Analiza tehnološkega, procesnega in organizacijskega vidika vpeljave sistema za upravljanje konfiguracije

Gregor Polančič, Gregor Jošt
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Inštitut za informatiko
{gregor.polancic, gregor.jost}@uni-mb.si

Povzetek

Organizacije, ki razvijajo programsko opremo, se v procesu razvoja izdelkov ali storitev soočajo s številnimi tehničnimi, procesnimi in organizacijskimi izzivi. Med osrednje spada nadzorovan in sledljiv razvoj množice izdelkov, ki se v različnih fazah razvoja preoblikujejo v končni izdelek ali storitev. Področje, ki pokriva omenjeno problematiko, se imenuje upravljanje konfiguracije. Prispevek obravnava vpeljavo upravljanja konfiguracije v majhnem podjetju, ki izdeluje programsko opremo. Predstavljena je analiza predhodnega stanja v podjetju, vpeljava sistematičnega upravljanja konfiguracije in analiza stanja po vpeljavi le-tega. Študija primera, ki smo jo izvedli, je razdeljena na tehnični, procesni in organizacijski vidik. Kvantitativni in kvalitativni rezultati raziskave podpirajo sistematično vpeljavo upravljanja konfiguracije v vseh vidikih, ki smo jih analizirali.

Abstract

THE ANALYSIS OF CONFIGURATION MANAGEMENT INTRODUCTION FROM TECHNOLOGICAL, PROCESS AND ORGANIZATIONAL VIEW

Software companies deal with several technological, process-based and organizational challenges during their software or service development process. One of focal challenges in these companies is managed and traceable development of intermediate or partial products which constitute final project outcomes. These important software engineering issues are addressed in the field of configuration management. The article presents a case study of introducing a systematic configuration management into a small software company from technological, process and organizational views. Our work also includes the analysis of previous practices in the company, the introduction of new practices and the analysis of software development team members' feedbacks. The qualitative and quantitative results of the performed case study support the systematic introduction of configuration management system in all of the analyzed aspects.

1 UVOD

V procesu razvoja programske opreme po navadi sodelujejo naročnik, vodja projekta in razvijalci. Interes vseh je, da v najkrajšem času zagotovijo kakovostno aplikacijo, ki bo pokrila vsa področja zahtev, vendar analiza, načrtovanje, implementacija in testiranje niso vedno dovolj za zagotovitev izpolnitve zahtev projekta ter upoštevanja časovnih ali denarnih mejnikov.

Za bolj plastično ponazoritev si predstavljajmo delo v okolju, v katerem (1) lokacije posameznih delov programske kode niso zmeraj znane, (2) ni znano, katera verzija se nahaja pri naročniku in katero ima posamezni razvijalec, (3) se na dnevni ravni prepisuje delo med razvijalci in (4) ni znano, kdo je urejal posamezni modul. Delo v takšnem okolju pušča negativne posledice na časovnih mejnikih, vzdušju v razvojni skupini in pri naročnikih. Za bolj organiziran pristop k izdelavi programske opreme moramo poleg obvladovanja zgoraj naštetih problemov imeti na voljo vse podrobnosti o verziji izdelka, ki vključuje naslednje podatke: kdo je delal na katerem modulu v posameznem časovnem obdobju, kakšne so razlike med aktualno in preteklo verzijo in kdaj lahko posamezni modul vključimo v celotno aplikacijo. Jasno je, da potrebujemo dodatno vodenje, ki nam bo zagotovilo obvladovanje problemov in večji pregled nad razvojem aplikacije skozi ves življenjski cikel programske opreme. Procesno področje, ki pokriva in obvladuje zgornje probleme, se imenuje upravljanje konfiguracije (v nadaljevanju UK) [1].

Standard ISO 10007 z naslovom »Kvalitetno vodenje – navodila za UK« (*Quality management – Guidelines for configuration management*) [13] definira upravljanje konfiguracije kot vedo o upravljanju, ki je usmerjena v tehnične in administrativne smeri razvoja, izdelave in vzdrževanja (podpore) življenjskega cikla konfiguracije. Povedano drugače, govorimo o identifikaciji, organiziranju in kontroli sprememb

gradnje programske opreme v projektni skupini. Veda je uporabna na področju strojne opreme, programske opreme, postopka materialov, pomoči in tehnične dokumentacije. Upravljanje konfiguracije predstavlja bistveni del vodenja življenjskega cikla razvoja izdelkov in storitev.

Upravljanje konfiguracije je zrelo področje, ki ga podpirajo številni prosto dostopni in lastniški programski paketi. Sodobna orodja in programski pripomočki za upravljanje konfiguracije so napredni in prijazni tako administratorju kakor navadnemu razvijalcu, a še vedno od celotne razvojne skupine terjajo motivacijo, odprtost do novih tehnologij in njihovo osvojitev. Proces uvajanja upravljanja konfiguracije prinese nekaj organizacijskih sprememb, ki prekinejo marsikatero rutino v razvojni skupini.

2 VIDIKI UPRAVLJANJA KONFIGURACIJE

Sistematično upravljanje konfiguracije zahteva vpeljavo in obvladovanje treh osnovnih vidikov upravljanja konfiguracije: tehnološkega, organizacijskega in procesnega. Tipični problemi, s katerimi se soočimo brez sistematičnega upravljanja konfiguracije, so:

- s **tehnološkega vidika** nimamo primernega orodja za upravljanje konfiguracije, kar pomeni, da nismo zmožni slediti spremembam (ni podrobnosti o tem, kdo je naredil kakšen popravek in zakaj). Nezmožni smo tudi pridobiti starejšo verzijo programa;
- ker nimamo urejenih pravil glede upravljanja konfiguracije, se soočamo z **organizacijskimi** problemi, npr. člani skupine ne poznajo svojih odgovornosti in pooblastil, ni natančnega pregleda nad rešenimi problemi oz. problemi, ki čakajo na rešitev. Soočamo se tudi s prepisovanjem trenutne programske kode s starejšo verzijo;
- nimamo vzpostavljenega upravljanja konfiguracije, kar pomeni, da se ne držimo nobenih standardov, ki bi definirali potrebne procese za zadovoljitev **procesnega vidika**. Zato se soočamo s težavami, kot so: odpravljene napake se ponovno pojavljajo v novejših verzijah in nedovoljene spremembe se vključijo v končno verzijo. Vse to pa botruje nepravilnemu delovanju programa.

V nadaljevanju so podane podrobnejše značilnosti posameznih vidikov upravljanja konfiguracije.

2.1 Tehnološki vidik

Začetki upravljanja konfiguracije programske opreme so bili ročni. Proces odjave kode (*check-out*) je zajemal pisanje imena razvijalca na tablo ali list poleg modula, katerega je urejal, medtem ko je bil proces prijave kode (*check-in*) zgolj brisanje imena s table ali z lista. Naprednejši pristop je zajemal barvne risalne žebljičke, pri čemer je imel vsak razvijalec svojo barvo. Te risalne žebljičke so potem zapičili v tablo poleg imena modula in določali, kdo ima pravico delati s posameznim modulom [1].

Tak pristop je danes neprimeren in neučinkovit, saj imamo na razpolago učinkovito programsko podporo. S tehnološkega vidika gledano pomeni upravljanje konfiguracije namestiti primerno programsko rešitev na strežnik in razvojne računalnike. Tabela 1 prikazuje najpogostejše programske rešitve in njihove osnovne lastnosti.

Večina rešitev upravljanja konfiguracije temelji na centralizirani podatkovni bazi, pri čemer se uporablja skupna shramba (*repository*). Razvoj se čedalje bolj nagiba k distribuiranemu pristopu, pri katerem ima vsak razvijalec svojo lokalno shrambo in je tako neodvisen od omrežja.[1]

Tabela 1: **Najpogostejše programske rešitve za upravljanje konfiguracije**

Sistem za upravljanje konfiguracije	Razvijalec	Podprt način[2]	Podprta računalniška okolja	Cena
ClearCase	IBM, l. 1992	Združi ali zakleni	Windows, *nix[3]	4380 USD
CVS	Dick Grune, l. 1986	Združi	*nix, Windows, MAC	Zastonj
Subversion (SVN)	CollabNet, Inc., l.2000	Združi ali zakleni	*nix, Windows, MAC	Zastonj
Team Foundation Server	Microsoft, l. 2006	Združi ali zakleni	Windows Server 2003; Windows	2799 USD[4]
Vault	SourceGear LLC,l. 2003	Združi ali zakleni	*nix, Linux, Windows	249 USD na uporabnika

[1] http://hgbook.red-bean.com/hgbookch1.html#x5-130001.4.
[2] Postopek *združi* omogoča souporabo informacij, ki se kasneje združijo, medtem ko postopek *zakleni* dopušča urejanje informacij samo avtorju zaklepa. Oba postopka rešujeta težavo souporabe datotek.
[3] Operacijski sistemi, ki temeljijo na Unixu.
[4] http://www.amazon.com/Microsoft-Visual-Studio-System-Foundation/dp/B000WM1Z5K/ref=sr_1_2?ie=UTF8&s=software&qid=1222270581&sr=8-2.

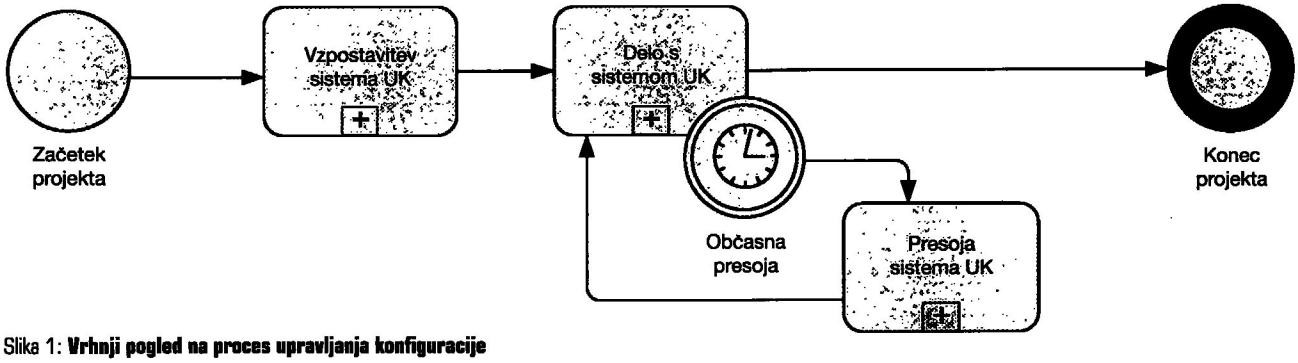
2.2 Organizacijski vidik

Najpogostejša praksa je, da se podjetje odloči za vpeljavo sistematičnega upravljanja konfiguracije, ko ugotovi, da obstajajo težave z obstoječim načinom dela. V takšnem primeru se podjetje sooča z mnogimi organizacijskimi spremembami. Novo vzpostavljen sistem upravljanja konfiguracije od skupine za razvoj zahteva, da se priuči ukazov okolja za upravljanje konfiguracije. Spremeniti ali vzpostaviti je treba tudi organizacijska pravila za poimenovanje in strukturiranje datotek in imenikov ter določiti vloge in odgovornosti v razvojni skupini. Izvajanje ukazov in prilagajanje na novi sistem po navadi prekine ustaljeno rutino v organizaciji, kar se lahko na začetku prikazuje v odporu do novega sistema, predvsem pri razvijalcih, ki so bili navajeni na sistem brez primernega upravljanja konfiguracije.

2.3 Procesni vidik

Proces je skupek med seboj povezanih ali vzajemno vplivajočih aktivnosti, ki pretvarja vhode v izhode [5]. Procesni vidik upravljanja konfiguracije ponazarja ključne procese, ki so sestavni del upravljanja konfiguracije. Najpogosteje uporabljan standard za definiranje procesov upravljanja konfiguracije je združeni zmožnostno-zrelostni model [6] (v nadaljevanju ZZZM), zato bomo v nadaljevanju predstavili procesni vidik, ki je skladen z njim.

ZZZM služi kot navodilo za oblikovanje vsebine procesov. Cilj ZZZM je, da povečajo uporabnost zrelostnih modelov, tako da združijo več modelov v eno ogrodje. ZZZM določuje procesna področja. Gre za množico povezanih praks, ki pomagajo doseči ključne cilje določenega procesnega področja [3]. Eno izmed teh procesnih področij je upravljanje konfiguracije. Slika 1 prikazuje vrhnji pogled na proces upravljanja konfiguracije, prikazan v notaciji BPMN (*Business Process Modelling Notation*).[5]



Slika 1: **Vrhnji pogled na proces upravljanja konfiguracije**

V ZZZM je področje upravljanja konfiguracije definirano na drugem nivoju in vključuje naslednje specifične cilje[6] in specifične prakse[7] (slika 1):

Vzpostavitev sistema za upravljanje konfiguracije

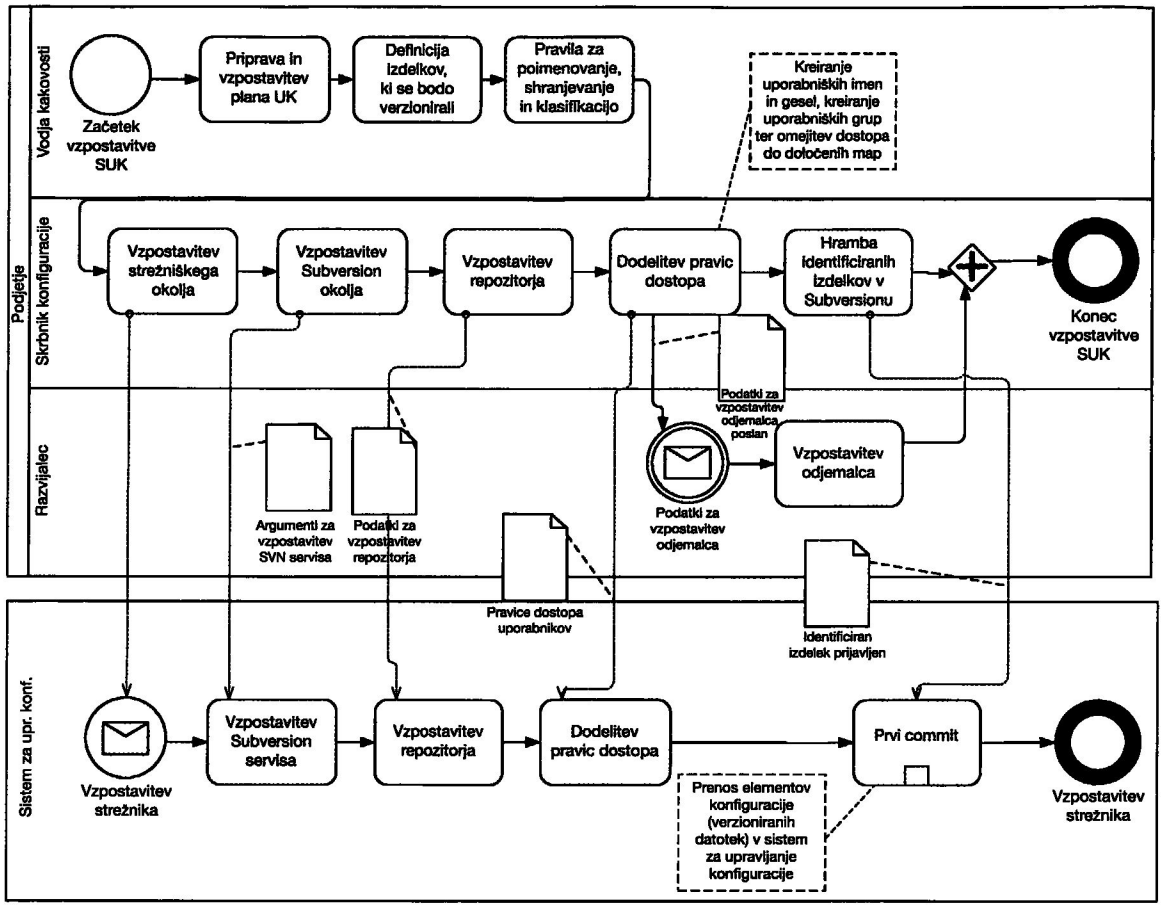
Pri vzpostavitvi sistema za upravljanje konfiguracije (okrajšano SUK) moramo najprej identificirati elemente konfiguracije (okrajšano EK), ki jih izberemo na podlagi dokumentiranih kriterijev. EK je skupek strojnih, programskih ali obojnih proizvodov, ki jih vključimo v upravljanje konfiguracije in jih obravnavamo kot samostojno entiteto v procesu upravljanja konfiguracije [2]. Vsak EK mora biti unikatno definiran ali prek poti ali prek imena. Prav tako moramo določiti, kdaj bomo posamezni EK vstavili v UK. Pri tem si pomagamo s pravili. Na primer, če programska koda predstavlja EK, se vstavi v UK, ko se prevede. Če je EK pisni dokument, ga vstavimo v UK, ko je slovnično pravilno oblikovan. Po vzpostavitvi SUK, moramo izdelati osnovno različico, za katero potrebujemo pooblastitev telesa za nadzor konfiguracije. Na naslednji sliki (Slika 2) je prikazan primer procesa vzpostavitve SUK v notaciji BPMN (*Business Process Modelling Notation*).

---

[5] BPMN tehnika nam omogoča, da grafično prikažemo korake znotraj poslovnih procesov.

[6] **Specifični cilji** (*specific goals* SG) veljajo samo za izbrano procesno območje in so obvezna komponenta modela. Za doseganje posameznega specifičnega cilja moramo izvajati določene aktivnosti, ki jim pravimo specifične prakse.

[7] **Specifične prakse** (*specific goals SG*) veljajo samo za izbrano procesno območje in so obvezna komponenta modela. Za doseganje posameznega specifičnega cilja moramo izvajati določene aktivnosti, ki jim pravimo specifične prakse.



Slika 2: **Primer procesa vzpostavitve SUK**

Iz zgornjega modela (slika 2) je razvidno, da pri vzpostavitvi SUK sodelujejo razvijalci, vodja kakovosti in (bodoči) skrbnik upravljanja konfiguracije.

Delo s sistemom za upravljanje konfiguracije

Sledenje zahtevam za spremembe predstavlja osrednje delo v SUK. Sledenje EK zahteva nadzor sprememb nad EK skozi ves življenjski cikel razvoja produkta. Na naslednji sliki (slika 3) je prikazan primer vrhnjega pogleda na proces dela s SUK.

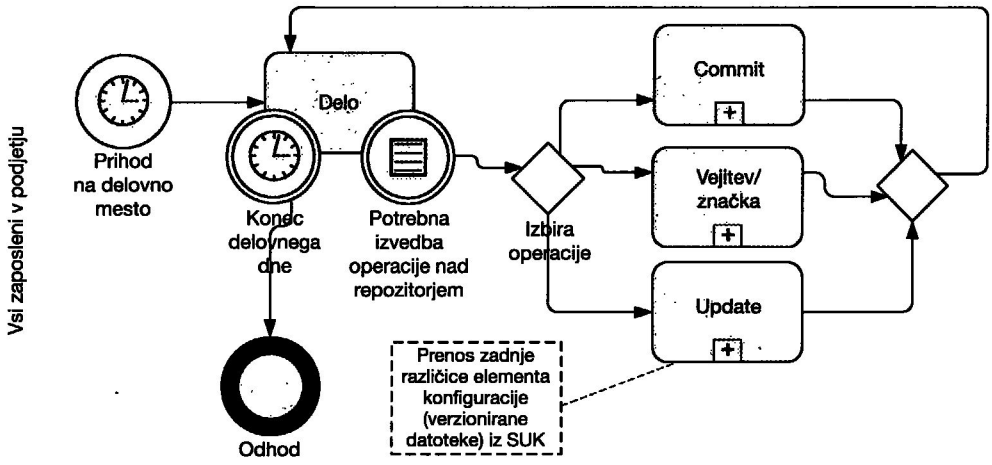
Presoje sistema za upravljanje konfiguracije

Vzpostavitev zapisov o upravljanju konfiguracije zahteva beleženje aktivnosti upravljanja konfiguracije do takšnih podrobnosti, da sta vsebina in status vsakega EK znana, pretekle verzije pa so po potrebi lahko povrnjene. Stanje in zgodovina posameznega EK morata biti vedno dosegljiva. Vsi pomembnejši udeleženci v projektu morajo imeti zagotovljen dostop in znanje o statusu EK. Ocenjevati in dokumentirati moramo verodostojnost osnovne verzije, pregledovati strukturo, pravilnost in popolnost EK ter slediti EK od presoj do zaključka. Na naslednji sliki (slika 4) je prikazan primer procesa izvedbe presoj SUK.

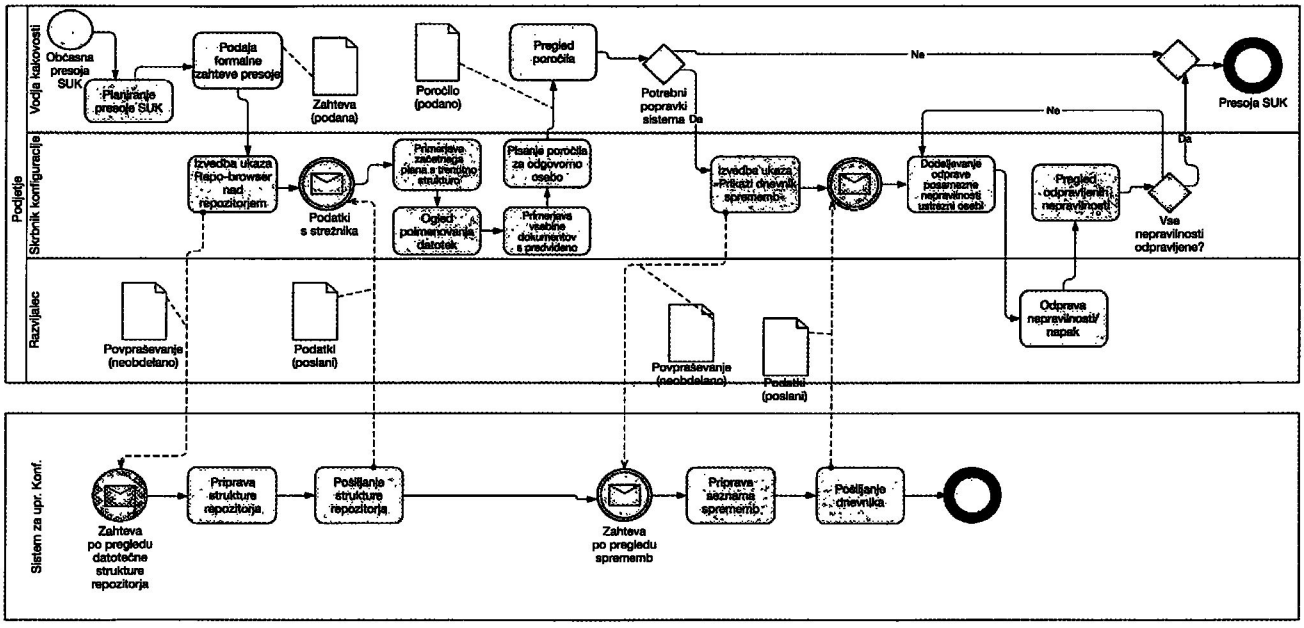
3 ŠTUDIJA PRIMERA VPELJAVE SISTEMATIČNEGA UPRAVLJANJA KONFIGURACIJE

3.1 Metoda raziskave

V raziskavi smo uporabili metodo študije primera (*case-study*), ki predstavlja poglobljeno analizo situacije v naravnem okolju [9]. Prav tako nudi raznolikost metod za zbiranje podatkov (npr. intervjuji, opazovanje, preučevanje dokumentacije, statistika uporabe). Študija primera se pogosto uporablja za opisovanje in razlago pri preiskovalnih raziskavah. Pri načrtovanju zbiranja podatkov moramo določiti, katere primere bomo izvajali in v katerih organizacijah. V analizi opravimo izbor metode, obrazcev in programskih orodij. Hkrati je treba upoštevati, da



Slika 3: **Vrhnji pogled na process dela s sistemom Subversion**



Slika 4: **Proces presoj sistema za upravljanje konfiguracije (SUK)**

metoda študije primera le pogojno dovoljuje posploševanje iz enega samega primera, kar velja tudi za našo raziskavo.

Naša študija primera je bila osredinjena na podjetje, ki se ukvarja z izdelavo programske opreme in ni imelo vzpostavljenega sistematičnega upravljanja konfiguracije. Posvetili smo se analizi preteklega ter vpeljavi in analizi prenovljenega sistema. Odgovorili smo na raziskovalno vprašanje »*Kakšen je odziv zaposlenih na vpeljavo sistema za upravljanje konfiguracije*«. Odziv zaposlenih smo analizirali s tehnološkega, organizacijskega in procesnega vidika:

- S **tehnološkega vidika** smo želeli ugotoviti, kako se shranjujejo projekti, kako se skrbi za verzije in kakšna orodja se uporabljajo. Po prenovi je tehnološki vidik predstavljal strežnik, na katerem je bilo nameščeno orodje za upravljanje konfiguracije. Pri vsakem posameznem razvijalcu je bilo treba namestiti in konfigurirati odjemalca za upravljanje konfiguracije.
- **Organizacijski vidik** je predstavljal analizo pravil, ki so se upoštevala znotraj organizacije. Ugotovili smo, kako so bili projekti razporejeni, kako poimenovani, ali so obstajala kakšna interna pra-

vila glede poimenovanja projektov in kdaj so posamezni elementi bili vključeni v osnovno verzijo. Med prenovo smo definirali primerno poimenovanje datotek in opredelili, kdaj se bodo elementi konfiguracije vključevali v osnovno verzijo. Po prenovi smo ugotavljali, v kolikšni meri smo izpolnili cilje, ki so skladni z ZZZM.

- V fazi **procesnega vidika** smo analizirali, katere vloge so bile vključene pri posameznih aktivnostih, kako so se procesi upravljanja konfiguracije izvajali in kateri procesi so že bili vzpostavljeni. Po vzpostavitvi novega sistema smo primerjali obstoječe stanje z novim glede na kompleksnost procesov oziroma glede na število vlog in aktivnosti pri izvajanju projektov.

3.2 Uporabljene metrike

Kvantitativni vidik raziskave smo izvedli na osnovi različnih metrik. Učinkovitost vpeljave v podjetju smo merili s pomočjo metrik po standardu ISO/IEC 9126-1. Prek števila konfliktov v datotekah smo ugotovili, ali na eni datoteki dela preveč razvijalcev. Odstotek datotek, ki se ne prevedejo, je prikazal, kako se upoštevajo pravila za dodajanje EK v osnovno verzijo. Število nepravilno poimenovanih datotek nam je bilo v pomoč pri ugotovitvi, kako so se upoštevala organizacijska pravila. Odstotek izdaj brez napak je prikazal, kako temeljito se testirajo datoteke pred vnosom v osnovno verzijo (tabela 2).

Tabela 2: **Metrike, ki so se uporabile v študiji primera**

Ime metrike	Meritev, formula in računanje podatkovnih elementov	Interpretacija merjene vrednote
Število nepravilno poimenovanih datotek	X = A A = Število nepravilno poimenovanih datotek	0 <= X Manj je bolje.
Število konfliktov v datotekah	X = A A = Število konfliktov	0 <= X Manj je bolje.
Odstotek datotek, ki se ne prevedejo	X = (A / B)*100 A = Število datotek, ki se ne prevede B = Število vseh datotek	0 % <=X<=100 % Manj je bolje.
Odstotek izdaj brez napak	X = (A / B) * 100 A = Vse izdaje B = Izdaje brez napak	0 % <= X <= 100 % Manj je bolje.

3.3 Rezultati

V nadaljevanju so podani rezultati raziskave, ki so razdeljeni na tehnološki, organizacijski in procesni vidik.

3.3.1 Tehnološki vidik

Analiza obstoječega stanja v organizaciji je podala naslednje ugotovitve. Vse aplikacije so v organizaciji so se izdelovale v odprtokodnem programskem jeziku PHP (PHP: Hypertext Preprocessor [4]). Uporabljal se je sistem spletnih predlog, katerih glavna naloga je ločitev logike programa od njegove predstavitve. Po potrebi se je v projekte vključil tudi XAJAX. XAJAX je odprtokodna implementacija AJAX-a v obliki knjižnice za PHP. Aplikacije so se programirale v razvojnem okolju Zend Studio for Eclipse. Glavna prednost Zend Studio for Eclipse je že vgrajena podpora dveh orodij za verzioniranje (Subversion in CVS).

Pred prenovo se je za izdelavo projektov, njihovo testiranje in hranjenje uporabljal strežnik za soupo-

rabo datotek, do katerega je lahko dostopal vsak razvijalec s svojim uporabniškim imenom in geslom. Na strežniku je bil nameščen operacijski sistem FreeBSD. Za deljenje imenikov prek omrežja se je uporabljala Samba, zastonjska implementacija Windows SMB protokola [11]. Samba izbrane imenike Unix sistema (ali katerega koli drugega sistema, ki temelji na Unixu; sem spada tudi FreeBSD) deli prek omrežja. Izbrani imeniki se uporabniku predstavljajo kot spletne mape.

Prvi sklop prenove sistema je zajemal vzpostavitev strežnika, definiranje lokacije repozitorija in odjemalcev. Sem je spadala tudi optimizacija obstoječega orodja za zagotavljanje združljivosti s sistemom.

Za strežnik SUK smo izbrali strežnik z operacijskim sistemom FreeBSD. Glavni razlog za izbor sistema FreeBSD je njegova brezplačnost, učinkovitost, zanesljivost in robustnost; kot tak se najpogosteje pojavlja na Netcraftovi lestvici neprekinjenega delovanja spletnih strežnikov [7].

SVN ne vključuje nobenega grafičnega uporabniškega vmesnika, ki bi bil namenjen odjemalcu. Ponuja sicer vmesnik z ukazno vrstico, vendar zahteva od uporabnikov dodatno učenje ukazov. V ta namen je bilo razvitih že mnogo odjemalcev z grafičnim uporabniškim vmesnikom in eden izmed njih je tudi prosti program TortoiseSVN, ki je izdan pod splošnim dovoljenjem GNU in je tako tudi odprtokoden. Prednosti TortoiseSVN so [10]: integracija v raziskovalca Windows, integracija z razvojnim okoljem in lokalizacija vmesnika v slovenski jezik.

3.3.2 Organizacijski vidik

Kot smo že dejali v prejšnjem razdelku, Samba izbrane imenike Unix sistema deli prek omrežja in ti se uporabniku operacijskega sistema Microsoft Windows predstavljajo kot mape, dosegljive prek omrežja. Te mape so predstavljale ključni element pri vodenju konfiguracije pred prenovo sistema. V podjetju so se uporabljale tri mape, do katerih je imel dostop vsak uporabnik. Organizacija map je bila takšna: mapa za projekte, mapa za materiale in začasna mapa.

Mapa za **projekte** se je uporabljala za shranjevanje datotek, ki skupaj predstavljajo celoten projekt. Na enem projektu je delalo več razvijalcev hkrati, saj so lahko vsi sočasno dostopali do vseh projektov v mapi za projekte in jih po potrebi spreminjali, ustvarjali ali brisali. Pred začetkom vsakega projekta se je določilo, kdo dela na njem. Tistemu so se nato dodelili posamezni moduli, ki jih je moral razviti, vendar je zaradi slabih dogovorov ali načina dela velikokrat prihajalo do prepisovanja ali brisanja pomembnih datotek. Mapa je imela dnevne varnostne kopije.

Mapa za **materiale** se je uporabljala za shranjevanje datotek, ki so bile povezane s projekti. Sem so spadale slikovne, avdio in video datoteke oz. tudi drugi tipi datotek, saj omejenost ni bila strogo določena le na naštete tri. Razvijalec je lahko ustrezno oblikovano sliko shranil v omenjeno mapo, a njena dejanska lokacija je bila na strežniku. Podjetje se je odločilo za mapo za materiale predvsem iz varnostnih razlogov. Če bi se datoteke, ki še niso bile del projekta, shranjevale na osebnem računalniku, bi jih lahko v primeru zrušitve sistema izgubili. Posledično bi izgubili čas, ki smo ga porabili za izdelavo omenjene datoteke. Mapa materiali je zato hranila dnevne varnostne kopije.

Manj pomembna je bila **začasna** mapa. Ni bila namenjena trajnemu shranjevanju podatkov, temveč je služila zgolj izmenjavi, varnostnih kopij ni imela. Tipičen scenarij za uporabo začasne mape je bil, da je želel uporabnik A posredovati določeno datoteko uporabniku B. Uporabnik A je to storil tako, da je datoteko vstavil v začasno mapo. Uporabnik B si je nato prekopiral datoteko na svoj računalnik in jo izbrisal iz mape.

Pri takšnem načinu izvajanja projektov so igrale najpomembnejšo vlogo varnostne kopije. Razvijalci v skupini so lahko tako pridobili vse podatke, ki so bili prepisani ali izbrisani iz različnih razlogov. Varnostne kopije so se izvajale dnevno in shranjevale le 14 dni, zato na ta način ni bilo mogoče pridobiti nazaj osnovne verzije projektov, ki so bili starejši od omenjenega časovnega obdobja. Edino vodenje verzij se je izvrševalo s kopiranjem celotnih projektov. Ko so razvijalci končali prvi večji sklop, je skrbnik sistema naredil kopijo celotnega projekta, ki je predstavljala zadnjo stabilno verzijo. Zato je prihajalo do številnih tipičnih težav, ki so povezane z upravljanjem konfiguracije:

1. Razvijalec je delal sam na projektu in ga uspešno zaključil ter predal stranki.
2. Izdelek je bil v uporabi in čez določen čas se je pojavila zahteva po dodatni funkcionalnosti.
3. Ker je delal na projektu sam, se ni zdelo potrebno izdelovati kopij. Razvijalec je lahko torej implementiral zahtevano funkcionalnost samo tako, da je nadgradil obstoječi izdelek.
4. V času implementacije dodatnih funkcionalnosti je stranka opozorila na napako v delovanju že obstoječega, predanega izdelka.
5. Razvijalec je prekinil proces implementacije zahtevanih funkcionalnosti in je nemudoma odpravil napake v že uporabljenem izdelku.
6. Težava je nastala, ko je želel razvijalec objaviti popravke obstoječega izdelka. Nova funkcionalnost je bila sicer delno implementirana, ampak ne do te stopnje, da bi jo lahko razvijalec objavil na produkcijskem strežniku.

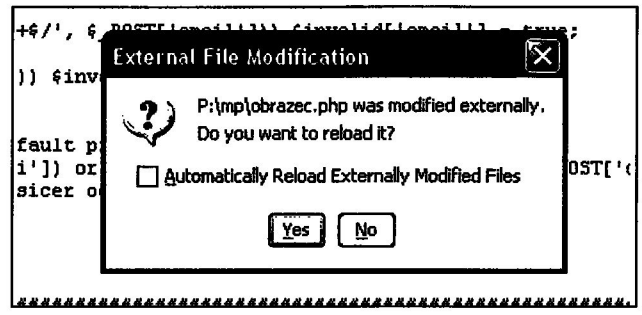
S pravilnim upravljanjem konfiguracije bi lahko razvijalec prekinil delo na novi verziji, si pridobil starejšo oz. enako, kot jo ima uporabnik, in odpravil napake. Takšno obliko rešitve bi lahko izvedli s pomočjo vejitev. V našem primeru si je razvijalec naredil dodatno delo s komentiranjem kode novih nedokončanih funkcionalnosti, kar je za podjetje pomenilo nepotrebno izgubo časa in denarja. Sorodna slaba praksa se je prav tako navezovala na vodenje verzij:

1. Projekt se je zaključil in šel v uporabo.
2. Čez čas je prišlo do večjih sprememb na obstoječem projektu, ki jih je bilo treba izvesti v zelo kratkem času.
3. Zaradi časovne omejitve se spremembe niso testirale dovolj temeljito, vendar se je nova verzija vseeno predala stranki.
4. Kasneje se je izkazalo, da nova verzija ni bila primerna za uporabo, staro staro, stabilno in delujočo pa smo prepisali z novo.

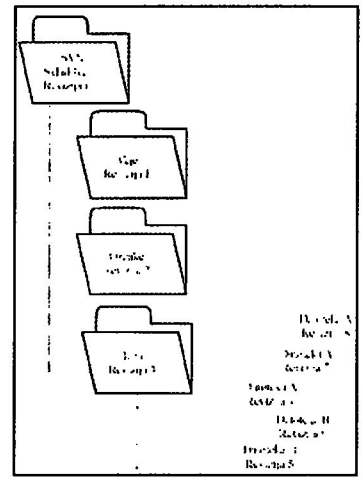
V tem primeru se sicer nepotrebnim finančnim in časovnim odstopanjem tudi s primernim upravljanjem konfiguracije ne bi izognili, vendar bi lahko stranki priskrbeli stabilno (sicer starejšo) verzijo projekta, medtem ko bi implementirali zahtevane spremembe.

Težave so potencialno nastale pri samem izvajanju projekta, na katerem je delalo več razvijalcev. Čeprav so se na začetku razdelile odgovornosti za posamezni modul aplikacije, je bilo nerealno pričakovati, da en razvijalec ne bo pregledoval ali celo spreminjal datoteke drugega razvijalca zavoljo testiranja ali same funkcionalnosti lastnega modula. Vse skupaj je zelo upočasnilo izdelavo aplikacije, v najslabšem primeru je sploh ni bilo mogoče razviti, če je na njej hkrati delalo več razvijalcev. Delo je potekalo tako, da so imeli vsi dostop do vseh datotek trenutnega projekta. Za primer vzemimo, da je iz kakršnih koli razlogov razvijalec A spreminjal modul razvijalca B in shranil spremembe. Programsko okolje je javilo razvijalcu B opozorilo, da je bila datoteka spremenjena zunaj njegovega okolja in jo je treba ponovno naložiti. Če je razvijalec B potrdil takšno opozorilo, se mu je celotna datoteka prepisala s spremembami, ki jih je shranil razvijalec A. To je lahko v najslabšem primeru pomenilo tudi izgubo dela razvijalca B, odvisno od obsežnosti popravkov razvijalca A. Če razvijalec B ni potrdil takšnega opozorila in je kasneje shranil svoje spremembe, se je isto opozorilo izpisalo razvijalcu A, ki pa je v primeru potrditve prepisa datoteke izgubil vse svoje spremembe. Če je odklonil opozorilo in nadaljeval z delom, so lahko bile kasnejše posledice hujše. Slika 5 prikazuje primer takšnega opozorila.

Po vzpostaviti upravljanja konfiguracije smo s ciljem učinkovite uporabe SVN upoštevali priporočila vzpostavitve primerne strukture. V praksi se večinoma uporablja pristop ustvarjanja treh imenikov [8]: **deblo** (*trunk*), **veje** (*branches*) in **značke** (*tags*), kot prikazuje slika 6. V deblu se razvija celotna aplikacija, za uporabo vejitev pa nam služi imenik veje. Pri uporabi vejitev je treba v določeni točki obe veji razvoja spojiti v celoto.



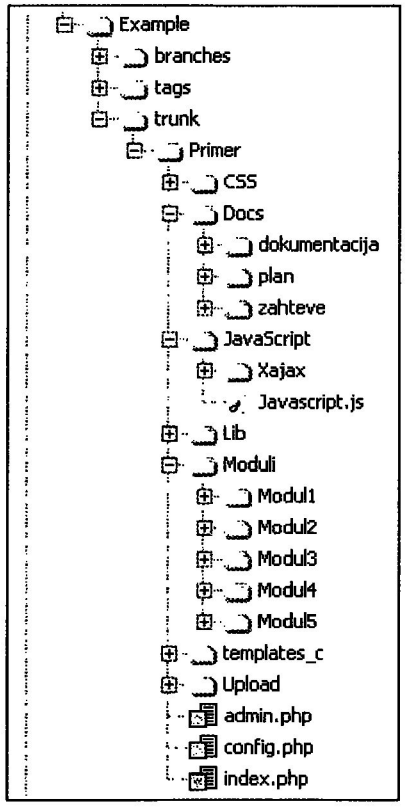
Slika 5: **Primer opozorila**



Slika 6: **Struktura map v skladišču (*repository*) SVN**

V imeniku oznake se shranjujejo posamezne zaključene verzije projekta, ki jih lahko poljubno poimenujemo. Priporočljivo je, da se lahko že iz imena imenika vidi, kaj predstavlja posamezni sklop. Ustaljena praksa je, da se izvede prevzem zgolj nad deblom, še posebno če ima projekt veliko oznak in je k razvoju pristopil še dodatni razvijalec, ki še nima svojega imenika z vsebino trenutne aplikacije. Če bi novi razvijalec prevzel celoten imenik, bi prav tako prevzel vse obstoječe oznake, ki v velikem številu zasedejo precej prostora in upočasnijo proces prevzema. Toda SVN omogoča, da lahko vidimo vse obstoječe oznake projekta, kljub temu da jih nismo prevzeli. Vsak projekt ima torej svoj imenik, ki je poimenovan po projektu. Znotraj tega imenika imamo strukturo, razdeljeno na deblo (*trunk*) in veje (*branches*).

Z uvajanjem upravljanja konfiguracije je prišlo do odločitve, da bodo vse aplikacije bolje strukturirane in da bodo imele večjo konsistenco. Takšna struktura omogoča, da je vsak EK unikatno identificiran že samo prek poti (slika 7). V strukturi je treba upoštevati posebnosti posameznih imenikov ali datotek.



Slika 7: **Struktura projektov**

Imenik »templates_c« potrebujemo za prevajanje in izvajanje aplikacije [12]. Ker ima vsak razvijalec svoj imenik, napolnjen s svojimi datotekami, ga izključimo iz SVN. Podobno velja za imenik »upload«, ki predstavlja prostor, kamor uporabniki prek aplikacije naložijo razne datoteke, najpogosteje so to slike (npr. slike artiklov pri spletnih trgovinah). Glede na vsebino se temu primerno poimenujejo podimeniki. Čeprav bi ta imenik oz. njegova vsebina lahko bila del EK, je treba upoštevati omejitve SVN. Vzemimo na primer, da posamezni razvijalec zavoljo testiranja aplikacije prek nje naloži sliko v imenik »upload« in po naslednji objavi začne SVN voditi verzije slike. Problem nastane, ko razvijalec kasneje prek iste aplikacije izbriše omenjeno sliko iz imenika upload. Ker sistem Subversion ne dopušča posrednega brisanja datotek, se tako pri naslednji objavi soočimo s težavami, ker smo nepravilno (prek aplikacije in ne SVN) brisali slike. Omenjeni imenik je posledično prav tako izključen iz SVN.

Kljub temu da vse PHP datoteke predstavljajo EK, obstajajo tudi izjeme. To je datoteka config.php, ki je edinstvena za vsakega posameznega razvijalca in je zato dodana na seznam prezrtih datotek ali imenikov. Datoteka je ključna za izvajanje aplikacije, saj vsebuje posamezne definicije, povezane z dostopom do podatkovne baze (kljub temu da je podatkovna baza skupna, lahko ima vsak razvijalec svojo različico s svojimi testnimi podatki) in lokacijami posameznih datotek na lokalnem disku.

Večino problemov, ki nastanejo zaradi nepravilnega vodenja konfiguracije, SVN uspešno razreši že s pridobivanjem starejših različic. Čeprav sistem olajša delo tudi v smislu prepisovanja datotek, se še vedno lahko pojavi problem, kadar delata dva razvijalca na istih datotekah. Vzemimo na primer, da en razvijalec popravlja napake, medtem ko drugi temeljito spreminja aplikacijo v smislu nadgradnje. S pogostimi objavami bi se tudi datoteke velikokrat nahajale v sporu in bi bilo treba vsak takšen spor reševati posebej. Sprotno reševanje lahko postane precej neprijetno in zamudno, zato se v takšnem primeru uporabljajo vejitve.

Vejitve so zelo uporabne tudi v primerih, ko se aplikacija razvija za dve različni okolji. Za primer lahko vzamemo uporabo Xajax-a. Če brskalnik končnega uporabnika ne podpira Javascript-a oz. je Javascript namerno onemogočen, mora aplikacija še vedno delovati.

3.3.3 Procesni vidik

Pred vpeljavo sistematičnega upravljanja konfiguracije organizacija ni imela vzpostavljenih nobenih standardiziranih procesov. Po vpeljavi smo ugotovili, da je že sama vpeljava SVN na strežniški in odjemalčevi strani zagotovila večino zahtev in praks po ZZZM. V nadaljevanju bomo predstavili, kako smo zadostili posameznim splošnim ciljem [6].

Z vidika specifičnega cilja **Vzpostavitev osnovnih različic** dosežemo identifikacijo enot konfiguracije z dokumentiranim spiskom delovnih produktov, ki medsebojno vplivajo drug na drugega, oz. pričakujemo, da se bodo spreminjali skozi čas. Struktura aplikacije skrbi, da je vsak EK unikatno identificiran prek poti, imena datoteke in končnice. Vsi pomembnejši podatki, kot sta npr. avtor elementa in tip dato-

teke, so zabeleženi v SUK. Čas vstavitve vsakega EK v upravljanje konfiguracije se tudi določi (datoteke je treba prevesti, dokumente HTML validirati in Wordove dokumente formalno pregledati). Namestili smo izbrano orodje za upravljanje konfiguracije, kar pomeni SUK s kontroliranimi delovnimi produkti in podatkovno bazo s spremembo na zahtevo.

Pred izdelavo osnovne verzije vodja projekta pregleda kandidate za osnovno verzijo in jih nato potrdi ali zavrne. Potrjeni kandidati postanejo del končnega izdelka in jih prek SUK vstavimo v osnovno verzijo. Tabela 3 prikazuje tipične izdelke, za katere moramo poskrbeti pri vzpostavitvi osnovnih različic.[8]

Tabela 3: **Tipični delovni izdelki vzpostavitve osnovnih različic**

Tipični delovni izdelek	Podprto	Podrobnosti
Identificirani EK	SVN	Vse datoteke, ki so v skladišču, predstavljajo EK.
SUK s kontroliranimi delovnimi produkti	Ročno	Namestitev SVN
Dostopni postopek SUK	SVN	Manipulacija kontrolnih datotek »authz« in »passwd«
Podatkovna baza s spremembo na zahtevo	SVN	SVN sam upravlja s podatkovno bazo.
Osnovna verzija	SVN	SVN omogoča, da se OV shranjuje v imenik deblo (*trunk*).
	Pravilo	Struktura vseh projektov je skladna s pravili in vnaprej določena; elementi v strukturi predstavljajo končni proizvod.
Opis osnovne verzije	SVN	Ob ukazu »Objavi« imamo možnost dokumentiranja sprememb. Vse spremembe in dela razvijalcev lahko kasneje pregledamo s pomočjo ukaza »Prikaži dnevnik«.

Z vidika specifičnega cilja »**Izvajanje presoj**« predstavlja zahteva za spremembo temeljne funkcionalnosti SVN. Njena uporaba se kontrolira s pomočjo avtorizacije uporabnikov sistema. Vse izdane zahteve se beležijo v dnevniku. Za nadzorovanje EK skrbi vodja projekta, tako da spremlja spremembe na nivoju celotnega projekta (ali aplikacija še deluje). Posamezni razvijalci skrbijo za zadolžene module. S tem preprečimo pojav nenamernih sprememb v osnovni verziji izdelka. Ob objavi EK lahko razvijalec vnese potreben komentar, nato beležimo spremembe EK in razloge za spremembo. Tipični delovni izdelki pri izvajanju presoj so predstavljeni v spodnji tabeli (Tabela 4).

Tabela 4: **Tipični delovni izdelki izvajanja presoj**

Tipični delovni izdelek	Podprto	Podrobnosti
Zahteva za spremembo	SVN	Zahteva za spremembo je temeljni del SVN. Njena uporaba se kontrolira s pomočjo avtorizacije uporabnikov sistema.
Zgodovina revizij EK	SVN	SVN dnevnik
Arhivi OV	SVN	Na strežniku se shranjujejo vse OV.

Pri splošnem cilju »**Vzpostavitev integritete**« skrbimo za integriteto prek beleženja aktivnosti upravljanja konfiguracije do podrobnosti, da sta vsebina in status vsakega EK znana, pretekle verzije pa so lahko povrnjene. Za slednje skrbi kar strežnik Subversion. Dostopanje in znanje o statusu konfiguracije EK sta zagotovljena vsem udeležencem; to dosežemo z dodelitvijo pravic posameznim udeležencem. Status EK, zapise o spremembah in razlike med osnovnimi verzijami pridobimo prek ukazov Subversiona. EK sledimo od presoj do zaključka, v pomoč so nam funkcionalnosti Subversiona, ki nam povedo, kako se je vsebina spreminjala skozi čas. Za pravilnost in popolnost EK skrbi sam SUK. Pri razreševanju morebitnih konfliktov morajo sodelovati razvijalci. Beležimo rezultate presoj in izdelke z opisi ukrepov ob morebitni ugotovitvi pomanjkljivosti pri presoji kakovosti. Tabela 5 prikazuje tipične delovne izdelke pri specifični praksi »Vzpostavitev integritete«.

[8] Celotna analiza je dosegljiva v diplomskem delu *ANALIZA TEHNOLOŠKEGA IN PROCESNEGA VIDIKA VPELJAVE SISTEMA SUBVERSION* (2008).

Tabela 5: **Tipični delovni izdelki vzpostavitve integritete**

Tipični delovni izdelek	Podprto	Podrobnosti
Revidiran potek EK	SVN	Nad želenim EK izvedemo ukaz »Pokaži dnevnik«.
Zapisi o spremembah	SVN	Ukaz »Pokaži dnevnik«
Kopija zahtevkov sprememb	SVN	Ukaz »Pokaži dnevnik« ima tudi podatke o tem, katera oseba je kaj spreminjala.
Status EK	SVN	Ukaza »Pokaži dnevnik« ali »Skladiščni brskalnik«
Razlike med OV	SVN	Rezultat ukaza »Pokaži dnevnik« omogoča ugotavljanje razlik med OV.
Rezultat presoj konfiguracije	SVN	Poročilo o presoji upravljanja konfiguracije
Vsebina korektivnih ukrepov	Ročno	Izdelki z opisi ukrepov ob morebitni ugotovitvi pomanjkljivosti pri presoji kakovosti

3.4 Interpretacija rezultatov

Za pridobitev odzivov uporabnikov smo izbrali kvantitativno-kvalitativni pristop, pri čemer smo podatke pridobili s vprašalniki zaprtega in odprtega tipa. Zaradi majhnega števila zaposlenih v podjetju, ki pri delu uporabljajo SUK, je bilo število respondentov razmeroma majhno (devet). Stopnjo strinjanja s trditvami so anketiranci izražali z ocenami med 1 in 7, pri čemer je ocena 1 predstavljala popolno nesoglasje in ocena 7 popolno soglasje (Likertova lestvica). Poglavitne ugotovitve so podane v nadaljevanju.

Prvič, ugotovili smo, da sta razumevanje in uporaba SVN preprosti, saj je skoraj polovica anketiranih (44 %) izrazila delno soglasje s trditvijo, medtem ko popolnega nesoglasja ni izrazil nihče. Menimo, da k visokemu odstotku soglasja pripomoreta preprosta sintaksa ukazov SVN na odjemalčevi strani in uporaba grafičnega orodja TortoiseSVN. Prav tako ima orodje dobro podporo in obsežno dokumentacijo, ki je dostopna tudi v slovenskem jeziku.

Drugič, ugotovili smo, da je s trditvijo »da se razvoj z uporabo sistema Subversion v skupini odvija hitreje«, popolnoma soglašalo 33 % anketirancev, enak odstotek jih je tudi delno soglašalo. Popolnega nesoglasja ni bilo. Na podlagi rezultatov sklepamo, da so slabe izkušnje s preteklim načinom delovanja igrale pomembno vlogo pri rezultatu, prav tako ne gre zanemariti uporabniku prijaznega načina objav in pridobitve elementov konfiguracije, ki jih nudi SVN.

Pod tretjo točko smo želeli pridobiti uporabnikovo mnenje, ali je bila uporaba sistema Subversion dobra zamisel. Večji delež (56 % anketirancev) je popolnoma soglašal s trditvijo. Hipoteza, da bodo razvijalci v podjetju, ki razvija programsko opremo, sprejeli vpeljavo sistema SVN, se je izkazala za pravilno. Večji odstotek anketirancev je namreč popolnoma soglašal s trditvijo, da je uporaba SVN dobra zamisel.

Med ostalimi trditvami se je izkazalo, da je bil večji odstotek anketirancev mnenja, da se SVN uporablja premalo, kar potrjuje zadovoljstvo ter zanimanje s strani razvijalcev. Izrazito soglašanje je dosegla tudi trditev, da se z uporabo SVN hitreje končajo projekti, na katerih dela več razvijalcev. Iz tega lahko sklepamo, da so anketiranci mnenja, da SVN prinaša izboljšave.

Pri rezultatih je treba upoštevati, da se je večina anketirancev opredelila kot eksperti preteklega načina dela (brez sistematičnega upravljanja konfiguracije).

5 SKLEP

V prispevku smo predstavili in analizirali vpeljavo sistema za upravljanje konfiguracije v manjše podjetje, ki razvija programsko opremo. Organizacija se je predhodno na omenjenem področju soočala s številnimi težavami, kot so oteženo vzdrževanje aplikacij, nesposobnost sledenju sprememb in težave pri projektih, na katerih je delalo več razvijalcev.

Iz navedenih razlogov smo izvedli vpeljavo sistema Subversion (SVN) in s tem poskrbeli za urejenost na področju upravljanja konfiguracije. Potrdimo lahko, da je bila vpeljava SVN uspešna na strežniški in odjemalčevi strani iz naslednjih razlogov: (1) dosegli smo specifične cilje združenega zmožnostno-zrelostnega modela; (2) z zanemarljivimi finančnimi in časovnimi stroški smo dosegli bolj strukturirano in tekoče delo v razvojni skupini; (3) vse verzije projektov pa so preprosto in vedno dosegljive; in (4) ni več večjih težav, kadar na enem projektu oz. eni datoteki dela več razvijalcev.

Analiza je pokazala, da je bila vpeljava sistema Subversion skoraj v celoti skladna z združenim zmožnostno-zrelostnim modelom. Določene podprakse, ki jih Subversion sam ne zagotavlja, so bile razrešene z definiranjem pravil, ki veljajo znotraj organizacije. Raziskava je pokazala, da sta področje upravljanja

konfiguracije in sistem Subversion tesno povezana, saj že sama vzpostavitev sistema Subversion zagotovi večino zahtev upravljanja konfiguracije po ZZZM (združenem zmožnostno-zrelostnem modelu), dodatne procesne in organizacijske spremembe pa omogočajo popolno upravljanje konfiguracije po tem modelu.

Ker se je spremenil tudi organizacijski vidik, nas je zanimalo, kako bodo razvijalci sprejeli sistem Subversion. Odgovore smo zbrali z vprašalniki. Raziskava je bila omejena na eno organizacijo in anketirancev je bilo le devet (zaradi majhnega števila zaposlenih). Zato raziskave ne moremo posploševati. Iz strogega metodološkega vidika velja le za organizacijo, v kateri je bila izvedena anketa.

Neodvisno od uspešno izvedene vpeljave sistematičnega upravljanja konfiguracije so v organizaciji ostala še odprta vprašanja. V organizaciji, v kateri je potekala raziskava, velja pravilo, da je treba na koncu vsakega uspešno končanega projekta napisati poročilo, v katerem so povzete zahteve projekta, opis posameznih modulov ter število ur, ki jih je posamezni razvijalec vložil v projekt. Sistem Subversion sicer omogoča vpisovanje komentarjev, vendar se zaradi oblike pri izpisu ti komentarji ne morejo uporabiti kot poročilo, tako da njihovo vpisovanje ob vsaki objavi pomeni le podvajanje informacij. Ti komentarji so povezani s spremembami posameznih modulov in bi lahko iz njih na koncu projekta s posebno programsko podporo ustvarili poročilo. Vendar trenutna struktura komentarjev ni določena z nobenimi pravili. Z uvedbo določenih pravil pri vpisu komentarjev bi lahko kasneje tvorili poročila posameznih projektov glede na pravilno strukturirane komentarje, ki bi se vpisovali prek sistema Subversion.

Organizacijski vidik bi lahko izboljšali tudi s samodejnim generiranjem osnovne oblike strukture. Omenili smo že, da so strukture aplikacij sedaj precej enotne in ravno zato bi se lahko pripravila ukazna datoteka, ki bi ustvarila celotno strukturo projekta (primerno poimenovani imeniki in vnaprej ustvarjene prazne pravilno poimenovane datoteke za posamezni modul). Končni cilj je, da se vsi projekti, ki so bili razviti na preteklem načinu in so še vedno izpostavljeni nadgradnjam, prestavijo v sistem Subversion.

6 VIRI

[1] Sajko, U. (2002). Upravljanje konfiguracije kot osnova za izboljšanje programskega, magistrsko delo, Univerza v Mariboru, Maribor.

[2] Paulk, M. C., Curtis, B., Chrissis, M. B. & Weber, C. V. (1993). Key Practices of the Capability Maturity Model, verzija 1.1, raziskovalno poročilo, Software Engineering Institute, Pittsburgh.

[3] Perc, D. (2007). Povezovanje CMMI in COBIT metode v metodo izdelave ali naročanja programske opreme, magistrsko delo, Univerza v Mariboru, Maribor.

[4] The PHP Group (2008). The PHP Group, pridobljeno 17. oktobra 2008, http://si2.php.net/manual/en/introduction.php.

[5] Vajde Horvat, R., Rozman, T., Harej, K., Jerčinović, A., Polančič, G. & Pavlič, L. (2004). Metodologija za vzpostavitev in vzdrževanje sistemov vodenja kakovosti, Univerza v Mariboru, Maribor.

[6] CMMI Product Team (2002). Capability Maturity Model Integration (CMMISM), verzija 1.1, Software Engineering Institute, Pittsburgh, pridobljeno 15. oktobra 2008, http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr029.pdf.

[7] Netcraft (2008). Netcraft, England and Wales, pridobljeno 16. aprila 2008, http://news.netcraft.com/archives/2008/04/14/april_2008_web_server_survey.html.

[8] Collins-Sussman, B., Fitzpatrick, B. W., Pilato, C. M. (2008). Version Control with Subversion, Creative Commons Attribution License, California, pridobljeno 17. oktobra 2008, http://svnbook.red-bean.com/en/1.4/svn.webdav.basic.html.

[9] Winston, T. (1997). 'Introduction to Case Study',The Qualitative Report, zvezek 3, številka 2, pridobljeno 15. septembra 2008, http://www.nova.edu/ssss/QR/QR3-2/tellis1.html.

[10] Čepon, M. (2008). TortoiseSVN Odjemalec za Subversion v operacijskem sistemu Windows, verzija 1.5.2, pridobljeno 17. oktobra 2008, http://tortoisesvn.net/docs/nightly/TortoiseSVN_sl/.

[11] The Samba Team (2008). The Samba Team, pridobljeno 17. oktobra 2008, http://us1.samba.org/samba/what_is_samba.html.

[12] The PHP Group (2008). The PHP Group, pridobljeno 16. oktobra 2008, http://www.smarty.net/rightforme.php.

[13] International Organization for Standardization (2003). ISO 10007:2003, Quality management systems – Guidelines for configuration management, Ženeva.

Gregor Polančič je asistent z doktoratom na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Med njegova interesna področja spadajo tehnološki in netehnološki vidiki sistemov za sodelovanje in obvladovanje informacij, med katere spada področje obvladovanja konfiguracije programske opreme.

Gregor Jošt je na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru vpisan na univerzitetni študij na področju računalništva in informatike. Med njegove interesne dejavnosti spada tudi področje upravljanja konfiguracije s poudarkom na upravljanju konfiguracije programske opreme.