UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

UPRAVLJANJE KONFIGURACIJ PROGRAMSKIH PROIZVODOV V INFORMACIJSKO-TEHNOLOŠKEM PODJETJU

Ljubljana, november 2023

DAVID NOVAK

IZJAVA O AVTORSTVU

Podpisani David Novak, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela z naslovom Upravljanje konfiguracij programskih proizvodov v informacijsko-tehnološkem podjetju, pripravljenega v sodelovanju s svetovalcem red. prof. dr. Mirom Gradišarjem

I Z J A V L J A M

1. da sem predloženo delo pripravil samostojno;

2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;

3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;

4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;

5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;

6. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil;

7. da sem pri pripravi predloženega dela ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;

8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;

9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;

10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

11. da sem preveril verodostojnost informacij, ki izhajajo iz zapisov na podlagi uporabe orodij umetne inteligence.

V Ljubljani, dne _____ Podpis študenta: _____

KAZALO

KAZALO TABEL

KAZALO SLIK

KAZALO PRILOG

SEZNAM KRATIC

angl. – angleško
BPMN – (angl. Business Process Model and Notation); Standardni način za grafično predstavitev poslovnih procesov
Ce – Stroški izobraževanja osebja
CI – (angl. Configuration Item); Konfiguracijski element
Cic – Stroški implementacije in konfiguracije
CMP – (angl. Configuration Management Plan); Načrt upravljanja konfiguracij
CNPV – (angl. Cumulative Net Present Value); Kumulativna neto sedanja vrednost
Co – Stroški orodja
CR – (angl. Change Request); Zahteva za spremembo
CRM – (angl. Customer Relationship Management); Upravljanje odnosov s strankami
Ct – Tekoči stroški
DBMS – (angl. DataBase Management System); Sistem za upravljanje baz podatkov
DF – Diskontni faktor
ERP – (angl. Enterprise Resource Planning); Načrtovanje virov podjetja
HRM – (angl. Human Resource Management); Upravljanje s človeškimi viri
KE – Konfiguracijski element
NPV – (angl. Net Present Value); Neto sedanja vrednost
P2P – (angl. Peer to Peer); Distribuiran komunikacijski model
PVCt – Sedanja vrednost tekočih stroškov
PVTS – Sedanja vrednost skupnih prihrankov
PZS – Portal za stranke
Sa – Prihranki zaradi avtomatizacije priprave namestitvenih paketov
SCM – (angl. Software Configuration Management); Upravljanje konfiguracij programskega proizvoda
Spvz – Prihranki pri stroških popravil in vzdrževanja
SQL – (angl. Structured Query Language); Strukturiran poizvedbeni jezik
Su – Prihranki pri učinkovitosti razvoja
TCu – Skupni stroški uvedbe
TS – Skupni prihranki
UKPP – Upravljanje konfiguracij programskega proizvoda

1 UVOD

Sodobno okolje za razvoj programskih proizvodov je lahko zelo kompleksno in prefinjeno. Včasih se za izdelavo posameznega programskega proizvoda združi tudi več različnih podjetij, ki se nahajajo v različnih delih sveta. Podobno se tudi znotraj istega podjetja lahko razvija več manjših podsistemov, ki združeni v celoto na koncu predstavljajo enovit programski proizvod. Upravljanje takih projektov razvoja programskih proizvodov brez ustreznih orodij lahko povzroči slabo kakovost, nenadzorovane stroške in časovne zamike, ki lahko privedejo tudi do končnega neuspeha projekta.

Upravljanje konfiguracij programskega proizvoda (v nadaljevanju UKPP) je idealna rešitev za obvladovanje kaosa in zmede pri razvoju programske opreme, njen primarni cilj je nadzor nad razvojnim procesom (Leon, 2015). Govorimo lahko tudi o UKPP kot organizacijskem okvirju za upravljanje razvoja programskih proizvodov v vseh fazah (Keyes, 2004).

Večina avtorjev ugotavlja, da je vzpostavitev sistema za UKPP v veliki meri odvisna od specifike posameznega razvojnega okolja, v katerem nastaja programski proizvod. Za uspešno vzpostavitev sistema za UKPP, kar nam bo omogočalo učinkovito delati kot ekipa, moramo razumeti, kako sodelujejo vsi deli razvojnega okolja med seboj in kako se tehnike UKPP prilegajo širši sliki prizadevanj za razvoj programske opreme (Berczuk in drugi, 2003).

Obstajajo skupna teoretična izhodišča, ki veljajo za vsa okolja in ki se kažejo predvsem v obliki uveljavljenih pojmov in razlag. Kot primer lahko izpostavimo npr. identifikacijo osnovnih elementov sistema za UKPP. Osnovna predpostavka upravljanja konfiguracije je razčlenitev velikih predmetov na najmanjše še smiselne koščke in nato zabeležiti spremembe in razvojno okolje, v katerem je posamezen košček izdelan (Kenefick, 2008). Upravljanje konfiguracije programskih proizvodov zajema identifikacijo, beleženje in poročanje o IT komponentah, vključno z njihovimi različicami, sestavnimi deli in odnosi (Blokdijk in Menken, 2008).

Vendar je potrebno te elemente v danem razvojnem okolju prepoznati v takšni obliki, kot dejansko nastopajo. Npr. izvorna koda nekega vnosnega obrazca je lahko v nekem razvojnem okolju v binarni obliki, spet v drugem je lahko zapisana v tekstualni obliki ali pa so npr. uporabniška navodila v obliki Word-ovih datotek, v drugem okolju pa v obliki HTML datotek, ki tečejo v spletni aplikaciji, v nekem tretjem okolju pa so lahko tudi kombinacija obojega ali morda nečesa tretjega.

A kljub ugotovljeni specifičnosti vsakega posameznega okolja je, glede na uporabljeno razvojno tehnologijo, možno razvojna okolja tudi združevati na nek skupni imenovalec: npr. razvojna okolja, kjer teče razvoj Java aplikacij, imajo zagotovo enake vrste izvornih datotek,

prav tako lahko to trdimo za razvojna okolja, npr. Oracle Forms, Oracle Reports in Oracle Apex, ker tehnologija sama do neke mere določa del osnovnih elementov UKPP.

Na voljo je kar nekaj literature s področja UKPP, a je večinoma izpod peres tujih avtorjev. Večina literature, ki je na voljo, opisuje podobno problematiko in tudi pristopi so bolj ali manj podobni. V glavnem avtorji izhajajo iz smernic, ki jih dajejo standardi ali metodologije, kot je npr. ITIL, a skoraj vsi so si edini, da je potrebno UKPP v okolje, kjer ga uvajamo, vpeljati z veliko mero previdnosti in z upoštevanjem posebnosti. Problematika uvedbe sistema za UKPP se deli na razumevanje težav, ki jih poskušamo rešiti (Kenefick, 2008), in izbiro ter uvedbo rešitve, ki upošteva značilnosti danega okolja.

Sodoben razvoj programskih proizvodov si je v organizaciji težko predstavljati brez upravljanja konfiguracij programske opreme. Pa vendar je pojem upravljanja konfiguracij precej na široko opredeljen in lahko zajema v različnih okoljih različne elemente in podsisteme.

Namen raziskave je spoznati teoretična izhodišča s področja sistemov za upravljanje konfiguracij in nato analizirati obstoječe stanje ter na podlagi ugotovljenih dejstev in posebnosti izbranega razvojnega okolja omogočiti izbiro najprimernejšega orodja za verzioniranje izvornih datotek ter razvoj lastne aplikacije z repozitorijem za shranjevanje metapodatkov za celostno implementacijo novega sistema za UKPP.

Kot cilj si v naši raziskavi postavljamo pripravo načrta za vzpostavitev novega sistema za upravljanje konfiguracij programskih proizvodov, razvitih v razvojnem okolju Oracle Forms, Oracle Reports in Oracle Apex, ki bo omogočal avtomatizacijo nekaterih procesov in obvladovano izdajo različic programskega proizvoda kot celote.

V pričujočem magistrskem delu si zastavljamo vprašanje, ali lahko s takšnim pristopom pričakujemo naslednje konkretne izboljšave:

- enostavnejše uveljavljanje sprememb, tako v obliki odpravljanja skritih napak kot v obliki dopolnitev in razširitev programskih proizvodov z novimi funkcionalnostmi, kar bo povzročilo hitrejše razvojne cikle in povečalo naklonjenost (tako razvijalcev kot tudi uporabnikov) pogostejšemu nameščanju novih različic;
- povečanje zanesljivosti procesa namestitve posodobljenih različic programske opreme za zmanjšanje napak in s tem dvig zaupanja v sam postopek namestitve programske opreme;
- zaradi definirane politike nameščanja in jasne določitve postopkov, kdo, kaj in kam lahko namešča nove različice, se bo dvignil nivo varnosti;
- z uvedbo avtomatiziranih opravil se bo zmanjšala količina ročnega dela in posledično se bo sprostila zasedenost človeških virov ter hkrati minimizirala možnost človeških napak, ki nastajajo pri ročnem izvajanju opravil.

Teoretični del, ki ga bomo obravnavali v prvem poglavju, bo osredotočen predvsem na sisteme za UKPP, prepoznavanje in konkretizacijo elementov sistema za UKPP in v določanje procesov in postopkov za izvajanje, nadzor, spremljanje stanja programske opreme ter tudi seznanitev z metodami, tehnologijami in orodji, ki omogočajo vse prej navedeno izvajati čim bolj avtomatizirano, standardizirano in s tem tudi bolj zanesljivo.

Praktični primer, ki ga bomo v raziskavi obravnavali v drugem in tretjem poglavju, se nanaša na podjetje, ki za svoje razvojno okolje uporablja Oracle Forms, Oracle Reports in Oracle Apex programska razvojna orodja. Programski proizvod, ki ga izbrano podjetje razvija, je lastni sistem za upravljanje poslovanja podjetja (angl. Enterprise Resource Planning, v nadaljevanju ERP). Zbrana teoretična spoznanja bomo aplicirali skozi praktični primer, pri čemer bomo na podlagi poznavanja metod in dobrih praks, opisanih v literaturi in virih, in nadalje na osnovi analize obstoječega stanja in poznavanja samega podjetja in dejanske problematike poizkušali izdelati načrt za vzpostavitev novega sistema za UKPP, vključno z analizo stroškov in koristi, ki jih prinaša implementacija novega sistema za UKPP.

Magistrska naloga bo temeljila na deduktivnem pristopu, za katerega je značilno, da izhajamo iz obstoječe teorije. To bo tako imenovana aplikativna poslovna raziskava, za katero je značilno, da je usmerjena v reševanje specifičnega problema.

V prvem delu naloge se bomo posvetili raziskavi področja upravljanja konfiguracije programskih izdelkov s pomočjo strokovne literature, člankov ter drugih relevantnih virov. Uporabili bomo tri glavne raziskovalne metode: metodo deskripcije za opisovanje ključnih konceptov, metodo klasifikacije za pojasnitev specifičnih pojmov ter metodo komparacije, s katero bomo primerjali stališča različnih avtorjev, zlasti v zvezi z različnimi pristopi in modeli pri vpeljevanju sistemov UKPP.

Na podlagi člankov in ostalih internetnih virov bomo zbrali objavljene ključne podatke podjetij IT panoge oz. ostalih podjetij, ki so se tega kakorkoli lotila in to javno predstavila, ter aktualne članke in prispevke, ki obravnavajo vzpostavitev UKPP v razvojno programsko okolje, ki smo si ga izbrali.

Drugi del naloge bo bolj dinamičnega značaja, saj bomo teoretična spoznanja aplicirali praktično v predmet preučevanja: to je vpeljavo sistema za UKPP v podjetje, ki kot razvojna orodja uporablja Oracle Forms, Oracle Reports in Oracle Apex. Pri tem bomo uporabili metodo sistemske analize in metodo sinteze rešitve.

Pri pripravi načrta vpeljave sistema za UKPP si bomo pomagali še s podatki in informacijami iz baz podatkov in z ostalo razvojno dokumentacijo podjetja, opravili bomo osebne razgovore in izvedli skupinsko diskusijo s ključnimi zaposlenimi v razvojnem oddelku.

Celota bo zaokrožena s teoretičnimi znanji in praktičnimi spoznanji, pridobljenimi skozi delo v obravnavanem podjetju.

Raziskovalne metode, ki bodo uporabljene, bodo tako kvalitativnega kot kvantitativnega značaja.

Večina podatkov bo kvalitativnih in tudi nekaj kvantitativnih, ki bodo imeli predvsem značaj opisne analize. Uporabili bomo:

a) sekundarne podatke:

- literaturo, ki zajema tako domače kot tuje vire na obravnavanem področju,
- članke in vire v tiskani ali digitalni obliki, pridobljene iz različnih baz podatkov,
- interne vire in dokumentacijo iz preučevanega podjetja,
- informacije, pridobljene iz virov, dostopnih preko spletnih strani konkurenčnih podjetij, svetovalnih firm, fakultet in medijskih objav,
- podatke, pridobljene iz virov ponudnikov sekundarnih podatkov, skupaj s pripadajočimi metapodatki.

b) primarne podatke:

- intervjuje z vodjo razvoja programske opreme preučevanega podjetja in ostalimi vodji posameznih aplikacijskih področij, ki sestavljajo celoto programskega proizvoda, v konkretnem primeru ERP sistema,
- skupinsko diskusijo in izvedeno anketo z ostalimi ključnimi razvijalci in vzdrževalci programske opreme.

2 UPRAVLJANJE KONFIGURACIJ PROGRAMSKIH PROIZVODOV

Računalniki in komunikacijska tehnologija so postali nepogrešljiv del našega vsakdana. Samo nekaj desetletij nazaj je potekala komunikacija predvsem med ljudmi, ena oseba je komunicirala z drugo. Sedaj lahko že nekaj časa opažamo premike, ko se vse bolj uveljavljajo neživi predmeti: knjige niso več samo vir informacij, ampak lahko blagajnam posredujejo informacijo, koliko stanejo; pametne kartice niso le sredstvo plačila, temveč lahko ključavnicam povedo, ali naj se odprejo ali ne; avtomatsko vodena vozila lahko svojemu gostiteljskemu računalniku sporočijo svojo lokacijo v trgovini, kaj prevažajo in kdaj bodo spet na voljo; na internetu pa se ljudje združujejo v živahnih razpravah na forumih in družabnih omrežjih ali igrajo igre, tudi če se nahajajo na različnih koncih sveta.

Glavni dejavnik te digitalne revolucije je računalniška programska oprema. Danes se programska oprema dotika in nadzoruje skoraj vse vidike našega življenja. Programska oprema nas naredi bolj učinkovite in produktivne, ljudem pomaga pri boljšem učenju in poučevanju. Programska oprema naredi entitete, vključno z našimi domovi, bankami in organizacijami, bolj varne, pomaga zdravnikom pri boljšem diagnosticiranju bolezni in

iskanju boljših zdravljenj, nadzoruje kritične aplikacije in opremo. Seznam je tako rekoč neskončen.

S tem, ko postaja programska oprema vse bolj ključna, se hkrati povečuje tudi zahtevnost procesa njenega razvoja. Kjer se uporablja programska oprema za kritične aplikacije in krmiljenje zahtevne opreme in sistemov, ima tudi majhna napaka lahko katastrofalne posledice. Sodobni programski izdelki postajajo vse bolj zapleteni – tako po obsegu, kompleksnosti kot po uporabljenih tehnologijah.

Veliko razvijalcev programske opreme se osredotoča na razvoj programskih izdelkov, ki segajo v različne dele sveta, podpirajo več jezikov in so v različnih velikostih in oblikah (npr. namizne, standardne, profesionalne in podjetniške programske rešitve). Primeri takšnih izdelkov vključujejo operacijske sisteme, urejevalnike besedil in tudi sisteme za upravljanje poslovanja (ERP sistemi), ki omogočajo podporo več jezikom in valutam. Skoraj vsi izdelki aplikacijske programske opreme (kot so urejevalniki besedil, ERP sistemi in celo orodja za UKPP) podpirajo več kot eno strojno ali programsko platformo. Obstajajo na primer sistemi ERP, ki delujejo na velikih računalniških sistemih (angl. Mainframe) in odjemalsko-strežniških sistemih (angl. Client-Server). Prav tako poznamo različne spletne brskalnike za računalnike PC in Mac ter sisteme za upravljanje baz podatkov (angl. DBMS – DataBase Management Systems), ki se prilagajajo različnim operacijskim sistemom, kot so VMS, Unix, Windows in Linux. Tekmovalnost med proizvajalci programske opreme in stalni napredek v tehnologiji spodbujata razvoj dodatnih funkcij ter novih možnosti v njihovih izdelkih, ker želijo ohraniti korak s konkurenco in ostati pomembni na trgu (Leon, 2015).

Pojav in naraščajoča priljubljenost računalništva v oblaku je ustvarila nove modele in prakse razvoja programske opreme. S pomočjo računalništva v oblaku razvijalci lahko delajo na projektih razvoja programske opreme od koderkoli na svetu, pod pogojem, da so opremljeni z računalnikom in imajo dostop do interneta.

Zahteven proces razvijanja aplikacij je ustvaril potrebo po novejših modelih za podporo razvoju programske opreme in podpornih funkcij, kot je UKPP, in zagotavljanju kakovosti programske opreme. Ti pristopi so ključni za premagovanje novih izzivov, ki se nanašajo na hitrost delovanja, neprekinjeno razpoložljivost ter varnost (Gechman, 2019).

Prav tako je smiselno opozoriti, da so uporabniki programske opreme postali bolj ozaveščeni, kar pomeni, da so napake v sistemu zaznane in tudi razkrite v javnosti hitreje kot kadarkoli prej.

Rok, v katerem mora podjetje obvladati nastalo škodo (kar vključuje prepoznavanje izvora programske napake, identifikacijo problematičnih področij, odpravo težave in zagotovitev, da odprava te napake ne povzroči novih težav, izvedbo regresijskega testiranja ter dostavo popravljene različice programske opreme stranki), je bistveno krajši kot prej. Organizacije se morajo izjemno hitro odzvati, da ohranijo svoj ugled.

Iz navedenega lahko ugotovimo, da je današnje okolje za razvoj programske opreme izjemno kompleksno in dinamično, pri čemer so pričakovani časi odziva izredno kratki.

Uspešno vodenje projekta razvoja programske opreme in zagotavljanje visoke kakovosti izdelka brez napak, ob upoštevanju časovnih omejitev in stroškov, predstavljata skoraj nerešljivo nalogo. V tem brezkompromisno konkurenčnem okolju organizacije potrebujejo učinkovit mehanizem, ki ohranja stvari pod nadzorom. Brez tega bi lahko zavladal popoln kaos in zmeda, ki bi ogrozila uspeh izdelkov, projektov ter celo obstoj organizacij. Eno izmed takšnih ključnih orodij je pravilno vzpostavljen sistem za UKPP.

2.1 Opredelitev upravljanja konfiguracij programskih proizvodov

UKPP je proces, ki se uporablja za učinkovitejši razvoj in vzdrževanje programske opreme, kar se doseže z izboljšanjem odgovornosti, ponovljivosti, sledljivosti in usklajevanjem. Vse funkcije UKPP, vključno z identifikacijo konfiguracije elementov, dokumentiranjem značilnosti in nadzorom nad spremembami, so ključnega pomena za zagotavljanje integritete, odgovornosti, prepoznavnosti, ponovljivosti, usklajevanja in sledljivosti projekta ter omogočajo formalni nadzor nad razvojem sistema in projekta (Leon, 2015).

V zadnjih desetletjih je zmogljivost in inovativnost programskih tehnologij daleč presegla našo sposobnost obvladovanja kompleksnosti problemov, ki jih mora obravnavati razvoj programske opreme (Keyes, 2004).

Sposobnost razvoja in distribucije zanesljive in uporabne programske opreme znotraj dogovorjenih rokov in finančnih okvirjev se, na žalost, še naprej izmika številnim organizacijam. Sistem za UKPP zagotavlja načine za upravljanje procesa razvoja programske opreme na strukturiran, urejen in produktiven način (Agarwal in drugi, 2011).

UKPP zajema vsa področja življenjskega cikla programske opreme in vpliva na vse podatke in postopke. Največja korist je torej dosežena, ko se UKPP obravnava kot inženirska disciplina in ne zgolj kot neka oblika umetnosti, kar je, na žalost, še vedno prisotno med mnogimi razvijalci. Kot inženirska disciplina UKPP zagotavlja raven podpore, nadzora in storitve za organizacijo iz različnih vidikov:

- PODPORA – UKPP je podporna funkcija, saj nudi podporo programskim inženirjem in razvijalcem, programom, podjetjem in v številnih situacijah tudi strankam;
- NADZOR – UKPP je nadzorna funkcija, saj nadzoruje specifikacije, dokumente, risbe, zahteve, orodja, programsko opremo in druge sodelujoče elemente;
- STORITEV – UKPP je ponudnik storitev, saj nudi podporo ljudem in nadzoruje podatke. Vloga vodje UKPP je zagotoviti: (1) da je osebje, ki uporablja sistem za UKPP ustrezno usposobljeno in ima potrebna sredstva (proračun in orodja) za učinkovito in uspešno delo; (2) da je ustrezno ravnovesje nadzora in podpore prilagojeno vsakemu programskemu proizvodu posebej, ki je podprt; in (3) da je funkcija UKPP prilagodljiva

tako, da se lahko prilagaja spreminjajočim se potrebam in zahtevam razvijalcev, strank, programskega proizvoda ter podjetja.

Proces UKPP se v zadnjih 20 do 30 letih ni bistveno spremenil, medtem ko se je okolje, v katerem UKPP deluje, močno spremenilo in se bo najverjetneje še spreminjalo. V zadnjih nekaj desetletij smo se preselili iz centraliziranih velikih računalnikov (angl. Mainframe) z uporabo samo nekaj programskih jezikov, kot sta COBOL in FORTRAN, do decentraliziranih, omrežnih, spletnih okolij, ki uporabljajo na tisoče naprav, stotine programskih paketov in desetine programskih jezikov (Keyes, 2004).

Največji vplivi na sistem za UKPP so bili povezani z avtomatiziranimi orodji in knjižničnimi sistemi, ki jih uporablja.

Vse do devetdesetih let prejšnjega stoletja je bil temeljni poudarek UKPP sistemov na nadzoru različic z majhnim številom ponudnikov. V današnjem času obstaja dobesedno na desetine majhnih do velikih UKPP ponudnikov, ki promovirajo različne izdelke: od preprostega nadzora različic do naprednih orodij, namenjenih vzpostavitvi in sledenju celotni programski opremi ter razvojno-proizvodnemu okolju.

Kljub tej raznolikosti ostaja osnovni proces UKPP v svojem bistvu nespremenjen. Sam proces se skorajda ne spreminja, spreminjajo se le elementi, ki jih proces upravlja.

To dejstvo omogoča uporabo UKPP tako v organizacijah z velikimi računalniškimi sistemi kot tudi npr. v tistih, kjer se vse aplikacije izvajajo preko interneta v varovanem omrežnem okolju. Ključ je torej v procesu (Keyes, 2004).

2.2 Zgodovinski razvoj

Zgodovina sistemov UKPP sega nazaj v čase začetkov razvoja programske opreme in informacijskih sistemov. Razviti so bili zaradi potrebe po učinkovitem nadzoru in sledenju sprememb v programski opremi. V zgodnjih letih razvoja programske opreme (1950–1960) so bili UKPP sistemi precej preprosti in so se osredotočali predvsem na shranjevanje in obvladovanje datotek s programsko kodo ter dokumentacije (Moreira, 2004). Kasneje, v 70-ih letih prejšnjega stoletja, se pri projektih razvoja programske opreme za vojaške namene uveljavijo standardi in smernice za upravljanje konfiguracij, kot sta bila na primer ameriška vojaška standarda (Quigley in Robertson, 2015):

- Military standard: Configuration management practices for systems, equipment, munitions, and computer programs (MIL-STD-483) in
- Military standard: Configuration management (MIL-STD-973).

Slednji je obravnaval širše prakse in načela upravljanja konfiguracij in postal »biblija« za upravljanje konfiguracij ter osnova za vse standarde upravljanja konfiguracij (Keyes, 2004).

V 80-ih so se začela pojavljati orodja, ki so omogočala bolj avtomatizirano upravljanje konfiguracij. To so bila pogosto namenska orodja, ki so omogočala sledenje sprememb, obvladovanje verzij in sestavljanje programske kode.

Med ključna in hkrati pionirska orodja iz tega obdobja lahko štejemo (McMillan, 2021):

- SCCS (Source Code Control System) je bilo eno izmed prvih orodij za upravljanje različic programske kode, ki je omogočalo sledenje spremembam v programskih datotekah, vzdrževanje različic ter sestavljanje različnih verzij programske kode. To orodje je postavilo temelje za kasnejše razvojne korake.

- RCS (Revision Control System) kot naslednik SCCS-ja je bil namenjen predvsem upravljanju različic posameznih datotek. Orodje je omogočalo enostavno spremljanje sprememb v posameznih datotekah in vzdrževanje različic.

- CVS (Concurrent Versions System) je bil prvo priljubljeno namensko orodje za upravljanje različic, ki je omogočalo hkratno delo več razvijalcev na isti kodi. CVS je postal ključno orodje za odprtokodne projekte in je omogočil boljšo skupinsko obvladovanje kode.

Kasneje, v 90-ih letih dvajsetega stoletja, se pojavijo orodja z večjim naborom funkcionalnosti za podporo UKPP. ClearCase orodje, razvito v podjetju Rational Software (pozneje prevzeto s strani IBM), je bilo eno izmed prvih širše uveljavljenih komercialnih orodij za upravljanje konfiguracij. ClearCase je ponujal naprednejše funkcije za sledenje sprememb in različic, predvsem za večje in bolj kompleksne projekte. Še eno priljubljeno komercialno orodje za upravljanje konfiguracij iz tega obdobja je bilo Perforce (zdaj Helix Core), ki je izstopalo zaradi svoje učinkovitosti pri obvladovanju velikih in hitro spreminjajočih se projektov razvoja programskih proizvodov (White, 2000).

Vse te tehnologije in orodja so odigrala ključno vlogo pri prehodu od ročnega upravljanja konfiguracij k bolj avtomatiziranemu in učinkovitemu procesu upravljanja razvoja programske opreme.

S pojavom razpršenih razvojnih timov, odprtokodnih projektov in agilnih metodologij na prehodu v novo tisočletje se je poudarek premaknil k decentraliziranemu upravljanju konfiguracij. Sistemi za sledenje sprememb (angl. Version Control Systems) so postali temelj za učinkovito sodelovanje in upravljanje kodne baze.

V zadnjem desetletju je postalo upravljanje konfiguracij ključno za prakse DevOps, ki združujejo razvoj in operacije. Sodobni sistemi za UKPP omogočajo hitro spreminjanje, testiranje in razvijanje programske opreme v realnem času.

Skozi celotno zgodovino njihovega obstoja so se sistemi UKPP razvijali in prilagajali potrebam razvijalcev ter spreminjajočemu se okolju razvoja programske opreme. Danes so

sistemi UKPP ključen gradnik pri učinkovitem upravljanju kodnih baz, skupinskega dela, avtomatizacije procesov in zagotavljanju kakovosti programske opreme.

2.3 Osnovni pojmi upravljanja konfiguracij programskih proizvodov

Obstaja več standardov, ki obravnavajo upravljanje konfiguracij programskih proizvodov. Ti standardi določajo smernice, prakse in zahteve za učinkovito upravljanje konfiguracij programskih proizvodov v razvojnih projektih. Nekateri od pomembnejših standardov v tem kontekstu so (Aiello in Sachs, 2011):

- IEEE 828 – Standard for Configuration Management in Systems and Software Engineering: določa smernice za upravljanje konfiguracij v projektih programske opreme. Opisuje načela, prakse in zahteve za konfiguracijsko upravljanje, vključno s planiranjem, identifikacijo, spremembami, sledenjem in revizijami;
- ISO 10007 – Quality management – Guidelines for configuration management: ta standard določa smernice za upravljanje konfiguracij in dokumentacije v celotnem življenjskem ciklu izdelka ali projekta. Pokriva načela in prakse za identifikacijo konfiguracije, obvladovanje sprememb, vodenje dokumentacije itd.;
- ISO 12207 – Systems and software engineering – Software life cycle processes: standard opredeljuje procese življenjskega cikla programske opreme in vključuje tudi smernice za upravljanje konfiguracij. Določa zahteve za načrtovanje, sledenje in nadzor konfiguracij v okviru projekta razvoja programske opreme;
- MIL-STD-973 – Configuration management: čeprav ni več aktualen, je bil MIL-STD-973 standard ameriškega vojaškega ministrstva, ki je kot prvi določal smernice za konfiguracijsko upravljanje.

V kontekstu IEEE 828 je upravljanje konfiguracij programskih proizvodov opredeljeno kot niz dejavnosti, namenjenih nadzoru sprememb z identifikacijo in definicijo elementov programske opreme, vzpostavljanjem odnosov med njimi, definiranjem mehanizmov za upravljanje z različicami teh elementov, nadzor naloženih sprememb ter revizija in poročanje o izvedenih spremembah (Institute of Electrical and Electronics Engineers, 2012). Standard ISO/IEC 10007 definira upravljanje konfiguracije kot dejavnost upravljanja, ki uporablja tehnično in administrativno usmerjanje v življenjskem ciklu izdelka in storitve, njegovo identifikacijo in status konfiguracije ter povezane informacije o konfiguraciji izdelka in storitve (International Organization for Standardization, 2017a).

Iz omenjenih standardov (IEEE 828, ISO/IEC 10007, MIL-STD-973) izhajajo številni skupni pojmi, ki se nanašajo na upravljanje konfiguracij in življenjskega cikla programske opreme. Čeprav se podrobnosti in izrazi v omenjenih standardih lahko razlikujejo, so temeljni pojmi, ki obravnavajo osnovne vidike upravljanja konfiguracij in razvoja programske opreme, skupni vsem.

Za boljše razumevanje procesa UKPP bomo v nadaljevanju predstavili glavne pojme.

Konfiguracijski element (v nadaljevanju KE) (angl. Configuration Item – CI) – osnovna enota programske opreme, dokumentacije, specifikacij, risb ali drugih komponent, ki je upravljana znotraj procesa upravljanja konfiguracij.

Verzija – določena točka v času, ki predstavlja specifično stanje konfiguracijskega elementa ali izdelka v določenem času.

Referenčna točka (angl. Baseline) – specifična verzija konfiguracijskega elementa ali izdelka, ki služi kot osnova za nadzor nad spremembami in kot referenca za ocenjevanje napredka.

Konfiguracijska baza podatkov (KB) – sistem za shranjevanje, sledenje in upravljanje konfiguracijskih elementov, verzij, sprememb in drugih relevantnih podatkov.

Sprememba – kakršnakoli modifikacija, dodatek ali odstranitev konfiguracijskega elementa, izdelka ali dokumentacije.

Odobritev spremembe – formalni postopek za ocenjevanje, odobritev in izvedbo sprememb.

Nadzor verzij – proces sledenja in upravljanja različnih verzij konfiguracijskih elementov skozi njihov življenjski cikel.

Nadzor sprememb – upravljanje sprememb, vključno z zahtevami za spremembe, oceno, odobritvijo in izvedbo teh sprememb.

Sledljivost – zmožnost sledenja povezav med konfiguracijskimi elementi, verzijami ter spremembami skozi celoten življenjski cikel.

Vzdrževanje konfiguracije – proces vzdrževanja doslednosti in urejenosti konfiguracije med uveljavljanjem sprememb.

Integracija – proces združevanja različnih konfiguracijskih elementov v celotno funkcionalno enoto.

Dokumentacija – pisna ali elektronska zabeležba informacij o konfiguracijskih elementih, verzijah, spremembah in drugih vidikih projekta.

Te definicije poudarjajo pomembnost načrtovanja, nadzora, identifikacije, dokumentacije in vzdrževanja konfiguracijskih elementov ter uporabo formalnih postopkov za zagotavljanje integritete, sledljivosti in učinkovitosti celotnega procesa upravljanja konfiguracij

Načrt upravljanja konfiguracij (angl. Configuration Management Plan, v nadaljevanju CMP) – dokument, ki opisuje, kako bo organizacija izvajala upravljanje konfiguracij v okviru svojih projektov ali procesov in opredeljuje pristope, postopke in odgovornosti za upravljanje konfiguracij v projektih ali procesih. Medtem ko tudi drugi standardi vključujejo

načela upravljanja konfiguracij, je standard IEEE 828 posebej usmerjen v načrtovanje in dokumentiranje tega procesa, kar vključuje tudi vključitev CMP.

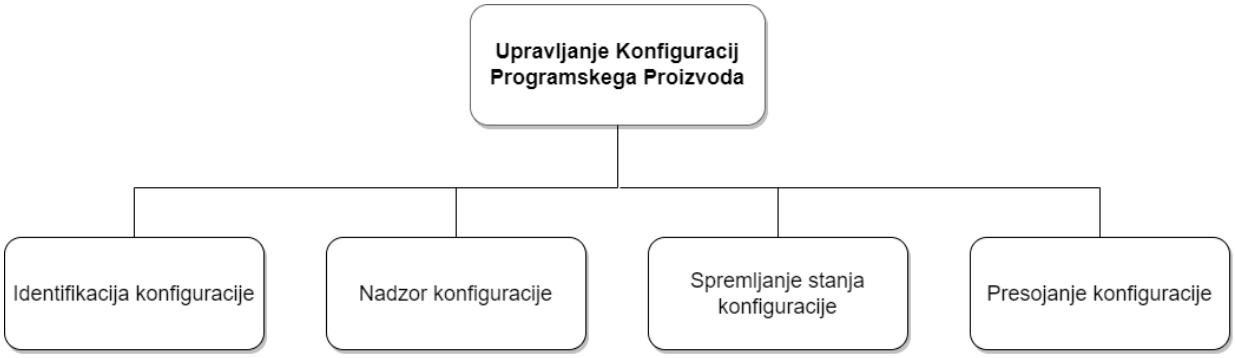
2.4 Predstavitev procesa upravljanja konfiguracij programskega proizvoda

Upravljanje konfiguracij programskega proizvoda je ključni proces pri razvoju programske opreme, ki omogoča dosledno in nadzorovano upravljanje vseh elementov programskega proizvoda skozi njegov celoten življenjski cikel. Ta proces se osredotoča na identifikacijo, nadzor, spremljanje, ocenjevanje in vzdrževanje konfiguracijskih elementov ter zagotavlja, da so vse spremembe temeljito ovrednotene in nadzorovane. Na sliki 1 so prikazana štiri funkcionalna področja procesa upravljanja konfiguracij programskega proizvoda, ki jih bomo v nadaljevanju podrobneje predstavili:

- identifikacija konfiguracije,
- nadzor konfiguracije,
- spremljanje stanja konfiguracije,
- presojanje konfiguracije.

Nadaljevali bomo s predstavitvijo postopkov upravljanja konfiguracij, ki omogočajo učinkovito izvajanje navedenih funkcionalnih področij upravljanja konfiguracij. Razumevanje tega je ključno za zagotavljanje integritete, sledljivosti in uspešnega razvoja programskega proizvoda v skladu z visokimi standardi kakovosti in učinkovitosti.

Slika 1: Funkcionalna področja sistema za UKPP



Vir: prirejeno po Keyes (2004).

2.4.1 Identifikacija konfiguracije

V fazi identifikacije konfiguracije se identificirajo in določijo ključni konfiguracijski elementi programskega proizvoda. Vsakemu KE se dodeli unikaten identifikator, ki omogoča sledenje in enostavno razlikovanje elementov med različnimi verzijami in spremembami. Identifikacija vključuje tudi določanje kriterijev za vključitev elementov v proces upravljanja konfiguracij.

Prej navedeni standardi obravnavajo fazo identifikacije konfiguracije vsak na svoj način, vendar s skupnim ciljem, in sicer: identificirati in označiti ključne KE za zagotavljanje sledljivosti in doslednosti med razvojem programskega proizvoda. ISO/IEC 10007 opredeljuje identifikacijo konfiguracije kot proces, ki vključuje dodeljevanje edinstvenih identifikatorjev konfiguracijskim elementom. Določa tudi, kako določiti kriterije za vključitev elementov v proces upravljanja konfiguracij ter kako opredeliti relacije med različnimi elementi (International Organization for Standardization, 2017a). IEEE 828 poudarja pomen identifikacije konfiguracijskih elementov in dodeljevanja identifikatorjev. Vključuje smernice za razvoj identifikacijske strukture, ki omogoča jasno razpoznavanje in sledenje elementov (Institute of Electrical and Electronics Engineers, 2012). ISO/IEC 12207 vključuje identifikacijo konfiguracije kot del procesa upravljanja konfiguracij v celotnem življenjskem ciklu programske opreme (International Organization for Standardization, 2017b).

Faza identifikacije konfiguracije torej vključuje določanje, kako bodo elementi identificirani, označeni in kako se bo zagotovila edinstvenost identifikatorjev. To omogoča doslednost, sledljivost in nadzor nad konfiguracijskimi elementi skozi celoten življenjski cikel proizvoda.

2.4.2 Nadzor konfiguracije

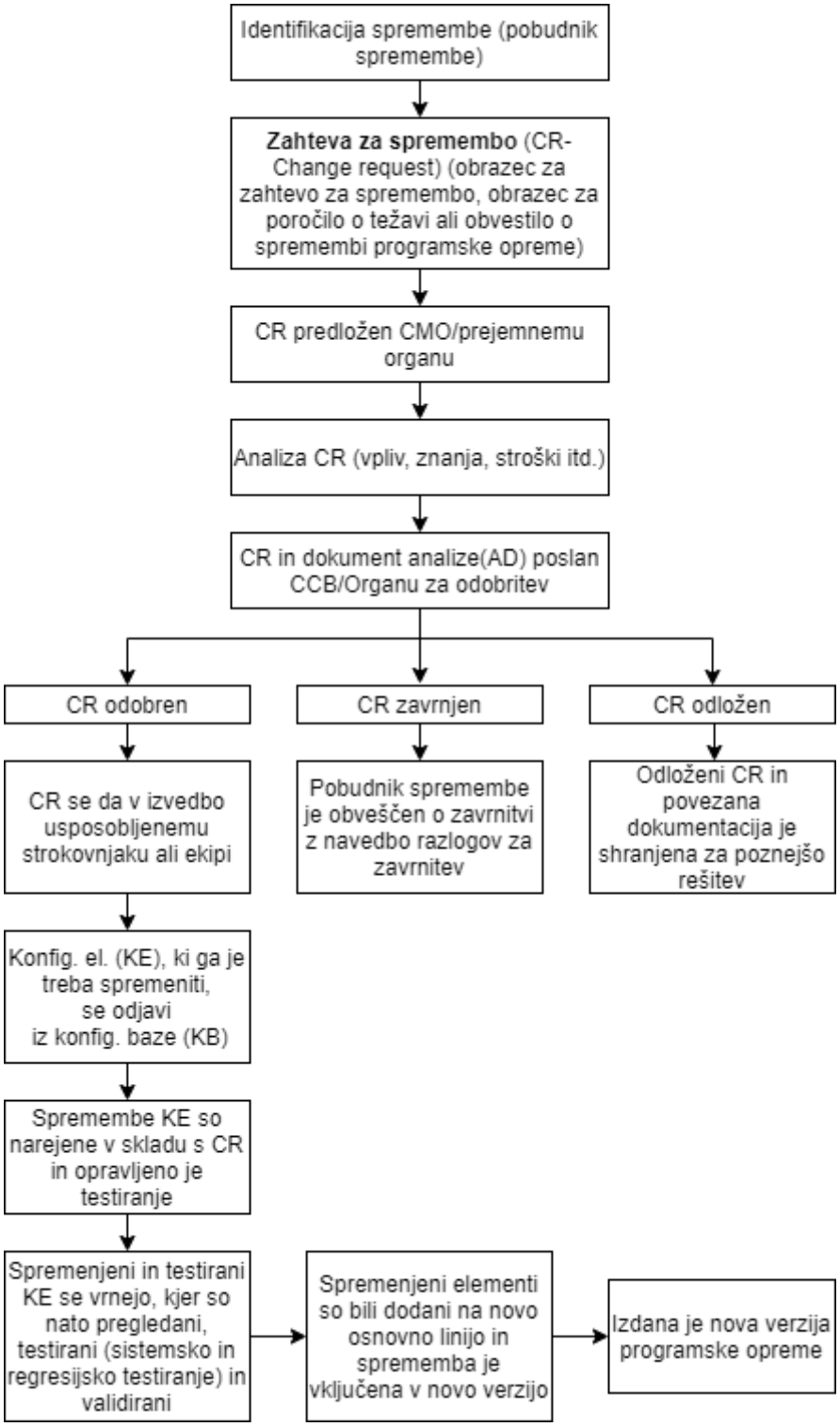
Nadzor konfiguracije zajema spremljanje in upravljanje sprememb v konfiguracijskih elementih. Vsaka sprememba se oceni glede na vpliv na proizvod, nato pa se izvede postopek odobritve spremembe. Konfiguracijske verzije se nadzorujejo skozi čas, z upravljanjem različnih verzij in zagotavljanjem doslednosti med njimi.

Posamezni standardi obravnavajo fazo nadzora konfiguracije kot ključno fazo procesa upravljanja konfiguracij, ki omogoča sledenje, upravljanje in nadzor sprememb v konfiguracijskih elementih. IEEE 828 vključuje postopke za nadzor konfiguracije, ki se osredotočajo na upravljanje verzij in uporabo referenčne točke (angl. Baseline). Standard določa smernice za spremljanje sprememb, ocenjevanje tveganj in zagotavljanje doslednosti med verzijami (Institute of Electrical and Electronics Engineers, 2012). ISO/IEC 10007 poudarja, da je nadzor konfiguracije usmerjen v upravljanje sprememb v konfiguracijskih elementih. Hkrati določa postopke za upravljanje sprememb, vključno z ocenjevanjem vpliva, odobritvijo sprememb in vzdrževanjem zgodovine sprememb (International Organization for Standardization, 2017a). ISO/IEC 12207 vključuje nadzor konfiguracije kot pomemben del življenjskega cikla programske opreme (International Organization for Standardization, 2017b).

Faza nadzora konfiguracije vključuje postopke za učinkovito sledenje in upravljanje sprememb v konfiguracijskih elementih, ki zagotavljajo doslednost, integriteto in sledljivost med razvojem proizvoda. To je ključno za zagotavljanje, da so vse spremembe ocenjene, odobrene in izvedene v skladu z načrtom programskega proizvoda.

Na sliki 2 je enostaven prikaz procesa uveljavljanja sprememb in nadzora konfiguracije. Omenjeni koraki so zelo splošni in se razlikujejo od podjetja do podjetja.

Slika 2: Proces uveljavljanja sprememb in nadzora konfiguracije



Vir: prirejeno po Leon (2015).

2.4.3 Spremljanje stanja konfiguracije

V tej fazi se vzdržuje ažurna konfiguracijska baza podatkov, ki vključuje informacije o konfiguracijskih elementih, verzijah, spremembah in odobritvah. Spremljanje stanja konfiguracije omogoča učinkovito sledenje napredka, sprememb in omogoča vpogled v trenutno stanje programskega proizvoda.

Standardi obravnavajo fazo spremljanja stanja konfiguracije kot proces vzdrževanja natančnih in ažurnih informacij o konfiguracijskih elementih, verzijah, spremembah ter drugih pomembnih podatkih. ISO/IEC 10007 poudarja vzdrževanje natančnih podatkov v konfiguracijski bazi podatkov in opisuje, kako slediti konfiguracijskim elementom in njihovim spremembam ter kako omogočiti dostop do ažurnih informacij (International Organization for Standardization, 2017a). IEEE 828 vključuje smernice za vzdrževanje ažurne konfiguracijske baze podatkov in poudarja, da mora biti konfiguracijska baza podatkov osnova za spremljanje in upravljanje sprememb ter za zagotavljanje sledljivosti (Institute of Electrical and Electronics Engineers, 2012).

Faza spremljanja stanja konfiguracije je ključna za zagotavljanje ažurnih in natančnih informacij o konfiguracijskih elementih ter omogoča pregled nad trenutnim stanjem programskega proizvoda. To je osnova za uspešno spremljanje napredka, ocenjevanje tveganj ter vzdrževanje sledljivosti in doslednosti med verzijami in spremembami.

2.4.4 Presojanje konfiguracije

Presojanje konfiguracije vključuje oceno preverjanja tako fizičnih (kako so bili izvedeni različni tehnični procesi razvoja od zahtev do testiranja) kot funkcionalnih značilnosti (kako je bil presojani element ali izdelek, kakršen je v času presoje, izdelan in kako so bile aplicirane zahtevane spremembe). To se lahko nanaša na katerikoli KE. Pogosta napaka je, da razvijalci verjamejo, da zadostuje sledenje konfiguracijskim elementom, kot so shranjeni v sistemu za upravljanje izvorne kode. Posamezen KE mora biti še vedno enolično prepoznaven, tudi ko je programski proizvod že izdan v produkcijsko okolje (Aiello in Sachs, 2011).

Faza presojanja konfiguracije je ključna za zagotavljanje ustreznosti dejanske konfiguracije sistema ali programske opreme nameravani konfiguraciji – kot je dokumentirano v konfiguracijski osnovi. Ta postopek je ključen za ohranjanje doslednosti, zanesljivosti in sledljivosti v okviru projekta ali sistema (The QA Lead Team, brez datuma).

2.4.5 Postopki upravljanja konfiguracije

V tej fazi so zajeti postopki in smernice za izvajanje vseh prejšnjih faz upravljanja konfiguracij. To vključuje pravila za identifikacijo, nadzor, spremljanje, presojanje in

vzdrževanje konfiguracijskih elementov. Postopki omogočajo doslednost, sledljivost ter usklajeno izvajanje upravljanja konfiguracij med vsemi člani projektnega tima.

Faza postopkov upravljanja konfiguracije vključuje opis smernic, pravil, in postopkov, ki jih organizacija uporablja za dosledno izvajanje vseh prejšnjih faz upravljanja konfiguracije. ISO/IEC 10007 vključuje postopke upravljanja konfiguracij, ki zajemajo identifikacijo, nadzor, spremljanje, presojanje in vzdrževanje konfiguracijskih elementov. Poudarja potrebo po dosledni izvedbi postopkov in upoštevanju smernic za zagotavljanje sledljivosti in integritete (International Organization for Standardization, 2017a). IEEE 828 opisuje načrtovanje postopkov za izvajanje nadzora konfiguracije in vključuje smernice za razvoj pravil za spremljanje sprememb, vzdrževanje konfiguracijske baze podatkov in upravljanje verzij (Institute of Electrical and Electronics Engineers, 2012). ISO/IEC 12207 opredeljuje postopke upravljanja konfiguracij kot del širšega procesa življenjskega cikla programske opreme (International Organization for Standardization, 2017b).

Faza postopkov upravljanja konfiguracije zajema izdelavo in implementacijo pravil, smernic ter postopkov za vsako od prej navedenih področij upravljanja konfiguracij. To je ključno za zagotavljanje doslednosti, sledljivosti in upravljanja v vseh fazah razvoja programskega proizvoda.

Tako strukturiran proces upravljanja konfiguracij omogoča organiziranost, sledljivost in urejenost med razvojem programskega proizvoda ter zagotavlja, da so vse spremembe in verzije nadzorovane in jih je možno slediti skozi celoten življenjski cikel programskega proizvoda.

3 ANALIZA OBSTOJEČEGA UPRAVLJANJA KONFIGURACIJ PROGRAMSKIH PROIZVODOV V IZBRANEM PODJETJU

3.1 Osnovni podatki o razvojnem okolju izbranega podjetja

Razvojno okolje v izbranem podjetju že od samih začetkov temelji na tehnologiji Oracle Forms. Gre za razvoj lastnih aplikacij ERP, ki je sprva zajemalo zgolj področja Upravljanja s kadrovskimi viri in Plačami, nato pa postopoma celotno podporo poslovnim procesom podjetja od Nabave, Proizvodnje, Skladiščnega poslovanja, Prodaje, Financ in Računovodstva z vsemi ostalimi podpornimi procesi.

V grobem lahko razdelimo razvoj v okolju Oracle Forms na dva dela:

- PODATKOVNO-BAZNI DEL in
- APLIKATIVNI DEL.

Osnova podatkovno-baznega dela je relacijska baza podatkov, ki vključuje različne vrste baznih objektov. Te vključujejo tabele (angl. Table), poglede (angl. View), omejitve (angl.

Constraints), zaporedja (angl. Sequence) in sprožilce (angl. Trigger), ki so splošno znani in prisotni v praktično vseh relacijskih bazah podatkov. Poleg teh osnovnih objektov obstajajo tudi drugi objekti, kot so npr. shranjene procedure in funkcije, ki služijo za izvajanje določenih operacij ali nalog neposredno v podatkovni bazi.

Shranjene procedure in funkcije se večinoma uporabljajo za izvajanje zapletenih obdelav, manipulacijo s podatki, izračune in druge operacije, pri čemer ni potrebno prenašati podatkov med strežnikom in odjemalcem. Razlika med funkcijami in procedurami je predvsem v tem, kako se obnašajo in kakšen je njihov izhod.

Procedura je niz korakov ali ukazov, ki so združeni v logično enoto. Izvaja se s klicem in se lahko uporablja za izvajanje določenih operacij, kot so vstavljanje, posodabljanje ali brisanje podatkov. Procedura se lahko izvaja neodvisno od konteksta, v katerem je bila ustvarjena, in lahko sprejema vhodne parametre ter vrača izhodne rezultate, lahko pa tudi ne.

Funkcija je podobna proceduri, vendar ima ključno razliko, da vedno vrača vrednost. Funkcija sprejme vhodne parametre, izvede določene operacije in nato vrne rezultat. Funkcije se pogosto uporabljajo za izračune ali obdelavo podatkov ter se lahko uporabijo kot del izračuna v SQL poizvedbah ali drugih operacijah v podatkovni bazi.

Pri uporabi funkcij in procedur, ki so vsebinsko enake ali podobne, jih je smiselno organizirati in združiti v paket, ki se v Oracle podatkovni bazi imenuje paket (angl. Package). To omogoča boljšo organizacijo in učinkovito obdelavo podatkov, saj lahko funkcije in procedure znotraj paketa delijo skupne komponente in se lažje vzdržujejo ter uporabljajo.

Postopek ustvarjanja neke nove aplikacije se praviloma prične v Oracle Designer orodju Entity Relationship Diagrammer, ki se uporablja za opredelitev informacijskih potreb podjetja v obliki entitetnih razmerij. Na tej stopnji se vzpostavi model entitet-povezav. Modeliranje odnosov med entitetami vključuje prepoznavanje pomembnih elementov v organizaciji (entitete), lastnosti teh elementov (atributi) in kako so med seboj povezane (povezave). Nastali informacijski model je neodvisen od nadalje izbranega načina dostopa, spreminjanja in shranjevanja podatkov.

Model entitet-povezav lahko izdelamo za celotne kompleksne sisteme ali pa ustvarimo manjše diagrame za podskupine sistemov. Posamezna entiteta se lahko pojavi v več diagramih, kar omogoča opredelitev številnih podrobnih podmnožic modelov.

Designer Entity Relationship Diagrammer ponuja (Dorsey in Koletzke, 1998):

- orodje za izdelavo diagramov, s katerim lahko izdelamo modele entitet-povezav;
- dostop do večuporabniškega repozitorija za ustvarjanje in vzdrževanje definicij entitet in povezav;
- dostop do čarovnika za načrtovanje baze podatkov, ki avtomatizira nekatere naloge razvoja sistema z izpeljavo privzete zasnove baze podatkov iz modelov entitet-povezav.

Na podlagi izdelanega modela entitet-povezav se nato s pomočjo čarovnika Database Design Wizard lahko izdela relacijske modele podatkov. Za pregled in urejanje teh načrtov se uporabi orodje Data Diagrammer, ki je grafično orodje za modeliranje načrtov logičnih in fizičnih shem baze podatkov. Objekte baze podatkov znotraj sheme je mogoče grafično predstaviti na relacijskih modelih, ki prikazujejo razmerje med tabelami, pogledi in posnetki v sistemu, ter integritetnih omejitev, določenih za te objekte.

Vsak relacijski model predstavlja zbirko objektov v bazi podatkov, ki jih je mogoče fizično razporediti med eno ali več baz podatkov. Sisteme porazdeljenih baz podatkov lahko modeliramo z več relacijskimi modeli, ki prikazujejo, kako se podatki razdelijo med lokalnimi in oddaljenimi bazami podatkov.

Data Diagrammer omogoča zasnovo sheme baze podatkov na dva glavna načina. Prvi je zasnova »od zgoraj navzdol« (angl. Top-Down), kjer začnemo z analitičnimi podatki, ustvarjenimi s pomočjo Entity Relationship Diagrammerja. Nato oblikujemo shemo baze podatkov na podlagi teh informacij. Drugi način omogoča pristop »od spodaj navzgor« (angl. Bottom-Up), kjer lahko na podlagi obstoječih podatkov v Oracle bazah rekonstruiramo zasnovo sheme nazaj v Data Diagrammer z uporabo orodij za obratno inženirstvo.

Poleg tega je na voljo tudi dodatna možnost, ki omogoča načrtovanje sheme baze podatkov neposredno v orodju Data Diagrammer, brez potrebe po predhodni analizi ali uporabi Entity Relationship Diagrammerja. To vključuje ustvarjanje novih tabel, pogledov, posnetkov ter določitev integritetnih omejitev.

Uporaba Data Diagrammerja omogoča (Dorsey in Koletzke, 1998):

- postopek pretvorbe podatkov iz analiz v tabele (skupaj z omejitvami tujih ključev), ki je avtomatiziran s čarovnikom za načrtovanje baze podatkov;
- modeliranje delov shem baz podatkov z uporabo relacijskih modelov, ki prikazujejo, kako so podatkovne strukture povezane v bazi podatkov;
- samodejno postavitev relacijskega modela z uporabo pripomočkov za samodejno postavitev;
- vzdrževanje in upravljanje obstoječih podatkov na ravni oblikovanja, centralno v repozitoriju in ustvarjanje novih elementov oblikovanja;
- modeliranje distribucije podatkov med lokalnimi in oddaljenimi zbirkami podatkov z uporabo več relacijskih modelov;
- povratni inženiring podatkov na ravni načrtovanja, ustvarjenih z uporabo Data Diagrammerja, na raven analize. Ta postopek je avtomatiziran s pripomočkom Table to Entity Retrofit;
- ustvarjanje poročil na podlagi podatkov iz repozitorija za izvajanje preverjanja kakovosti in za namene dokumentacije.

S pomočjo orodja Module Data Diagrammer se nato prične z gradnjo t. i. modula, v katerega se dodaja posamezne osnovne (angl. Base) in iskalne (angl. Lookup) tabele, na podlagi katerih je nato orodje sposobno samo generirati osnovni izgled vnosnega obrazca z vsemi polji, ki so bili v času izgradnje označeni, da naj se prikazujejo na obrazcu. Hkrati se npr. v primeru izgradnje vnosnega obrazca po principu Glave-Postavke (angl. Master-Detail) vzpostavijo vsi mehanizmi, ki zagotavljajo pravilno delovanje vnosnega obrazca.

Module Data Diagrammer ponuja način za hitro in enostavno ustvarjanje modulov tipa Vnosni obrazec (angl. Screen), Poročilo (angl. Report) in Graf (angl. Chart) z uporabo podatkovnih modelov. Generatorji uporabljajo informacije iz podatkovnih modelov in z njimi vplivajo na strukturo, funkcionalnost in postavitev ustvarjenih aplikacij.

Module Data Diagrammer omogoča opravljanje naslednjih glavnih nalog (Dorsey in Koletzke, 1998):

- ustvarjanje in vzdrževanje definicij modulov,
- ustvarjanje in vzdrževanje podrobne uporabe podatkov modula,
- opredeljevanje strukture generiranih aplikacij,
- ustvarjanje in vzdrževanje uporabe povzetkov podatkov modula.

Vsaka aplikacija je ustvarjena iz definicij modulov. Če definicije modulov ne obstajajo, jih je mogoče ustvariti iz Module Data Diagrammerja na naslednje načine:

- ročno opredeljevanje nove definicije modula,
- kopiranje obstoječe definicije modula,
- ustvarjanje definicij modulov s pomočjo čarovnika za načrtovanje aplikacij.

Kaj je Module Data Diagram oz. podatkovni diagram modula?

Vsak podatkovni diagram modula temelji na definiciji modula. Podatkovni diagram modula prikazuje podrobno uporabo podatkov modula in povezave med uporabami tabel.

Vsak podrobni element diagrama uporabe tabele, ki se pojavi na podatkovnem diagramu modula, predstavlja bodisi uporabo osnovne tabele ali uporabo iskalne tabele.

Uporaba osnovne tabele predstavlja skupino elementov v ustvarjeni aplikaciji. Postavke zagotavljajo informacije na podlagi te tabele. Če je uporaba osnovne tabele npr. ustvarjena za definicijo modula, potem uporaba osnovne tabele predstavlja blok na ustvarjenem obrazcu z elementi, ki omogočajo povezljivost z bazo podatkov.

Uporaba iskalne tabele predstavlja informacije, ki jih zahteva skupina postavk, vendar niso na voljo v osnovni podrobni uporabi tabele, na kateri temeljijo postavke. Vsaka uporaba iskalne tabele na diagramu je neposredno ali posredno povezana z uporabo osnovne tabele. Povezave med podrobnimi uporabami tabel temeljijo na omejitvah tujih ključev.

Po osnovnem modeliranju v Module Data Diagrammerju in določanju izgleda vnosnega obrazca na podlagi modula v naslednjem koraku izberemo skupno predlogo (angl. Template) vnosnega obrazca, ki poskrbi za čim bolj poenoten izgled vseh modulov oz. vnosnih obrazcev, ki se generirajo s pomočjo Oracle Designer orodja.

Prav tako se je uveljavil razvojni pristop, kjer so na vsak vnosni obrazec pripete skupne knjižnice, ki zagotavljajo enotno delovanje določenih funkcionalnosti ob zagonu vnosnih obrazcev. Te knjižnice na primer omogočajo prepoznavanje in branje podatkov iz tabel s pred nastavljenimi lastnostmi uporabniškega imena, s katerim je končni uporabnik prijavljen v aplikacijo, ter izvajanje avtorizacije, ki npr. določa, katere elemente vnosnega obrazca prikazati in katere načine upravljanja s prikazanimi podatki dovoliti.

Celoten repozitorij Oracle Designer orodij je shranjen v tabelah Oracle podatkovne baze, v posebej za to namenjeni bazni shemi, do katere ima direktni dostop samo administrator baze. To pomeni, da se elementi shranjujejo v nekakšni obliki metapodatkov kot zapisi v tabelah in služijo kot navodilo za kasnejše generiranje fizičnih aplikativnih datotek (*.fmb za vnosne obrazce in *.rdf za poročila oz. reporte) in datotek baznih objektov (*.tab,*.con,*.trg, …).

Vsak razvijalec uporablja svoje uporabniško ime za prijavo v aplikacijska področja, ki vključujejo tabele in druge bazne objekte specifičnega aplikacijskega področja.

Med postopkom generiranja razvijalec določi ciljno mesto, kamor naj se aplikacijska datoteka generira. To običajno vključuje mapo na njegovem lokalnem računalniku, ki je prav tako zabeležena v registru njegovega računalnika.

Po tem, ko razvijalec v svojem lokalnem razvojnem okolju zaključi programiranje za zagotovitev zahtevanih funkcionalnosti, sledi korak namestitve izvršilnih datotek v skupno okolje, ki se nahaja na predhodno določeni mapi na skupnem razvojnem strežniku. Tam praviloma shrani tako izvorno kot tudi izvršilno kodo, pri čemer vsak tip datoteke shranjuje v svojo ustrezno mapo.

Do teh skupnih map so vzpostavljene bližnjice, ki omogočajo posameznim uporabnikom zagon aplikacij in izvajanje vnosa ali manipulacije podatkov v bazi podatkov preko izbrane izvršilne datoteke.

Oracle Designer sam po sebi ne vključuje funkcionalnosti za verzioniranje posameznih modulov, zato mora razvijalec, če želi zagotoviti prejšnjo verzijo modula, ročno ustvariti kopijo obstoječega modula v repozitoriju, preden začne izvajati spremembe na izbranem modulu.

3.2 Zgodovina in razvoj razvojnega okolja izbranega podjetja

Prvi začetki razvoja lastnih programskih proizvodov segajo v obdobje začetka 80-ih let z uvedbo takrat revolucionarne Oracle relacijske podatkovne baze in razvojem aplikacij s

pomočjo SQL*Forms, ki je veljalo kot eno izmed prvih orodij za razvoj uporabniških vmesnikov za baze podatkov in se je takrat izvajalo še v terminalskem okolju. V tem obdobju je omogočalo razvijalcem ustvarjanje enostavnih vmesnikov za urejanje podatkov in izvajanje poizvedb.

V času, ko sem se sam prvič srečal z razvojnim okoljem v izbranem podjetju, so takratne aplikacije razvijali v verziji Oracle Forms 4.5 s pomočjo Oracle Designer orodja. Šlo je za proces razvoja v dvonivojskem načinu (2N), kar pomeni, da je bil vzpostavljen odjemalec – strežnik (angl. Client-Server) način delovanja. Tak pristop zahteva namestitev izvajalnega okolja Oracle Forms tudi pri vsakemu končnemu uporabniku posebej, poleg tega pa tudi postavitev lokalnih bližnjic, ki vsebujejo povezave za zagon menijev posameznih aplikacij. Končni uporabniki se v aplikacije prijavijo z ustreznimi uporabniškimi imeni. Osebni računalniki (angl. PC – Personal Computer), na katerih teče Oracle Forms in Reports, se v Oracle podatkovno bazo prijavljajo preko omrežnega protokola SQL*NET.

Razvoj 2N aplikacij poteka tako, da se pri vsakemu razvijalcu namesti in vzpostavi lokalno razvojno okolje, ki vsebuje razvojna orodja Oracle Forms in Reports Developer, v katerem so t. i. moduli, ki so v osnovi izdelani in generirani s pomočjo Oracle Designer generatorja, nato pa se po potrebi ročno dopolnjujejo s kodiranjem dodatne programske kode za zagotovitev delovanja zahtevanih funkcionalnosti aplikacij. Vsa dodatna programska koda se praviloma dodaja v (k modulu pripete) knjižnice, katerih vsebino je kadarkoli možno spreminjati, ne da bi bilo potrebno iz Designerja ponovno generirati osnovni modul.

2N način izdelave aplikacij v Oracle Forms se nanaša na organizacijo razvoja aplikacij na dveh glavnih nivojih, pri čemer se aplikacijska logika in uporabniški vmesnik združita v eno enoto. Ta pristop je manj abstrakten in strukturiran kot trinivojski (3N) način.

Osnovno strukturo 2N izdelave aplikacij v Oracle Forms predstavljata:

- nivo baze podatkov (angl. Database Tier): na najnižjem nivoju je še vedno baza podatkov, ki hrani vse podatke za aplikacijo, praviloma Oracle podatkovna baza. V njej se izvajajo operacije za shranjevanje, prebiranje, posodabljanje in brisanje podatkov;
- nivo aplikacijske logike in uporabniškega vmesnika (angl. Application Logic and User Interface Tier): v dvonivojskem načinu se aplikacijska logika (kot so poslovna pravila, validacije itd.) in uporabniški vmesnik (obrazci, poročila) združita v isti enoti. To pomeni, da se v okviru iste aplikacije razvijajo tako funkcionalnost aplikacije kot tudi vmesnik za uporabnike. Razvoj poteka v okviru Oracle Forms razvojnega okolja, kjer so obrazci in poročila ustvarjeni ter opremljeni z aplikacijsko logiko.

Prednosti 2N pristopa vključujejo preprostost razvoja in manjšo kompleksnost aplikacij, saj ni potrebno skrbeti za ločevanje aplikacijske logike od uporabniškega vmesnika. Ta pristop je lahko primeren za manjše ali manj kompleksne aplikacije, vendar ima omejitve pri večji skalabilnosti, vzdrževanju in ponovni uporabi kode v primerjavi s 3N pristopom. Odjemalci

v 2N računalniškem modelu so t. i. debeli odjemalci (angl. Fat Client), kjer se nahaja večina procesorske moči in aplikacijske logike. Zaradi tega je vzdrževanje odjemalcev zamudno in posledično drago. Poleg tega lahko odjemalci delujejo na različnih platformah, zaradi česar je potrebna uvedba različic aplikacij, specifičnih za posamezno platformo.

Z verzijo Oracle Forms 6i (in nadaljnjimi verzijami) se uveljavi 3N način izdelave aplikacij. Nanaša se na organizacijo razvoja aplikacij z uporabo Oracle Forms razvojnega okolja na treh glavnih nivojih. To omogoča boljšo organizacijo in vzdrževanje aplikacij. Trinivojski način predstavljajo:

- nivo baze podatkov (angl. Database Tier): tako kot pri 2N načinu je na najnižjem nivoju baza podatkov, ki hrani vse podatke, potrebne za aplikacijo. Praviloma gre za Oracle podatkovno bazo, ki je močno integrirana z Oracle Forms. Na tem nivoju se izvajajo operacije, kot so shranjevanje, prebiranje, posodabljanje in brisanje podatkov;
- nivo aplikacijske logike (angl. Application Logic Tier): ta nivo vsebuje Oracle Forms aplikacijo, ki je razvita za uporabniški vmesnik in aplikacijsko logiko. Aplikacija vključuje obrazce (Forms), poročila (Reports) in druge elemente za uporabnikov vmesnik. Aplikacijska logika se uporablja za obdelavo podatkov med vmesnikom in baznim nivojem, vključno z obdelavo poslovnih pravil, validacijo podatkov in izvajanjem transakcij;
- nivo uporabniškega vmesnika (angl. User Interface Tier): na vrhu je nivo uporabniškega vmesnika, kjer uporabniki dostopajo do aplikacije na strani strežnika. V tej trinivojski arhitekturi je odjemalska programska oprema (stopnja odjemalca) dovolj lahka, da jo je mogoče prenesti na zahtevo, in predstavlja zgolj uporabniški vmesnik za aplikacijo na strani strežnika. Večji del logike aplikacije je implementiran v aplikacijskem strežniku ali v bazi podatkov. Ta nivo omogoča uporabnikom, da vnašajo podatke, pregledujejo poročila in izvajajo operacije v okviru aplikacije.

Prednosti 3N pristopa vključujejo boljšo organizacijo aplikacij, lažje vzdrževanje in možnost večkratne uporabe aplikacijske logike. Poleg tega omogoča boljšo ločitev med razvojem aplikacijske logike in uporabniškim vmesnikom ter podpira večjo skalabilnost in učinkovitost.

V obdobju po letu 2000 je Oracle Forms napredoval tudi v smislu večplatformske podpore. Orodje je postalo združljivo z različnimi operacijskimi sistemi in spletnimi tehnologijami. To je omogočilo razvijalcem ustvarjati obrazce, ki so dostopni prek spleta.

V naslednjem desetletju in vse do danes se je osredotočenost razvoja Oracle Forms premaknila k spletnemu razvoju. Oracle je izdal Oracle Forms 11gR2, ki je omogočal razvijalcem, da svoje obstoječe obrazce pretvorijo v spletne aplikacije z uporabo Oracle Forms to WebLogic Server (Oracle, 2017). To je omogočilo enostavno uporabo obstoječih veščin in programske kode ter hkrati zagotovilo, da so aplikacije dostopne prek spleta.

Čeprav so se v zadnjih letih pojavile druge tehnologije za razvoj spletnih in mobilnih poslovnih aplikacij, Oracle še vedno podpira Oracle Forms in redno izdaja posodobitve. Razvijalci, ki so se zanašali na to tehnologijo za svoje aplikacije, lahko še naprej izkoristijo prednosti in vzdržujejo svoje obstoječe rešitve. V času pisanja tega dela je aktualna verzija Oracle Forms 12c.

Oracle Forms ostaja pomembno orodje za organizacije, ki temeljijo na Oracle podatkovnih bazah in imajo obstoječe aplikacije razvite v tem okolju. Medtem ko so se trendi v razvoju programske opreme razvijali, je Oracle Forms še vedno uporaben za tiste, ki želijo ohraniti in nadgraditi svoje obstoječe aplikacije.

3.3 Analiza trenutnega stanja razvojnega okolja izbranega podjetja

Razvojno okolje za ERP programski proizvod v izbranem podjetju temelji na Oracle tehnologiji. V razvojnem procesu sodeluje 13 aplikativnih inženirjev, ki so razvijalci oz. programerji, ter 10 strokovnjakov za poslovno vsebino oz. poslovnih analitikov.

Tehnološko gledano je razvojno okolje razdeljeno na dva glavna dela: aplikativni del, ki se izvaja na aplikacijskem strežniku IAS (angl. Internet Application Server), ter bazni del, kjer je osrednji del enotna baza podatkov Oracle, nameščena na baznem strežniku.

Aplikativni del poslovno-informacijskega sistema ERP je razdeljen na 5 osnovnih poslovnih področij (Komerciala, Proizvodnja, Finance in Računovodstvo, Standardizacija – Skupno področje in Upravljanje s človeškimi viri (angl. Human Resource Management – HRM) s Plačami), pri čemer vsako poslovno področje dodatno vsebuje različno število aplikacij. Skupno je 76 aplikacij, ki vključujejo več kot 1500 vnosnih obrazcev (Oracle Forms). Od teh je približno 1000 splošnih, ki veljajo za vse stranke, medtem ko so preostali izdelani posebej za pokrivanje specifičnih funkcionalnosti poslovnih procesov določenih strank. Podobno razmerje med splošnimi in strankam prilagojenimi oblikami velja tudi za poročila (Oracle Reports), katerih skupno število je približno 1700.

V baznem delu se skupno nahaja približno 5500 tabel, preko 1000 paketov (angl. Package) z različnimi procedurami in funkcijami, več kot 1000 pogledov (angl. View) in približno 700 prožilcev (angl. Trigger).

Definicije aplikativnih modulov in baznih objektov so osnovno shranjene v Oracle Designerju, kjer je vzdrževana samo ena trenutna verzija oz. konfiguracija programskega proizvoda.

V orodju Oracle Designer so postavljena različna aplikacijska področja in tudi dostop do teh področij je porazdeljen in avtoriziran po posameznih skupinah aplikativnih inženirjev. Do poslovnega področja Komerciala lahko na primer dostopajo samo 4 razvijalci in 3 poslovni analitiki. Poimenovanje posameznih aplikacijskih področij, aplikacij, modulov in baznih

objektov je standardizirano s predponami oziroma kraticami, kot npr. Skladišče – SDE, Prodaja – POJ, Proizvodnja – PRO, Inventure – IVT itn.

Na tak način se z uporabo govorečih šifer posamezni elementi združujejo glede na pripadnost določeni vsebini. Enoličnost poimenovanja se zagotavlja v sklopu posameznega aplikacijskega področja in hkrati z uporabo omenjenih predpon.

Posamezna verzija izvornih datotek se zagotavlja z generiranjem modulov (vnosni obrazci, poročila) in definicij baznih objektov (tabele, pogledi, zaporedja, prožilci) ter shranjevanjem na skupno mapo strežnika, kjer so na drugem nivoju postavljene mape za posamezne stranke. Dostop do skupne mape je omogočen vsem razvijalcem, kar pomeni, da obstaja precejšnje tveganje nekontroliranega spreminjanja vsebine teh map. Sicer se izvaja dnevno varnostno kopiranje vsebine strežniških diskov, ampak to ne zagotavlja zaščite pred, na primer, nenamernim spreminjanjem vsebine v mapah, prav tako se v takem primeru nikjer ne beleži, kdo, kaj in kdaj je spremembo vsebine datotečnih map izvedel.

Razvojna dokumentacija (funkcijske specifikacije, slike, diagrami, zapisniki sestankov, uporabniška navodila) se prav tako shranjuje v skupnih mapah na strežniških diskih. Tudi tu se lahko pojavlja podobno tveganje kot pri aplikativnih datotekah z izvorno kodo.

Razvoj novih modulov ali spreminjanje obstoječih poteka znotraj skupin, glede na poslovno vsebino. Povod za spreminjanje informacijskega sistema so lahko ugotovljene napake, zahteve po novih funkcionalnostih ali zakonske spremembe, ki jih je potrebno implementirati v programsko rešitev. Zaradi kompleksnosti vsebine praviloma razvijalec, ki je razvil nek določen modul, postane tudi vzdrževalec tega modula, enako je z baznimi objekti. Kadar je vsebina širša, se sestane ekipa, ki vključuje strokovnjake iz različnih področij in sprejme odločitve glede načina in postopka vpeljave novih funkcionalnosti. Ker je poslovni sistem precej dobro parametriziran, se določene zahteve za spremembe lahko uveljavi tudi brez poseganja v programsko kodo in se izvede zgolj sprememba določenih nastavitev, ki vplivajo na način delovanja posamezne aplikacije.

Za potrebe razvoja se uporabljata skupna razvojna baza in skupni aplikativni strežnik. Razvijalec že v času kodiranja izvaja sprotna testiranja delovanja spreminjajočega modula. Kasneje, ko spremembe uveljavi, preda novo verzijo v testiranje vsebinskemu analitiku, ki izvede verifikacijo uveljavljenih sprememb. To praviloma poteka tako, da razvijalec pripravi namestitveni paket, ki vsebuje tako datoteke za spremembo baznih objektov, kot tudi aplikativne datoteke spremenjenih modulov (če so bili spremenjeni), in izvede namestitev v testno okolje stranke. Izvedbo verifikacije spremenjenega sistema v skupnem razvojnem okolju praviloma ni možno izvesti dovolj kakovostno, ker v poslovnem informacijskem sistemu obstaja kar nekaj posebnosti, ki veljajo samo za določene stranke. Razlog za to je med drugim tudi ta, da v skupnem razvojnem okolju ni dostopa do drugih sistemov, ki so lahko povezani s poslovnim informacijskim sistemom pri določeni stranki in so na voljo le v okolju pri tej stranki ali pa so podatki in nastavitve sistema v podatkovni bazi toliko

specifični, da je testiranje možno izvesti zgolj v testnem okolju pri stranki. Zato se verifikacija s strani vsebinskih analitikov in nato še klasična validacija s strani ključnih uporabnikov stranke praviloma izvede v testnem okolju pri določeni stranki.

Če se zgodi, da testiranje oziroma verifikacija ali validacija ni uspešna, se zadevo vrne razvijalcu (skupaj s pripombami) in razvijalec prične nov cikel spreminjanja programske kode.

Ko pridemo do faze, da uporabnik potrdi ustreznost izvedenih sprememb (uspešna validacija), se lahko namestijo v produkcijsko okolje. V večini primerov tudi ta del izvede razvijalec sam, pri nekaterih strankah pa dostop do produkcijskega okolja ni omogočen (npr. banke in zavarovalnice). Takrat namestitev izvede oseba/strokovnjak pri stranki, ki ima ustrezna znanja in je sposoben v dogovoru z aplikativnimi inženirji v razvojnem podjetju izvesti zadano nalogo.

Ker so aplikacijska področja ločena, načeloma ne prihaja do primerov, kjer bi več razvijalcev spreminjalo isto programsko kodo. Lahko pa se pojavi situacija, kjer sprememba neke procedure ali funkcije vpliva na delovanje neke druge, ki se posredno v svoji programski kodi sklicuje nanjo. Glede na to, da se uporablja skupna enotna razvojna baza, se to takoj opazi, ker paketi, ki so nosilci procedur in funkcij, postanejo neveljavni (angl. Invalid). V takem primeru razvijalec, ki izvaja spremembe v razvojni bazi, obvesti skrbnika neveljavnega paketa in praviloma skupaj z medsebojnim sodelovanjem in dogovorom ugotovita, kako ustrezno spremeniti programsko kodo tudi v tem paketu, da bo zopet postal veljaven.

Vsaka sprememba baznega objekta se mora praviloma uveljaviti tudi v Oracle Designerju, ki je skupni repozitorij vseh baznih objektov in modulov. Če razvijalec to pozabi, se kasneje pri izvedbi nadgradnje celotnega informacijskega sistema neke stranke ta verzija baznega objekta ne bo uveljavila, kar bo posledično povzročilo napake in neveljavnost določenih baznih objektov ali morda okrnjeno delovanje modulov. V takih primerih je sprememba navadno uveljavljena zgolj v verziji baznega objekta, ki je nameščen v skupni razvojni bazi.

Ker Oracle Designer ne omogoča verzioniranja posameznih elementov v repozitoriju, posledično tudi ni možno direktno iz njega ugotoviti, katera verzija baznega objekta ali modula je nameščena pri stranki. To se lahko ugotovi zgolj ročno, z izvozom dotičnega baznega objekta iz podatkovne baze in nato primerjavo z verzijo, ki je bila odložena kot namestitvena datoteka na skupno mapo določene stranke na skupnem strežniku ob izdaji oz. nadgradnji na takratno zadnjo verzijo.

Način obravnave in uveljavljanja zahtev za spremembe informacijskega sistema je določen v organizacijskih predpisih Razvoj/izvedba projekta in Servis/vzdrževanje, ki ju ima podjetje opredeljeno kot temeljna procesa in sta tudi del dokumentacije vzpostavljenega sistema kakovosti po standardu ISO 9001:2015 (International Organization for Standardization, 2015).

Kadar ne gre za nove interne razvojne projekte, dopolnitve/razširitve programskega proizvoda v obliki novih funkcionalnosti oziroma novih aplikacij, se večina sprememb uveljavlja preko storitvenih zahtevkov (angl. Service Request), ki jih podjetje obvladuje z uporabo sistema za elektronsko podajanje zahtevkov. Vse stranke, ki imajo sklenjeno vzdrževalno pogodbo, imajo preko Portala Za Stranke (v nadaljevanju PZS) omogočen dostop do omenjenega sistema za prijavo storitvenih zahtevkov, bodisi da gre za prijavo napak ali zahtevkov za dograditev. Gre za interno razvito aplikacijo za upravljanje odnosov s strankami (angl. Customer Relationship Management, v nadaljevanju CRM), ki temelji na Microsoft Dynamics CRM platformi.

Aplikacija PZS omogoča zajem zahtevkov, ki se v CRM okolje zapisujejo v obliki Projektov. Posamezni Projekti imajo postavljene statuse, ki določajo, v kakšnem stanju se nahaja Projekt. Statusi so naslednji 1 – Prijavljen, 2 – Ocenjen, 3 – Odobren, 4 – Razporejen, 5 – V delu, 6 – Delo končano, 7 – Verificiran, 8 – Potrjen, 9 – Fakturiran, 10 – Zaprt, 11 – Preklican, 12 – Ustavljen, 13 – Zavrnjen. Prav tako se z lastnostjo Tip projekta določa, za kakšne vrste projekta gre. Projekti so lahko Prodajni, Vzdrževalni, PZS (vezani na Storitvene zahtevke), Razvojno-izobraževalni, Interna režija.

Vsak aplikativni inženir ima odprt svoj račun/uporabniško ime, s katerim se v sistem prijavlja preko interne aplikacije VP (Vodenje projektov) in preko nje dostopa do Projektov, v katere je vključen preko njemu dodeljenih Planov. V tem primeru gre za dostop do CRM sistema, kot ga potrebuje aplikativni inženir za izvedbo in evidentiranje opravljenih zadanih nalog.

Vsak Projekt je najprej obravnavan s strani Skrbnika projektov in po pregledu vsebine razporejen naprej glavnemu skrbniku poslovnega področja, ki postane Vodja projekta. To je praviloma eden izmed aplikativnih inženirjev, poslovnih analitikov.

Vloga vodje projekta je skrb za pravilno in pravočasno izvedbo projekta. Naloge vodje projekta so:

- preverjanje, opremljanje in po potrebi spreminjanje osnovnih podatkov projekta;
- spreminjanje statusov, skladno s potekom projekta;
- odpiranje planov za vpletene aplikativne inženirje (vsebinske analitike, programerje);
- začetno komuniciranje s stranko (ocena dela, roki, po potrebi pridobivanje dodatnih informacij);
- po verificiranju opravljenega dela poskrbi za potrditev ustreznosti rešitve s strani stranke (validacija);
- po strankini potrditvi obveščanje skrbnika stranke in prodajalca ob zaključku projekta (projekt pripravljen za fakturiranje).

Na sliki 3 je z grafično notacijo za modeliranje poslovnih procesov in delovnih tokov (angl. Business Process Model Notation, v nadaljevanju BPMN) prikazan proces uveljavljanja sprememb in nadzora konfiguracij.

Začetek procesa lahko aktivira stranka, interni razvoj ali prodajalec, ki je zaznal tržno priložnost za razvoj programske rešitve. Vsak tak zagon sproži ustvarjanje novega Projekta v CRM sistemu s statusom »1 – Prijavljen«. Ob nastanku novega Projekta je najprej obveščen Skrbnik projektov, ki preko aktivnosti Analiza opisa in opremljanje Projekta z zahtevanimi podatki in glede na zabeleženo vsebino v Projektu ugotovi, ali je vsebina sprejemljiva za nadaljnje izvajanje sprememb programskega proizvoda. Če sam ne more sprejeti odločitve in obstajajo dvomi, se Skrbnik projektov sestane s širšim krogom strokovnjakov iz zadevnega področja in če ugotovijo, da vsebina ni primerna za izvedbo, prestavi status projekta na »11 – Preklican«. V nasprotnem primeru se določi tip projekta in imenuje vodjo projekta.

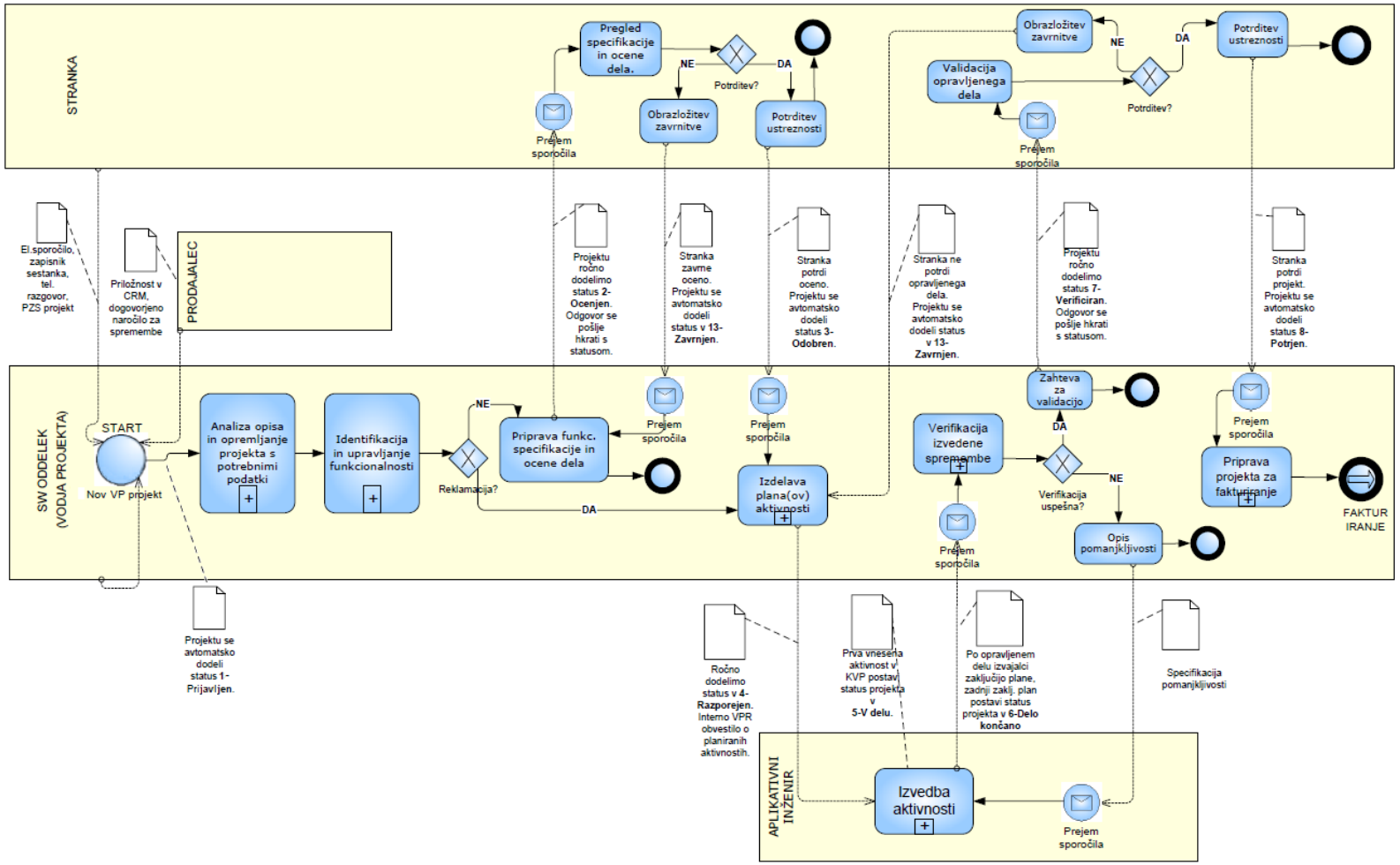
Vodja projekta nato na podlagi zabeležene vsebine na projektu določi, ali gre za napako (angl. Bug) ali pa za dodajanje nove funkcionalnosti (angl. Feature) ali izboljšavo obstoječe.

Če gre za napako v obstoječi funkcionalnosti, vodja projekta pripravi ustrezen načrt, kjer določi naloge, ki jih mora opraviti izvajalec nalog. Te naloge se običajno nanašajo na spremembe nastavitev sistema ali prilagoditev programske kode, kot so spremembe baznih objektov ali aplikativnih datotek. Komunikacija s stranko praviloma poteka preko aplikativnih inženirjev, poslovnih analitikov, ne pa neposredno s programerji. Če gre za dodajanje novih funkcionalnosti ali izboljšav, vodja projekta pripravi funkcijsko specifikacijo in oceni obseg dela, ki se posreduje stranki. Stranka prejme sporočilo s povezavo do projekta, katerega status se samodejno spremeni v »2 – Ocenjen«. Po pregledu vsebine lahko stranka odobri (status »3 – Odobren«) ali zavrne (status »13 – Zavrnjen«) predlagane spremembe.

Če stranka odobri oceno dela, se začne faza priprave načrta aktivnosti (status »4 – Razporejen«). Obvesti se aplikativnega inženirja razvijalca, ki lahko začne izvajati načrtovane aktivnosti. Ob vnosu zapisa prve aktivnosti, se status projekta samodejno spremeni v »5 – V delu«. Po dokončanju dela, vključno z namestitvijo v testno okolje, razvijalec spremeni status projekta v »6 – Delo končano«. O tem obvesti vsebinskega analitika zadevnega področja, ki opravi verifikacijo izvedenih sprememb. Če je verifikacija uspešna, se status projekta spremeni v »7 – Verificiran«, o čemer je obveščen predstavnik stranke, ki je prijavil zahtevek. Pri stranki se začne izvajati postopek validacije. Če je ta uspešen, se status projekta spremeni v »8 – Potrjen«. V nasprotnem primeru projekt dobi status »13 – Zavrnjen«. O tem je obveščen vodja projekta, ki preuči vzroke in po potrebi pripravi nove načrte za dopolnitev zahtevanih funkcionalnosti. Postopek se ponovi.

Ko ima projekt status »8 – Potrjen«, je pripravljen za proces fakturiranja. Po izvedbi faze fakturiranja projekt dobi status »9 – Zaprt«.

Slika 3: Proces uveljavljanja sprememb in nadzora konfiguracije v izbranem podjetju



Vir: lastno delo.

Da bi dobili čim bolj podrobne informacije o obravnavni problematiki, so bili opravljeni osebni razgovori in izvedena skupinska diskusija s ključnimi zaposlenimi v razvojnem oddelku. Na podlagi anketnega vprašalnika, ki ga je izpolnilo 13 aplikativnih inženirjev razvijalcev, je bilo ugotovljeno, da ima večina razvijalcev težave pri upravljanju s programsko kodo brez ustreznega orodja za upravljanje s programsko kodo. Ti rezultati podpirajo vpeljavo takega orodja v obstoječe razvojno okolje. Anketni vprašalnik je bil razdeljen na tri sklope vprašanj:

- splošne informacije o poznavanju orodja za upravljanje s programsko kodo;
- težave pri upravljanju s programsko kodo brez orodja za upravljanje s programsko kodo;
- uporaba orodij za upravljanje programske kode v preteklosti.

Skoraj 80 % vprašanih je navedlo, da imajo težave pri sledenju spremembam v programski kodi, če ne uporabljajo orodja za upravljanje s programsko kodo. Prav tako je 65 % vprašanih izjavilo, da je bila uporaba orodja za upravljanje s programsko kodo v preteklosti koristna. Medtem ko večina vprašanih pozna koncept uporabe orodja za upravljanje s programsko kodo, jih je manj kot 40 % dejansko že uporabljalo takšno orodje.

Rezultati podpirajo dejstvo, da ima večina razvijalcev težave pri upravljanju s programsko kodo, če ne uporabljajo orodja za upravljanje s programsko kodo. Iz tega izhaja, da bi bila dopolnitev sistema za upravljanje konfiguracij z vključitvijo orodja za upravljanje s programsko kodo zelo koristna. S tem bi lahko izboljšali kakovost programske kode in skrajšali čas, ki ga porabijo razvijalci za sledenje spremembam. Prav tako iz ankete izhaja, da je manj kot 40 % anketirancev v preteklosti dejansko že uporabljalo takšno orodje, kar kaže na potrebo po boljšem ozaveščanju o tej tehnologiji in njenih prednostih. Med orodji, ki so jih anketiranci v preteklosti že uporabljali, so navedli tako Subversion (SVN), Git kot tudi Mercurial.

Anketni vprašalnik je omogočil, da bolje razumemo, kako razvijalci vidijo upravljanje s programsko kodo in kaj lahko storimo, da izboljšamo obstoječi razvojni proces.

4 PREDLOG NOVEGA UPRAVLJANJA KONFIGURACIJ PROGRAMSKIH PROIZVODOV

4.1 Analiza možnih orodij za verzioniranje konfiguracijskih elementov za podporo sistemu za upravljanje konfiguracij programskih proizvodov

Izhajajoč iz analize trenutnega stanja upravljanja konfiguracij in predstavljenega koncepta vzpostavitve sistema za UKPP se bomo v nadaljevanju osredotočili na izbiro primernega orodja, ki bo omogočalo vzpostavitev konfiguracijske baze z možnostjo verzioniranja posameznih konfiguracijskih elementov in celotne konfiguracije posameznega programskega proizvoda. Trenutno se v podjetju uporablja Oracle Designer kot centralno

skladišče, v katerem so shranjene definicije tako baznih objektov kot aplikativni moduli, ki jih je potrebno generirati pred končno namestitvijo pri stranki. Trenutno največja pomanjkljivost obstoječega sistema za UKPP je nezmožnost sistemskega verzioniranja posameznih konfiguracijskih elementov, zato se bomo osredotočili na orodja za upravljanje z izvorno programsko kodo. Na podlagi dejstva, da obstajajo odprtokodna orodja, ki so v praksi že vrsto let uveljavljena in med razvijalci dokončno sprejeta, bomo ožji izbor izvedli zgolj med temi in zavestno izpustili komercialna orodja. Med najbolj zastopanimi odprtokodnimi orodji za upravljanje z izvorno programsko kodo in nadzorom verzij najdemo Git, SVN in Mercurial. V nadaljevanju sledi kratka predstavitev vsakega izmed njih.

4.1.1 Predstavitev možnih orodij za podporo sistemu za upravljanje konfiguracij programskih proizvodov

Glavna razlika na splošno med sistemi za upravljanje z izvorno programsko kodo je v načinu delovanja. Lahko so centralizirani – bazirani na enotnem strežniku ali porazdeljeni Peer-to-Peer (P2P), kjer so posamezni računalniki medsebojno enakovredni. Prvi uporabljajo centralizirano skladišče, kjer se koda odjavi in vrne s spremembami, pri drugih pa gre za bolj decentraliziran pristop, kjer se koda pogosto posodablja iz enakovrednih virov, da jo ohranijo posodobljeno.

Apache Subversion (SVN) je brezplačna in odprtokodna programska oprema, ki jo je razvila Apache Software Foundation in deluje kot nadzorni sistem za sledenje sprememb datotek, map in imenikov. Uporablja se za pomoč pri obnavljanju podatkov in beleženju zgodovine sprememb, ki so se zgodile v nekem časovnem obdobju. Zasnovan je bil kot nadomestilo za sistem Concurrent Versions System (CVS), ki je vseboval številne prirojene napake in pomanjkljivosti funkcij. Razvoj SVN se je pričel leta 2000 in se je razvil kot odprtokodni projekt. Začetnika razvoja sta bila Karl Fogel in Ben Collins-Sussman. Vizija SVN sistema je biti centraliziran nadzorni sistem za upravljanje različic, ki deluje kot zanesljivo varno zavetje dragocenih podatkov. Prva stabilna različica Subversion 1.0 je bila izdana februarja 2004 (Collins-Sussman in drugi, 2009). To je pomenilo začetek širše uporabe tega sistema v razvojni skupnosti. Skozi leta je bil Subversion deležen številnih nadgradenj in izboljšav za izboljšanje delovanja, varnosti in funkcionalnosti. Subversion vključuje tudi orodja za upravljanje repozitorija, zaradi česar je uporaben za ekipe, ki delajo na večjih projektih (Nagel, 2005). Kljub široki uporabi v preteklih letih je Subversion izgubil priljubljenost v korist porazdeljenih sistemov za nadzor različic, kot sta Git in Mercurial. Ti sistemi so zaradi svoje večje učinkovitosti in fleksibilnosti postali bolj priljubljeni, predvsem za ekipe, ki delajo na večjih projektih ali projektih z več vejami (Collins-Sussman in drugi, 2008).

Git je brezplačna in odprtokodna programska oprema. Gre za porazdeljeni sistem za upravljanje različic, ki ga je razvil Linus Torvalds, znan po ustvarjanju operacijskega sistema Linux. Git je nastal zaradi potrebe po boljšem sistemu za upravljanje verzij znotraj razvojne

ekipe operacijskega sistema Linux. Takratni sistem Monotone enostavno ni zadostil potrebam Linusa in drugih razvijalcev. Linus Torvalds je aprila 2005 izdal prvo javno verzijo Gita. Razvil ga je s poudarkom na hitrosti, preprostosti in stabilnosti. Git je hitro pridobil na priljubljenosti zaradi svoje učinkovitosti in zmožnosti podpore velikim in kompleksnim projektom. Različni odprtokodni projekti in podjetja so začela uporabljati Git za upravljanje različic svojih programskih projektov. Git je postal standard v industriji razvoja programske opreme in številna podjetja, od majhnih, novoustanovljenih podjetij do velikih korporacij, ga uporabljajo za upravljanje svojih projektov. V preteklih letih je bil Git razvit in izboljšan s številnimi funkcijami in orodji, zaradi katerih je še bolj uporaben za razvijalce, vključno z razvejanjem, združevanjem, povrnitvijo nazaj, zgodovino sprememb in mnogimi drugimi. Njegova zgodovina priča o potrebi po boljšem sistemu za nadzor različic pri razvoju programske opreme in o tem, kako je inovacija Linusa Torvaldsa privedla do ustvarjanja orodja, ki je spremenilo način razvoja programske opreme (Chacon in Straub, 2014).

Mercurial je prav tako brezplačna in odprtokodna programska oprema in je podobno kot Git porazdeljen sistem za upravljanje različic in razvijalcem in razvojnim skupinam omogoča sledenje spremembam v njihovih projektih razvoja programske opreme skozi čas. Vsak uporabnik Mercuriala ima popolno kopijo celotnega repozitorija. To poveča hitrost in omogoča delo brez dostopa do interneta. Lahko se ga enostavno integrira z različnimi razvojnimi orodji in platformami, vključno z IDE (integriranimi razvojnimi okolji) in spletnimi platformami za gostovanje repozitorija. Razvoj Mercuriala je pričela Olivia Mackall v začetku leta 2005. Glavnina je razvita s programskim jezikom Python. Namen razvoja novega orodja za nadzor različic je bil nuditi podporo pri razvoju Linux Kernela, vendar so se kasneje odločili, da izberejo Git. Kljub temu je bil izbran za podporo pri razvoju znanih projektov, kot npr. Facebook, W3C in Mozilla (O'Sullivan, 2009).

Mercurial je priljubljen v določenih skupnostih, zlasti med uporabniki, ki imajo raje njegove porazdeljene funkcije ali jih že uporabljajo v svojih projektih.

Pomembno je omeniti, da čeprav je Mercurial zmogljivo orodje za nadzor različic, njegova priljubljenost ni dosegla ravni Gita, ki je danes najbolj razširjen distribuiran sistem za nadzor različic. Kljub temu Mercurial ostaja aktualen in se uporablja v različnih okoljih.

4.1.2 Predlog izbire najprimernejšega orodja na podlagi prednosti/slabosti

Izmed vseh možnih orodij za upravljanje programske kode, ki se pojavljajo na trgu, smo se osredotočili na tri najbolj pogosto zastopane v razvijalskih krogih, katerih prednosti in slabosti, kot jih prepoznavamo v povezavi z našim razvojnim okoljem, bodo odločale o dokončni izbiri. V nadaljevanju so, povzeto po (Cortuk, 2019), prikazane poglavitne prednosti in slabosti posameznega orodja za upravljanje programske kode.

Glavne prednosti SVN sistema so:

- centraliziran sistem – SVN uporablja centraliziran model za shranjevanje in upravljanje datotek. To pomeni, da je en sam centralni strežnik, ki vsebuje vse verzije datotek. To lahko v določenih primerih poenostavi upravljanje projektov in omogoča boljši nadzor nad verzijami;
- stabilnost – SVN je bil dolgo časa eden najbolj priljubljenih sistemov za verzioniranje in je znan po svoji stabilnosti in zanesljivosti;
- obsežna skupnost – SVN ima veliko skupnost uporabnikov in razvijalcev, kar pomeni, da lahko najdemo veliko informacij, orodij in dodatkov za podporo svojim projektom;
- vgrajene funkcije za nadzor dostopa – SVN omogoča natančen nadzor dostopa do datotek in projektov, kar je pomembno za zagotavljanje varnosti in zaščite občutljivih podatkov;
- enostavna vejitev in združevanje – SVN omogoča preprosto ustvarjanje vej in združevanje sprememb med vejami, kar je ključno za razvoj večjih projektov z več razvijalci.

Slabosti SVN sistema pa so:

- centraliziran model – čeprav je centraliziran model lahko prednost za nekatere projekte, lahko v nekaterih drugih predstavlja tudi omejitev, saj lahko povzroči težave, če centralni strežnik postane nedostopen ali počasen;
- zapletenost pri upravljanju vej – medtem ko SVN omogoča vejitev in združevanje, se lahko procesi vejitve in združevanja zdijo bolj zapleteni in manj intuitivni v primerjavi z nekaterimi drugimi sistemi;
- počasne operacije na velikih projektih – SVN lahko postane počasen pri izvajanju operacij, kot so posodobitve in združevanja, na velikih projektih z velikim številom datotek in sprememb;
- omejen ekosistem orodij – SVN ima manjši ekosistem orodij in storitev v primerjavi z nekaterimi drugimi sistemi za verzioniranje, kar lahko omeji izbiro orodij za integracijo v sam razvojni proces.

Glavne prednosti sistema Git so:

- distribuirano verzioniranje – Git je distribuiran sistem za verzioniranje, kar pomeni, da ima vsak razvijalec celotno zgodovino projekta na svojem lokalnem računalniku. To omogoča hitro delo brez internetne povezave in boljšo odzivnost;
- enostavna vejitev in združevanje – Git ima zmogljivo funkcijo za vejitve, ki omogoča, da razvijalci ustvarijo ločene veje za različne funkcionalnosti ali popravke napak. Nato lahko enostavno združijo svoje spremembe nazaj v glavno vejo. To olajša sodelovanje več razvijalcev in upravljanje različnih funkcionalnosti;
- hitrost – Git je znan po svoji hitrosti. Opravljanje operacij, kot so prenos delovne kopije repozitorija (angl. Checkout), oddajanje (angl. Commit), združevanje (angl. Merge) in primerjava (angl. Diff), je običajno zelo hitro;

- velika skupnost in ekosistem – Git ima ogromno skupnost razvijalcev in velik ekosistem orodij in storitev, kot so GitHub, GitLab in Bitbucket, ki omogočajo enostavno sodelovanje, gostovanje projektov in sledenje spremembam;
- vgrajena varnost – Git zagotavlja varnost in celovitost datotek s pomočjo algoritmov za preverjanje integritete.

Tudi Git ima nekaj slabosti:

- krivulja učenja – začetno učenje Gita se lahko zdi zahtevno, še posebej za začetnike. Zapletenost vejitev in ukazne vrstice lahko predstavlja izziv;
- velikost metapodatkov – Git shranjuje veliko metapodatkov o vsaki spremembi, kar lahko povzroči, da projekti s številnimi datotekami in zgodovino postanejo precej veliki.
- problemi s konflikti – včasih se lahko pojavijo konflikti pri združevanju vej, kar zahteva ročno reševanje;
- zloraba v večjem obsegu – zaradi svoje distribuirane narave Git omogoča več svobode pri spreminjanju zgodovine projekta, kar lahko povzroči težave, če razvijalci ne spoštujejo najboljših praks;
- brez vgrajenega nadzora dostopa – Git sam po sebi nima vgrajenega sistema za nadzor dostopa, kar pomeni, da se je potrebno zanašati na druge storitve ali orodja za nadzor dostopa, če je to potrebno.

Kljub tem slabostim je Git postal standard v razvoju programske opreme in ima številne prednosti, ki močno olajšajo sodelovanje in sledenje spremembam v razvojnih projektih.

Glavne prednosti sistema Mercurial so:

- enostavna uporaba – Mercurial je znan po svoji enostavni in intuitivni uporabniški izkušnji. Uporabniki, še posebej začetniki, se lahko hitro naučijo osnovnih operacij;
- distribuirano verzioniranje – Mercurial je distribuiran sistem za verzioniranje, podobno kot Git, kar omogoča vsakemu razvijalcu, da ima svojo lokalno kopijo celotnega projekta, kar povečuje odpornost in omogoča delo brez povezave;
- hitrost – Mercurial se ponaša s hitrimi operacijami za prenos, oddajanje, združevanje in primerjavo sprememb, kar omogoča hitro delo;
- vgrajena podpora za veje in združevanje – Mercurial ima robustno podporo za vejitve in združevanje sprememb, kar omogoča enostavno upravljanje različnih funkcionalnosti v projektu;
- dober nadzor dostopa – Mercurial omogoča nadzor dostopa z vgrajenimi mehanizmi, ki omejujejo dostop do določenih vej in datotek.

Slabosti sistema Mercurial so:

- zapletene operacije zgodovine – nekatere napredne operacije zgodovine (npr. Rebase) so v Mercurialu bolj zapletene v primerjavi z Git-om;

- manjša skupnost in ekosistem – Mercurial ima manjšo skupnost razvijalcev in manj razpoložljivih orodij in storitev v primerjavi z Git-om. To lahko omeji izbiro orodij za sodelovanje in gostovanje projektov;
- manjša priljubljenost – Git je postal »de facto« standard v svetu razvoja programske opreme, zato lahko Mercurial težje najde svoje mesto v velikih projektih ali organizacijah;
- omejen ekosistem storitev za gostovanje – medtem ko obstajajo nekatere storitve za gostovanje projektov, ki podpirajo Mercurial, je izbira manjša v primerjavi z gostovanjem na platformah, kot sta GitHub in GitLab, ki sta osredotočeni na Git.

V tabeli 1 je prikazana matrika ključnih lastnosti obravnavanih orodij za verzioniranje.

Tabela 1: Primerjava ključnih lastnosti obravnavnih orodij za verzioniranje

ORODJE / LASTNOST	Git	SVN	Mercurial
ARHITEKTURA	Porazdeljena	Centralizirana	Porazdeljena
ENOSTAVNOST UPORABE	Zmerna do težka, strma krivulja učenja	Enostaven, linearen pristop	Zmeren, bolj intuitiven kot Git
ZMOGLJIVOST IN HITROST	Odlično, še posebej pri velikih repozitorijih	Dobra, lahko je počasnejši z velikimi kodnimi bazami	Dobra, obvladuje tako majhne kot velike repozitorije
RAZVEJANJE IN ZDRUŽEVANJE	Učinkovito in brezhibno	Lahko je okoren	Učinkovit, podoben Git-u
PRILJUBLJENOST	Zelo priljubljena, obsežna podpora skupnosti	Še vedno pogosto uporabljen, manj kot Git	Manj priljubljen, vendar ima predano skupnost
ATOMARNO UVELJAVLJANJE SPREMEMB (angl. COMMIT)	Ne	Da	Da
NADZOR DOSTOPA	V osnovi ne, odvisno od platforme gostovanja (npr. GitHub)	Podroben nadzor dostopa	Odvisno od platforme gostovanja
PRIMERNOST	Idealen tako za majhne kot velike projekte	Najboljši za projekte, ki zahtevajo linearno zgodovino	Dober tako za majhne kot velike projekte

Vir: prirejeno po LinuxConcept (brez datuma).

Na podlagi prikazanih prednosti in slabosti posameznih orodij lahko sklepamo, da je temeljna razlika med orodjem SVN na eni strani in Git ter Mercurial na drugi v tem, da prvi podpira centraliziran način delovanja, druga dva pa porazdeljenega. V našem obravnavanem

razvojnem okolju prednosti porazdeljenega sistema za nadzor različic ne pridejo toliko do izraza, ker način razvoja aplikacij že zdaj poteka centralizirano. Razvojna ekipa deluje v skupnem razvojnem okolju, kjer so vloge in odgovornosti jasno opredeljene in razmejene in ni potrebe po omogočanju razvoja brez možnosti povezave.

Orodje za nadzor različic se ne bo uporabljalo zgolj za shranjevanje izvorne programske kode, ampak tudi za sledenje verzijam dokumentacije, kot so funkcionalne specifikacije, uporabniška navodila, sheme, risbe in podobno. Zato je centraliziranost, ki omogoča upravljanje podrobnega nadzora dostopnih pravic tako za datoteke kot tudi mape, v tem primeru prednost orodja SVN pred ostalima orodjema. Ta funkcionalnost omogoča, da lahko prenesemo (angl. Checkout) samo določene sklope repozitorija, ki so avtorizirani za dostop glede na funkcijo ali delovno mesto uporabnika. Poleg tega je Subversion znan po boljši podpori za upravljanje binarnih datotek, kar je za obravnavano okolje še posebej pomembno, saj razvoj v Oracle Forms razvojnem okolju vključuje veliko binarnih datotek aplikativnih modulov, kot so FMB, RDF in PLL. Vse to so dodatni razlogi, ki v našem primeru govorijo v prid uporabe SVN-ja pred Gitom in Mercurialom.

Eden izmed ključnih razlogov, da je SVN boljši pri upravljanju z binarnimi datotekami, je, da SVN že v osnovi uporablja centraliziran model, kar pomeni, da je v uporabi en sam strežnik, ki vodi repozitorij in nadzoruje vse spremembe (Mason, 2006). To lahko poenostavi upravljanje binarnih datotek, saj ni potrebno skrbeti za konflikte pri združevanju več različic, kot je pogosto pri porazdeljenih sistemih, kot sta Git in Mercurial. Kot drugo SVN ponuja robusten mehanizem za upravljanje dostopnih pravic, kar je ključno pri delu z binarnimi datotekami, saj omogoča natančno nadzorovanje, kdo lahko spreminja te datoteke.

Glede na omenjene zahteve in specifičnosti obravnavanega razvojnega okolja predlagamo izbiro Subversiona (SVN) kot orodje za upravljanje z verzijami izvornih datotek.

4.2 Razvoj lastnega programskega orodja z repozitorijem za shranjevanje metapodatkov kot podpora novemu sistemu za upravljanje konfiguracij programskih proizvodov

Ob upoštevanju dejstva, da je trenutni centralni repozitorij shranjen v Oracle Designer orodju, ki ga Oracle več ne razvija in je tudi ukinil podporo zanj, se je pojavilo vprašanje, ali bi bilo smiselno izkoristiti metapodatke iz obstoječega Oracle Designer repozitorija in razmisliti o izdelavi lastnega orodja za podporo sistemu UKPP za zagotovitev lastnega repozitorija metapodatkov s prenosom vseh informacij o različnih aplikacijskih področjih in aplikacijah, vseh baznih objektih, aplikativnih datotekah oz. modulih. Poleg tega bi omogočili dodatno razširitev repozitorija metapodatkov z informacijami o strankah in različnih namestitvenih okoljih pri njih. Iz analize trenutnega stanja razvojnega okolja je razvidno, da moramo, poleg splošnih funkcionalnosti, ki veljajo za vse stranke, zagotavljati tudi razvoj prilagojenih rešitev za stranko, ki jih druge stranke ne zahtevajo ali ne morejo uporabiti zaradi njihovih specifičnih zahtev, ki se nanašajo izključno na to stranko in za

katero smo razvili posebno funkcionalnost. Pomeni, da je na nek način potrebno zagotoviti tudi beleženje metapodatkov o teh posebej razvitih modulih in baznih objektih, ki veljajo samo za določene stranke. Tudi te informacije bomo vključili v načrt izdelave lastne programske rešitve z repozitorijem metapodatkov.

Izgradnja lastne programske rešitve za podporo celotnemu sistemu UKPP bi tako zajemala vzpostavitev centralnega repozitorija metapodatkov z naslednjimi seznami:

- seznamom vseh aplikacijskih področij in aplikacij, ki tvorijo celoto ERP sistema;
- seznamom vseh različnih tipov baznih objektov, ki morajo biti nameščeni pri implementaciji ERP sistema;
- seznamom vseh različnih tipov aplikativnih datotek, ki zajemajo tako vnosne obrazce, poročila kot tudi ostale tipe modulov;
- seznamom vseh strank, pri katerih so implementirane posamezne aplikacije, moduli in bazni objekti programske rešitve ERP;
- seznamom vseh različnih tipov namestitvenih okolij, ki so vezana na posamezne stranke, in omogočajo testiranje, razvoj in produkcijsko uporabo programske rešitve ERP pri stranki.

Načrt izdelave lastnega programskega orodja s centralnim repozitorijem metapodatkov kot podpora UKPP sistemu zajema pripravo podatkovno entitetnega relacijskega modela, ki bo omogočal izgradnjo baznega dela programske rešitve za shranjevanje in urejanje podatkov vseh prej navedenih seznamov. Prav tako se pripravi načrt za izdelavo vseh vnosnih obrazcev.

Nadalje se zagotovi povezljivost lastnega orodja z izbranim orodjem za verzioniranje izvornih datotek, pri čemer se upošteva in zagotovi vključitev vseh posebnosti, ki so bile za posamezno stranko razvite in implementirane.

Centralni repozitorij fizičnih datotek s programsko kodo, tako baznih objektov kot tudi aplikativnih datotek, se bo nahajal v izbranem orodju za verzioniranje izvornih datotek, predvidoma SVN.

S pomočjo metapodatkov iz seznama centralnega repozitorija lastnega sistema za podporo UKPP se bo z uporabo avtomatiziranih skriptov omogočila priprava ustreznega nabora datotek, ki bodo vključene v namestitveni paket za posodobitev aplikacij programske rešitve ERP pri posamezni stranki.

Avtomatizirani skripti bodo na podlagi prebranih informacij, ki so zabeležene v skupnem repozitoriju metapodatkov o posameznih baznih objektih in aplikativnih datotekah, zagotovili zajem tako splošnih modulov kot tudi vseh posebnosti za izbrano stranko. Na ta način se bo zelo zmanjšalo tveganje možnih človeških napak pri pripravi namestitvenega paketa.

Najprej se bo, ob pripravi nove izdaje programskega proizvoda, v repozitoriju orodja za verzioniranje programske kode (SVN) izvedla vejitev (angl. Branching) celotnega debla (angl. Trunk) in s tem zagotovila enolična določljivost posameznih verzij datotek oz. konfiguracijskih elementov, ki bodo vključeni v izdajo programske rešitve oz. v namestitveni paket.

Pred zagonom avtomatiziranih skriptov se bo izvedla odjava (angl. Checkout) veje (angl. Branch) iz SVN repozitorija na izbrano mesto na strežniku, od koder se potem izvede nabor vseh tistih datotek, ki so za izbrano stranko vključene v repozitoriju metapodatkov o baznih objektih in aplikativnih datotekah.

Vsak bazni objekt in tudi modul bo enolično določen s svojim nazivom in se bo lahko pojavljal zgolj enkrat v celotnem repozitoriju metapodatkov. Pripadal bo natanko eni aplikaciji ali enemu aplikacijskemu področju. Prav tako se bo v repozitoriju sistema za verzioniranje izvornih datotek (SVN) nahajala ena njegova fizična datoteka, ki bo nosila enak naziv, kot je zapisan v repozitoriju metapodatkov. Za posamezne tipe baznih objektov in modulov iz seznamov centralnega repozitorija se bodo uveljavile ustrezne končnice datotek, ki bodo fizično shranjene v sistemu za verzioniranje izvornih datotek (SVN).

V nadaljevanju predstavljamo seznam konfiguracijskih elementov (KE) datotek Oracle Forms razvojnega okolja za različne tipe modulov z njihovimi končnicami. Prikazane so tako izvorne kot tudi izvršilne datoteke, vendar slednjih ne bomo shranjevali v repozitorij sistema za verzioniranje izvornih datotek, saj jih je možno kadarkoli ustvariti iz izvornih datotek s pomočjo kompilacije:

a) Vnosni obrazci
– FMB (izvorna oz. source datoteka)
– FMX (izvršilna oz. executable datoteka)

b) Poročila
– RDF (izvorna in hkrati izvršilna datoteka)

c) Meniji
– MMB (izvorna oz. source datoteka)
– MMX (izvršilna oz. executable datoteka)

d) Apex moduli
– APX (izvorna oz. source datoteka).

Poleg osnovnih datotek modulov se pojavljajo tudi pomožne datoteke modulov:

a) Knjižnice (opcijsko pri FMB in RDF datotekah)

– PLL (izvorna oz. source datoteka)
– PLX (izvršilna oz. executable datoteka)

b) Datoteke za navigacijo
– NAV (zagotavlja navigacijo v določenih modulih)
– opcijska pri FMB datotekah

c) Datoteke dinamičnih prevodov (obvezno pri FMB datotekah)
– TRN (txt datoteka z insert SQL stavki)

d) Datoteke parametrov izpisov (obvezno pri RDF datotekah)
– PAR (txt datoteka z insert SQL stavki).

Na podoben način lahko opredelimo tudi konfiguracijske elemente datotek (KE) za različne tipe baznih objektov, vključno z njihovimi končnicami:

– TAB (txt datoteka z Data Definition Language (DDL) SQL stavki za izdelavo tabel)
– CON (txt datoteka z DDL SQL stavki za izdelavo omejitev)
– IND (txt datoteka z DDL SQL stavki za izdelavo indeksov)
– SEQ (txt datoteka z DDL SQL stavki za izdelavo zaporedij)
– DDL (txt datoteka z DDL SQL stavki za spreminjanje definicij obstoječih baznih objektov)
– DML (txt datoteka z Data Manipulation Language (DML) SQL stavki za vnos, izbris ali spremembo podatkov v tabelah)
– VW (txt datoteka z DDL SQL stavki za izdelavo pogledov)
– PA (txt datoteka z DDL SQL stavki za izdelavo paketov)
– TRG (txt datoteka z DDL SQL stavki za izdelavo prožilcev).

Vsa omenjena pravila bodo vključena v dokumentacijo ISO sistema za obvladovanje kakovosti v obliki organizacijskih predpisov (OP) ali navodil za delo (ND).

Poleg omenjenih aplikativnih datotek in datotek baznih objektov obstajajo za vsako posamezno aplikacijo tudi uporabniška navodila (trenutno v obliki Word datotek), ki se bodo prav tako shranjevala v repozitorij sistema za verzioniranje izvornih datotek (SVN). Na ta način se bo zagotovila usklajenost uporabniških navodil z aplikativnimi datotekami.

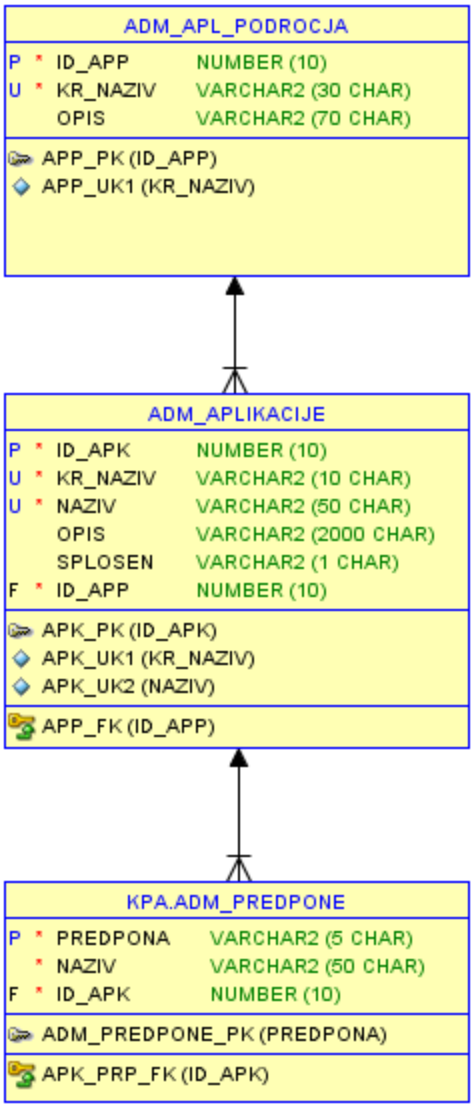
4.2.1 Načrt razvoja lastne aplikacije z repozitorijem metapodatkov

Preden pričnemo z razvojem nove aplikacije, je smiselno določiti in uporabiti enoten način poimenovanja vključenih elementov. Ker gre za aplikacijo, ki se bo uporabljala za administracijo in upravljanje metapodatkov razvojnega okolja, smo se odločili, da bomo vse tabele in bazne objekte, ki jih bomo ustvarili v okviru te nove aplikacije, opremili s predpono

»ADM_« in nato nadaljevali z izrazom, ki se nanaša na vsebino, ki jo bo tabela oz. katerikoli drug element vsebovala. Na podoben način bomo poimenovali tudi aplikativne module.

V nadaljevanju predstavljamo relacijske modele podatkov, ki bodo služili kot načrt za vzpostavitev centralnega repozitorija metapodatkov vseh prej navedenih seznamov. Na relacijskih modelih so prikazane tabele s povezavami med njimi. Posamezna tabela, prikazana v obliki pravokotnika, ima na vrhu zapisan naziv, spodaj so navedeni atributi tabele in pod njimi ključi, ki določajo enoličnost zapisov v tabeli. Kjer je pred atributom navedena črka P, pomeni, da je atribut vključen v primarni ključ (angl. Primary Key). Če je navedena črka U, potem je atribut vključen v edinstven oz. unikaten ključ (angl. Unique Key). Kadar pa je črka F, takrat pomeni, da je atribut vključen v tuj ključ (angl. Foreign Key). Pri opisovanju diagramov se bomo osredotočili zgolj na opis atributov, ključev in ostalih podrobnosti tabele ne bomo opisovali.

Slika 4: Relacijski model strukture Aplikacijskih področij, Aplikacij in Predpon



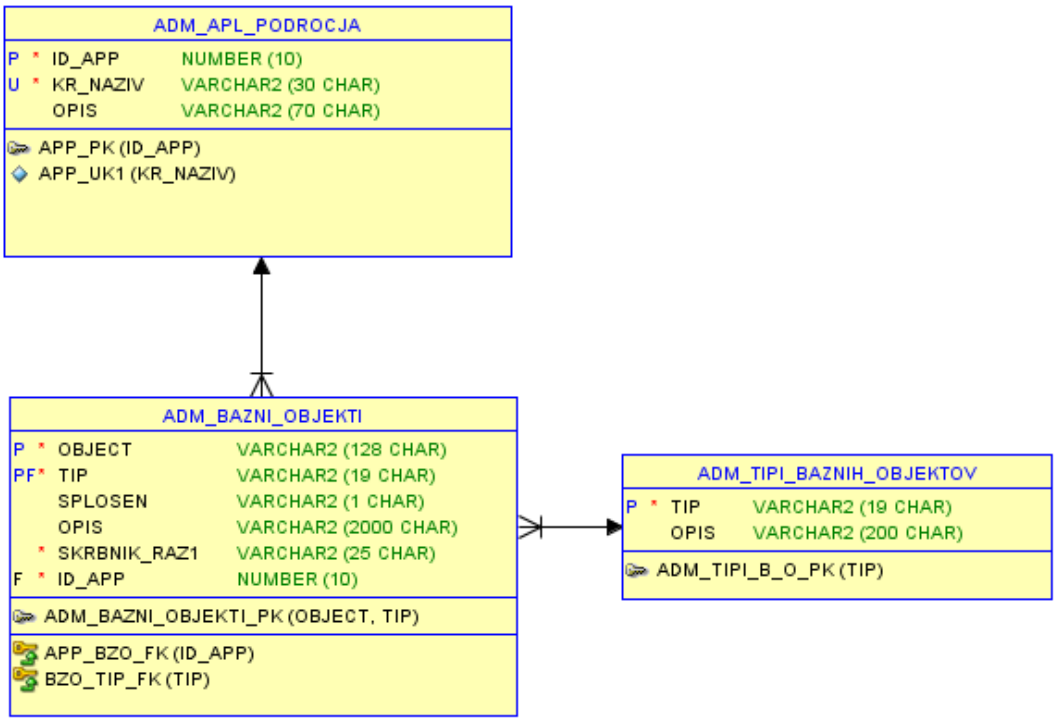
Vir: lastno delo.

Na sliki 4 je prikazan relacijski model s seznamom tabel, ki bodo omogočale shranjevanje podatkov o hierarhični strukturi aplikacij, ki so vključene v ERP sistem. V tabeli ADM_APL_PODROCJA, ki vsebuje atribute ID_APP – števec zapisa, KR_NAZIV – naziv aplikacije, OPIS – opis aplikacije, se bodo shranjevali podatki o aplikacijskih poslovnih področjih, ki predstavljajo najvišji nivo v sklopu ERP sistema. Naslednji nivo predstavljajo poslovne aplikacije, katerih podatki se bodo zapisovali v podrejeni tabeli ADM_APLIKACIJE. Več jih lahko pripada posameznemu aplikacijskemu področju. V tabeli je poleg osnovnih atributov ID_APK – števec zapisa, SIF_APK – šifra aplikacije, NAZIV – naziv aplikacije, OPIS – opis aplikacije, dodan tudi atribut SPLOSEN, ki določa, ali je aplikacija razvita za vse stranke ali kot posebnost za določeno stranko. Za potrebe vzpostavitve seznama poenotenega poimenovanja vseh elementov sistema za UKPP se na tabelo aplikacij poveže tabela ADM_PREDPONE, v katero bomo shranjevali vse kratice (atribut PREDPONE), ki se lahko pojavljajo kot predpone pri poimenovanju različnih objektov, katerih vsebina se nanaša na dotično aplikacijo.

Postavljena struktura aplikacijskih področij in aplikacij bo osnova za določitev nadaljnjih seznamov, v katere bomo vključili tako bazne objekte kot tudi aplikativne module.

Podatke o baznih objektih bomo vezali direktno na aplikacijska področja, ker jih ni smiselno podrobneje deliti po aplikacijah, saj so medsebojno odvisni in se jih združene tudi namešča v podatkovno bazo na skupno shemo.

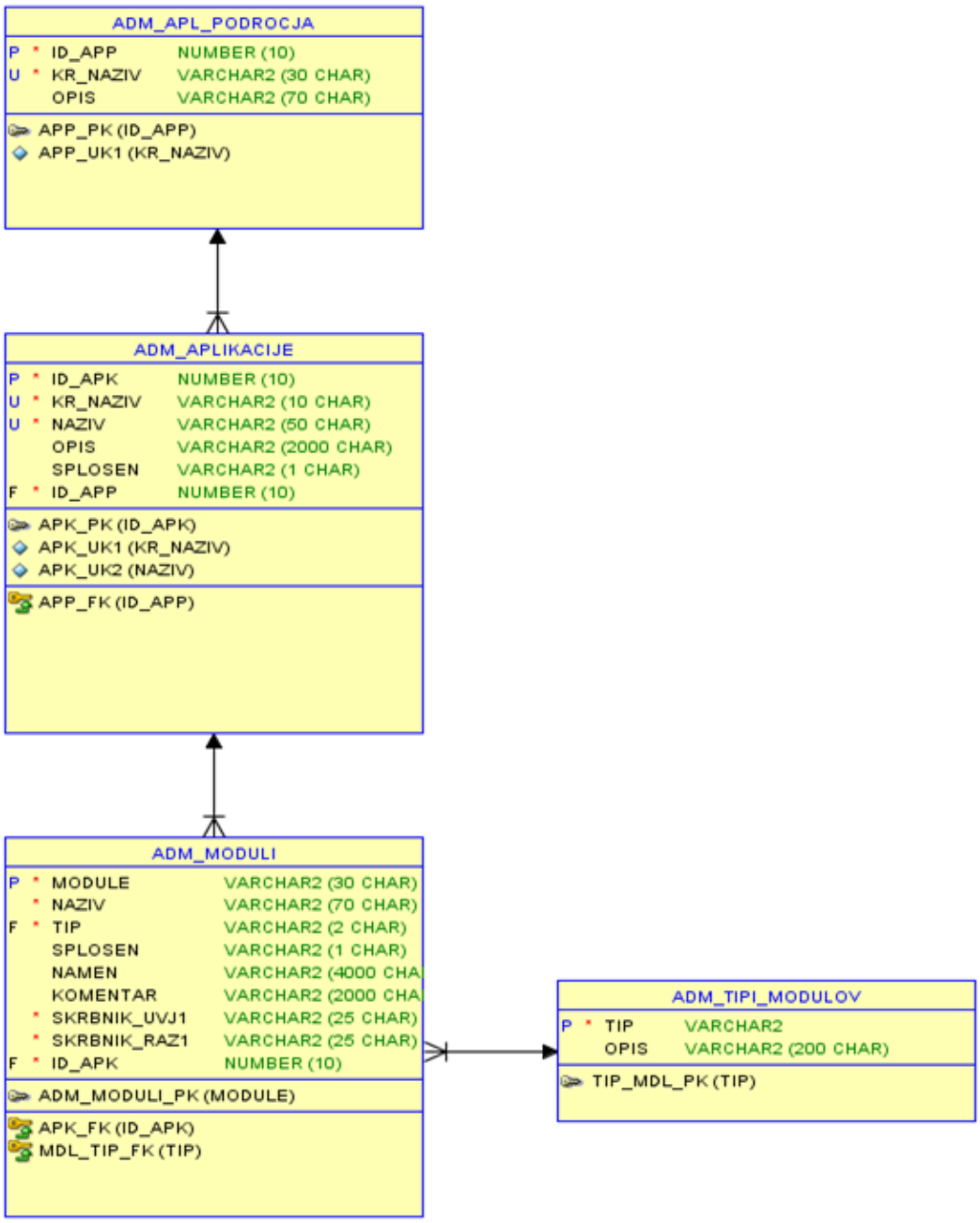
Slika 5: Relacijski model strukture Aplikacijskih področij, Baznih objektov in njihovih Tipov



Vir: lastno delo.

Na sliki 5 je prikazan relacijski model, ki povezuje seznam aplikacijskih področij (tabela ADM_APL_PODROCJA) s seznamom baznih objektov (tabela ADM_BAZNI_OBJEKTI), katerih lastnosti so v osnovi opredeljene s tipom baznega objekta (atribut TIP povezan s tabelo ADM_TIPI_BAZNIH_OBJEKTOV, ki se v strukturi pojavlja kot iskalna tabela), skrbnikom razvijalcem (atribut SKRBNIK_RAZ1) in določitvijo, ali gre za splošen bazni objekt ali pa gre za posebnost in je bazni objekt implementiran zgolj pri nekaterih strankah (atribut SPLOSEN). Atribut OPIS je namenjen dodatnemu širšemu opisu baznega objekta.

Slika 6: Relacijski model strukture Aplikacijskih področij, Aplikacij, Aplikacijskih modulov in njihovih Tipov

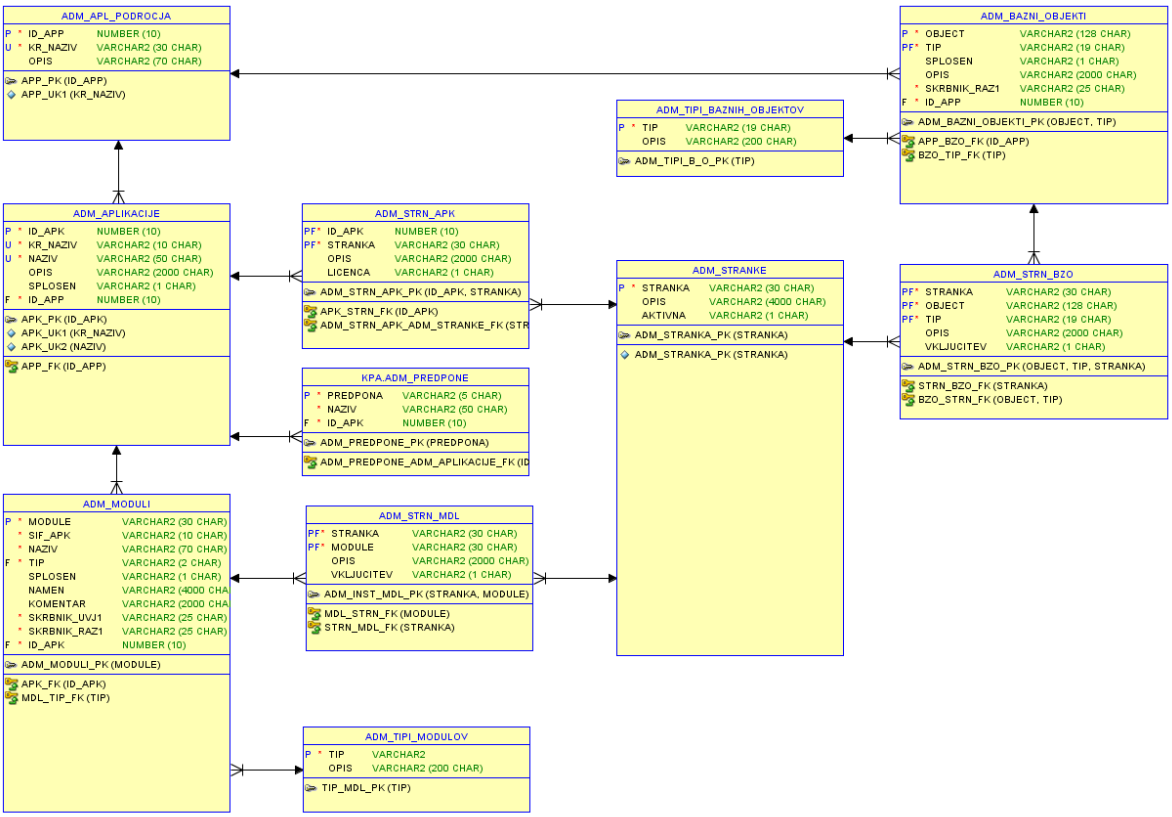


Vir: lastno delo.

Pri opredelitvi seznama aplikativnih modulov se bomo v strukturi vezali na seznam aplikacij, torej nivo nižje v hierarhiji, kot smo to naredili pri baznih objektih. Posamezni element aplikativnih modulov se bo lahko pojavljal zgolj in samo na eni sami aplikaciji.

Na sliki 6 je prikazana struktura tabel, ki prikazuje umestitev aplikativnih modulov (tabela ADM_MODULI) v strukturo aplikacij (tabela ADM_APLIKACIJE), pri čemer se ravno tako uporablja lastnost tip modula (atribut TIP povezan s tabelo ADM_TIPI_MODULOV, ki se v strukturi pojavlja kot iskalna tabela) in določa za kakšno vrsto aplikativnega modula gre. Modul je lahko na primer vnosni obrazec, klasično poročilo, generiran izpis, PL/SQL obdelava, Apex obrazec itn. Poleg opisnih atributov NAMEN in KOMENTAR vsebuje tudi atribut SKRBNIK_UVJ1, ki lahko določa, kdo je vsebinski skrbnik modula. Podobno kot pri baznih objektih se pojavljata tudi v tej tabeli atributa SPLOSEN in SKRBNIK_RAZ1 z enako funkcionalnostjo.

Slika 7: Relacijski model strukture seznamov Stranke, Aplikacije, Aplikativni moduli in Bazni objekti



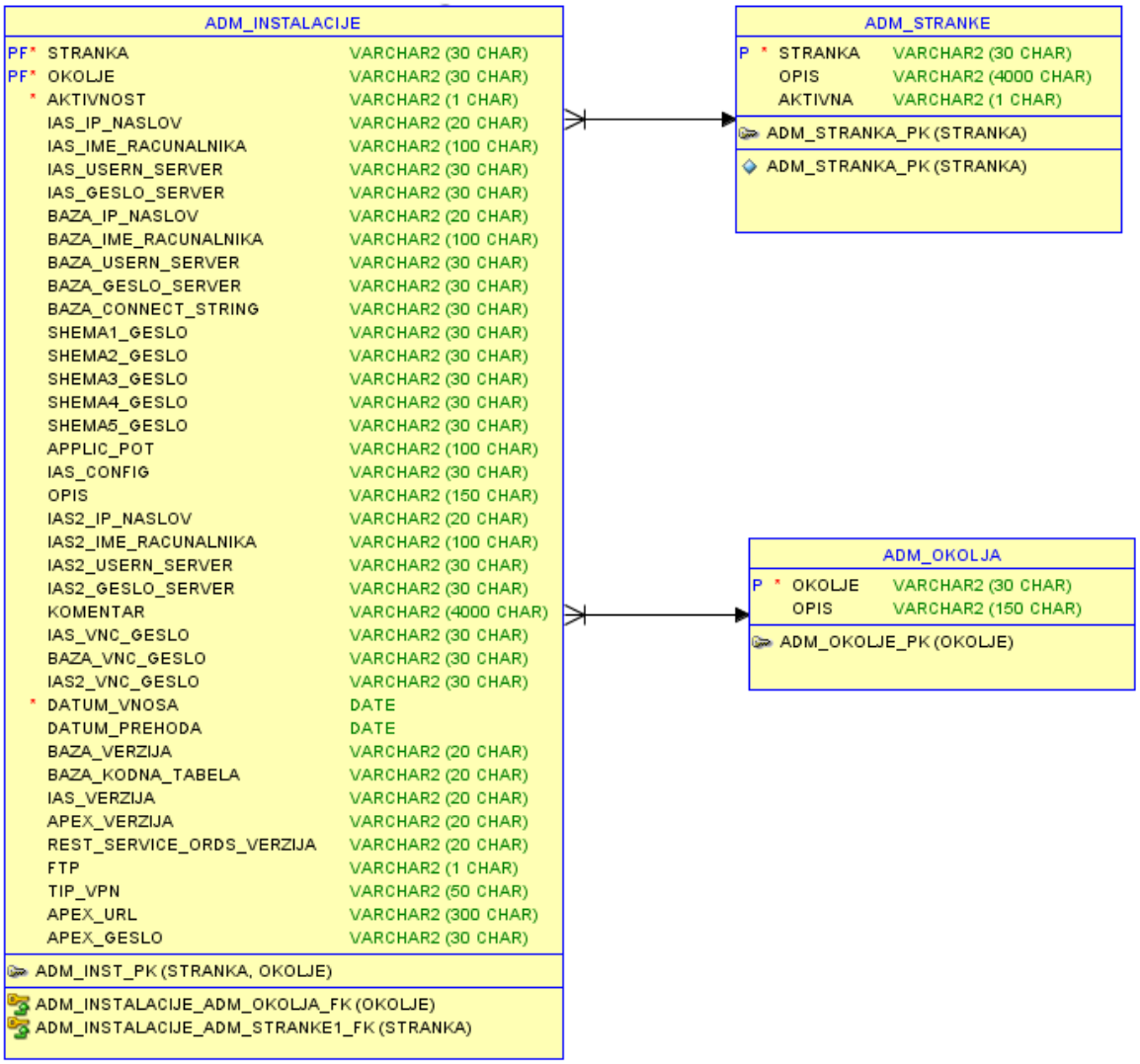
Vir: lastno delo.

Na sliki 7 je prikazan relacijski model, ki prikazuje umestitev seznama Strank (tabela ADM_STRANKE), pri katerih je programski proizvod nameščen, pri čemer se seznam Strank povezuje v prvi vrsti z Aplikacijami (tabela ADM_APLIKACIJE), ki jih bo posamezna stranka imela nameščene v svojem okolju, nato pa dodatno še s seznamom

Aplikativnih Modulov (tabela ADM_STRN_MDL) in seznamom Baznih Objektov (tabela ADM_STRN_BZO), pri čemer se praviloma na posamezno stranko povezuje samo tiste module in bazne objekte, ki so označeni kot posebnost za določeno stranko. Na ta način bo pri pripravi konfiguracije možno nedvoumno opredeliti, kateri bazni objekti in aplikativni moduli bodo nameščeni v izbrano okolje pri stranki.

Pri vsaki stranki se pred namestitvijo poslovno-informacijskega sistema ERP pripravi vsaj testno in produkcijsko okolje, v nekaterih primerih pa tudi replikacijo produkcijskega okolja. Kjer stranke izvajajo samostojni razvoj, je potrebno upoštevati tudi razvojno okolje. Zaradi tega je potrebno vedeti, v katero okolje pri stranki se bo izvedla namestitev in ustrezno pripraviti seznam namestitvenih okolij oz. instalacij, kamor bomo vnašali podatke o njih.

Slika 8: Relacijski model strukture Stranke, Instalacije in Okolja



Vir: lastno delo.

Slika 8 prikazuje relacijski model podatkov, ki ilustrira povezavo med seznamom Instalacij (tabela ADM_INSTALACIJE), povezanim s seznamom Strank (tabela ADM_STRANKE) ter seznamom predhodno pripravljenih Tipov okolij za stranko (tabela ADM_OKOLJA), ki so pred namestitvijo programske rešitve pripravljeni pri stranki.

V tabeli ADM_INSTALACIJE bomo tako vodili različne informacije o namestitvenih okoljih, ki jih bomo lahko vključili v avtomatizirane skripte za pripravo namestitvenih paketov. V nadaljevanju predstavljamo samo najpomembnejše atribute, saj se število različnih atributov, ki jih je smiselno vključiti v to tabelo, lahko razlikuje glede na posebnosti posameznega razvojnega okolja.

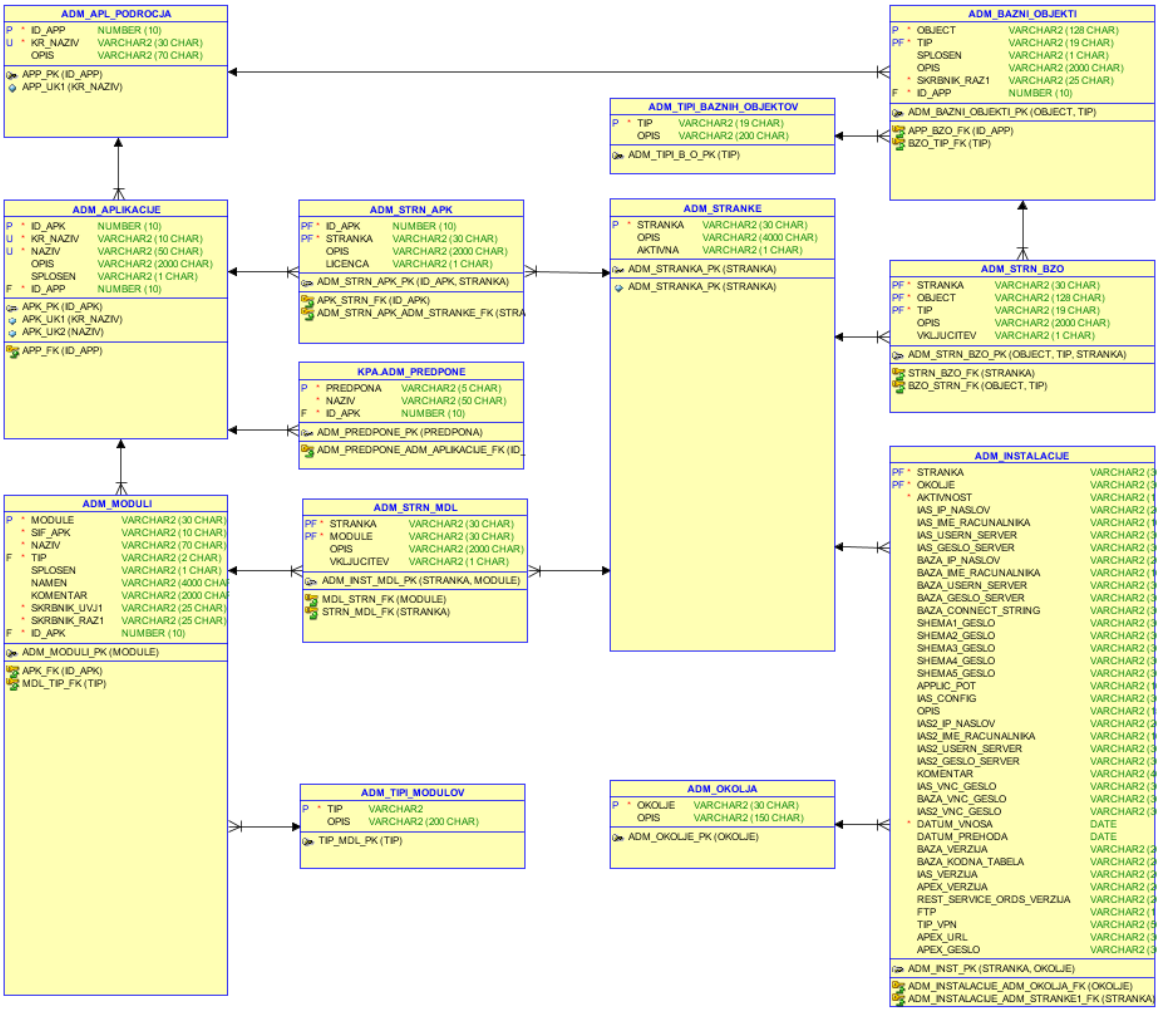
Med drugim bodo v tabeli shranjene informacije o IP naslovih IAS strežnikov in tudi baznih strežnikov, skupaj z uporabniškimi imeni in gesli, kar bo omogočilo avtomatizirano prijavo na posamezna okolja.

Prav tako bodo shranjena gesla različnih baznih shem, ki se v bazi podatkov pojavljajo kot lastnice baznih objektov (atributi SHEMAx_GESLO), kar bo omogočilo avtomatsko prijavo in namestitev baznih objektov na ustrezne sheme. Ker različne stranke zahtevajo različne načine dostopa do svojih omrežij, se bo v tabelo shranjevalo tudi različne tipe dostopov preko virtualnih zasebnih omrežij (angl. VPN – Virtual Private Network), kar posledično zahteva lokalno namestitev in uporabo različnih VPN odjemalcev za vzpostavitev povezave do strežnikov pri stranki. Določene stranke še vedno omogočajo protokol za prenos datotek (angl. FTP – File Transfer Protocol) in tudi Virtual Network Computing (VNC) dostop, zato bo v tabeli navedena tudi ta informacija.

Pri pripravi namestitvenih paketov je zelo pomembno vedeti, katera verzija Oracle Forms IAS strežnika, katera verzija ORACLE podatkovne baze in Apex-a je nameščena v izbranem okolju pri posamezni stranki. Na podlagi teh informacij se izvede ustrezna kompilacija izvornih datotek, ker so lahko v nekem časovnem trenutku verzije omenjenih orodij različne od stranke do stranke in v nekaterih primerih tudi od okolja do okolja pri isti stranki. Tudi te informacije bodo shranjene v tabeli ADM_INSTALACIJE. Za potrebe zagona aplikacij, razvitih z Oracle Apex razvojnim orodjem, bomo v tabelo shranjevali tudi Apex URL naslov in APEX geslo. Če se bo kadarkoli v prihodnosti pojavila potreba po shranjevanju dodatnih informacij v zvezi z namestitvenimi okolji, bomo lahko tabelo ADM_INSTALACIJE razširili z dodatnimi atributi. To je ena izmed velikih prednosti, ki jo prinaša lasten razvoj programske rešitve za podporo sistema za UKPP.

Slika 9 prikazuje celoten relacijski model podatkov z vsemi tabelami, ki smo jih predhodno obravnavali. Na podlagi tega relacijskega modela bomo nadaljevali s pripravo načrta za izdelavo vnosnih obrazcev in pregledov. S pomočjo teh obrazcev in pregledov bo možno učinkovito upravljati z metapodatki o konfiguracijskih elementih, ki so vključeni v posamezno konfiguracijo programskega proizvoda.

Slika 9: Celoten relacijski model strukture seznamov Aplikacijskih področij, Aplikacij s Predponami, Aplikacijskih modulov in njihovih Tipov, Baznih objektov in njihovih Tipov, Strank, Instalacij in Okolij



Vir: lastno delo.

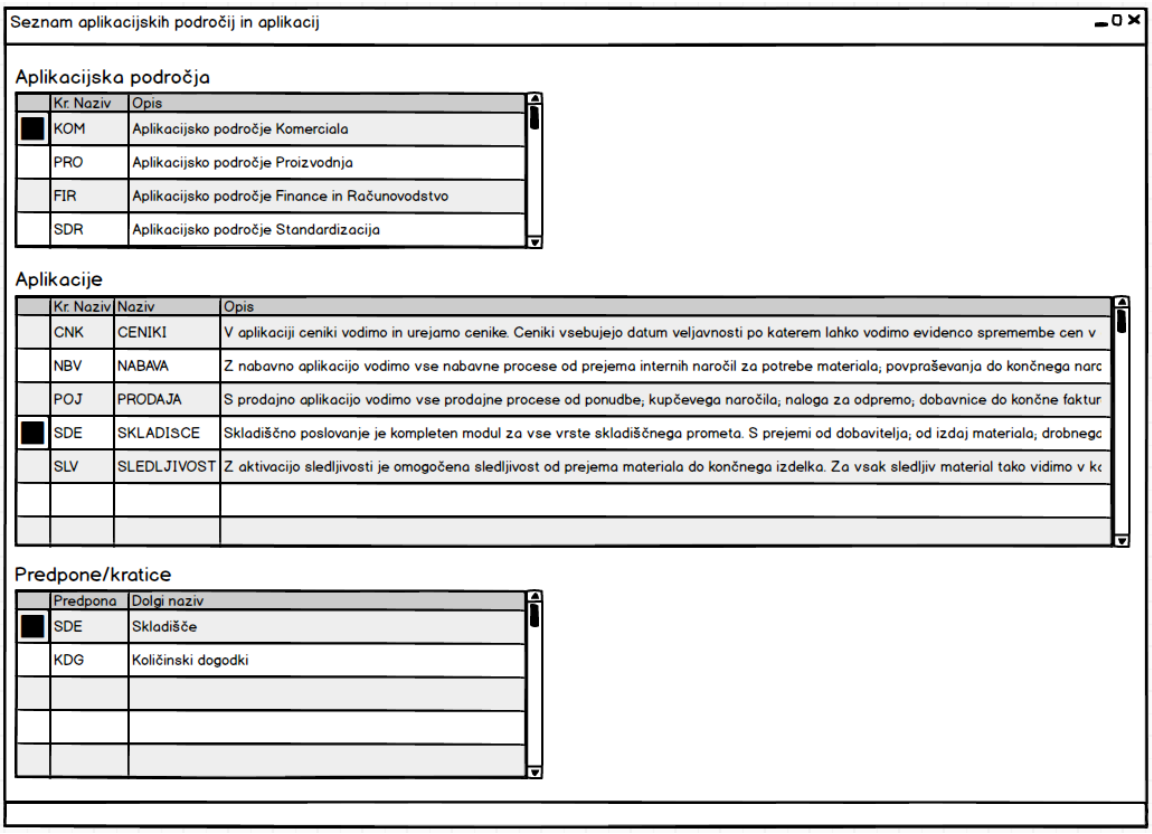
Po pripravi relacijskega modela podatkov sledi načrt izdelave vnosnih obrazcev oziroma modulov, s katerimi bomo omogočili upravljanje s podatki, ki bodo shranjeni v prej prikazani strukturi tabel.

Na sliki 10 je prikazana struktura oz. model (angl. Mockup) vnosnega obrazca, pri katerem je v najvišjem bloku omogočeno upravljanje s podatki seznama Aplikacijskih področij, ki je kot glava (angl. Master), povezan z blokom postavk (angl. Detail) seznama Aplikacij.

Na tretjem nivoju se nahaja seznam Predpon, ki predstavlja blok postavk, povezan s seznamom Aplikacij, ki v tej povezavi predstavlja njihovo glavo. Konkretno lahko iz obrazca razberemo, da ima blok s podatki o Aplikacijskih področjih v prvem stolpcu, ki označuje trenutno označeni zapis, postavljen črn kvadratek na zapisu s kratkim nazivom KOM. Na drugem bloku, kjer so podatki o Aplikacijah, so tako izpisane Aplikacijskemu

področju KOM podrejene aplikacije CNK – CENIKI, NBV – NABAVA, POJ – PRODAJA, SDE – SKLADIŠČE, SLV – SLEDLJIVOST. Ker je črn kvadratek na drugem bloku na zapisu SDE – SKLADIŠČE, se posledično na tretjem bloku poimenovanem Predpone/kratice prikazujejo podatki o predponah in kraticah, ki so podrejene Aplikaciji SDE – SKLADIŠČE.

Slika 10: Vnosni obrazec za urejanje strukture Aplikacijskih področij, Aplikacij in Predpon



Vir: lastno delo.

Na sliki 11 je prikazan model vnosnega obrazca, pri katerem je v najvišjem bloku omogočeno upravljanje s podatki seznama Aplikacijskih področij, ki je kot glava povezan z blokom postavk, kjer se ureja seznam Aplikacij.

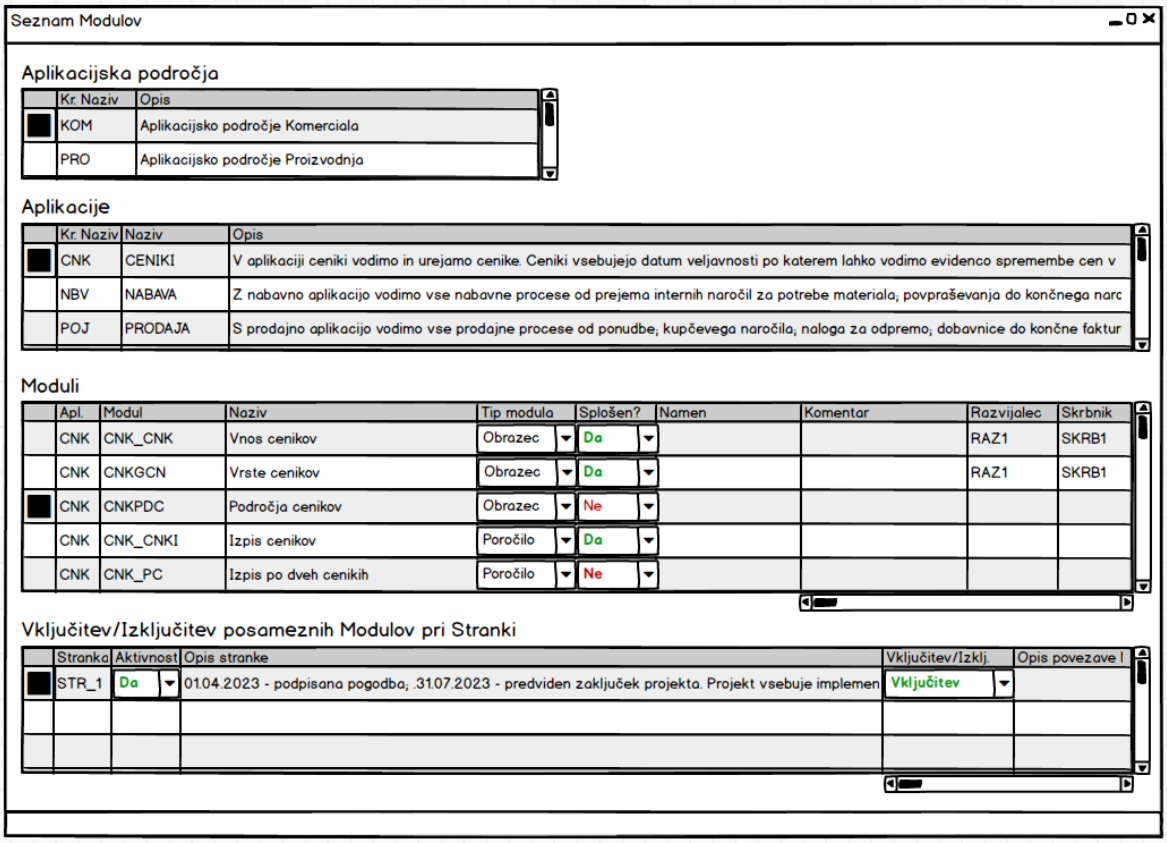
Na tretjem nivoju se nahaja seznam Modulov, ki je kot blok postavk povezan s seznamom Aplikacij iz drugega bloka in v tej povezavi omogoča enolično dodelitev Modula natanko eni sami Aplikaciji. Na najnižjem nivoju vnosnega obrazca se nahaja še blok Stranke, ki je povezan z blokom Modulov in omogoča vključitev ali izključitev posameznega Modula za izbrano Stranko.

Konkretno lahko iz tega obrazca razberemo, da ima blok s podatki o Aplikacijskih področjih v prvem stolpcu, ki označuje trenutno označen zapis, postavljen črn kvadratek na zapisu s kratkim nazivom KOM. Na drugem bloku, kjer so podatki o Aplikacijah, so tako izpisane Aplikacijskemu področju KOM podrejene aplikacije CNK – CENIKI, NBV – NABAVA,

POJ – PRODAJA. Ker je črn kvadratek na drugem bloku na zapisu CNK – CENIKI, se posledično na tretjem bloku, poimenovanem Moduli, prikazujejo podatki o modulih, ki so podrejeni Aplikaciji CNK – CENIKI. Iz zapisa, ki je označen s črnim kvadratkom, lahko razberemo, da gre za modul s kratkim nazivom CNKPDC in nazivom Področja cenikov. Gre za modul tipa vnosni obrazec, ki je bil razvit kot posebnost za točno določeno stranko, zaradi česar je polje Splošen? označeno z vrednostjo Ne. Iz četrtega bloka, kjer so prikazani podatki o vključitvi posameznih Modulov pri Stranki, lahko razberemo, da je bil modul CNKPDC razvit kot posebnost za stranko z nazivom STR_1, zaradi česar je polje Vključitev/Izključitev označeno z vrednostjo Vključitev. Če bi obstajal nek modul, ki bi bil razvit kot posebnost za neko stranko, in bi ga ustvarili in preimenovali kot kopijo nekega drugega splošnega modula (tega splošnega pa stranka ne bi želela uporabljati), potem bi to kopijo osnovnega splošnega modula Vključili na to stranko, osnovnega splošnega pa Izključili.

S tem mehanizmom imamo možnost Vključitve/Izključitve kateregakoli modula za katerokoli stranko in posledično tudi vključitev oz. izključitev iz konfiguracije za to stranko.

Slika 11: Vnosni obrazec za urejanje seznama Modulov oz. aplikativnih datotek



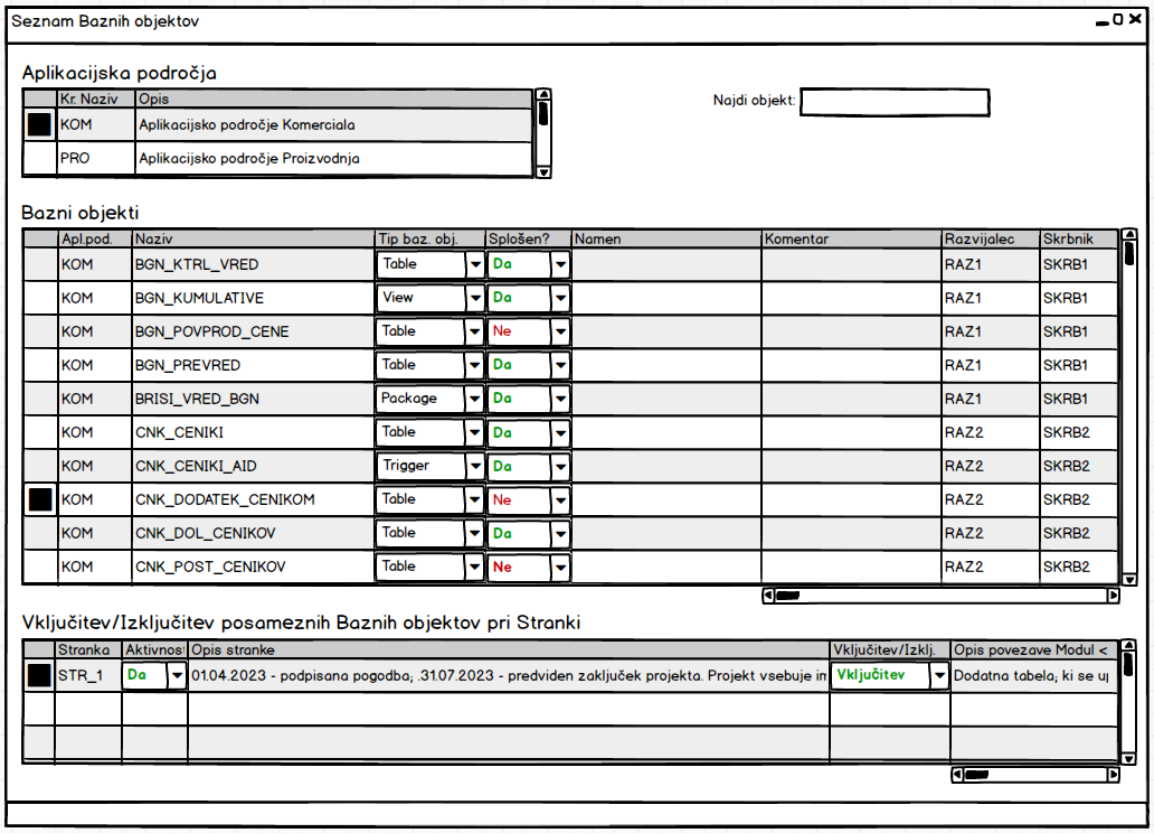
Vir: lastno delo.

Podobno kot pri vnosnem obrazcu za urejanje podatkov seznama Modulov imamo na sliki 12 prikazan model vnosnega obrazca za urejanje podatkov seznama Baznih objektov. Razlika je zgolj v tem, da seznam Baznih objektov povežemo direktno s seznamom

Aplikacijskih področij in ne s seznamom Aplikacij. Torej, posamezni Bazni objekt je enolično dodeljen natanko enemu Aplikacijskemu področju. Tudi pri Baznih objektih lahko vključujemo ali izključujemo katerikoli Bazni objekt za katerokoli Stranko. Mehanizem deluje na enak način kot pri Modulih. Konkretno lahko iz tega obrazca razberemo, da ima blok s podatki o Aplikacijskih področjih v prvem stolpcu, ki označuje trenutno označen zapis, postavljen črn kvadratek na zapisu s kratkim nazivom KOM. Na drugem bloku, kjer so podatki o Baznih objektih, so tako izpisani Aplikacijskemu področju KOM podrejeni Bazni objekti:

- BGN_KTRL_VRED, ki je tip baznega objekta Table,
- BGN_KUMULATIVE, ki je tip baznega objekta View,
- BGN_POVPROD_CENE, ki je tip baznega objekta Table,
- BGN_PREVRED, ki je tip baznega objekta Table,
- BRISI_VRED_BGN, ki je tip baznega objekta Package,
- CNK_CENIKI, ki je tip baznega objekta Table,
- CNK_CENIKI_AID, ki je tip baznega objekta Trigger,
- CNK_DODATEK_CENIKOM, ki je tip baznega objekta Table,
- CNK_DOL_CENIKOV, ki je tip baznega objekta Table,
- CNK_POST_CENIKOV, ki je tip baznega objekta Table.

Slika 12: Vnosni obrazec za urejanje seznama Baznih objektov

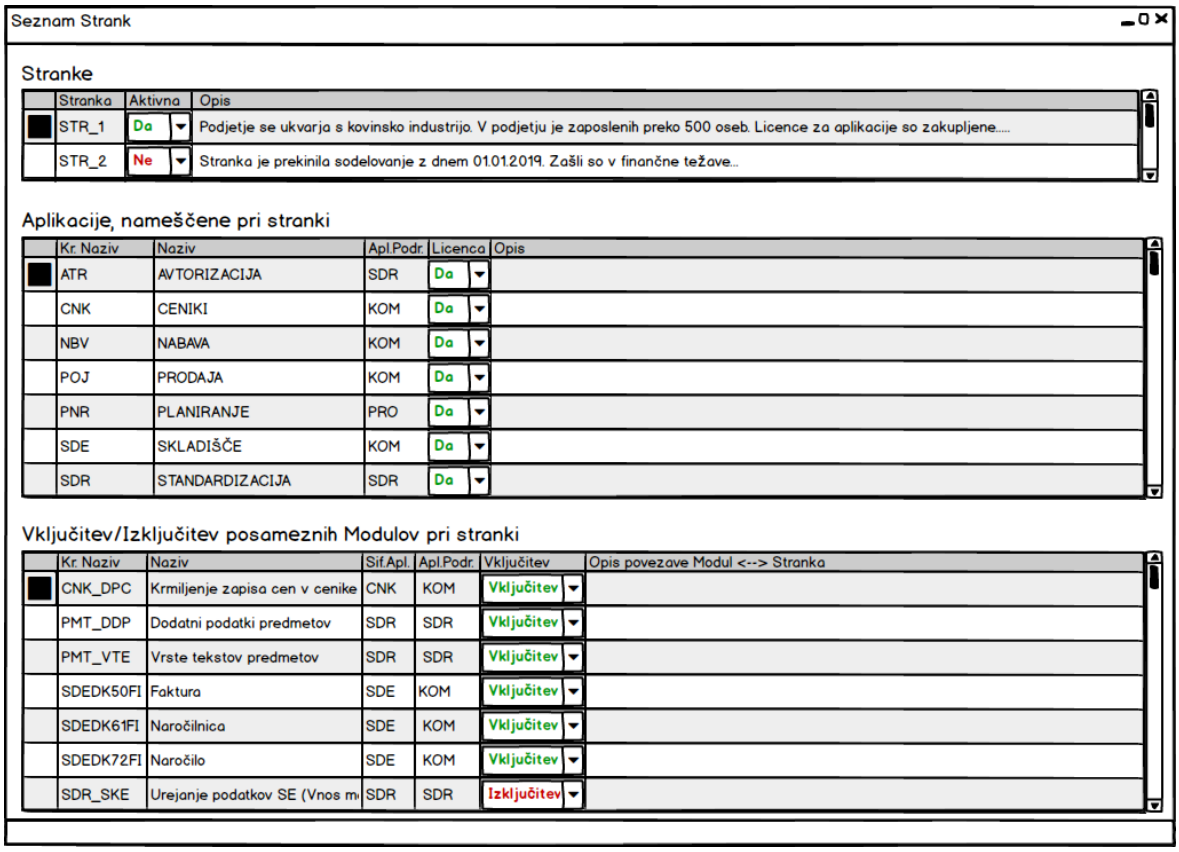


Vir: lastno delo.

Kjer je na drugem bloku na zapisu CNK_DODATEK_CENIKOM črn kvadratek, lahko iz podatkov razberemo, da gre za bazni objekt tipa Table, ki je bil razvit kot posebnost za točno določeno stranko, zaradi česar je polje Splošen? označeno z vrednostjo Ne.

Iz tretjega bloka, kjer so prikazani podatki o Vključitvi/Izključitvi posameznih Baznih objektov pri Stranki, lahko razberemo, da je bil Bazni objekt CNK_DODATEK_CENIKOM razvit kot posebnost za stranko z nazivom STR_1, zaradi česar je polje Vključitev/Izključitev označeno z vrednostjo Vključitev.

Slika 13: Vnosni obrazec za urejanje seznama Strank z Aplikacijami in Moduli



Vir: lastno delo.

Slika 13 nam prikazuje model vnosnega obrazca, ki povezuje seznam Strank s seznamom Aplikacij, ki jih ima posamezna Stranka nameščene, oz. je do njih upravičena.

Na tretjem, najnižjem bloku, se nahaja prikaz seznama Modulov, ki jih ima Stranka bodisi vključene bodisi izključene iz konfiguracije. Torej se ta blok povezuje z blokom Strank in ne z blokom Aplikacij.

V tem modelu imamo tako en sam blok (seznam Strank), ki predstavlja glavo dvema blokoma postavk (seznam Aplikacij in seznam vključenih oz. izključenih Modulov).

Konkretno lahko iz tega obrazca razberemo, da ima blok s podatki o Strankah v prvem stolpcu, ki označuje trenutno označen zapis, postavljen črn kvadratek na zapisu z nazivom Stranke STR_1. Na drugem bloku, kjer so podatki o Aplikacijah, so tako izpisane Stranki STR_1 dodeljene Aplikacije:

- ATR – AVTORIZACIJA,
- CNK – CENIKI,
- NBV – NABAVA,
- POJ – PRODAJA,
- PNR – PLANIRANJE,
- SDE – SKLADIŠČE,
- SDR – STANDARDIZACIJA.

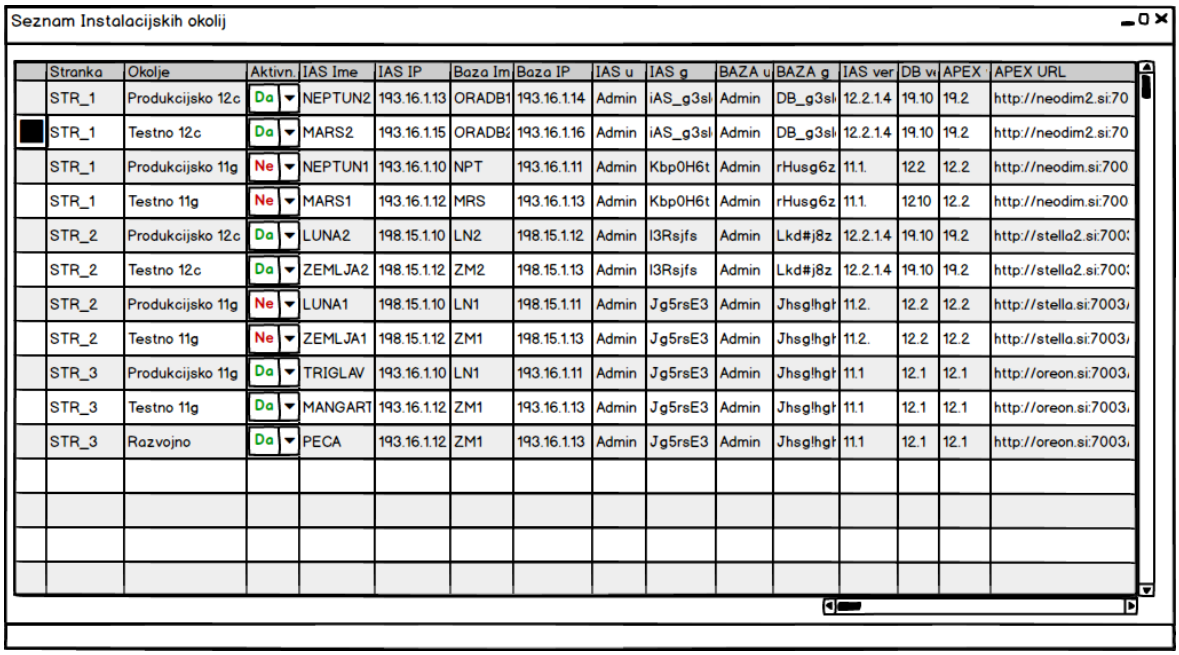
Ker je tretji blok neposredno dodeljen prvemu, so na njem izpisani Moduli, ki so Vključeni ali Izključeni na Stranko z nazivom STR_1:

- CNK_DPC – Krmiljenje zapisa cen v cenike,
- PMT_DDP – Dodatni podatki predmetov,
- PMT_VTE – Vrste tekstov predmetov,
- SDEDK50FI – Faktura,
- SDEDK61FI – Naročilnica,
- SDEDK72FI – Naročilo,
- SDR_SKE – Urejanje podatkov SE, ki je tudi edini od naštetih modulov Izključen za stranko STR_1.

Za kasnejšo izdelavo avtomatiziranih skriptov, ki bodo omogočali pripravo namestitvenega paketa in avtomatsko izvedbo namestitve v izbrano okolje pri stranki, je potrebno zagotoviti seznam vseh pred pripravljenih okolij s podatki o IAS in DB (angl. DataBase) strežnikih, vključno s podatki o verzijah in uporabniških imenih z gesli.

Na sliki 14 je prikazan model vnosnega obrazca s primeri podatkov o različnih okoljih pri strankah. Okolja, ki sčasoma postanejo neaktivna zaradi novih verzij Oracle Forms, se ustrezno označijo s stolpcem Aktivnost z vrednostjo Ne. Ker je črn kvadratek na zapisu Stranke STR_1 z Okoljem Testno 12c, lahko iz tega razberemo, da gre za testno okolje pri stranki STR_1, ki ima nameščeno okolje verzije Oracle Forms 12c. Gre za aktivno okolje, katerega IAS strežnik nosi naziv MARS2, njegov IP naslov pa je 193.16.1.15. Naziv baznega strežnika je ORADB2 z IP naslovom 193.16.1.16. Na IAS strežnik se lahko prijavimo z uporabniškim imenom Admin in geslom iAS_g3_slo, na bazni strežnik pa z uporabniškim imenom Admin in geslom DB_g3_slo. Natančna verzija Oracle Forms je 12.2.1.4, verzija Oracle baze 19.10 in verzija Apex 19.2. Na obrazcu je prikazan še Apex URL naslov za dostop do Apex aplikacij.

Slika 14: Vnosni obrazec za urejanje seznama namestitvenih Okolij pri Stranki



Vir: lastno delo.

Naveden seznam vnosnih obrazcev bo tako predstavljal lastno aplikacijo, ki bo omogočala zajem metapodatkov vseh konfiguracijskih elementov programskega proizvoda. Na podlagi teh podatkov bo možno s pomočjo avtomatiziranih skriptov pripraviti seznam konfiguracijskih elementov, vključno s podatki za posamezno okolje pri stranki, kamor bomo nameščali datoteke v obliki namestitvenih paketov. Za shranjevanje verzij posameznega konfiguracijskega elementa bo vpeljano orodje za verzioniranje izvorne programske kode (SVN). Struktura repozitorija v izbranem orodju za verzioniranje (SVN), kamor se bodo shranjevale fizične datoteke, bo postavljena po enakem principu, kot je postavljena hierarhična struktura repozitorija metapodatkov. Pri tem bo znotraj osnovnega debla (angl. Trunk) postavljena struktura map na podlagi seznama Aplikacijskih področij, kjer bodo moduli in bazni objekti v obliki posameznih tipov datotek predstavljali množico vseh elementov iz seznama vseh Aplikacij za zadevno Aplikacijsko področje.

Za pripravo lokalne kopije repozitorija SVN bo uporabljena aplikacija Tortoise SVN, ki bo nameščena na vse računalnike aplikativnih inženirjev, ki bodo avtorizirani za dostop do sistema za verzioniranje programske kode SVN. V začetni fazi bo pripravljeno testno okolje s testnim repozitorijem, kjer bo možno testno izvajati vse glavne funkcije sistema za verzioniranje: odjava (angl. Checkout), posodobitev (angl. Update), oddajanje (angl. Commit), primerjava (angl. Diff), združevanje (angl. Merge).

Z izbiro orodja za verzioniranje (SVN) in izdelavo aplikacije za shranjevanje metapodatkov o konfiguracijskih elementih bomo občutno izboljšali obstoječi sistem za UKPP. Če se ozremo na štiri funkcionalna področja procesa upravljanja konfiguracij programskega proizvoda, lahko ugotovimo, da smo z opredelitvijo konfiguracijskih elementov razvojnega

okolja Oracle Forms, skupaj z baznimi objekti in pripravo načrta za izdelavo aplikacije, ki bo omogočala strukturirano shranjevanje enolično določenih konfiguracijskih elementov, uspešno pokrili fazo identifikacije konfiguracije.

Drugo fazo, nadzor nad spremembami, bomo učinkovito pokrili z nadaljevanjem izvajanja procesa uveljavljanja sprememb in nadzora konfiguracije. Vse faze procesa uveljavljanja sprememb in nadzora konfiguracije so zagotovljene z uporabo interno razvite aplikacije, ki temelji na Microsoft Dynamics CRM platformi in omogoča prijavo servisnih zahtevkov v obliki projektov. Grafični prikaz delovanja procesa smo predstavili z uporabo BPMN in bo služil kot osnova za prikaz delovanja procesa vsem obstoječim in novo zaposlenim sodelavcem.

Fazo spremljanja stanja konfiguracije bomo omogočili z uporabo interno razvite aplikacije za shranjevanje metapodatkov o vseh elementih konfiguracije, ki bo vključevala uporabo orodja za verzioniranje izvorne kode (SVN), v katerem bo vzpostavljen repozitorij fizičnih datotek z enako strukturo kot repozitorij metapodatkov. S tem bomo omogočili vzdrževanje natančnih informacij o konfiguracijskih elementih, verzijah, spremembah ter drugih pomembnih podatkih. Omogočen bo pregled nad trenutnim stanjem programskega proizvoda, kar je osnova za uspešno spremljanje napredka, ocenjevanje tveganj ter vzdrževanje sledljivosti in doslednosti med verzijami in spremembami.

S pomočjo avtomatiziranih skriptov, ki bodo omogočili pripravo izdaje programskega proizvoda in sledenje konfiguracijskim elementom skozi celotni življenjski cikel programskega proizvoda, bomo zagotovili možnost presojanja konfiguracije tako fizičnih lastnosti kot tudi funkcionalnih značilnosti posameznih konfiguracijskih elementov, kar je zahteva faze presojanja konfiguracije.

Za vsako fazo procesa UKPP bodo izoblikovana pravila in navodila za delo, ki bodo vključena v ISO dokumentacijo sistema kakovosti. Načrtujemo tudi izobraževanje vseh zaposlenih, ki bodo aktivno sodelovali pri uporabi sistema za UKPP. Pripravljen bo seznam delovnih mest, ki bodo upravičena do avtoriziranega dostopa do sistema za UKPP prek navedenih orodij. Vzpostavljene bodo ustrezne vloge z dodeljenimi pravicami, ki bodo omogočile dostop do repozitorijev glede na vsebino, do katere ima posamezno delovno mesto avtoriziran dostop. Z jasno definirano politiko dostopa in določitve postopkov, kdo, kaj in kam lahko namešča nove verzije, bomo dvignili nivo varnosti.

4.3 Ekonomika uvedbe predlaganega novega sistema za upravljanje konfiguracij programskih proizvodov

Ekonomika uvedbe se nanaša na analizo stroškov in koristi, ki jih prinaša implementacija nove programske rešitve v organizaciji ali podjetju. Ta analiza je ključnega pomena, saj pomaga odločevalcem razumeti, ali je naložba v novo programsko opremo smiselna in ali bo prinesla pričakovane koristi (Hass, 2003).

Te odločitve sprejema vodstvo podjetja kot ločena raven odločevalcev, ki morajo odobriti ali zavrniti naložbene odločitve glede vzpostavitve sistema za UKPP. Vodstvo praviloma zanima, kaj se dogaja z razvojem lastnih izdelkov, predvsem glede kakovosti in izvedenimi storitvami za stranko. Skrbijo predvsem za finančne koristi in stroške. Uporaba sistema za UKPP zagotavlja konkurenčno prednost, saj omogoča hitrejše izdaje programskih proizvodov ali skrajšanje časa, potrebnega za odgovor na zahteve in poizvedbe končnega uporabnika. Vse te koristi so težko ekonomsko merljive, a lahko igrajo ključno vlogo pri odločitvah o izvajanju UKPP v podjetju. Poleg tega je potrebno upoštevati tudi številne neposredne stroške, kot so licence za orodje, stroški usposabljanja za vse osebje podjetja, stroški vzdrževanja (Borraci, 2005).

Vsebino ekonomske analize uvedbe programske rešitve UKPP predstavljajo tako stroški kot koristi. Lahko vključujejo naslednje stroške ali zahteve (UpGuard Team, 2021):

- stroške nabave strojne opreme (mrežna infrastruktura, strežniki),
- stroške nabave orodja za vzpostavitev sistema za UKPP (pri nakupu komercialnega orodja so lahko pomembni dodatni parametri kot vrsta licence ali predvideno število uporabnikov orodja) ali stroške razvoja lastne programske rešitve,
- stroške implementacije in konfiguracije orodja za vzpostavitev sistema za UKPP, vključno z integracijo z obstoječimi orodji,
- stroške izobraževanja obstoječih zaposlenih in kasneje novih sodelavcev;
- nekoliko povečan obseg dela zaradi vpeljav dodatnih aktivnosti v proces razvoja, vključno s spremljanjem in poročanjem o konfiguracijah sistema za UKPP.

Koristi pa so:

- povečana učinkovitost razvoja,
- zmanjšanje števila prijavljenih napak,
- skrajšanje časa za odpravo prijavljenih napak,
- skrajšanje časa za pripravo namestitvenih paketov zaradi uporabe avtomatiziranih skriptov,
- izboljšana sledljivost in nadzor,
- izboljšana konkurenčnost zaradi hitrejšega razvoja programskih proizvodov.

Pretvorba pričakovanih koristi uvedbe sistema za UKPP v številčne kazalnike za ekonomsko analizo zahteva kvantifikacijo teh koristi. To pomeni, da moramo izraziti koristi v merljivih enotah, kot so denar, čas ali druge relevantne metrike. Glede na to, da pričakujemo povečano učinkovitost razvojnega procesa, lahko uvedemo časovno metriko, s pomočjo katere lahko primerjamo čas, ki smo ga uporabili za določeno aktivnost znotraj izvajanja procesa UKPP, pred in po uvedbi novega sistema za UKPP. Recimo, da smo pred uvedbo sistema za UKPP za odpravo napake potrebovali povprečno 5 ur, zdaj pa z uvedbo UKPP samo še 2 uri. To pomeni, da smo prihranili 3 ure pri vsaki odpravi napake. Druga metrika bi lahko predstavljala odstotek zmanjšanja števila prijavljenih napak. Pred uvedbo sistema za UKPP

smo imeli na primer na leto prijavljenih 10 večjih napak, po uvedbi pa pričakujemo samo še 3, kar predstavlja 70 % zmanjšanje napak.

UKPP je disciplina, ki je bistveno odvisna od konteksta uporabe. Možne koristi in zadevni faktor zaslužka se lahko razlikujejo glede na različne primere uporabe. Vsled tega je nemogoče ustvariti popolno formulo, ki bi veljala za vse možne primere. Ker je UKPP subjektivna disciplina, lahko uporabimo le subjektivni model ROI, katerega vrednost mora podjetje samo izračunati, pri tem pa upoštevati kontekst, v katerem se uporablja sistem za UKPP, ter način izvajanja discipline (Borraci, 2005).

V nadaljevanju bomo, na podlagi subjektivne ocene, prikazali primer izračuna donosnosti naložbe (angl. ROI – Return on Investment) vzpostavitve sistema za UKPP za obdobje petih let s prilagoditvijo modela in uporabo formul za izračun (Boracci, 2005).

ROI je metrika, ki jo uporabljamo za oceno smiselnosti predlagane naložbe in časovni okvir za povračilo vložka v podjetju. ROI naj bi nam pomagal ugotoviti, v kolikšnem času se bo donosnost naložbe v sistem za UKPP povrnila (Borraci, 2005).

Najprej bomo izračunali enkratne skupne stroške uvedbe sistema (TCu) za UKPP in skupne prihranke (TS) v povezavi z uvedbo sistema za UKPP, ki jih pričakujemo v posameznem letu.

Kot je razvidno iz enačbe (1), so v našem primeru skupni stroški uvedbe (TCu) enaki vsoti stroškov orodja (Co), stroškov implementacije in konfiguracije (Cic) ter stroškov izobraževanja osebja (Ce).

$$TCu = Co + Cic + Ce \tag{1}$$

Skupne prihranke (TS) izračunamo kot vsoto prihrankov pri učinkovitosti razvoja (Su), prihrankov pri stroških popravil in vzdrževanja (Spvz) in prihrankov zaradi avtomatizacije priprave namestitvenih paketov (Sa), kar je razvidno iz enačbe (2).

$$TS = Su + Spvz + Sa \tag{2}$$

Za izračun sedanje vrednosti prihodnjega denarnega toka bomo uporabili diskontni faktor (DF), ki ga izračunamo s pomočjo diskontne stopnje (r) in števila let (n) v prihodnosti na način, kot prikazano v enačbi (3).

$$DF = \frac{1}{(1-r)^n} \tag{3}$$

Sedanjo vrednost skupnih prihrankov (PVTS) na letnem nivoju za vsako posamezno leto izračunamo tako, da pomnožimo skupne prihranke (TS) z diskontnim faktorjem (DF) za izbrano leto (m). Izračun je prikazan v enačbi (4).

$$PVTS_m = TS_m * DF_m \tag{4}$$

Na podoben način bomo izračunali sedanjo vrednost tekočih stroškov (PVCt). Tekoče stroške na letnem nivoju (Ct) bomo pomnožili z diskontnim faktorjem (DF) za posamezno leto (m), kot je prikazano v enačbi (5).

$$PVCt_m = Ct_m * DF_m \tag{5}$$

Neto sedanjo vrednost prihrankov in stroškov (NPV) za posamezno leto izračunamo tako, da od sedanje vrednosti skupnih prihrankov (PVTS) odštejemo skupne stroške uvedbe (TCu) in sedanjo vrednost tekočih stroškov (PVCt) za izbrano leto (m), kar prikazuje enačba (6).

$$NPV_m = PVTS_m - TCu_m - PVCt_m \tag{6}$$

Ker na začetku investicije še nimamo skupnih prihrankov (TS), niti tekočih stroškov (Ct), je začetna neto sedanja vrednost prihrankov in stroškov (NPV) enaka skupnim stroškom uvedbe (TCu) in izražena z negativno vrednostjo.

Za izračun kumulativne neto sedanje vrednosti prihrankov in stroškov (CNPV) za izbrano leto (m) uporabimo enačbo (7), kjer neto sedanji vrednosti prihrankov in stroškov (NPV) izbranega leta (m) prištejemo kumulativno neto sedanjo vrednost prihrankov in stroškov (CNPV) preteklega leta (m-1).

$$CNPV_m = NPV_m + CNPV_{m-1} \tag{7}$$

Vse dokler bo kumulativna neto sedanja vrednost prihrankov in stroškov preteklega leta (CNPV) negativna in višja od neto sedanje vrednosti prihrankov in stroškov (NPV) izbranega leta, bo kumulativna neto sedanja vrednost prihrankov in stroškov (CNPV) za izbrano leto ostala negativna.

Primer izračuna ROI za 5-letno obdobje:

Za potrebe prikaza izračuna ROI za 5 letno obdobje bomo, na podlagi preteklih izkušenj in lastne subjektivne ocene, predpostavili naslednje vrednosti stroškov in prihrankov na nivoju enoletnega razvoja programskih proizvodov v izbranem podjetju:

- skupni stroški razvoja na letnem nivoju pred uvedbo sistema za UKPP: 500.000,00 €,
- skupni stroški (TCu) uvedbe sistema za UKPP: 80.000,00 €,
- tekoči stroški (Ct) na letnem nivoju (povečan obseg dela zaradi vpeljave novih aktivnosti v proces razvoja): 1.000,00 €,
- skupni prihranki (TS) po uvedbi sistema za UKPP na letnem nivoju: 50.000,00 € (vključeni prihranki pri učinkovitosti razvoja (Su) + prihranki pri stroških popravil in vzdrževanja (Spvz) + prihranki zaradi avtomatizacije priprave namestitvenih paketov (Sa)).

Zaradi poenostavitve izračuna v danem primeru predvidevamo, da bodo skupni prihranki (TS) po uvedbi sistema za UKPP na letnem nivoju predstavljali 10 % od skupnih stroškov razvoja na letnem nivoju pred uvedbo sistema za UKPP.

V našem primeru bomo izbrali diskontno stopnjo 4 %, ki jo najprej pretvorimo v decimalno obliko 0,04 in nato za vsako leto posebej izračunamo diskontni faktor, kot je prikazano v tabeli 2.

Tabela 2: Izračun diskontnega faktorja za petletno obdobje

	r	n	DF (1/1-r)^n
PRVO LETO	0,04	1	0,9615
DRUGO LETO	0,04	2	0,9246
TRETJE LETO	0,04	3	0,8890
ČETRTO LETO	0,04	4	0,8548
PETO LETO	0,04	5	0,8219

Vir: lastno delo.

V tabeli 3 je prikazan izračun v sedanjo vrednost diskontiranih kumulativnih neto prihrankov in stroškov (CNPV) in ostalih vrednosti za izračun ROI po posameznih letih za obdobje petih let. V 10 vrstici je prikazan tudi izračun obdobja, ki bo preteklo do povrnitve investicije. Iz izračuna lahko ugotovimo, da bo investicija povrnjena po 1 letu in 265 dneh.

Kot je razvidno iz enačbe (8), je mogoče ROI izračunati kot kvocient med kumulativno neto sedanjo vrednostjo prihrankov in stroškov (CNPV) in vsoto skupnih stroškov uvedbe (TCu) ter kumulativne sedanje vrednosti tekočih stroškov (CPVCt), pomnoženo s 100 %.

$$ROI = \left(\frac{CNPV}{TCu+CPVCt}\right) * 100\ \% \quad (8)$$

Če vnesemo ustrezne vrednosti iz tabele 3 v enačbo za izračun ROI za obdobje petih let, dobimo naslednji rezultat:

ROI = (138.138,20 € / (80.000,00 € + 4.451,80 €)) x 100 % = 163,57 %

V tem primeru je ROI za petletno obdobje 163,57 %, kar pomeni, da je naložba v uvedbo sistema za UKPP precej donosna, čeprav je kumulativna neto sedanja vrednost prihrankov in stroškov (CNPV) v prvem in večjem delu drugega leta negativna.

Tabela 3: Izračun kumulativne neto sedanje vrednosti prihrankov in stroškov ter ostalih vrednosti za izračun ROI za petletno obdobje

Z.št.	ANALIZA STROŠKOV IN PRIHRANKOV	LETO 0	LETO 1	LETO 2	LETO 3	LETO 4	LETO 5	SKUPAJ
1	**Skupni prihranki (TS)**		50.000,00 €	50.000,00 €	50.000,00 €	50.000,00 €	50.000,00 €	
2	**Diskontni faktor (DF)**	1	0,9615	0,9246	0,8890	0,8548	0,8219	
3	**Sedanja vrednost skupnih prihrankov (PVTS)**	0,00 €	48.075,00 €	46.230,00 €	44.450,00 €	42.740,00 €	41.095,00 €	222.590,00 €
4	**Skupni stroški uvedbe (TCu)**	80.000,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	80.000,00 €
5	**Tekoči stroški (Ct)**		1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	1.000,00 €	
6	**Diskontni faktor (DF)**	1	0,9615	0,9246	0,889	0,8548	0,8219	
7	**Sedanja vrednost tekočih stroškov (PVCt)**		961,50 €	924,60 €	889,00 €	854,80 €	821,90 €	4.451,80 €
8	**Neto sedanja vrednost prihrankov in stroškov (NPV)**	-80.000,00 €	47.113,50 €	45.305,40 €	43.561,00 €	41.885,20 €	40.273,10 €	
9	**Kumulativna neto sedanja vrednost prihrankov in stroškov (CNPV)**	-80.000,00 €	-32.886,50 €	12.418,90 €	55.979,90 €	97.865,10 €	138.138,20 €	
10	**Obdobje odplačila investicije:**	1 leto + 32.886,5 / (32.886,5 + 12.418,90) = 1 + 0,7256 = 1 leto + (0,7256 * 365 dni) ali 1 leto in 265 dni						
11	**5 letni ROI:**	(222.590,00 - (80.000 + 4.451,80)) / (80.000 + 4.451,80) = 1,6357 * 100 % = 163,57 %						

Vir: lastno delo.

5 SKLEP

Na podlagi preučevanja teoretičnih osnov in standardov, ki obravnavajo proces UKPP, smo analizirali obstoječe razvojno okolje v izbranem podjetju. Ob upoštevanju ugotovljenih spoznanj smo prišli do zaključka, da obstaja prostor za izboljšave obstoječega sistema za UKPP. Obravnavali smo možnost izbire primernega orodja in vpeljave sistema za verzioniranje izvornih datotek, kar bo omogočilo vzpostavitev repozitorija konfiguracijskih elementov v obliki fizičnih datotek, ki predstavljajo celotno konfiguracijo programskega proizvoda. S tem bomo nadomestili pomanjkljivost trenutnega repozitorija orodja Oracle Designer, ki sam nima vgrajenega sistema za verzioniranje posameznih elementov. Na osnovi funkcionalnosti in metapodatkov iz obstoječega centralnega repozitorija orodja Oracle Designer smo pripravili načrt za izdelavo lastne aplikacije. Ta aplikacija bo nadomestila obstoječi repozitorij in omogočila pripravo lastnega repozitorija metapodatkov o vseh konfiguracijskih elementih programskega proizvoda. S pomočjo avtomatiziranih skriptov bomo lahko na podlagi informacij iz repozitorija metapodatkov kadarkoli pripravili seznam vseh konfiguracijskih elementov za vsako posamezno stranko, pri čemer bodo upoštevane vse posebnosti, ki so zanje implementirane in dokumentirane. Seznam teh elementov bo služil za avtomatsko pripravo namestitvenih paketov, ki bodo vsebovale fizične datoteke iz repozitorija sistema za verzioniranje izvornih datotek. Zaradi uvedenega avtomatizma pričakujemo občutno zmanjšanje števila napak pri nameščanju posodobljenih različic programske opreme, kar bo povečalo zaupanje v proces nameščanja programske opreme ter enostavnejše uveljavljanje sprememb, kar bo povzročilo hitrejše razvojne cikle. Hkrati pričakujemo občutno zmanjšanje količine ročnega dela in posledično sprostitev zasedenosti človeških virov ter hkrati minimiziranje možnih človeških napak, ki nastajajo pri ročnem izvajanju opravil.

Z uvedbo sistema za verzioniranje izvornih datotek in možnostjo avtomatizirane vejitve pred pripravo izdaje različice programskega proizvoda za posamezno stranko bomo omogočili nedvoumno določitev verzije vsakega posameznega elementa konfiguracije in verzije programskega proizvoda kot celote.

V osnovnem deblu sistema za verzioniranje izvornih datotek se bodo nahajale vse datoteke kot fizična kopija konfiguracijskih elementov, zabeleženih v repozitoriju metapodatkov. V mislih imamo tako splošne elemente, do katerih so upravičene vse stranke, kot tudi vse posebnosti, ki bodo razvite in ustrezno zabeležene v repozitoriju metapodatkov na izbrano stranko ali stranke, pri katerih bo posebna funkcionalnost implementirana. Za enolično določljivost posameznih konfiguracijskih elementov bodo skrbela pravila, ki jih bomo v največji možni meri implementirali v samo aplikacijo preko omejitev in kontrol knjiženja in hkrati že v samo relacijsko strukturo tabel, kamor se bodo metapodatki shranjevali.

Pravila in smernice za uporabo sistema za UKPP bodo zapisana in vključena v ISO dokumentacijo sistema za kakovost v obliki organizacijskih predpisov in navodil za delo.

Pripravljen bo seznam delovnih mest, ki bodo upravičena do avtoriziranega dostopa do sistema za UKPP prek navedenih orodij. Vzpostavljene bodo ustrezne vloge z dodeljenimi pravicami, ki bodo omogočile dostop do repozitorijev glede na vsebino, do katere ima posamezno delovno mesto avtoriziran dostop in posledično tudi zaposleni na teh delovnih mestih, ki bodo uporabljali sistem za UKPP. Z jasno definirano politiko dostopa in določitve postopkov, kdo, kaj in kam lahko namešča nove verzije, bomo dvignili nivo varnosti.

Pred produkcijsko vpeljavo posodobljenega sistema za UKPP v obstoječe razvojno okolje bomo izvedli ustrezna izobraževanja zaposlenih in pripravili pilotni projekt s postavitvijo testnega razvojnega okolja, kjer bo možno testirati vse funkcionalnosti orodij, vključenih v sistem za UKPP. Na podlagi ugotovljenih pomanjkljivosti in priložnosti bomo sistem ustrezno korigirali in prilagodili pravila in smernice za uporabo.

Poleg izvornih datotek s programsko kodo bomo v glavno deblo repozitorija shranjevali tudi vso dodatno dokumentacijo, kot so funkcijske specifikacije, uporabniška navodila, risbe, skice, BPMN diagrame in podobno. S pomočjo procesa za upravljanje sprememb, ki omogoča postopno premikanje skozi faze življenjskega cikla izvedbe projekta (vsakega enolično določenega s šifro) in vključuje vse elemente, kot so vsebinske zahteve, dokumentirane izvedene aktivnosti in testni scenariji s potrditvami tako s strani vsebinskih analitikov kot tudi končno validacijo s strani uporabnikov pri stranki, bomo zagotovili dvig kakovosti. S tem se upravičeno pričakuje povečanje naklonjenosti, tako razvijalcev kot tudi uporabnikov, za pogostejše namestitve novih različic programskega izdelka.

Analizo stroškov in prihrankov v povezavi z uvedbo sistema UKPP smo uporabili za izračun donosnosti naložbe (angl. ROI – Return on Investment) za obdobje petih let, pri čemer je potrebno poudariti, da smo določene vhodne vrednosti predvideli na podlagi preteklih izkušenj in lastne subjektivne ocene. Ugotovili smo (z upoštevanjem prej navedenih omejitev), da se bo predvidena investicija povrnila v le 1 letu in 265 dneh.

ROI za petletno obdobje pa znaša 163,57 %, kar pomeni, da je naložba v sistem za UKPP poleg vseh ostalih koristi tudi precej donosna.

LITERATURA IN VIRI

1. Agarwal, B. B., Dhall, S. in Tayal, S. P. (2011). *Software Project Management*. Laxmi Publications Pvt Limited.
2. Aiello, B. in Sachs, L. (2011). *Configuration Management Best Practices*. Addison-Wesley.
3. Berczuk, S. P., Appleton, B. in Brown, K. (2003). *Software Configuration Management Patterns: Effective Teamwork, Practical Integration.* Addison-Wesley.
4. Blokdijk, G. in Menken, I. (2008). *Configuration Management Best Practice Handbook.* The Art of Service.

5. Borraci, L. (2005). *A Return on Investment Model for Software Configuration Management*. Lund Institute of Technology.
6. Chacon, S. in Straub, B. (2014). *Pro Git* (2. izd.). Apress.
7. Collins-Sussman, B., Fitzpatrick, B. W. in Pilato, C. M. (2008). *Version Control with Subversion* (2.izd.). O'Reilly Media, Inc.
8. Collins-Sussman, B., Fitzpatrick, B. W. in Pilato, C. M. (2009). *Subversion 1.6 Official Guide – Version Control with Subversion*. Fultus Corporation.
9. Cortuk, D. (2019, 28. oktober). *Version Control Software Comparison:Git, Mercurial, CVS, SVN.* https://medium.com/@derya.cortuk/version-control-software-comparison-git-mercurial-cvs-svn-21b2a71226e4
10. Dorsey, P. in Koletzke, P. (1998). *Oracle Designer Handbook* (2.izd.). McGraw-Hill Osborne Media.
11. Gechman, M. (2019). *Project Management of Large Software-Intensive Systems.* CRC Press.
12. Hass, A. M. J. (2003). *Configuration Management Principles and Practice.* Addison-Wesley.
13. Institute of Electrical and Electronics Engineers. (2012). *IEEE/ANSI 828-2012 Standard for Software Configuration Management Plans.* Institute of Electrical and Electronics Engineers.
14. International Organization for Standardization. (2015). *ISO 9001:2015 Quality management systems – Requirements.* International Organization for Standardization.
15. International Organization for Standardization. (2017a). *ISO 10007:2017 Quality management - Guidelines for configuration management.* International Organization for Standardization.
16. International Organization for Standardization. (2017b). *ISO 12207:2017 Systems and software engineering - Software life cycle processes.* International Organization for Standardization.
17. Kenefick, S. (2008). *Real World Software Configuration Management.* Apress.
18. Keyes, J. (2004). *Software Configuration Management.* CRC Press.
19. Leon, A. (2015). *Software Configuration Management Handbook* (3. izd.). Artech House.
20. LinuxConcept. (brez datuma). *The battle of Version Control: Git vs. SVN vs. Mercurial.* Pridobljeno 13. septembra 2022 s https://linuxconcept.com/git-vs-svn-vs-mercurial/
21. Mason, M. (2006). *Pragmatic Version Control using Subversion* (2. izd.). The Pragmatic Programmers LLC.
22. McMillan,T. (2021, 4. september). *A History of Version Control.* https://tarynwritescode.hashnode.dev/a-history-of-version-control
23. Moreira, M. E. (2004). *Software Configuration Management Implementation Roadmap.* Wiley.
24. Nagel, W. (2005). *Subversion Version Control - Using The Subversion Version Control System in Development Projects.* Prentice Hall.
25. O'Sullivan, B. (2009). *Mercurial: The Definitive Guide.* O'Reilly Media, Inc.

26. Oracle. (2017, september). *Oracle Fusion Middleware.* https://docs.oracle.com/middleware/12213/formsandreports/deploy-forms/using-oracle-forms-services-http-listener-and-oracle-weblogic-server.htm#FSDEP227
27. Quigley, J. M. in Robertson, K. L. (2015). *Configuration Management – Theory, Practice, and Application.* Auerbach Publishers Inc.
28. The QA Lead Team. (brez datuma). *The Software Configuration Management Process: 5 steps.* Pridobljeno 13. julija 2021 s https://theqalead.com/topics/software-configuration-management-process/
29. UpGuard Team. (2021, 25. maj). *What Is Configuration Management and Why Is It Important?.* Pridobljeno 13. julija 2021 s https://www.upguard.com/blog/5-configuration-management-boss
30. White, B. A. (2000). *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction.* Addison-Wesley.

PRILOGE

Priloga 1: Anketni vprašalnik o uporabi in izkušnjah razvijalcev s sistemi za upravljanje programske kode

Prosimo vas, da si vzamete nekaj minut časa za izpolnitev te ankete o uporabi in vaših izkušnjah z orodji za upravljanje s programsko kodo pri razvoju programskih rešitev. Vaše mnenje je izjemno pomembno za razumevanje potreb pri pripravi predloga in kasnejši implementaciji orodja za upravljanje s programsko kodo v sistem za upravljanje konfiguracij programskih proizvodov (UKPP).

1) Ali uporabljate orodje za upravljanje s programsko kodo pri svojem delu?

   a) Da

   b) Ne

2) Če ne uporabljate orodja za upravljanje s programsko kodo, ali ste kdaj imeli težave pri sledenju spremembam v programski kodi?

   a) Da

   b) Ne

3) Ali ste v preteklosti že uporabljali orodje za upravljanje s programsko kodo?

   a) Da

   b) Ne

4) Če ste v preteklosti uporabljali orodje za upravljanje s programsko kodo, ali menite, da je bila njegova uporaba koristna?

   a) Da

   b) Ne

5) Ali poznate koncept uporabe orodja za upravljanje s programsko kodo?

   a) Da

   b) Ne

6) Ali ste že slišali za orodja, kot so Subversion (SVN), Git in Mercurial?

   a) Da

   b) Ne

7) Ali ste že uporabljali katero od naštetih orodij (Subversion, Git, Mercurial) pri svojem delu?

   a) Da

   b) Ne

8) Ali menite, da bi vključitev orodja za upravljanje s programsko kodo v sistem za upravljanje konfiguracij izboljšala kakovost programske kode in skrajšala čas, ki ga porabljate za sledenje spremembam?

   a) Da

   b) Ne

   c) Ne vem

9) Ali menite, da bi bilo koristno povečati ozaveščenost o orodjih za upravljanje s programsko kodo in njihovih prednostih med razvijalci?

   a) Da

   b) Ne

   c) Ne vem