicano komponento ali množico možnih komponent, ki bodo klicane. V SOA sistemih pa temu ni tako, saj je sistem lahko opisan skozi delovni tok abstraktnih storitev, ki so avtomatsko povezane z dejanskimi storitvami, ki jih sistem pridobi iz enega ali več imenikov med izvajanjem instanc delovnega toka.

Pomanjkanje nadzora

V tradicionalnih sistemih so komponente oz. knjižnice fizično integrirane v programski sistem, medtem ko v SOA sistemih temu ni tako. Storitve se izvajajo na neodvisni infrastrukturi in se razvijajo pod nadzorom ponudnika storitve. Uporabniki storitve se torej ne odločajo o strategiji razvoja storitve, o tem kdaj bodo prešli na uporabo nove različice storitve, …

Cena testiranja

Klicanje dejanskih storitev na strežniku njenega ponudnika je lahko ključnega pomena za ceno testiranja. Če naš sistem uporablja zunanje storitve, lahko le-ta zaračunava uporabo storitev. Problemi se pojavijo predvsem pri tako imenovanih »masovnih testih« (kadar je med testiranjem številko klicev storitve zelo veliko). V takšnih primerih lahko pri ponudniku pride do pojava zanikanja storitve (denile-of-service), kar lahko povzroči, da ponavljajoče testiranje storitev ne bo več mogoče, kadar ima storitev poleg navadnih odgovorov stranske učinke.

5 INTEGRACIJSKO TESTIRANJE

Integracijsko testiranje je vmesna faza med testiranjem enot in varnostnim ter sistemskim testiranjem. Pri testiranju integracije nas zanima povezava med posameznimi enotami in medsebojno delovanje in složnost sestavljenega podsistema [16].

Integracijsko testiranje se izvaja postopoma, med sestavljanjem posameznih enot v manjše ali večje podsisteme. Podsistem je uspešno integriran, če je bil uspešno preveden, povezan, naložen in so vmesniki vseh povezanih enot uspešno prestali vse integracijske teste. Največji podsistem, ki ga lahko sestavimo iz posameznih enot, je celoten sistem. Takrat že govorimo o sistemskem testiranju.

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

Integracijsko testiranje je lahko eden najbolj kritičnih trenutkov v razvoju programske opreme, saj se v tej fazi testiranja pokažejo morebitne težave v komunikaciji med razvijalci, pomanjkljivosti specifikacij in slabosti, ki so nastale pri načrtovanju programske opreme. Ker je razvoj kompleksnega programskega sistem zelo obširen, zahteva večje število razvijalcev in pogosto je komunikacija med njimi omejena. Zaradi tega se pojavijo tako imenovani komunikacijski otoki, med katerim je prenos informacij in znanj otežen.

5.1 Pristopi k integracijskemu testiranju

Obstaja več pristopov k integracijskemu testiranju oz. h gradnji testnega podsistema, ki ga lahko sestavimo na več načinov [17]:

- Big-bang,
- 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,
- kombinacija pristopa od spodaj navzgor in pristopa od zgoraj navzdol.

5.1.1 Big-bang

Pri gradnji podsistema z Big-bang pristopom pravzaprav ne gradimo podsistema, ampak že kar celoten sistem. Vse enote naenkrat združimo in jih nato skupaj testiramo. V praksi takšen pristop ni priporočljiv, saj z njegovo pomočjo ne bomo odkrili vseh napak. Lahko pa se ga uporabi, ko so integracijski testi v večini že uspešno opravljeni.

5.1.2 Od spodaj navzgor

Pri gradnji od spodaj navzgor začnemo s testiranjem osnovnih enot na najnižji ravni hierarhije. Nato k osnovni enoti povežemo enoto, ki je višje in vključuje že testirano enoto. Podsistem dopolnjujemo in gradimo toliko časa, dokler niso vse enote vključene v testiranje.

Pri takšni gradnji oz. testiranju potrebujemo posebne module, ki nadomestijo manjkajoče enote na višjih ravneh, ki jih še nismo testirali. Tem modulom pravimo testni gonilniki (angl. drivers). Z njihovo pomočjo kličemo testirane komponente. Postopoma odstranjujemo gonilnike in jih nadomeščamo s pravimi enotami.

5.1.3 Od zgoraj navzdol

Pri gradnji podsistema s pristopom od zgoraj navzdol se podsistemov lotimo ravno na obratni način kot pri pristopu od spodaj navzgor. Začnemo na najvišji ravni s kontrolnim podsistemom, ki mu postopoma dodajamo testne enote. Manjkajoče podsisteme simuliramo z lupinami (angl. stubs).

5.1.4 Kombinirana gradnja

Če si predstavljamo, da je sistem razdeljen na tri ravni (ciljna, spodnja in zgornja raven), potem lahko uporabimo kombinacijo gradnje od spodaj navzgor in od zgoraj navzdol. Podsistem gradimo v smereh od spodaj navzgor in od zgoraj navzdol proti ciljni ravnini. Leto si je potrebno izbrati tako, da pri gradnji potrebujemo čimmanj testnih gonilnikov in lupin.

5.2 Umestitev integracijskega testiranja v celoten postopek testiranj

Integracijsko testiranje se izvaja za testiranjem enot, saj je potrebno pred preverjanjem pravilnosti integracije preveriti ali delujejo pravilno posamezne storitve, ki bodo del širšega sistema. Izvaja pa se pred sistemskim in varnostim testiranjem, saj je potrebno pred preverjanjem ustrezne zmogljivosti sistema in zadostne varnosti zagotoviti, da sistem deluje funkcionalno pravilno.

Namen

Namen integracijskega testiranja je preverjanje pravilnosti skupnega delovanja povezanih storitev, ki naj bi same zase že delovale pravilno (predhodno uspešno opravljeno testiranje enot). Kljub temu, da posamezne storitve/enote delujejo pravilno, to še ne pomeni da ne more priti do napak pri interakciji med storitvami ter med storitvami in drugimi komponentami sistema.

5.3 Vrste napak

Napake, ki jih iščemo z integracijskim testiranjem, so sledeče:

- napake v interakcijah med posameznimi storitvami (največkrat izvirajo iz napačnega razumevanja opisa storitev),
- napake v integraciji storitev s preostalimi komponentami sistema,
- storitve niso skladne s standardi interoperabilnosti,
- ali sistem deluje kot pričakujemo tudi z namestitvami, ki so identične namestitvam, kot jih bo uporabljal končni uporabnik sistema,
- funkcionalne napake.

5.4 Metodologija integracijskega testiranj

Integracijsko testiranje sestavljajo tri zaporedne (pod)faze [6]:

1. faza: preverjanje interakcij med različnimi komponentami. Zajema testne primere, ki prožijo vse primerke medsebojne komunikacije in klicev komponent.
2. faza: testiranje upoštevanja smernic pravilne souporabe različnih specifikacij o storitvah za doseganje višje ravni interoperabilnosti predvsem med storitvami našega in zunanjih sistemov.
3. faza: funkcionalno testiranje. Zajema preverjanje funkcionalnosti razvitih storitev in procesov v skladu z uporabniškimi zahtevami.

V nadaljevanju bom podrobno opredelila vse tri podfaze integracijskega testiranja.

5.4.1 Testiranje interakcij

5.4.1.1 Opis

Testni primeri preverjanja interakcij morajo zajemati vse primere medsebojnih komunikacij in klicev storitev. Gre torej za testiranje sestavljenih storitev. Testni primeri izhajajo iz že opredeljenih testnih primerov enot, s tem da nas v tej fazi zanimajo le primeri, ki vključujejo interakcije med storitvami, in da se testiranje ne izvaja z nadomestnimi partnerskimi storitvami, temveč z dejanskimi storitvami, nameščenimi v razvojno oziroma testno okolje.

5.4.1.2 Napake

Napake, ki jih iščemo s testiranjem interakcij, navadno izvirajo iz napačnega razumevanja opisov partnerskih storitev oziroma nepravilne obravnave prejetih sporočil. WSDL opisi storitev namreč določajo le sintakso storitev, ne pa tudi semantike, kot je na primer: pomen posameznih parametrov, obnašanje v primeru napak itd. Primer je neskladno obravnavanje enote parametra, se pravi, če določena storitev vrne podatek tipa integer, pri čemer se ta nanaša na velikost v metrih, storitev, ki podatek prejme, pa ga obravnava kot decimetre.

5.4.1.3 Postopek

Izdelava testnih specifikacij

Naš namen je opredeliti vse primerke medsebojne komunikacije, pri čemer je še posebej pomembno preveriti, ali so podatki, ki se pri tem prenašajo, na obeh straneh komunikacije razumljeni na pravilen način. Pri opredeljevanju testnih primerov preverjanja interakcij izhajamo iz testnih primerov enot, pri čemer bo, če smo natančno opredelili testne primere enot, za vse možne poti skozi kodo posameznega procesa ali sestavljene storitve vsak testni primer preverjanja interakcij sovpadal z določenim testnim primerom enote (ne pa tudi obratno).

Priporočeno ogrodje za zapis testne specifikacije je prikazano v spodnji tabeli:

Glava	**Oznaka** Enolična oznaka testa.
	Opis Kratek opis testnega primera, vključno z interakcijo, ki jo testiramo.
Telo	**Storitev oziroma procesi in operacije** Storitev oziroma proces in operacija, ki jo prožimo za začetek izvajanja primera uporabe, ter morebitne druge operacije med testnim ogrodjem (odjemalcem) in storitvami, ki jih zahteva primer uporabe. Pri vsaki operaciji je potrebno označiti, če je vhodna ali izhodna (glede na storitev).

	Vhodni podatki Vhodni podatki, s katerimi prožimo operacijo. Določimo jih za vsako operacijo, kjer so potrebni.
	Pričakovani izhodni podatki Opis pričakovanih izhodnih podatkov, s katerimi preverimo dejanske izhodne podatke, ki jih pošlje testna storitev oziroma BPEL proces. Določimo jih za vsako operacijo, kjer so potrebni.
	Zaporedje storitev oziroma procesov z operacijami in pripadajočimi prenesenimi sporočili Veljavna zaporedja storitev oziroma BPEL procesov, ki sodelujejo v interakciji, pripadajočih operacij in pogojev, ki jim morajo zadoščati prenesena sporočila.
	Pogoji Pogoji, ki morajo veljati za pravilno izvedbo testiranja.
	Število ponovitev Določene testne primere je potrebno izvesti večkrat.
	Rezultat Uspešno/neuspešno zaključen test (za vsako ponovitev, če jih je več).

Tabela 2: Priporočeno ogrodje testne specifikacije

Opredelitev testne strukture

Opredeliti moramo testno strukturo, ki ni nič drugega kot združevanje testnih primerov. Pri testiranju interakcij testne primere razvrščamo v skupine glede na testne skupine vhodnih podatkov. Strukturo je potrebno oblikovati tako, da se posamezne enakovredne interakcije ne bodo prožile večkrat kot je to potrebno. Če se določeno zaporedje interakcij proži v okviru določenega testnega primera sestavljene storitve, kot podzaporedje vseh interakcij tega testnega primera, podzaporedja ni potrebno posebej testirati. To bi zgolj ponovilo enak postopek .

Izdelava testov in analiza rezultatov

Testiranje interakcij izvajamo s preprostimi klici storitev. Pri tem je potrebno beležiti storitve in njihove operacije, ki se v okviru interakcije prožijo, ter sporočila, ki si jih prenašajo.

Uspešnost testov preverjamo z zaporedjem teh operacij in prenesenimi sporočili, in sicer je potrebno preveriti naslednje:

- zaporedje storitev in operacij mora ustrezati zaporedju storitev oziroma zaporedju procesov z operacijami opredeljenimi v testnih specifikacijah,
- prenesena sporočila morajo ustrezati pogojem opredeljenim v testnih specifikacijah.

Izvedba testov proizvede rezultate in statistike, ki so predstavljene uporabniku. Za analizo rezultatov testiranja interakcij se navadno uporablja metrika boolean rezultatov izvajanja, ki jih navadno omogočajo vsa testna ogrodja: test uspe ali ne uspe. Neuspeh testa lahko razdelimo v dve kategoriji, in sicer:

- napačno zaporedje operacij storitev ali neustrezno sporočilo (failure): program se je zaključil, vendar se ni pravilno izvedel – interakcija je povzročila nepravilno pot po kodi storitev, na primer zaradi napačno interpretiranih izmenjanih podatkov.
- napaka v izvajanju (error): med izvajanjem programa je prišlo do napake in program se je zaključil nepravilno – interakcija je povzročila nepravilno izvajanje določene storitve, na primer zaradi napačno interpretiranih izmenjanih podatkov, ki so povzročili, da se je sprožila izjema, ki ni bila ujeta.

5.4.2 Testiranje skladnosti s smernicami WS

5.4.2.1 Opis

Ključne storitvene tehnologije, kot so SOAP, WSDL in XML, so načrtovane z namenom poenostavljanja interoperabilnosti in integracije. Kljub vsemu pa zgolj uporaba teh tehnologij sama po sebi še ne zagotavlja popolne interoperabilnosti. Eden izmed večjih problemov dandanes je namreč v nepazljivi implementaciji storitev, ki včasih vsebujejo lastnosti značilne za platformo, na kateri so razvite.

Zato je poleg izvajanja testiranja, ki preverja pravilnost delovanja storitev in njihove funkcionalne ustreznosti, potrebno preveriti tudi, ali zagotavljajo ustrezno raven interoperabilnosti. Ker je interoperabilnost ena izmed ključnih prednosti, ki jih prinašajo storitve, in pogosto igra pomembno vlogo pri odločanju o izbiri te tehnologije, so testni primeri za preverjanje interoperabilnosti zelo pomembni.

Za preverjanje interoperabilnosti uporabljamo pristop preverjanja ustreznosti s standardi. S tem področjem se ukvarja organizacija *Web Service Interoperability Organization* (WS-I), ki je razvila in daje na razpolago različne vire za izdelavo interoperabilnih storitev in preverjanje skladnosti rezultatov z njihovimi smernicami. Poglavitni viri, ki jih ponujajo, so:

- Profili, ki vsebujejo smernice, ki pojasnjujejo kako uporabljati različne specifikacije storitvenih tehnologij z namenom doseganja interoperabilnosti. Danes je na razpolago

več različic ključnih smernic, in sicer Basic Profile 1.0 [WS-I BP1.0], 1.1 [WS-I BP1.1], 1.2 [WS-I BP1.2] in 2.0 [WS-I BP2.0].

- Vzorčne aplikacije, ki prikazujejo primere storitvenih aplikacij skladnih s smernicami WS-I. Razvite so z uporabo različnih platform, programskih jezikov in orodij.
- Testna orodja, ki omogočajo testiranje skladnosti sporočil komunikacije storitev s smernicami WS-I.

5.4.2.2 Napake

Napaka, ki jih iščemo pri testiranju skladnosti s smernicami WS, izhajajo iz neskladnosti z izbranim profilom.

5.4.2.3 Postopek

Izdelava testnih specifikacij

Testne specifikacije v primeru preverjanja ustreznosti opisa storitev in pri primerjanju ustreznosti SOAP sporočil se razlikujejo. Njihovo ogrodje prikazujeta spodnji tabeli (Tabela 3 in Tabela 4). V obeh primerih lahko večino pogojev preverjanja ustreznosti, t.j. pogoje, ki se nahajajo v polju Izhodni podatki specifikacij testiranja interoperabilnosti SOAP sporočila in v polju Pogoji opisa storitve specifikacij testiranja interoperabilnosti opisa storitve, zapišemo s pomočjo XPath izrazov. Včasih XPath ne zadostuje, na primer v opisu testnega primera z zaporedno številko 21 (razpoložljivost WSDL), in je potrebno pogoje, če želimo teste avtomatizirati, implementirati na drugačen način, na primer s programskim jezikom, ki ga uporabljamo za izdelavo testnega ogrodja.

Glava	**Oznaka** Enolična oznaka testa.
	Opis Kratek opis testnega primera.
	Zunanja storitev Zunanja storitev, ki jo testiramo.
	Operacije Operacije storitve, ki jo testno ogrodje proži z opredeljenimi vhodnimi podatki, ter morebitne druge operacije potrebne za interakcijo med odjemalcem in zunanjo storitvijo, na primer operacija povratnega klica, če gre za asinhrono interakcijo.
Telo	**Vhodni podatki** Vhodni podatki, s katerimi prožimo storitev. Določimo jih za vsako operacijo,

	kjer so potrebni.
	Izhodni podatki Opis oziroma pogoji, ki jim bi naj ustrezalo pričakovano izhodno sporočilo testirane zunanje storitve. Določimo jih za vsako izhodno operacijo, kjer so potrebni.
	Pogoji Pogoji, ki morajo veljati za pravilno izvedbo testiranja.
	Rezultat Uspešno/neuspešno zaključen test (za vsako ponovitev, če jih je več).

Tabela 3: Ogrodje testne specifikacije testiranja interoperabilnosti SOAP sporočila

Glava	**Oznaka** Enolična oznaka testa.
	Opis Kratek opis testnega primera.
	Zunanja storitev Zunanja storitev, ki jo testiramo.
Telo	**Pogoji opisa storitve** Pogoji, ki jim bi opis storitve naj zadoščal, če ustreza smernicam izbranega profila.
	Rezultat Uspešno/neuspešno zaključen test.

Tabela 4: Ogrodje testne specifikacije testiranja interoperabilnosti opisa storitev

Opredelitev testne strukture

Ker se testni primeri preverjanja interoperabilnosti izvajajo le za zunanje storitve, testno strukturo opredelimo v treh ravneh, in sicer na:

- testne primere testiranja SOAP sporočil in na testne primere testiranja opisov storitev,
- testne primere razvrstimo v skupine glede na ponudnike storitev,

- testne primere razvrstimo še na testne primere posameznih storitev.

Izdelava testov in analiza rezultatov

Za testne primere, ki se nanašajo na sporočila, je pristop k izvedbi testiranja podoben kot v predhodnih fazah testiranja. Zunanjo storitev prožimo z vhodnim sporočilom določenim s testnimi specifikacijami, ter, če testni primer zahteva preverjanje ustreznosti izhodnih sporočil, zabeležimo izhodna sporočila. Če so sporočila skladna z zahtevami, test uspe, sicer ne.

Za testne primere, ki se nanašajo na opise storitev, le preverimo ali opis storitve zadošča pogojem v testnih specifikacijah. Če je pogojem zadoščeno, test uspe, sicer ne.

Ker gre za testiranje zunanjih storitev, je v primeru neuspešnega testa možnih več scenarijev. Če nimamo nobenega vpliva na spremembo storitve, se je potrebno odločiti, ali bomo storitev kljub temu uporabljali, s tem da bomo upoštevali neskladje, ali bomo uporabili drugo zunanjo storitev, ki bi bila primerna. Če med ponudnikom zunanje storitve in med nami ali naročnikom sistema, ki ga razvijamo, obstaja dogovor o zahtevani ravni interoperabilnosti, lahko včasih vplivamo na zunanjega ponudnika, da odpravi neskladnost.

5.4.3 Funkcionalno testiranje

5.4.3.1 Opis

Funkcionalno testiranje je zadnja faza integracijskega testiranja, saj je potrebno pred preverjanjem funkcionalne pravilnosti integracije storitveno usmerjenega sistema, zagotoviti, da je skupno delovanje storitev pravilno in da storitve ustrezajo standardom interoperabilnosti. Če interakcije med storitvami ne potekajo pravilno, potem sistem prav gotovo ne zadošča funkcionalnim zahtevam. S funkcionalnim testiranjem preverimo, če sistem, ki ga razvijamo, ustreza funkcionalnim zahtevam, ki so navadno opredeljene s primeri uporabe. Zato testiramo pravilnost izvajanja vseh scenarijev posameznih primerov uporabe. Za razliko od predhodnih faz testiranja se funkcionalno testiranje izvaja skupaj z drugimi deli sistema, v katerem se bo nahajal sistem, ki ga razvijamo.

5.4.3.2 Napake

S funkcionalnim testiranjem želimo odkriti morebitne funkcinalne napake in probleme, ki nastopijo, ko izvajamo primere uporabe storitveno usmerjenega sistema, ki ga razvijamo. Namen tega je, da napake v uporabniških scenarijih odkrijemo, preden končni uporabnik začne sistem uporabljati. Ker bi naj na tem mestu sistem, ki smo ga razvijali, sam zase že deloval pravilno, se napake lahko pojavijo tudi v njegovem povezovanju z ostalimi deli sistema.

5.4.3.3 Postopek

Izdelava testnih specifikacij

Testne primere funkcionalnega testiranja identificiramo na podlagi arhitekturnega načrta, na kar jih dopolnimo s podrobneje določenimi funkcionalnimi zahtevami podrobnega načrta, ki je izdelan v razvojni fazi načrtovanja. Pri tem so funkcionalne zahteve navadno opredeljene z modelom primerov uporabe. Vsak primer uporabe je lahko podan z diagramom in enim ali več scenariji, ki opisujejo glavno zaporedje korakov in alternativna zaporedja korakov. Za vsak scenarij opredelimo vsaj en testni primer.

Scenariji lahko zajemajo izvajanje različnih aplikativnih sistemov, ki so s storitveno usmerjenim sistemom, ki ga razvijamo, tako ali drugače povezani, na primer proženje določenega scenarija iz uporabniškega vmesnika, ki v ozadju proži zaporedje BPEL procesov in drugih storitev. Zato je poleg zaporedja operacij storitev in prenesenih sporočil potrebno določiti aplikativni sistem in morebitni podsistem oziroma funkcionalnost, v kateri se izvajanje posameznega scenarija začne, ter ustrezne vhodne podatke, na primer preko katere uporabniške aplikacije se primer uporabe začne, s katerimi uporabniškimi pravicami in s katerimi podatki. Prav tako je potrebno določiti morebitne pogoje, ki morajo biti izpolnjeni med izvajanjem ter po končanem izvajanju, ter interakcije z ostalimi aplikativnimi sistemi in podsistemi, ki sodelujejo v primeru uporabe. To vključuje tudi celotno zaporedje proženih operacij storitev in sporočil, ki se prenašajo, in sicer:

- ali so v zaporedje interakcij vključene pričakovane storitve in ali gre za pravilen vrsti red nastopa posamezne storitve v zaporedju interakcij,
- ali sporočila, ki si jih v interakcijah storitve pošiljajo med seboj, ustrezajo funkcionalnim zahtevam.

Če so na tem mestu že opredeljeni testni scenariji prevzemnega testiranja, jih vključimo v testne specifikacije funkcionalnega testiranja.

Tabela 5 prikazuje in razlaga posamezna polja ogrodja testne specifikacije posameznega testnega primera funkcionalnega testiranja.

Glava	**Oznaka** Enolična oznaka testa.
	Opis Kratek opis testnega primera.
	Primer uporabe in scenarij

	Primer uporabe in pripadajoči scenarij zaporedja korakov, ki ga testiramo.
Telo	**Aplikativni sistemi in podsistemi** Aplikativni sistem, s katerim prožimo primer uporabe, na primer uporabniški vmesnik, ter morebitni ostali aplikativni sistemi, ki so potrebni za izvedbo primera uporabe. Potrebno je natančno podati tudi podsisteme in funkcijo, če sicer ni razvidno, katero funkcijo je potrebno prožiti. Način, na katerega opišemo proženje posameznega aplikativnega sistema, je v veliki meri odvisen od same implementacije.
	Vhodni podatki Vhodni podatki, s katerimi prožimo primer uporabe, v aplikativnem sistemu, ter ostali vhodni podatki aplikativnih sistemov, ki so potrebni za izvedbo primera uporabe. Navadno ponazarjajo podatke, ki jih posredujejo zunanji sistemi in drugi odjemalci testnega sistema. Lahko vključujejo tudi uporabniške podatke, če so ti zahtevani za dostop do zahtevanega aplikativnega sistema.
	Pričakovani izhodni podatki Opis oziroma pogoji, ki jim morajo zadoščati izhodni podatki, ki predstavljajo odgovore v proženih funkcijah aplikativnih sistemov, če so zahtevani; na primer odgovor stranki na določeno poizvedbo v aplikativnem sistemu.
	Zaporedje storitev oziroma procesov in operacij Veljavna zaporedja storitev in procesov skupaj s pripadajočimi operacijami, ki bi se naj prožile v okviru izvajanja primera uporabe.
	Pričakovani vhodni podatki storitev oziroma procesov in operacij Opis pričakovanih vhodnih podatkov, s katerimi preverimo dejanske vhodne podatke, s katerimi so prožene posamezne operacije.
	Pričakovani izhodni podatki storitev oziroma procesov in operacij Opis pričakovanih izhodnih podatkov, s katerimi preverimo dejanske izhodne podatke, ki jih pošlje posamezna storitev oziroma proces.
	Pogoji Pogoji, ki morajo veljati za pravilno izvedbo testiranja. Vključujejo lahko tudi potrebno konfiguracijo aplikativnih sistemov.
	Rezultat Uspešno/neuspešno zaključen test (za vsako ponovitev, če jih je več).

Tabela 5: Ogrodje testne specifikacije testnega primera funkcionalnega testiranja

Opredelitev testne strukture

Testne primere funkcionalnega testiranja strukturiramo v skupine po posameznih primerih uporabe. Ker so navadno v arhitekturnem načrtu primeri uporabe že razporejeni v skupine, za testno strukturo uporabimo enako razvrstitev.

Izdelava testov in analiza rezultatov

Testne specifikacije funkcionalnega testiranja navadno vključujejo večje število pogojev kot specifikacije v predhodnih fazah testiranja. Izvedeni test mora za uspešno izvedbo zadostiti vsem pogojem. Razlog za neuspešen test je lahko bodisi v predhodnih fazah še neodkrita napaka v sistemu, ki ga razvijamo, bodisi napaka, ki izhaja iz njegove integracije z drugimi komponentami sistema. Rezultate neuspešnih testov je potrebno posredovati odgovornim razvijalcem.

6 INTEGRACIJSKO TESTIRANJE SOA V PRAKSI

Vsi operaterji, ki ponujajo telefonske storitve, morajo izvajati prenosljivost številk na osnovi Zakona o elektronskih komunikacijah (ZEKom) (Uradni list RS, št. 13/2007 in št. 110/2009) in Splošnega akta o prenosljivosti številk (Uradni list RS, št. 75/2005, št. 25/2006, št. 39/2006, št. 16/2007 in št. 71/2008).

6.1 Osnovni pojmi

Osnovni pojmi so povzeti iz Splošnega akta o prenosljivosti številk [18].

Centralna baza podatkov (CBP) je tehnična podpora procesov za prenos številk v Republiki Sloveniji, ki beleži status vsake transakcije med postopkom prenosa in vsebuje vse potrebne podatke za usmerjanje klicev na prenesene številke, s katerimi se pravočasno ažurirajo lokalne baze prenesenih številk. Centralna baza podatkov je enotna za vse operaterje in je izven domene operaterjev/ponudnikov storitve .

Lokalna baza prenesenih številk (LBPŠ) je baza aktivnih podatkov, ki se nahajajo pri operaterjih in vsebuje podatke, potrebne za usmerjanje klicev na prenesene številke. Praviloma jo operaterji posodabljajo vsak delovni dan.

Operater je operater javno dostopnih telefonskih storitev, vključno z mobilnimi storitvami, kakor tudi prenosnih omrežij, ki posredujejo nacionalne (medkrajevne) oziroma mednarodne klice.

Operater dajalec številke (donor) je operater, ki zagotavlja javno dostopne telefonske storitve naročniku ali uporabniku pred prenosom številke.

Operater prejemnik številke (recipient) je operater, ki zagotavlja javno dostopne telefonske storitve naročniku ali uporabniku, ki uporablja prenesene številke.

Prenesena številka je nacionalna značilna številka, ki se nespremenjena prenese iz mobilnega telekomunikacijskega omrežja enega operaterja v mobilno telekomunikacijsko omrežje drugega operaterja ali iz fiksnega telekomunikacijskega omrežja enega operaterja v fiksno telekomunikacijsko omrežje drugega operaterja.

Upravljalec centralne baze podatkov (CBP) je fizična ali pravna oseba, ki na podlagi pravnega posla z operaterji upravlja s centralno bazo podatkov.

Naročnik je vsaka pravna ali fizična oseba, ki ima pri določenem operaterju sklenjeno naročniško razmerje in želi svojo telefonsko številko prenesti na drugega operaterja ali s prekinitvijo naročniškega razmerja ali pa brez nje.

'Todo' lista je nalog, ki ga dobi operater iz CBP in vključuje vse aktivnosti, ki jih mora izvršiti v naslednji noči. Navedene so številke, ki jih mora zaradi prenosa številk izključiti

(dajalec številke) in potrditi izključitev ter tiste, ki jih mora zaradi prenosa številk vključiti (prejemnik številke).

6.2 Splošno o prenosu telefonskih številk

Prenos telefonskih številk zahteva regulacija EU. Tako pospešuje prehode naročnikov med operaterji in jim pri tem omogoča ohranjanje telefonske številke. Trenutno je to možno znotraj fiksnih in znotraj mobilnih telefonskih omrežji.

Pri prenosu številk lahko ima vsak operater dve vlogi, in sicer:

- dajalca številke (angl. donor),
- prejemnika številke (angl. recipient).

Stik z naročnikom ima praviloma le operater prejemnik številke.

V Splošnem aktu o prenosljivosti številk [18] je zapisan postopek prenosa. Uporabnik, ki želi prenesti svojo telefonsko številko, poda povpraševanje o možnosti prenosa številke na prodajnem mestu pri izbranem operaterju, prejemniku številke.
Pri tem mora uporabnik, ki je predplačnik mobilnih storitev, predložiti svojo telefonsko številko in PUK številko, uporabnik naročnik telefonskih storitev pa telefonsko številko in številko računa, ki ni starejši od treh mesecev.
Izbrani operater takoj nato pri trenutnem operaterju (dajalcu številke) preveri možnost prenosa številke. O tem je uporabnik praviloma obveščen v roku 15 minut in v primeru možnosti lahko takoj vloži pisno zahtevo za prenos številke.
Izbrani operater nato izvede vse potrebne aktivnosti v zvezi z naročilom prenosa in po izključitvi številke iz omrežja dajalca številke tudi vključitev številke v svoje omrežje.

Čas, v katerem mora biti prenos telefonske številke opravljen

Prenos telefonske številke v mobilnih omrežjih se izvede najkasneje v 3 delovnih dneh od dneva, ko je operater dajalec številke prejel zahtevo za prenos številke. Enak rok 3 delovnih dni je predpisan za prenos številk v fiksnih omrežjih.

Dejanski prenos številke, t.j. izključitev iz omrežja dajalca številke in vključitev v omrežje prejemnika številke, se mora izvesti v najkrajšem možnem času med 00:00 in 06:00 uro, tedaj je uporabniku onemogočena uporaba komunikacijskih storitev.

Stroški prenosa telefonske številke

Operater dajalec številke lahko naročniku, ob dogovoru med operaterji pa operaterju prejemniku številke, za prenos številke k drugemu operaterju zaračuna enkratni znesek, ki

krije dejanske stroške prenosa številke, in sicer največ v višini 5 EUR za vsak zahtevek. Če posamezni uporabnik istočasno vloži zahtevke za večje število številk se skupni znesek za kritje stroškov prenosov vseh številk omeji na največ 200 EUR.

6.3 Postopek prenosa pri enem izmed ponudnikov TK storitev

V praktičnem delu diplomskega dela se bom osredotočila na prenose fiksnih številk v fiksnem omrežju.

6.3.1 Preveritev

Potek preveritve prikazuje slika 6. Operater prejemnik številke je dolžan identificirati naročnika s telefonsko številko in številko računa za TK storitve operaterja dajalca številke. Račun ne sme biti starejši od 3 mesecev.

CBP predhodno sama avtomatsko preveri ali je številka v fazi prenosa. Če je številka že v fazi prenosa se dodatno povpraševanje zavrne, sicer pa se ga posreduje dajalcu številke.

Številka je v fazi prenosa od pozitivnega povpraševanje do konca delovnega dne, če ne pride do naročila oz. do pridobitve negativnega odgovora oz. določitve roka za prenos in tudi ves čas trajanja postopka prenosa.

Posredovanje preveritve:

Dajalcu številke so preko CBP posredovani podatke o:

- telefonski številki,
- številki računa.

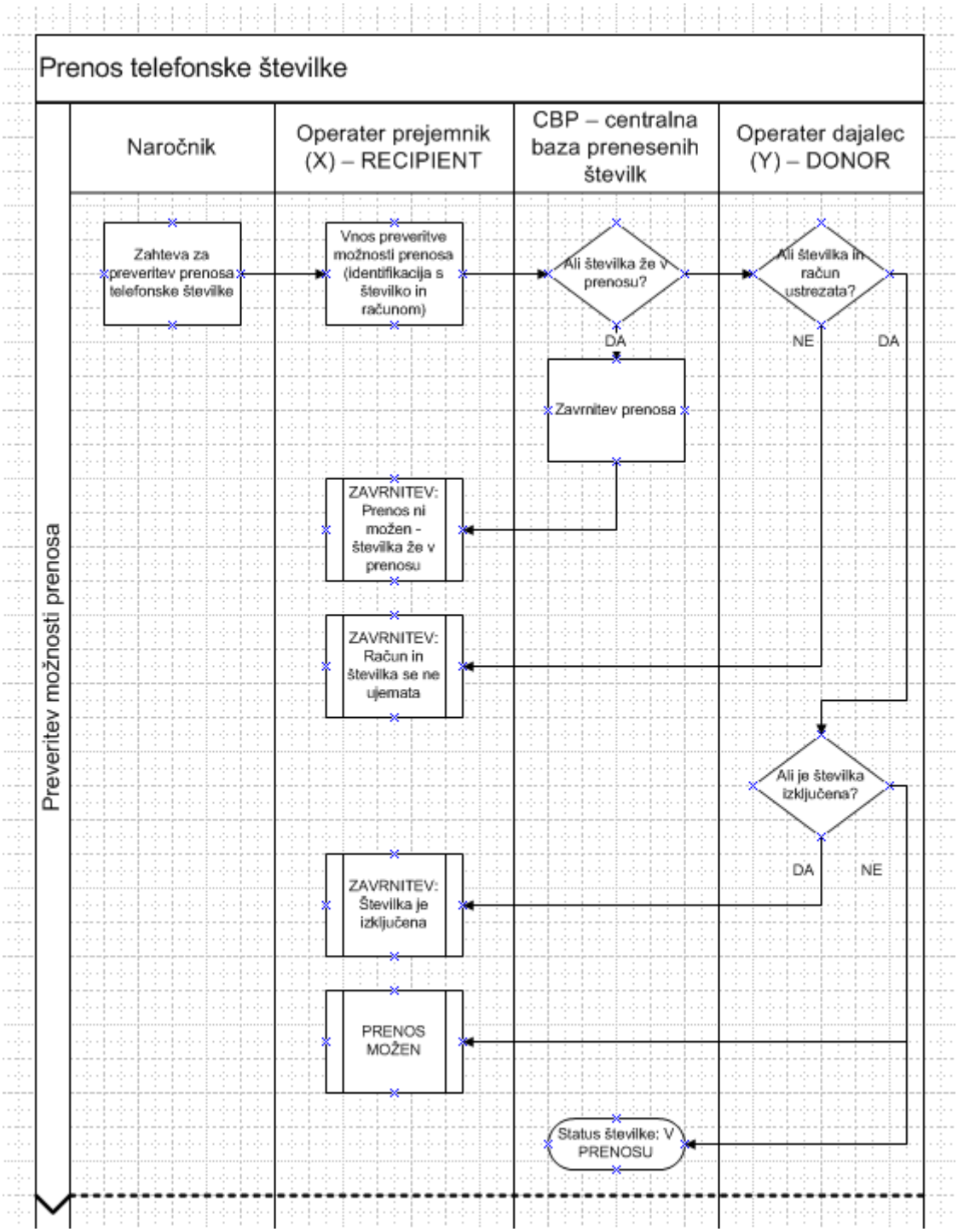
Dajalec številke mora preveriti sledeče:

- če številka, za katero se zahteva prenos, obstaja na posredovanem računu,
- če številka, za katero se zahteva prenos, ni stalno oz. začasno izključena.

Če sta izpolnjena navedena pogoja, je preveritev pozitivna.

»Delovni čas« CBP in roki za odgovor:

Delovni čas za prejemanje preveritev in naročil za vse operaterje določa Splošni akt o prenosljivosti številk in je vsak delovni dan od 08:00 do 16:00 ure. Na prejete zahteve mora dajalec številke odgovoriti še isti delovni dan in sicer na preveritev v največ 15 minutah in na naročilo v največ 3 urah.



Slika 6: Prikaz preveritve možnosti prenosa telefonske številke

6.3.2 Naročilo

Naročilo je možno na osnovi pozitivne preveritve. Potek naročanja prikazuje slika 7. Naročilo se posreduje v elektronski obliki preko CBP. Ob prejemu naročila se zagotovi obvestilo, da je naročilo prejeto.

V naročilu se k podatkom o preveritvi doda še odločitev naročnika o nadaljevanju naročniškega razmerja pri operaterju dajalcu številke.

Naloge operaterja prejemnika številke:

Pred naročilom je operater prejemnik številke (torej novi operater) dolžan seznaniti naročnika o načinu prenosa številke:

- časovnem obdobju, v katerem bo onemogočena storitev,
- obsegu in načinu uporabe storitev v omrežju operaterja prejemnika številke,
- razlogih za zavrnitev prenosa,
- razlogih za zakasnitev prenosa številke.

Če se naročnik strinja s temi pogoji, podpiše pri novem operaterju potrebne obrazce.

Naročilo prek CBP se posreduje za vsako številko posebej.

Operater prejemnik je dolžen operaterju dajalcu številke po faksu ali po e-pošti najpozneje v roku 30 minut poleg naročila za prenos posredovati še:

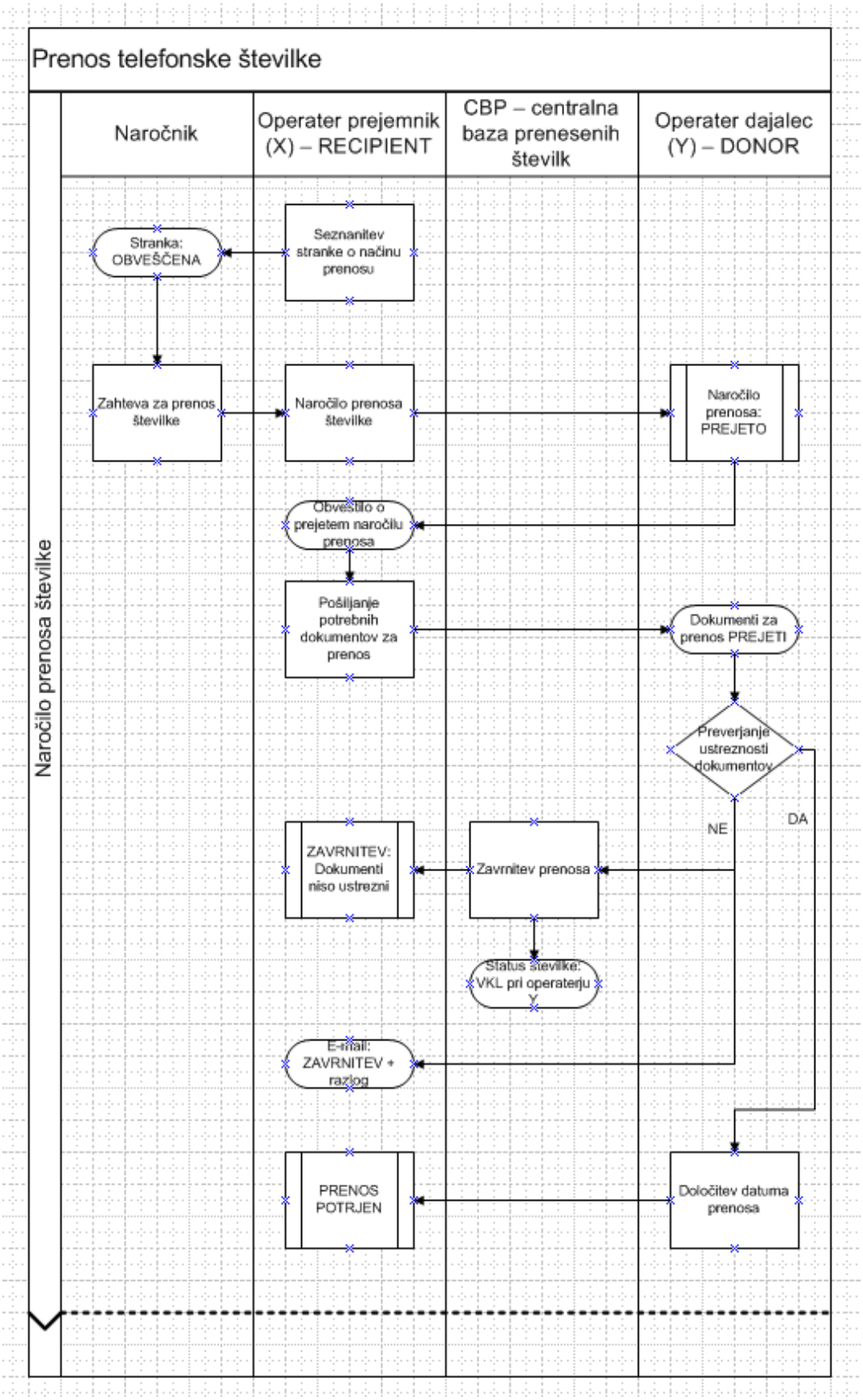
- v celoti izpolnjen in ožigosan Zahtevek za prenos številke in
- kopije potrebnih dokumentov

Naloge operaterja dajalca številke:

Dolžnosti operaterja dajalca številke (torej starega operaterja) so:

- prejeto naročilo prek CBP se natisne in zapiše nanj hh:mm (uro in minuto) prejema,
- počakati do 30 minut na prejem izpolnjenega obrazca Zahtevka za prenos številke s prilogami. Če popolno izpolnjenega zahtevka in/ali prilog ni, se naročilo zavrne in po e-pošti prejemnika številke obvesti o vzroku,
- v primeru prejete dokumentacije se:
    - preveri čitljivost podatkov,
    - preveri, če zahteva za prenos vsebujejo vse potrebne podatke (podpis naročnika (fizične osebe) ali samostojnega podjetnika (samostojni podjetniki) ali zastopnika/pooblaščenca (pravne osebe), kraj in datum, ter žig in podpis operaterja prejemnika številke,
    - preveri skladnost podatkov o prekinitvi naročniškega razmerja na zahtevi posredovani pred CBP in na prejetem Zahtevi za prenos.

V primeru neizpolnjevanja navedenega se naročilo zavrne prek CBP. Hkrati se operaterju prejemniku številke posreduje po e-pošti razlog za zavrnitev prenosa.



Slika 7: Prikaz naročila prenosa telefonske številke

6.3.3 Izvedba

V primeru izpolnjevanja navedenega in ujemanja podatkov se na zahtevek odgovori prek CBP. CBP se posreduje datum prenosa, ki je T+3 delovnih dni (T je datum prejema naročila). Izvedba je nujno na delovni dan. Če določimo kakšen drug dan, to CBP zavrne. Izvedbo prenosa telefonske številke prikazuje slika 8.

Naloge operaterja dajalca številke:

Operater dajalec mora izvesti izključitev točno na dan prenosa, ki je bil določen v CBP. Odstopanja niso dovoljena.

Preko CBP je operaterju dajalcu na dan prenosa posredovana »todo« lista za izvedbo izključitev. Ta lista se neposredno uporabi za izključevanje na komutacijski opremi, za delo na terenu (dostopovno omrežje in delo pri naročniku) pa se uporabijo običajni delovni in tehnični nalogi.

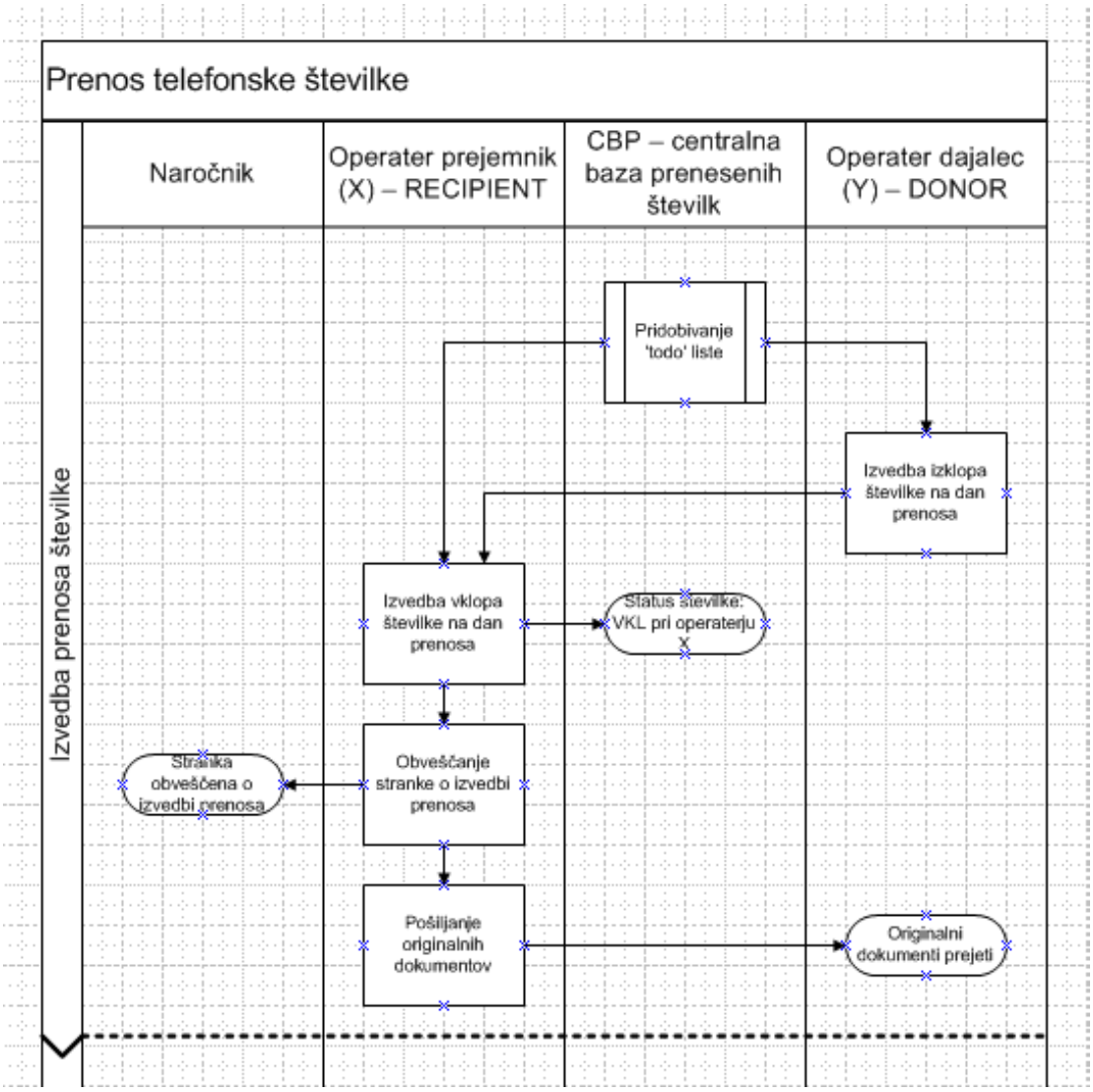
Izključitve na komutacijski opremi zaradi prenosljivosti se izvajajo samo v nočnem času in sicer vsak delovni dan od 00:00 do 06:00, in sicer praviloma v noči na ponedeljek, torek, sredo, četrtek in petek. Izključitve se potrdi preko CBP.

Naloge operaterja prejemnika številka:

Telefonska številka se k operaterju prejemniku prenese ponoči v času od 00:00 do 06:00.

Izvedbo na terenu se izvede prednostno in to na isti delovni dan, kot je bil izveden prenos številke. Izvedbo se takoj, najpozneje pa naslednji delovni dan, vnese v vse sisteme za informacijsko podporo in s tem zagotovi pravilen prikaz statusa številke v sistemih informacijske podpore.

Operaterju dajalcu številke je potrebno do 5. v naslednjem mesecu posredovati vse originalne dokumente Zahtevkov za prenos naročniške številke s prilogami, ki so bili posredovani v fazi naročila preko faksa ali elektronske pošte.



Slika 8: Prikaz izvedbe prenosa telefonske številke

6.4 Uporaba metodologije na primeru prenosa telefonskih številk

6.4.1 Testiranje interakcij

Namen je opredeliti vse primere medsebojne komunikacije, do katerih pride pri celotnem procesu prenosa telefonske številke. Ker je teh primerov veliko in vseh ne morem zajeti v diplomskem delu, bom spodaj predstavila interakcije samo pri primeru uporabe »Preveritev možnosti prenosa« in primer izpolnjene specifikacije za enega izmed testnih primerov.

Izvedba testov in analiza rezultatov:

Test interakcij smo izvedli z navadnimi klici storitev. Beležili smo storitve in njihove operacije, ki so se prožile, sporočila, ki so se prenašala ter ali zaporedje operacij in prenesenih sporočil ustreza zaporedju operacij, zapisanim v testni specifikaciji.

Zajeti so bili vsi možni primeri medsebojne komunikacije in klicev storitev v procesu prenosa telefonske številke. Testi so bili po večini uspešno izvedeni. Do neuspehov je prišlo samo zaradi motenj na omrežju enega od operaterjev.

Na osnovi uspešnosti testiranja lahko trdimo, da če so opisi storitev ustrezni in če jih upoštevamo pri razvoju, potem v tej fazi testiranja ne sme priti do večjih napak. Če WSDL opisi storitev ne bi natančno podajali kakšne podatke WS prejema, bi lahko prišlo do semantičnih napak. Na primer: telefonska številka ni v pravem formatu (npr. 12345678 namesto 012345678, telefonska številka je prekratka/predolga (npr. 012345, 01234567899), …

Testirali smo relativno enostaven proces, ki ima le nekaj možnih vej. Posamezni koraki so jasno določeni in dokumentirani, zato je bilo v tej fazi razvoja možno razviti učinkovit sistem. Kot posledica tega je bilo testiranje uspešno in učinkovito.

Problem je nastal, ker mora pri testiranju sodelovati več deležnikov. Sodelujejo trije deležniki: 2x operater (eden v vlogi dajalca številke, drugi v vlogi prejemnika številke) in CBP, ki služi za izmenjavo podatkov med operaterji in nadzira časovni potek preverjanja, naročanja in izvedbe prenosljivosti. Hkrati omogoča APEKu nadzor in sankcioniranje izvajanja.

6.4.2 Testiranje skladnosti s smernicami WS

Vsi profili WS-I so javno objavljeni na internetni strani organizacije Web Service Interoperability Organization [19].

Za preverjanje interoperabilnosti smo uporabili pristop preverjanja ustreznosti s standardi. Vse testirane storitve so ustrezale izbranim profilom WS-I.

Izvedba testov in analiza rezultatov:

Ker je smernic, ki smo jih preverjali v tej fazi testiranja, veliko in vseh ne morem zajeti v diplomskem delu, bi rada navedla vsaj nekatere izmed njih. Vse se nanašajo na pošiljanje SOAP sporočil in skladnost z Basic Profile 2.0 (BP).

- BP zahteva, da XML procesorji podpirajo kodiranje znakov UTF-8 in UTF-16. Za izboljšano interoperabilnost se mora za določanje pravilnega kodiranja znakov sporočila uporabiti parameter *charset* polja *Content-Type* HTTP glave.
- Preverimo, ali struktura ovojnice od storitve prejetega SOAP sporočila ustreza strukturi določeni v SOAP 1.2

- Preverimo, ali ima ovojnica SOAP sporočila natanko nič ali en podelement elementa telesa sporočila (*soap:Body*).
- Preverimo, če je podelement elementa *soap:Body* kvalificiran z imenskim prostorom.
- Preverimo, če SOAP ovojnica ne vsebuje XML ukazov procesiranja.
- Sporočilo mora biti poslano z uporabo ene od različic HTTP protokola: HTTP /1.1 ali HTTP /1.0. Priporočljivo je, da se uporablja različica HTTP /1.1.

V tem sklopu testiranja smo preveriti še, ali so opisi storitev v skladu z izbranim oziroma uporabljenim profilom. V našem primeru je to Basic Profile 2.0 (BP).

Spet podajam le nekaj izmed smernic, na katere smo bili pozorni:

- Preverimo, če XML stavek v uvoženi XML shemi uporablja kodiranje znakov UTF-8 ali UTF-16.
- Preverimo, če deklarativni XML stavek v WSDL dokumentu uporablja kodiranje UTF-8 ali UTF-16.
- V opisu storitve se morajo elementi *wsdl:types* nahajati pred vsemi drugimi elementi WSDL imenskega prostora, z izjemo elementov *wsdl:documentation* in *wsdl:import*.
- Preverimo, če se povezava tipa *rpc-literal* v opisu storitve v podelementih tipa *wsoap12:body* nanaša le na elemente tipa *wsdl:part*, ki so bili opredeljeni z uporabo atributa *type.*
- Preverimo, če se povezava tipa *document-literal* v opisu storitve v podelementih tipa *wsoap12:body* nanaša le na elemente tipa *wsdl:part*, ki so bili opredeljeni z uporabo atributa *element.*
- Preverimo, če se v opisu storitve *wsdl:message*, ki vsebuje *wsdl:part* z atributom *element*, v tem atributu nanaša na globalno deklariran element.
- Preverimo, če je vrstni red elementov v *soap:body* ovojnice SOAP sporočila enak, kot je vrstni red *wsdl:parts* v *wsdl:message*, ki ga opisuje za vsakega izmed elementov *wsdl:part*, vezanih na pripadajoči element *wsoap12:body* ovojnice.
- Preverimo, če v opisu storitve posamezen element tipa *wsdl:portType* uporablja operacije z različnimi vrednostmi atributov *names* (vsak atribut *names* ima svojo enolično vrednost).

Test skladnosti smo izvedli z navadnimi klici storitev, tako kot v predhodni fazi testiranja. Zaporedje operacij in prenesenih sporočil ustreza zaporedju operacij, zapisanimi v testni specifikaciji.

Za testne primere, ki se nanašajo le na opise storitev, smo preverili ali opis zadošča pogojem v testni specifikaciji.

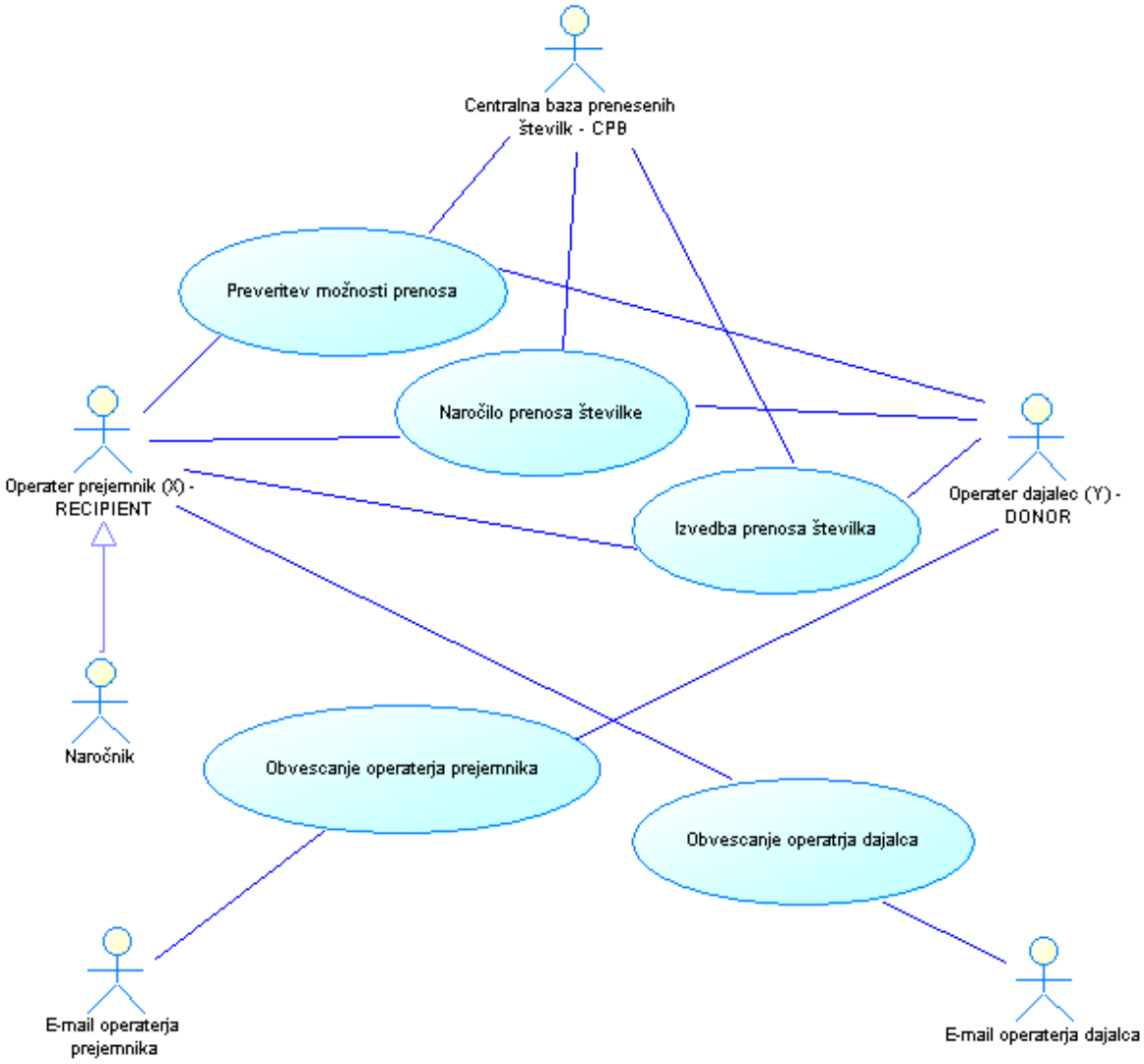
V opisu sem omenila, da gre pri tem koraku za testiranje zunanjih storitev in če je test neuspešen, je možnih več scenarijev. Med drugimi tudi ta, ki pravi: »Če nimamo vpliva na spremembo storitve, pa se moramo odločiti, ali bomo storitev kljub temu uporabljali, s tem da bomo upoštevali neskladje, ali pa bomo uporabili drugo zunanjo storitev«. V primeru prenosa telefonskih številk te možnosti ni, ker obstaja le ena centralna baza podatkov, od katere moramo dobiti ustrezen odgovor na našo preveritev, ali pa kateri moramo podati ustrezen odgovor na preveritev drugega operaterja. Če gre za napako pri ponudniku storitve, ga moramo nanjo opomniti, če pa gre za našo napako, moramo narediti vse, da se jo odpravi.

6.4.3 Funkcionalno testiranje

S funkcionalnim testiranjem preverimo, če sistem, ki ga razvijamo, ustreza funkcionalnim zahtevam, ki so običajno opredeljene s primeri uporabe.

Primere uporabe smo predstavili s pomočjo diagrama primerov uporabe – slika 9. Vsak primer uporabe vsebuje glavno zaporedje korakov in možna alternativna zaporedja korakov. Pri testiranju je potrebno pokriti čim več poti (najboljše vse), tako da je dobro za vsak scenarij oziroma za vsako zaporedje korakov kreirati vsaj en testni primer.

6.4.3.1 Izdelava diagrama primerov uporabe



Slika 9: Diagram primerov uporabe

6.4.3.2 Izdelava primerov uporabe

PU1: Preveritev možnosti prenosa

Kratek opis

Primer uporabe omogoča operaterju, prejemniku številke, preveritev prenosa želene številke. Vključuje identifikacijo naročnika s številko in računom.

Osnovni tok

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Operater v roku 15 minut dobi pozitiven odgovor na poizvedbo.
5. Sistem prikaže opcijo za prenos.

Alternativni tokovi

a) telefonska številka ni v pravem formatu

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Sistem obvesti operaterja, da številka ni v pravem formatu. Sistem ponovno prikaže formular za vnos preveritve.
5. Alternativni tok se nadaljuje po osnovnem toku pri št.2.

b) telefonska številka je že v fazi prenosa

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Operater v roku 15 minut dobi negativen odgovor na poizvedbo.
5. Sistem izpiše razlog negativne preveritve: »Številka že v prenosu pri operaterju XY«.

c) telefonska številka izključena

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Operater v roku 15 minut dobi negativen odgovor na poizvedbo.
5. Sistem izpiše razlog negativne preveritve: »Številka je izključena«.

d) račun je neveljaven

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Operater v roku 15 minut dobi negativen odgovor na poizvedbo.
5. Sistem izpiše razlog negativne preveritve: »Račun je neveljaven«.

e) številka in račun se ne ujemata

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Operater v roku 15 minut dobi negativen odgovor na poizvedbo.
5. Sistem izpiše razlog negativne preveritve: »Številka in račun se ne ujemata«.

Predpogoji

Želja stranke po prenosu številke in račun, ki ni starejši od 3 mesecev.

Popogoji

Če se primer uporabe uspešno zaključi, se status številke spremeni na »V prenosu«. Če se primer uporabe ne zaključi uspešno, se stanje v sistemu ne spremeni.

PU2: Naročilo prenosa številke

Kratek opis

Primer uporabe omogoča operaterju, prejemniku številke, naročilo prenosa želene številke.

Osnovni tok

1. Sistem prikaže formular za vnos naročila prenosa.
2. Operater prejemnik vpiše telefonsko številko, številko računa, željo o odpovedi ali ohranitvi naročniškega razmerja in odda naročilo.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 3 ur.
4. Operater dajalec mora obvestiti operaterja prejemnika, da je naročilo prejeto.
5. Operater prejemnik v roku 30 minut posreduje operaterju dajalcu številke potrebne dokumente (faks, e-pošta).
6. Operater dajalec ima 3 ure časa, da preveri posredovane dokumente in obvesti operaterja prejemnika o uspešnosti naročila prenosa.
7. Operater prejemnik v dogovorjenem času dobi pozitiven odgovor na naročilo.

Alternativni tokovi

a) operater prejemnik po prejetem obvestilu ne pošlje potrebnih dokumentov

1. Sistem prikaže formular za vnos naročila.
2. Operater prejemnik vpiše telefonsko številko, številko računa, željo o odpovedi ali ohranitvi naročniškega razmerja in odda naročilo.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 3 ur.
4. Operater dajalec mora obvestiti operaterja prejemnika, da je naročilo prejeto.
5. Operater dajalec v roku 30 minut ne prejme potrebnih dokumentov od operaterja prejemnika.
6. Operater dajalec zavrne prenos: »Prenos zavrnjen – ni bilo posredovanjih ustreznih dokumentov«.

b) dokumenti so nečitljivi

1. Sistem prikaže formular za vnos naročila.
2. Operater prejemnik vpiše telefonsko številko, številko računa, željo o odpovedi ali ohranitvi naročniškega razmerja in odda naročilo.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 3 ur.

4. Operater dajalec mora obvestiti operaterja prejemnika, da je naročilo prejeto.
5. Operater prejemnik v roku 30 minut posreduje operaterju dajalcu številke potrebne dokumente.
6. Operater dajalec zavrne prenos: »Prenos zavrnjen – dokumenti so nečitljivi«.

c) zahteva ne vsebuje vseh podatkov naročnika

1. Sistem prikaže formular za vnos naročila.
2. Operater prejemnik vpiše telefonsko številko, številko računa, željo o odpovedi ali ohranitvi naročniškega razmerja in odda naročilo.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 3 ur.
4. Operater dajalec mora obvestiti operaterja prejemnika, da je naročilo prejeto.
5. Operater prejemnik v roku 30 minut posreduje operaterju dajalcu številke potrebne dokumente.
6. Operater dajalec zavrne prenos: »Prenos zavrnjen – pomanjkljivi podatki naročnika«.

d) podatki o prekinitvi naročniškega razmerja na zahtevi za prenos in v centralni bazi podatkov se ne ujemajo

1. Sistem prikaže formular za vnos naročila.
2. Operater prejemnik vpiše telefonsko številko, številko računa, željo o odpovedi ali ohranitvi naročniškega razmerja in odda naročilo.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 3 ur.
4. Operater dajalec mora obvestiti operaterja prejemnika, da je naročilo prejeto.
5. Operater prejemnik v roku 30 minut posreduje operaterju dajalcu številke potrebne dokumente.
6. Operater dajalec ugotovi da se podatki o zahtevi za prekinitev naročniškega razmerja na zahtevi in v CBP ne ujemajo. Operater dajalec zavrne prenos.
7. Operater dajalec zavrne prenos: »Prenos zavrnjen – podatki o prekinitvi naročniškega razmerja se ne ujemajo«.

Predpogoji

Naročilo je možno le, če je bila predhodno opravljena preveritev prenosa telefonske številke ter odgovor operaterja dajalca pozitiven. Naročnik, ki zahteva prenos, mora izpolniti obrazec za prenos naročniške številke.

Popogoji

Če se primer uporabe uspešno zaključi, se status številke ne spremeni in ostaja »V prenosu«. Če se primer uporabe ne zaključi uspešno, se status številke spremeni iz »V prenosu« na »Vključena pri operaterju Y«.

PU3: Izvedba prenosa številke

Kratek opis

Primer uporabe omogoča izvedbo prenosa številke.

Osnovni tok

1. Sistem na dan dogovorjenega prenosa iz CBP pridobi 'todo' listo za tekoči dan.
2. Operater dajalec izključi telefonsko številko.
3. Operater dajalec obvesti operaterja prejemnika, da je številko izključil.
4. Sistem omogoči vklop telefonske številke šele, ko je s strani operaterja dajalca izključena.
5. Operater prejemnik še isti dan vključi telefonsko številko.
6. Operater prejemnik do petega dne v naslednjem mesecu posreduje originalne dokumente za prenos operaterju dajalcu.

Alternativni tokovi

a) na 'todo' listi ni želene številke

1. Sistem na dan dogovorjenega prenosa iz CBP pridobi 'todo' listo za tekoči dan.
2. Operater dajalec ne more izključiti želene telefonske številke, ker je ni na listi.
3. Operater dajalec zavrne prenos: »Prenos zavrnjen – številke ni na 'todo' listi«.

b) operater številke dajalec na dan prenosa ne izključi v dogovorjenem času 00:00 - 06:00

1. Sistem na dan dogovorjenega prenosa iz CBP pridobi 'todo' listo za tekoči dan.
2. Operater dajalec ne izključi telefonske številke v dogovorjenem času.
3. Telefonska številka se mora prihodnji dan ponovno pojaviti na 'todo' listi.

c) operater dajalec ne obvesti operaterja prejemnika, da je številko izključil

1. Sistem na dan dogovorjenega prenosa iz CBP pridobi 'todo' listo za tekoči dan.
2. Operater dajalec izključi telefonsko številko.

3. Operater prejemnik ne dobi obvestila, da je številka izključena.
4. Številka ostane v statusu »V prenosu«.
5. Operater prejemnik lahko številko kadarkoli vključi.

Predpogoji

Izvedba prenosa telefonske številke je možna le, če je bilo predhodno naročilo uspešno opravljeno ter če je bil dogovorjen datum za prenos.

Popogoji

Če se primer uporabe uspešno zaključi, se status številke spremeni na »Vključena pri operaterju X«.

Izvedba testov in analiza rezultatov:

Zaradi obsežnosti sistema, smo pozabili na določene veje procesa. Primere uporabe je bilo potrebno dopolniti in napisati pripadajoče testne primere. Spodaj podajam alternativni tok d) za primer uporabe PU3.

PU3: Izvedba prenosa številke

d) sistem kljub uspešni izključitvi na dajalčevi strani ne dovoli vklopa številke.

1. Sistem na dan dogovorjenega prenosa iz CBP pridobi 'todo' listo za tekoči dan.
2. Operater dajalec izključi telefonsko številko.
3. Operater dajalec obvesti operaterja prejemnika, da je številko izključil.
4. Sistem omogoči vklop telefonske številke šele, ko je s strani operaterja dajalca izključena.
5. Operater prejemnik še isti dan vključi telefonsko številko.

Pri kreiranju primerov uporabe smo pozabili na alternativne tokove, ki bi ponazarjali testne primere glede na časovno omejitev. Na primer: operater prejemnik v dogovorjenem času ne dobi odgovora na preveritev, itd.

K obstoječim primerov uporabe smo dodali nove alternativne tokove. Spodaj podajam alternativne tokove f) in g) za primer uporabe PU1 in e) za primer uporabe PU2.

PU1: Preveritev možnosti prenosa

f) odgovora ni v roku 15 min

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 15 minut.
4. Operater prejemnik v roku 15 minut ne dobi odgovora.
5. Sistem izpiše sporočilo: »Preveritev je potekla«.

g) preveritev je oddana izven delovnega časa

1. Sistem prikaže formular za vnos preveritve.
2. Operater prejemnik vpiše telefonsko številko, številko računa in odda preveritev.
3. Sistem obvesti operaterja, da poizvedbe ne more oddati izven delovnega časa.

PU2: Naročilo prenosa številke

e) operater prejemnik po prejetem obvestilu ne pošlje potrebnih dokumentov v roku 30 min

1. Sistem prikaže formular za vnos naročila prenosa.
2. Operater prejemnik vpiše telefonsko številko, številko računa, željo o odpovedi ali ohranitvi naročniškega razmerja in odda naročilo.
3. Sistem obvesti operaterja, da bo dobil odgovor v roku 3 ur.
4. Operater dajalec mora obvestiti operaterja prejemnika, da je naročilo prejeto.
5. Operater dajalec v roku 30 minut ne pridobi potrebnih dokumentov od prejemnika številke.
6. Operater dajalec zavrne prenos.

Pri testiranju negativnih vej smo naleteli na težave pri naboru testnih podatkov (ne poznamo številk, ne vemo, katere številke so vključene in katere ne, kateri računi so pravilni, kateri ne).

Težavo nam je pomagal odpraviti upravljalec centralne baze, ki nam je z dovoljenjem drugega operaterja posredoval ustrezne številke in račune, ki smo jih lahko uporabili pri testu.

Težave smo nastale tudi pri testiranju spletnih storitev drugega ponudnika, saj se je pogosto zgodilo, da spletne storitve niso bile dostopne. Ponudnik spletne storitve je bil o tem obveščen in dostopnost svojih storitev izboljšal tako, da je le te prestavil na zmogljivejši strežnik.

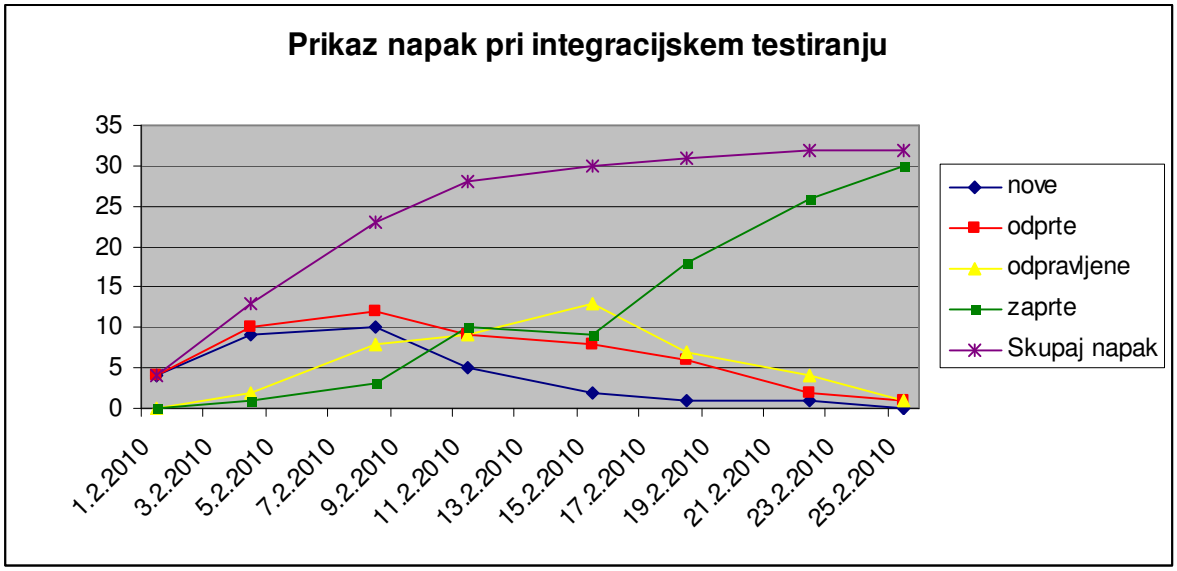
6.5 Analiza rezultatov testiranja

Integracijsko testiranje smo izvajali 2x tedensko po vsaki izdaji nove verzije procesov. Opravljenih je bilo 8 testnih ciklov. Vsak testni primer smo izvedli vsaj enkrat tekom meseca. Testne primere, na katerih so bile odkrite napake, smo izvajali večkrat. Testnega primera nismo ponovno izvedli, dokler ni bila odpravljena napaka, ki je bila odkrita na njem. Napake z večjo prioriteto, ter napake prijavljene pri scenarijih z večjo frekvenco pojavljanja smo odpravljali prioritetno.

	1.feb	4.feb	8.feb	11.feb	15.feb	18.feb	22.feb	25.feb
nove	4	9	10	5	2	1	1	0
odprte	4	10	12	9	8	6	2	1
odpravljene	0	2	8	9	13	7	4	1
zaprte	0	1	3	10	9	18	26	30
Skupaj napak	4	13	23	28	30	31	32	32

Tabela 6: Pregled števila napak

Podajam še grafični prikaz.



Slika 10: Prikaz napak pri integracijskem testiranju

Pri enomesečnem testiranju smo odkrili 32 napak. Število vseh napak je na začetku testiranja hitro naraščalo, vendar se je ta rast proti koncu testiranja zmanjšala. Na grafu tudi vidimo, da je število novo odprtih napak v 2/3 testnega obdobja začelo padati. V zadnjih nekaj testnih ciklih smo odkrili le 1 ali 2 novi napaki.

Iz grafa vidimo, da število zaprtih napak ni nujno vedno naraščajoče. Napaka, ki je že bila odpravljena, ponovno testirana in zaprta, se lahko pojavi ponovno v naslednjem testnem ciklu.

Izračunajmo še testno pokritost sistema in stopnjo odpravljenih napak za vsak testni cikel:

	1.feb	4.feb	8.feb	11.feb	15.feb	18.feb	22.feb	25.feb
št. izvedenih testov	130	102	158	80	40	127	90	145
št. vseh testov	158	158	158	158	158	158	158	158
testna pokritost sistema	82,28%	64,56%	100,00%	50,63%	25,32%	80,38%	56,96%	91,77%

Tabela 7: Testna pokritost sistema

Slika 11: Prikaz testne pokritosti sistema

Pri spodnjem izračunu smo med odpravljenimi napakami upoštevali seštevek odpravljenih in zaprtih napak.

	1.feb	4.feb	8.feb	11.feb	15.feb	18.feb	22.feb	25.feb
št. vseh napak	4	13	23	28	30	31	32	32
št. odpravljenih napak	0	2	8	9	13	7	4	1
št. zaprtih napak	0	1	3	10	9	18	26	30
delež odpravljenih napak	0,00%	23,08%	47,83%	67,86%	73,33%	80,65%	93,75%	96,88%

Tabela 8: Delež odpravljenih napak

Slika 12: Prikaz deleža odpravljenih napak

Delež odpravljenih napak s časom raste in se približuje meji 100%, ki pa je ne doseže.

7 ZAKLJUČKI

Cilj diplomskega dela je bil prikazati integracijsko testiranje v SOA sistemih.

V diplomskem delu sem predstavila testiranje kot del razvojnega procesa ter opisala prednosti in slabosti SOA sistemov. Predstavila sem metodologijo, s pomočjo katere izvajamo integracijske teste in le to predstavila na realnem primeru.

Sistemi, ki se v današnjih časih vedno bolj poslužujejo SOA arhitekture, so povečini veliki, povezujejo večje število poslovnih procesov in so prilagodljivi neprestanim spremembam. Gradnjo tako velikih sistemov mora spremljati testiranje, ki se izvaja sistematično.

Prednost, ki jo ima uporabljena metodologija, je ravno ta sistematičnost, o kateri govorim. Le tako lahko pretestiramo celoten sistem. Testni scenariji in testni primeri morajo biti dobro definirani, s točno znanimi vhodi in izhodi.

Ob testiranju skladnosti smernic WS bi bilo dobro razmisliti o avtomatizaciji testiranja. Le ta bi nam prihranila ogromno časa in denarja, obseg testiranja pa bi lahko povečali. Testna ekipa bi bila tako manj obremenjena s ponavljajočimi monotonimi testi smernic in bi lahko več časa posvetila testiranju funkcionalnosti, pri katerih smo tudi sami naleteli na največ problemov.

Največ težav nam je povzročalo testiranje zunanje storitve. Vzrok za to je bila pogosta nedosegljivost spletne storitve in pa pomanjkljive informacije o številkah drugih operaterjev. Nekaterim testnim primerov smo tako le s težka določili vhodne podatke. Težave smo imeli predvsem pri pokrivanju negativnih vej procesa. Vedeli smo, kakšen rezultat želimo, vhodnih podatkov pa nismo mogli določiti drugače kot s poskušanjem. Ravno zaradi teh poskušanj je testna stopnja pokritosti sistema znotraj nekaterih ciklov zelo nizka. Sredi testnega meseca je celo padla pod 30%.

Stopnja odpravljenih napak tekom testiranja raste in se približuje 100%, vendar te meje zaradi omejenega časa testiranj ne doseže. Kratki časovni roki za izdelavo sistema ali programske opreme so v precejšnji meri najpogostejši razlog za slabo kvaliteto sistema. Testiranju se pogosto posveti premalo časa in resursov. Z vsako novo verzijo se praviloma veča tudi obseg sistema in številko testnih primerov. Zaradi kratkih časovnih rokov pa, žal, na premnoge od teh primerov naletimo šele v produkciji.

SEZNAM SLIK

SEZNAM TABEL

SEZNAM ENAČB

8 VIRI in LITERATURA

[1] Software Testing Fundamentals – Integration testing. Dostopno na: http://www.softwaretestingfundamentals.com/2009/05/integration-testing.html

[2] Drake T., Testing Software Based Systems: The Final Frontier. Dostopno na: http://www.softwaretechnews.com/

[3] Solina F., Projektno vodenje razvoja programske opreme, 1.izdaja. Ljubljana: Fakulteta za računalništvo in informatiko, 1997.

[4] Myers Glenford J., The art of software testing. New York: John Wiley ans Sons, 1997.

[5] Zorko Samo R., Lovci na žuželke. Moj mikro, marec 2004.

[6] Interna dokumentacija Telekoma Slovenije

[7] Perry, William E., Effective Methods for Software Testing, 2nd Edition. New York: Joh Willy & Sons, Inc, 2000.

[8] Hutcheston M.L., Software Testing Fundamentals: Methods and Matrics. Indianapolis: Wiley Publishing, 2003.

[9] Tian J., Software Quality Engineering. New York: John Wiley & Sons, Inc. 2005.

[10] Kung, David C., Testing object-oriented software. Los Alamitos, IEEE Computer Society, 1998.

[11] Elfriede D., Effective Software Testing, Addison-Wesley, 2002.

[12] Torry Harris: SOA Test Methodology. Dostopno na: http://www.thbs.com/pdfs/SOA_Test_Methodology.pdf

[13] Jurič B. Matjaž, Mathew B. & Sarang P., Bussiness Process Execution Language for Web Services [Jezik za orkestracijo poslovnih procesov]. Birmingham: Packt Publishing Ltd, 2004.

[14] Gačnik, M., Arhitektura spletnih storitev, Usmeritve za načrtovanje. Ljubljana: Microsoft d.o.o., Ljubljana, 2003.

[15] Tyler, D., SOA – Is it right for you? Part I in II. Dostopno na: http://it.toolbox.com/blogs/soa-business/soa-is-it-right-for-you-part-i-5834 in http://it.toolbox.com/blogs/soa-business/soa-is-it-right-for-you-part-ii-5928

[16] Wikipedia - Integration testing. Dostopno na: http://en.wikipedia.org/wiki/Integration_testing

[17] Beizer B., Software Testing Techniques, New York: The Coriolis Group, 1990.

[18] Splošni akta o prenosljivosti številk. Dostopno na: http://www.apek.si/sl/splosni_akt_o_spremembah_in_dopolnitvah_splosnega_akta_o_prenosljivosti_stevilk

[19] Web Services Interoperability Organization. Dostopno na: http://www.ws-i.org/

[20] Bertolino A.: Approaches to testing service-oriented software systems, 2009. Dostopno na: http://portal.acm.org/citation.cfm?id=1596473.1596475

[21] Bartolini C., Bertolino A., Elbaum S., Marchetti E.: Whitening SOA testing, 2009. Dostopno na: http://portal.acm.org/citation.cfm?id=1595696.1595721