Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Peter Fabbro

Orodje za popis metode razvoja programske opreme

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Marko Bajec

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00079 / 2013
Datum: 9.4.2013

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

Kandidat: **PETER FABBRO**

Naslov: **ORODJE ZA POPIS METODE RAZVOJA PROGRAMSKE OPREME**
A TOOL FOR DOCUMENTING SOFTWARE DEVELOPMENT METHODS

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Dokumentiranje in vzdrževanje metod za razvoj programske opreme znotraj podjetja pomembno vpliva na uspešnost IT projektov in prispeva k večji kakovosti razvite programske opreme. Kljub temu pa podjetja v veliki meri še vedno nimajo popisane lastne metode razvoja in delujejo v skladu s pravili, ki temeljijo predvsem na znanju in izkušnjah posameznikov. Kandidat naj v diplomskem delu predstavi različne metode razvoja programske opreme in se spozna z omenjeno problematiko. Na podlagi pridobljenega znanja naj zasnuje in razvije orodje, ki bo podjetjem omogočalo popis metode razvoja programske opreme. Rešitev naj predstavi skozi realen primer uporabe.

Mentor:

izr. prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Peter Fabbro, z vpisno številko **63080067**, sem avtor diplomskega dela z naslovom:

Orodje za popis metode razvoja programske opreme

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Bajca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30. januarja 2014 Podpis avtorja:

Iskreno se zahvaljujem mentorju izr. prof. dr. Marku Bajcu in as. Marku Jankoviću za pomoč in usmerjanje med izdelavo diplomskega dela.

Zahvaljujem se tudi družini in prijateljem za spodbujanje med izdelavo diplomskega dela, kot tudi študijem.

Staršem in starim staršem.

Kazalo

Povzetek

V sklopu diplomske naloge je bila izdelana spletna aplikacija, ki omogoča računalniškim podjetjem vnos njihove metode razvoja programske opreme. Spletna aplikacija omogoča vnos projektov, vseh njihovih pomembnih elementov, kot npr. discipline, sprotno urejanje vnešenih podatkov med nadaljnjim razvojem opreme in izris na projektu uporabljene metode s pomočjo dveh diagramov.

V prvem delu diplome je opisan pomen uporabe ustrezne metode pri razvoju programske opreme. Opisane so tudi metode, ki so se skozi leta razvijale.

Nadaljuje se s podrobnejšim opisom izdelave spletne aplikacije, kjer imamo specifikacijo, kakšna naj bo aplikacija in kaj naj omogoča uporabnikom, ki jo uporabljajo. Opisana so tudi orodja, s katerimi je bila spletna aplikacija izdelana, od sistema za upravljanje s podatkovno bazo (kratica SUPB) MySQL, do integriranega razvojnega okolja (ang. kratica IDE) NetBeans in vseh njegovih orodij in knjižnic. Prikazan je tudi podatkovni model, ki je bil izdelan med izdelavo diplomske naloge.

V zadnjem delu je opisano delovanje spletne aplikacije na primeru uporabe, pri kateremu so v sistem vnešeni podatki, kot bi jih lahko vneslo pravo podjetje, ki bi spletno aplikacijo uporabljalo.

Ključne besede: spletna aplikacija, razvoj programske opreme, metoda, projekt, disciplina, orodje, MySQL, NetBeans, podatkovni model, primer uporabe.

Abstract

In the making of this Bachelor's thesis, a web application was developed, which gives the option to software development companies to record their software development method. The web application enables recording of projects and all of their important elements, like disciplines and their editing during the following software development. It also gives the option of drawing the project method with two diagrams.

The meaning of using a method to develop software is described in the first part of the thesis. It also describes some well known methods, developed through the years.

It goes on with a detailed description of the web application making process. We begin with the specification, of how the application should look and what options it should give it's users. There are also descriptions of the tools, used during development, from the database management system (DBMS) MySQL, to the integrated development environment (IDE) NetBeans, it's tools and libraries. The database model, made during the development of this thesis is also shown and described.

In the last part we have a description of how the web application works through a use case, which uses data, a real software company could insert in the system.

Keywords: web application, software development, method, project, discipline, tool, MySQL, NetBeans, database model, use case.

Poglavje 1

Uvod

Z razvojem informacijskih tehnologij skozi desetletja smo prišli do točke, kjer je vsakdanje življenje postalo odvisno od uporabe računalniških sistemov. Skozi čas je zaradi tega razvoj programske opreme postal vedno bolj pomemben in število podjetij, ki opremo razvijajo se je povečalo. Razvoj kvalitetne programske opreme pa predstavlja cilj podjetij, ki opremo razvijajo. Pomembno je torej, da razvoj programske opreme poteka po postopkih, ki zagotovijo, da bo končen izdelek (programska oprema) kvaliteten, izdelan v določenem roku in vzdrževanje enostavno. Tak skupek postopkov imenujemo metoda razvoja programske opreme.

Podjetja pri razvoju programske opreme lahko sledijo znanim metodam, kot pa pravi vir [12], analize kažejo, da večina današnjih podjetij v večini primerov nima zabeleženih in točno določenih metod. Uporabljene metode se tipično spreminjajo od projekta do projekta in le te obstajajo samo v glavah razvijalcev. Kot rezultat, je kvaliteta razvite programske opreme nizka in nadaljnje vzdrževanje zahtevnejše.

V diplomskem delu smo se zaradi tega odločili izdelati orodje, ki bo podjetjem omogočalo beleženje projektno specifične metode razvoja programske opreme. Orodje bo omogočalo tudi naknadna popravljanja metode. S tem

bo podjetjem v končni fazi omogočen ogled in analiza projektne metode in s tem ugotavljanje morebitnih težav in nadaljnjih spremenjenih načinov razvoja drugih projektov. Primerjanje uporabljene metode s splošno znanimi lahko namreč razkrije pomankljivosti oz. podrobnosti, katere bi bilo primerno izboljšati. Pri definiranju metod nadaljnjih projektov, bo omogočen uvoz načinov in postopkov razvoja prejšnjih projektov, nova spoznanja pa bo z urejanjem mogoče vključiti. Razvito orodje bo prav tako podpiralo aktivno sodelovanje razvijalcev projekta na razvijanju in ogledu metode.

Orodje bo z zgoraj definiranimi funkcionalnostmi podjetjem omogočalo beleženje metod, katerega se v večini primerov do sedaj ni izvajalo. Poleg tega se bo s sprotnim spoznavanjem pomankljivosti in izboljševanjem le teh omogočilo izboljšanje kvalitete končnega izdelka in enostavnejše vzdrževanje. S sodelovanjem projektnih razvijalcev pri razvoju metode, se bo omogočilo spoznavanje dejanskega načina razvoja programske opreme s strani razvijalcev, prav tako pa se jim bo postavilo dobro definirane cilje in napotke, katerim naj sledijo.

Poglavje 2

Metode razvoja programske opreme

Kot je bilo omenjeno v uvodu, je pri razvoju programske opreme pomembno, da razvijalci sledijo nekemu postopku. Pomembno je, da je problem, ki se ga s programsko rešitvijo skuša rešiti dobro raziskan in analiziran, v nasprotnem primeru kakovost končnega izdelka ni zagotovljena. Iz tega sledi, da razvoj programske opreme ni zgolj sestavljen iz programiranja ampak večjega števila opravil. Skupek teh opravil se imenuje **metoda razvoja programske opreme**. Razlogov za uporabo metode pri razvoju je veliko, v knjigi "Information system development methodologies, techniques and tools" [11] pa so, kot razlogi navedeni **višja kvaliteta končnega izdelka**, **višja kvaliteta razvojnega procesa** in **standardizacija procesa**.

2.1 Življenjski cikel razvoja programske opreme

Večina današnjih tradicionalnih metod razvoja programske opreme obsega znane postopke oz. faze, katere so tipično **analiza**, **načrtovanje**, **implementacija**, **testiranje** in **uvedba**. Ti postopki skupaj sestavljajo življenjski cikel razvoja programske opreme. [9]

Analiza je postopek zbiranja informacij o poslovnem okolju in zahtevah ter analiza le teh. V tem postopku tipično poteka izvajanje sestankov, izpolnjevanje anket s strani ključnih uporabnikov itd. Iz pridobljenih informacij se kasneje sestavi opis poslovnega okolja in strukturirano specifikacijo zahtev, kaj naj nova programska rešitev obsega in omogoča. Postopek analize je zelo pomemben, saj predstavlja temelj za kvaliteten in učinkovit nadaljnji razvoj programske opreme.

Načrtovanje je postopek sestave natančnega načrta razvoja programske opreme po specifikaciji, izdelani v postopku analize. Tipično se to realizira s pomočjo opisov in različnih vrst grafičnih vizualizacij funkcionalnosti in strukture sistema.

Implementacija je postopek dejanskega razvoja programske opreme po načrtu, izdelanem v prejšnjem postopku. Ta postopek proizvede programsko opremo, katero je naročnik naročil.

V postopku testiranja poteka pregled delovanja izdelane programske opreme in identifikacija napak oz. morebitnih popravkov.

V postopku uvedbe poteka predaja in uvedba novo razvite programske opreme pri naročniku.

2.2 Tradicionalne metode

Tradicionalne metode razvoja programske opreme so starejša vrsta metod. Uvajajo formalizirane postopke, ki pokrivajo celoten življenjski cikel razvoja in s tem odpravljajo pomankljivosti, ki nastanejo pri razvoju brez uporabe določene metode razvoja. Obstaja več modelov razvoja po tradicionalnih metodah, ki definirajo, kako poteka življenjski cikel in prehod med posameznimi postopki.

2.2.1 Modeli razvoja programske opreme

Kot je bilo zgoraj omenjeno, razvoj programske opreme je v tradicionalnih metodah razdeljen v večje število postopkov. Kako si ti postopki sledijo in kdaj se kakšen postopek lahko izvaja pa imenujemo **model razvoja programske opreme**.

Slapovni model

To je nastarejši model razvoja programske opreme. Temelji na zaporednem izvajanju postopkov razvoja. Faze si sledijo zaporedno ena za drugo, ko se ena faza konča, vrnitev nazaj ni več mogoča. Prav zaradi tega se model tako imenuje, saj pri slapu tok vedno teče v eno smer, vrnitev nazaj pa ni mogoča. Model je primeren za projekte, kjer so zahteve dobro poznane in se le te med izvajanjem projekta ne bodo spreminjale.

Z uporabo tega modela, dosežemo dobro projektno vodenje, saj je model enostaven in zahteva zelo striktno izvajanje faz. Slabosti tega modela razvoja pa so enostavno razvidne. Glavna slabost je, da model ni fleksibilen. Ker model ne predvideva vračanja nazaj, je ob morebitnih spremembah v kasnejših fazah, spreminjanje zelo težko, saj zahteva veliko dela. Zato je v primeru slabega poznavanja zahtev, uporaba tega modela odsvetovana, saj obstaja visoka verjetnost, da bo razvita programska oprema nekvalitetna. Poleg tega z uporabo tega modela, paralelno izvajanje postopkov ni mogoče, saj si le ti morajo slediti zaporedno.

Iterativni model

Iterativni model je nastal, kot odgovor na slabosti slapovnega modela. Model prav tako zahteva zaporedno izvajanje postopkov, vendar se prehod čez postopke ponavlja, dokler ne izdelamo končnega izdelka. Prehod navadno poteka čez vse postopke razvoja. Nek postopek se torej ne izvede samo enkrat ampak večkrat. En prehod čez vse postopke razvoja imenujemo **iteracija**.

V prvih iteracijah navadno razvijemo najbolj kritične dele sistema. Ker si iteracije sledijo, se lahko ena iteracija prične le, ko je prejšnja zaključena. Spremembe projekta se ne spreminjajo med samimi izvajanji iteracij ampak po zaključku iteracije, ko dobimo povratne informacije.

V primerjavi s slapovnim modelom, iterativni ima nekatere ključne prednosti. Ker razvoj poteka skozi iteracije, prve iteracije pa obsegajo najbolj kritične dele sistema, se le te reši še preden postane investicija velika. Ker so iteracije časovno kratke v primerjavi s celotnim projektom pa stalno prejemamo povratne informacije, napredek je merljiv in izdelek lahko predamo še preden je celoten projekt zaključen. Ker je razvoj razdeljen v iteracije in plan lahko spreminjamo, je vodenje projekta zahtevnejše v primerjavi s slapovnim modelov. Poleg tega ne moremo točno vedeti, koliko iteracij bo potrebnih za dokončanje projekta.

Prototipni model

Prototipni model je le različica iterativnega modela. Model temelji na izdelavi **prototipov**, izdelava pa poteka v iteracijah, dokler kakovost in funkcionalnosti izdelanega prototipa niso zadovoljive.

Inkrementalni model

Inkrementalni model temelji na razdelitvi celotnega sistema na zaokrožene funkcionalnosti - **inkremente**. Razvoj poteka po inkrementih, ko je nek inkrement izdelan pa sledi njegova predaja naročniku.

Pri inkrementalnem modelu razvoja je glavna prednost ta, da razvoj inkrementov lahko poteka paralelno. Poleg tega se z izdelavo posameznih inkrementov in predaje le teh naročniku lahko izdelan sklop že uporablja, ob izdelavi naslednjih inkrementov pa se le te doda zraven v uporabo.

2.2.2 Strukturni razvoj

Strukturni pristop je starejši in temelji na ločitvi podatkov od funkcionalnosti sistema. Z uvajanjem objektno usmerjenih programskih jezikov, se ta postopek počasi ukinja. Pri strukturnem pristopu so podatki predstavljeni s podatkovno bazo, funkcionalnosti pa so na voljo s programskimi moduli, kateri se gradijo okoli podatkovne baze.

Na voljo je več metod za razvoj po strukturnem pristopu, najbolj znana pa je **Informacijski inženiring** (IE).

Informacijski inženiring

Informacijski inženiring [7] (v nadaljevanju IE) je starejša metoda, razvita v začetku osemdesetih let prejšnjega stoletja, katere avtor je James Martin. Glavna značilnost te metode je, da uvaja strateško planiranje, kot eno izmed faz razvoja programske opreme.

IE sestavlja več postopkov:

Strateško planiranje V postopku strateškega planiranja se izvede pregled poslovnega sistema. Identificira se pomankljivosti v trenutni izvedbi informacijske podpore in pripravi nadaljnji plan za vlaganje v informatiko.

Analiza V fazi analize poteka izdelava modela sistema t.j. izdelava modela podatkovne strukture in funkcij, ki uporabljajo podatke iz podatkovne strukture.

Načrtovanje Glavna izdelka načrtovanja sta načrt podatkovne baze in podatkovnih modulov.

Izvedba V postopku izvedbe poteka dejanska izdelava programske opreme, katero smo v fazi načrtovanja definirali. Ob zaključku izdelave poteka prenos in zamenjava obstoječe z novo razvito programsko opremo.

2.2.3 Objektni razvoj

Objektni pristop razvoja je nastal z uvedbo objektno usmerjenih programskih jezikov. Od strukturnega pristopa se razlikuje v tem, da podatke in funkcionalnosti sistema ne več obravnavamo ločeno. Pri objektnem pristopu imamo objekte, ki vsebujejo podatke in metode, ki podatke uporabljajo.

Z razvojem objektno usmerjenih podatkovnih jezikov in objektnega pristopa, so se pojavile tudi objektno usmerjene metode razvoja programske opreme, med katerimi je verjetno najbolj znana **Rational Unified Process** (RUP).

Rational Unified Process

Rational Unified Process [8] (v nadaljevanju RUP) je objektno usmerjena metoda, ki jo je razvilo podjetje Rational Software. RUP temelji na zmanjšanju količine dokumentacije in predlaga uporabo Unified Modeling Language (kratica UML) za izdelavo modelov in diagramov v aktivnostih procesa. RUP sestavljata dve dimenziji:

Horizontalna dimenzija Ponazarja čas (zato v nadaljevanju časovna dimenzija). Sestavljena je iz štirih faz. **Inception**, **Elaboration**, **Construction**, **Transition**. Vsaka faza je lahko sestavljena iz več iteracij.

Vertikalna dimenzija Dimenzija, ki vsebuje postopke. RUP definira 6 osnovnih (poslovno modeliranje, zajem zahtev, analiza in načrtovanje, izvedba, testiranje in uvedba) in 3 dodatne.

Vsaka faza časovne dimenzije se zaključi s točno določenim mejnikom, v katerem so določeni cilji procesa zaključeni, znotraj vsake faze pa se lahko izvaja več iteracij.

V prvi fazi (Inception) se določi problemsko domeno in identificira problem. Pozornost v tej fazi je na prvih dveh postopkih vertikalne dimenzije (Poslovno modeliranje in Zajem zahtev), torej poteka identifikacija primerov uporab in

opis nekaterih, identifikacija kritičnih dejavnikov ter časovna postavitev glavnih mejnikov.

V drugi fazi (Elaboration) je pozornost na tretjem postopku vertikalne dimenzije (Analiza in razvoj). V tej fazi se torej tipično v glavnem izvede dopolnitev primerov uporabe in akterjev (nadaljnja identifikacija in izdelava večine opisov) in vseh ostalih rezultatov prve faze ter izdela arhitekturo sistema in plan razvoja.

V tretji fazi (Construction) je pozornost na četrtem in petem postopku vertikalne dimenzije (Izvedba in testiranje), izvede se torej dokončanje razvoja programske opreme in testiranje le te. Izdela se tudi vsa potrebna navodila za uporabo opreme.

V četrti fazi (Transition) pa se izvede prenos nove programske opreme k končnim uporabnikom, ko programska oprema izpolnjuje zadostni nivo razvoja in kvalitete. Pozornost je torej na šestem postopku vertikalne dimenzije (Uvedba). Tipično po pričetku uporabe s strani uporabnikov, se odkrijejo novi problemi in zahteve za spremembe. Pri razvijalcih programske opreme je ta faza tipično pomembna in znotraj nje se izvede več iteracij zaradi odkritja novih problemov in želj po spremembah in popravkih.

2.2.4 Prednosti in slabosti tradicionalnih metod

Tradicionalne metode razvoja programske opreme ponujajo visoko stopnjo formalnosti kar pomeni, da so le te težko prilagodljive in priporočajo načine, ki so pri npr. majhnih projektih nepotrebni. Med razvojem po tradicionalnih metodah, nastaja veliko dokumentacije in vmesnih izdelkov. To pripelje do dodatne porabe virov in časa, katerega z uporabo v nadaljevanju opisanih agilnih metod ni potrebno. To je tudi eden izmed razlogov za nizko uporabo metod pri podjetjih.

Visoka stopnja formalnosti pa ne vedno predstavlja prav slabosti, saj je le ta pri velikih in kritičnih projektih zaželena. S pomočjo dobro definiranih in znanih postopkov lahko namreč zagotovimo, da bo razvoj zahtevnega projekta kvaliteten. Obsežna dokumentacija pa ponuja enostavnejše vodenje in spreminjanje v nadaljevanju, katero je lahko brez uporabe metode težavno.

2.3 Agilne metode

Zaradi visoke formalnosti in težavnega prilagajanja obstoječih tradicionalnih metod, se je v devetdesetih letih prejšnjega stoletja začel razvoj **agilnih metod** [2]. Z izidom manifesta agilnega razvoja programske opreme leta 2001, se je zavedanje o obstoju nove filozofije razvoja začelo širiti. Manifest je poudarjal štiri glavna načela, ki so pomembnejša od predhodno upoštevanih:

Posamezniki in komunikacija nad procesi in orodji.

Delujoča programska oprema nad razumljivo dokumentacijo.

Sodelovanje s stranko nad usklajevanjem pogodbe.

Odziv na spremembe nad upoštevanjem definiranega načrta.

Agilne metode temeljijo na sodelovanju skupine razvijalcev skozi iteracije razvoja, ki so časovno kratke. Rezultat vsake iteracije, je že uporabna programska koda, katero se potem v naslednji iteraciji ponovno ureja, glede na nove zahteve stranke, ki konstanto sodeluje pri razvoju. Glede na navedeno, je agilni razvoj najbolj podoben inkrementalnemu modelu razvoja. V nadaljevanju je opisana najbolj pogosto uporabljena agilna metoda SCRUM.

2.3.1 Scrum

Pri metodi Scrum, razvoj poteka preko iteracij imenovanih **sprinti**. Vsak sprint proizvede uporaben izdelek oz. inkrement. Razvijalci so med razvojem združeni v razvojne skupine, zato jih imenujemo **člani razvojne skupine**

(team members). Vodja skupine se imenuje **Scrum master**, ki je odgovoren za vodenje in nadzor izvajanja projekta. Pred začetkom vsakega sprinta, se člani razvojne skupine zberejo na sestanku za načrtovanje sprinta. Tukaj člani skupine sestavijo **seznam zahtev** (backlog items), ki predstavlja zadeve, ki bodo tekom sprinta razvite. Pri Scrum, vsak sprint poteka v štirih fazah:

Razvoj člani skupine, navedene zahteve razvijejo, rezultat testirajo in zapišejo spremembe.

Sestava med sestavo, člani sestavijo izdelek v ustrezno delujočo obliko za predstavitev.

Pregled pri pregledu, se izdelek predstavi, določi nove (popravjene) zahteve in oceni tveganja.

Popravki v zadnji fazi, skupina vse nove zahteve določene pri pregledu združi.

Po zaključku faz se izvede nov sestanek, na katerem se naredi celoten pregled, kjer se oceni napredek izveden tekom sprinta in uredi seznam zahtev, glede na prioriteto posameznih zahtev.

2.3.2 Prednosti in slabosti agilnih metod

Agilne metode postavljajo v ospredje boljšo neposredno komunikacijo med razvijalci in disciplinirano, namesto visoke formalnosti in obsežne dokumentacije. Z dobrim komuniciranjem, širjenjem znanja in boljšimi povratnimi informacijami, potrebe po točno definiranih in težko prilagodljivih postopkih ni več, prav tako se zmanjša število vmesnih izdelkov.

V večini primerov pa je pri agilnih metodah problem, da so le te prisotne samo v glavah razvijalcev. Prav tako je pri projektih z visoko kritičnostjo, višja formalnost in rigoroznost zaželena, saj ponuja postopke, ki problem dobro definirajo in rešijo.

Poglavje 3

Orodje za popis metode razvoja programske opreme

3.1 Specifikacija zahtev

Naloga orodja, izdelanega v sklopu diplomskega dela je beleženje projektno specifičnih metod razvoja programske opreme. Uporabnikom mora omogočati vnos, sprotno urejanje in končni prikaz metod ter tako identifikacijo pomankljivosti in podobnosti uporabljene metode s splošno priznanimi. Podjetja lahko tako identificirajo morebitne popravke metode, z namenom izboljšave končne kvalitete razvite programske opreme. V tem poglavju bo predstavljena specifikacija vseh funkcionalnosti, ki jih mora orodje, ki bo v sklopu diplomske naloge razvito omogočati.

3.1.1 Pravice skrbnika

Orodje mora podpirati dve vrsti uporabniških računov (navadnega uporabnika in skrbnika). Skrbnik ima dovoljenje do upravljanja vseh uporabniških računov in kreacije projekta. Ko se uporabnik registrira v sistem, lahko njegovo registracijo potrdi le skrbnik. Ta lahko prav tako ureja vse podatke uporabnikov ter uporabnike tudi sam kreira. Skrbnik prav tako mora biti edini, ki lahko projekt začetno kreira. Med kreacijo projekta, poleg osnovnih

podatkov navede tudi projektnega vodjo, ki delo na projektu nadaljuje.

3.1.2 Pravice vodje projekta

Orodje naj omogoča nadaljnje urejanje projekta projektnemu vodji. Pravice in funkcionalnosti, ki naj jih orodje omogoča projektnemu vodji pri urejanju projekta, lahko razdelimo po podpoglavjih.

Upravljanje podatkov projekta

Projektni vodja naj ima pravico do ponovnega urejanja podatkov o projektu, ki jih je vnesel skrbnik. Prav tako lahko vnaša razvijalce projekta, ki z določeno vlogo na projektu nastopajo in te naknadno briše. Omogočen naj mu bo tudi vnos in urejanje disciplin, ki projekt sestavljajo z zaporednim izvajanjem. Na voljo naj bo tudi grafični prikaz zaporedja aktivnosti z diagramom in uvažanje podatkov projekta iz starejših obstoječih projektov.

Upravljanje disciplin

Pri disciplinah, projektni vodja lahko ponovno ureja osnovne podatke ter vnaša aktivnosti, ki sestavljajo disciplino in predstavljajo neko točno določeno izvajanje znotraj discipline. Pri aktivnostih naj bo omogočeno tudi hranjenje treh pod elementov: **ciljev**, **pridobitev** in **omejitev**. Cilji predstavljajo naše želje o tem, kaj želimo z izvajanjem aktivnosti doseči. Pridobitve predstavljajo pozitivne dosežke, ki jih aktivnost prinese, če je bila njena izvedba kvalitetna. Omejitve pa predstavljajo vse dejavnike, ki lahko negativno vplivajo na kvalitetno izvajanje aktivnosti. S pomočjo dodatnih elementov (AND/OR vejitvena člena ter sinhronizacijski element) naj ima možnost izdelave in naknadnega urejanja sosledja aktivnosti, katerih izvajanje se za razliko od disciplin lahko veji v več vzporednih ter nazaj združuje v en tok. Omogočen naj mu bo tudi ogled sosledja aktivnosti z grafičnim prikazom (diagramom sosledja aktivnosti).

Upravljanje aktivnosti

Projektni vodja lahko pri aktivnostih ponovno ureja vnešene podatke. Aktivnostim lahko tudi pripiše razvijalce projekta (vnešene pri urejanju projekta), ki imajo potem pravico aktivnost prav tako urejati.

Upravljanje artefaktov

Aktivnosti, kot rezultat svojega izvajanja proizvedejo **artefakte**, ki predstavljajo izdelek. Določena aktivnost poleg tega, da artefakte ustvari, lahko tudi pri svojem izvajanju uporablja že obstoječe, ki so nastali v prejšnjih aktivnostih. Projektni vodja naj ima zato omogočen tudi vnos artefaktov in pripis le teh aktivnostim (ali jih aktivnost ustvari ali uporablja).

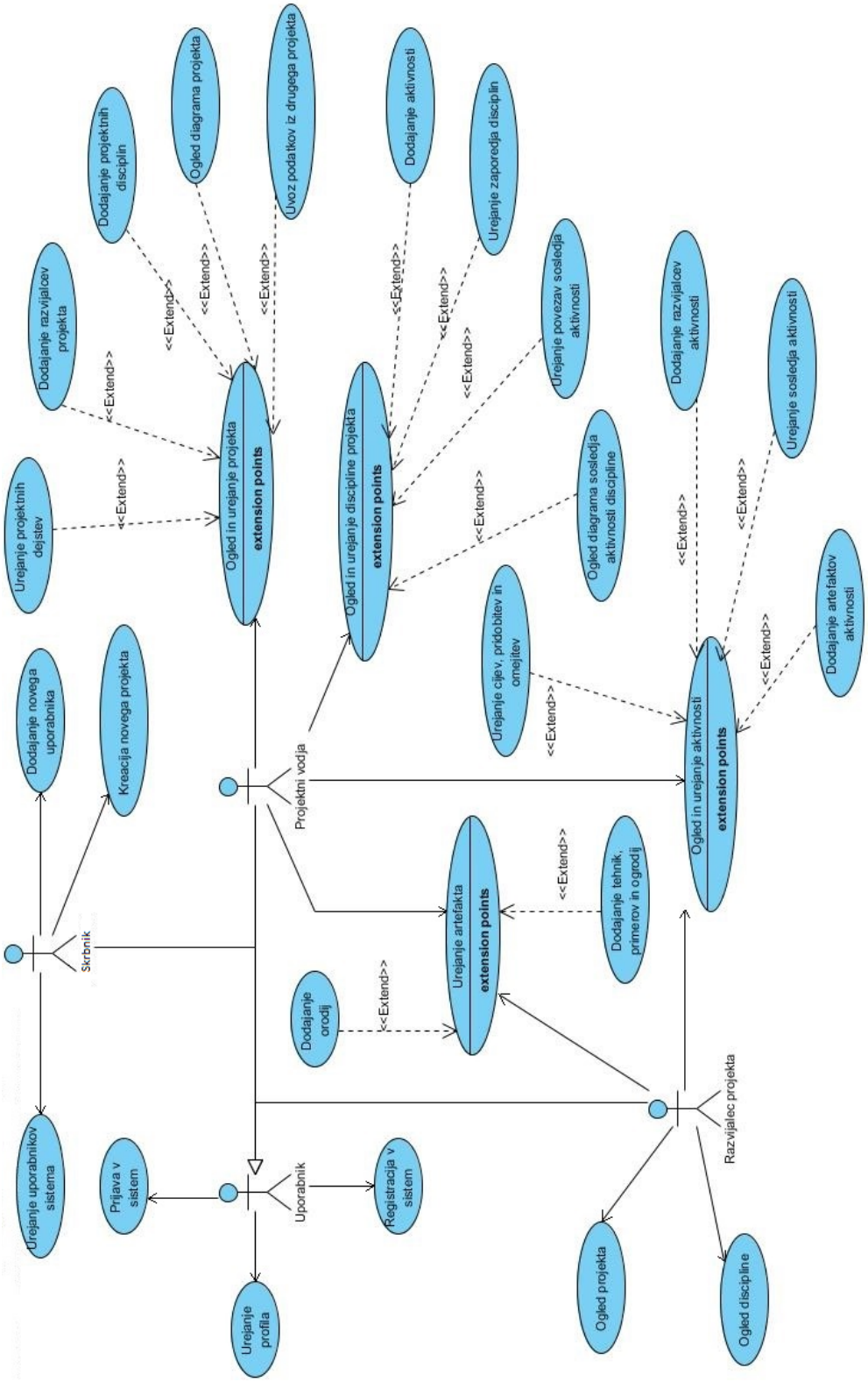
Izdelava artefaktov je mogoča s pomočjo orodij, katerih hranjenje podatkov je prav tako omogočeno. Poleg tega pri kreiranju artefakta, si lahko uporabniki, ki aktivnost izvajajo, pomagajo s pomočjo **tehnik**, **primerov** in **ogrodij**. Tehnike predstavljajo sredstvo, ki je v pomoč pri izdelavi artefakta in je podprto z orodjem. Primeri predstavljajo že izdelano različico podobnega artefakta, po katerem se lahko zgledujemo pri izdelavi trenutnega. Ogrodja pa predstavljajo izdelane predloge, po katerih lahko izdelamo vse pomembne aspekte artefakta.

3.1.3 Pravice razvijalcev projekta

Razvijalec projekta naj ima možnost ogleda podatkov celotne metode projekta, na kateri nastopa z določeno vlogo. Če je projektni vodja razvijalca pripisal določeni aktivnosti, naj ima razvijalec na njej tudi možnost urejanja. Prav tako naj lahko ureja vse njene pod elemente (cilje, pridobitve, omejitve in artefakte).

3.1.4 Diagram primerov uporabe

Funkcionalnosti orodja lahko grafično prikažemo s pomočjo diagrama primerov uporabe, pri katerem uporabimo **Unified Modeling Language** (UML) [10]. V diagramu primerov uporabe, so prikazani akterji, ki so v sistemu v interakciji z določenimi primeri uporabe. Primeri uporabe pa ponazarjajo neko funkcionalnost sistema.



Slika 3.1: Diagram primerov uporabe.

3.2 Predstavitev uporabljenih tehnologij

Ker tema diplomske naloge dopolnjuje večji projekt, ki je v izvedbi znotraj laboratorija, so nekatere tehnologije bile izbrane, glede na že uporabljene v celotnem projektu. To je seveda predstavljalo dodaten in zanimiv izziv, saj predhodnega znanja iz določenih tehnologij ni bilo. Glavni parameter za izbiro ostalih orodij in tehnologij, pa je bila odprtokodnost, saj to pomeni, da je njihova uporaba prosta.

3.2.1 MySQL

MySQL je **sistem za upravljanje s podatkovno bazo** (SUPB), v katerem je bila realizirana podatkovna baza, na kateri sloni aplikacija. MySQL ponuja možnost realizacije relacijske podatkovne baze, ki za delo s podatki uporablja znan jezik SQL. Glavni razlog za izbiro MySQL proti konkurenci, je seveda že zgoraj navedena odprtokodnost.

3.2.2 Integrirano razvojno okolje NetBeans

NetBeans je integrirano razvojno okolje (IDE), ki je namenjeno za razvoj v Javi in drugih jezikih, kot C, C++, PHP in HTML5. Za potrebe diplomske naloge, smo naložili NetBeans 7.3 za Java EE razvijalce, poleg tega je bil dodan tudi odprtokodni aplikacijski strežnik GlassFish 3.1.1.

Java EE

Java Enterprise Edition [5] (v nadaljevanju Java EE) je platforma, ki razširja Java Standard Edition (Java SE) z novimi komponentami. Komponente med seboj komunicirajo z uporabo ustreznih pravil. Vsaka komponenta se izvaja znotraj ustreznega okolja imenovanega Container, ki ji zagotavlja določene storitve.

Java Server Faces (v nadaljevanju JSF) je komponenta, ki se nahaja v web containerju in razvijalcem omogoča razvoj spletnih aplikacij za javanske

strežnike.

Enterprise Java Beans (EJB) je komponenta, ki se nahaja v EJB containerju. EJB ponuja vrsto storitev, med glavnimi pri razvoju spletne aplikacije pa je bila Java Persistence API (JPA), ki predstavlja vmesnik za dostop do podatkov v bazi s pomočjo preslikovanja v objekte preko Object Relational Mapping (ORM). Delo nad podatki pa poteka s pomočjo Java Persistence Query Language (JPQL), ki je jezik podoben SQL, za upravljanje z objekti v podatkovni bazi.

HTML in XHTML

Hyper Text Markup Language [4] (v nadaljevanju HTML) je označevalni jezik za določitev prikaza in vsebine spletne strani. Značke (tags) so vrste ukazov, ki se nahajajo med znakom < in znakom >. Tako je npr. primer značke <html>, ki označuje začetek html dokumenta (začetna značka), značka </html> pa konec dokumenta (končna značka). Razvijalci spletnih strani imajo možnost med značke navesti tudi razne vrste atributov, kot npr. id elementa, poravnanost elementa in še mnoge druge.
Extensible Hyper Text Markup Language [4] (v nadaljevanju XHTML) se uporablja za enak namen kot HMTL, vendar njegova zasnova je usklajena z XML sintakso, kar pomeni, da obstajajo strožji pogoji pri tvorjenju. Medtem ko se pri HTML posamezne značke lahko izpusti, mora biti struktura XHTML hierarhično pravilno zapisana (vsaka začetna značka mora imeti tudi ustrezno končno značko na pravem nivoju hierarhije). Uporaba strukture usklajene z XML sintakso tudi pomeni, da se lahko uporablja enostavnejše razčlenjevalnike od HTML.

CSS

Cascading Style Sheets [3] (CSS) je slogovni jezik za izdelavo izgleda spletne strani. S pomočjo CSS izdelamo predlogo za HTML ali XHTML stran in s tem ločimo izgled od vsebine strani, kar nam omogoča večjo preglednost, ponovno uporabo definiranih izgledov in s tem krajšo programsko kodo. La-

stnosti, ki jih s CSS urejamo so tako npr. barve, velikosti, obrobe, dodatni barvni efekti, odmiki in ostale lastnosti izgleda gradnikov kot tudi dodatne efekte ob interakciji s temi gradniki, kot npr. ob prehodu miške.

Primer CSS kode, uporabljene za izgled naslova znotraj izdelane spletne aplikacije:

```
.caption {
        font-family: arial;
        font-size: large;
        color:darkslategray;
}
```

Primefaces

PrimeFaces je javanska knjižnica, ki vsebuje široko paleto gradnikov za kreacijo enostavnega in funkcionalnega grafičnega vmesnika za JSF spletno aplikacijo v enotni .jar datoteki. Znotraj XHTML datoteke je mogoče s pomočjo definiranih PrimeFaces značk s predpono "p:", vključiti vizualne gradnike, ki spletno stran sestavljajo. Namen nekaterih gradnikov je samo vizualni, kot npr. <p:separator/>, ki postavi ločno črto v dokument, drugi pa se sklicujejo na podatke (npr. <p:inputText/>) in metode (npr. <p:commandButton/>) na strežnikovi strani.

Knjižnica tudi definira razrede za kreacijo raznih podatkovnih struktur, na katere se gradniki sklicujejo (npr. TreeNode).

AJAX

Asynchronous JavaScript And XML [1] (v nadaljevanju AJAX) je koncept uporabe obstoječih tehnologij za osveževanje samo potrebnih gradnikov in ne celotne spletne strani. AJAX deluje na način, da na strežnik pošilja samo potrebne podatke. Ko spletna stran čaka na odgovor strežnika, lahko uporabnik še vedno nemoteno opravlja svoje delo. Ko strežnik vrne podatke, aplikacija osveži samo ustrezne podatke, brez celotnega osveževanja strani.

Primer uporabe AJAXa v naši spletni aplikaciji, je osvežitev ustrezne forme ob izbiri določenega podatka iz kombinirane izvlečne liste, brez celotnega osveževanja strani.

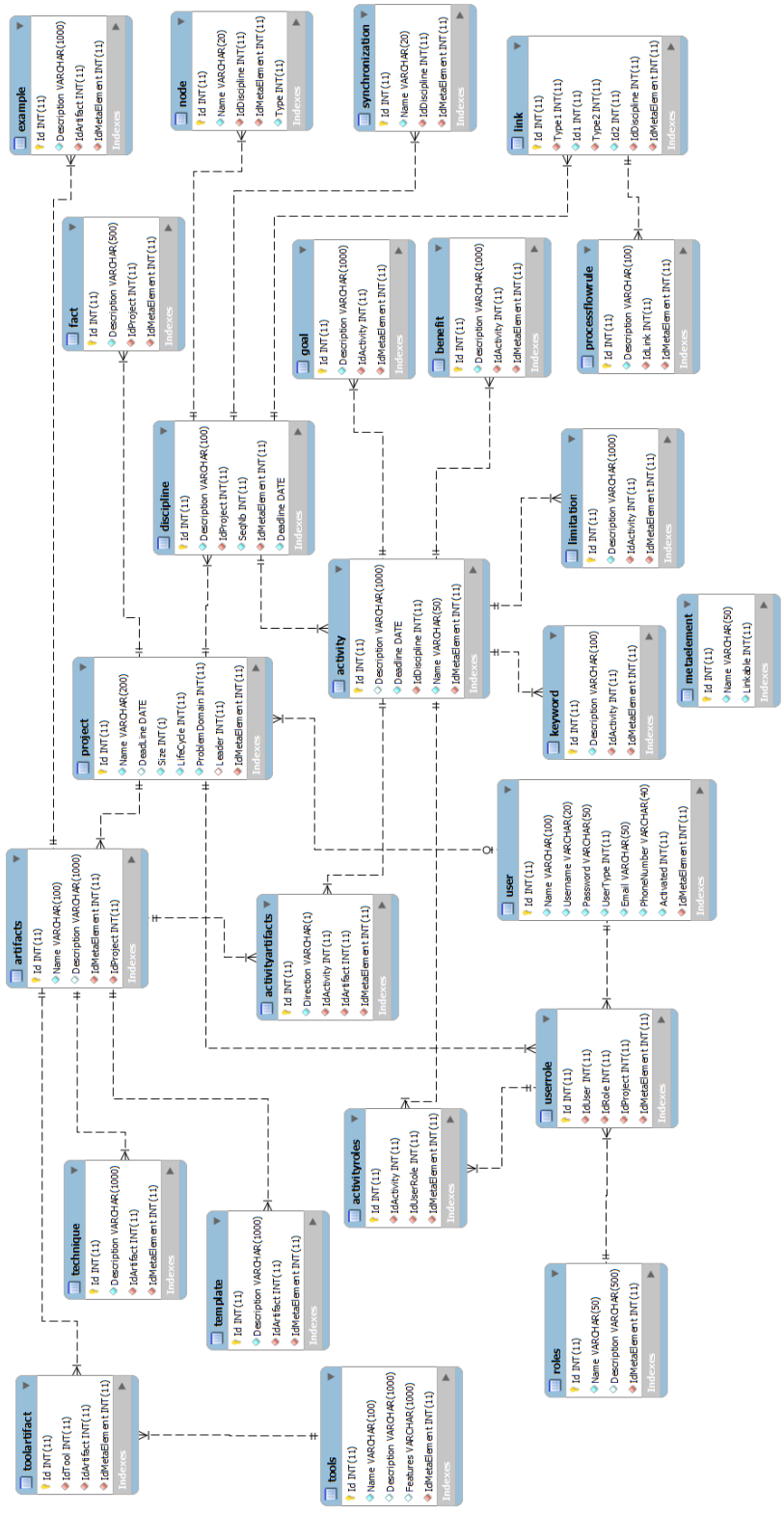
JointJS

JointJS [6] je javascript knjižnica za HTML5. Knjižnica ponuja široko paleto možnosti in načinov za vizualizacijo grafov in diagramov, tako statičnih kot tudi interaktivnih z vektorsko grafiko. JointJS temelji na ločitvi grafa, elementov in povezav med elementi (Model) od pogleda (View). Model skrbi za definicijo elementov, njihovih oblik, View pa skrbi za prikaz le teh na zaslonu. JointJS ponuja poleg osnovnih diagramskih elementov (pravokotnik, elipsa, tekst...) tudi diagramske elemente raznih standardov (npr. UML), animacije itd.

V spletni aplikaciji je bila knjižnica uporabljena za izdelavo diagramov, ki prikazujejo sosledje izvajanja postopkov (disciplin) in aktivnosti projekta.

3.3 Podatkovni model

Kot je bilo omenjeno, je bila podatkovna baza izdelana v MySQL. Za namen prikaza strukture podatkovne baze, je bil preko programskega okolja MySQL Workbench izvožen entitetno-relacijski (ER) diagram obstoječe baze.



Slika 3.2: ER diagram podatkovne baze.

ER diagram vsebuje 24 entitetnih tipov. Z namenom razumljivega prikaza, so bila na diagramu razmerja entitetnega tipa metaelement odstranjena, saj je ta entitetni tip povezan z vsemi ostalimi, preko atributa [IdMetaElement] v ostalih entitetnih tipih. Enolični identifikatorji vseh entitetnih tipov so identifikacijske številke entitete [id].

Osnovni entitetni tipi bodo skozi primer obrazloženi v poglavju 4 tako, da bodo tukaj predvsem pojasnjena razmerja in povezovalni entitetni tipi.

V entitetnem tipu userrole se hranijo entitete projektnih razvijalcev, ki so uporabniki [IdUser], ki z določeno vlogo [IdRole] nastopajo na projektu [IdProject], v entitetnem tipu activityroles pa so ti razvijalci [IdUserRole] povezani na določeno aktivnost [IdActivity], katero lahko urejajo.

Entitetni tip activityartifacts vsebuje entitete, ki povezujejo določen artefakt [IdArtifact] na aktivnost [IdActivity], entitetni tip toolartifact pa povezuje orodja [IdTool] z artefakti [IdArtifact] torej, ko določeno orodje pripišemo artefaktu, se entiteta vnese v entitetni tip toolartifact.

Med sestavljanjem sosledja aktivnosti, se za vsako povezavo vstavi entiteto v entitetni tip link. Atribut [Type1] hrani vrsto elementa na začetku povezave, [Id1] pa njegov enolični identifikator. Princip je enak za atributa [Type2] in [Id2] le, da se tukaj hrani podatek o elementu na koncu povezave. Ob vnosu entitete, se torej ustvari povezava od [Type1, Id1] do elementa [Type2, Id2].

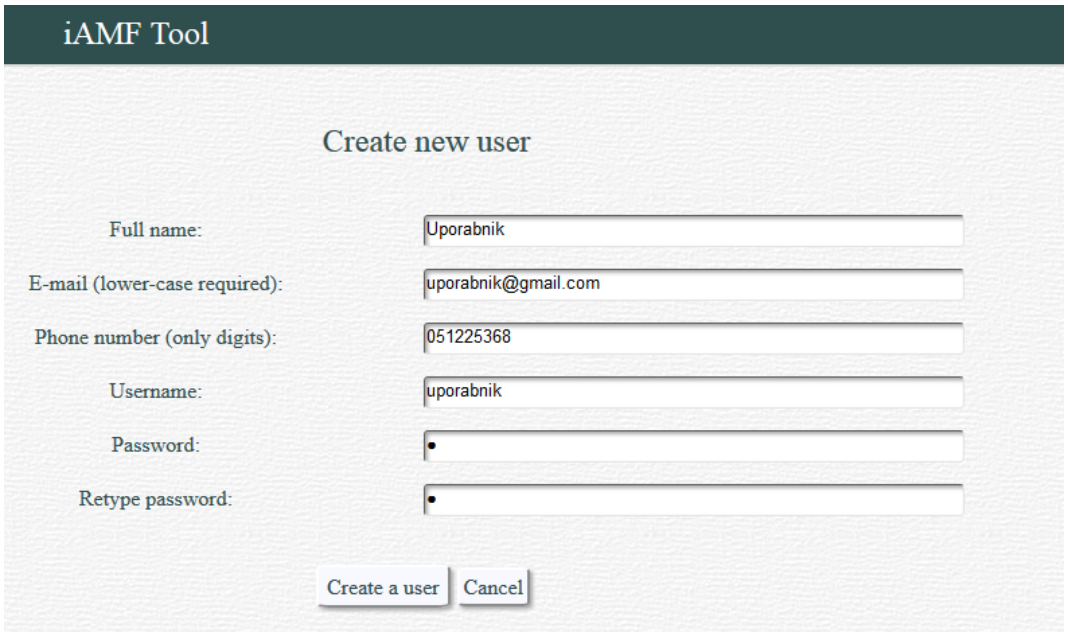
Poglavje 4

Primer uporabe orodja

V tem poglavju bo podrobneje prikazan postopek vnosa podatkov v spletno aplikacijo. Primer naj bi prikazoval način vnosa, ki bi potekal v podjetju. Predpostavljamo, da podjetje, ki uporablja aplikacijo, med razvojem programske opreme sledi na RUP zasnovani metodi. V primeru uporabe bo prikazan vnos za postopek zajema zahtev, ki bo prikazal vse funkcionalnosti, s katerimi se lahko projekt vnese do konca.

4.1 Registracija uporabnika

Ob zagonu spletne aplikacije se najprej odpre prijavno okno. Preden lahko uporabnik dostopa do nadaljnjih vsebin, si mora ustvariti svoj uporabniški račun. Ta postopek lahko izvede s klikom na povezavo **Create new account**. Sedaj se uporabniku odpre nova forma, v katero vnese vse potrebne podatke. Novemu uporabniku so na voljo za vnos naslednji uporabniški podatki.



Slika 4.1: Registracija novega uporabnika.

Ob vnosu vseh potrebnih podatkov in kliku na gumb **Create user**, je sedaj uporabnik vnešen v sistem, vendar prijava v sistem še ni mogoča. Uporabnik mora sedaj počakati, da skrbnik aktivira njegov račun. Ob aktivaciji bo prijava v sistem mogoča.

4.2 Glavni meni spletne aplikacije

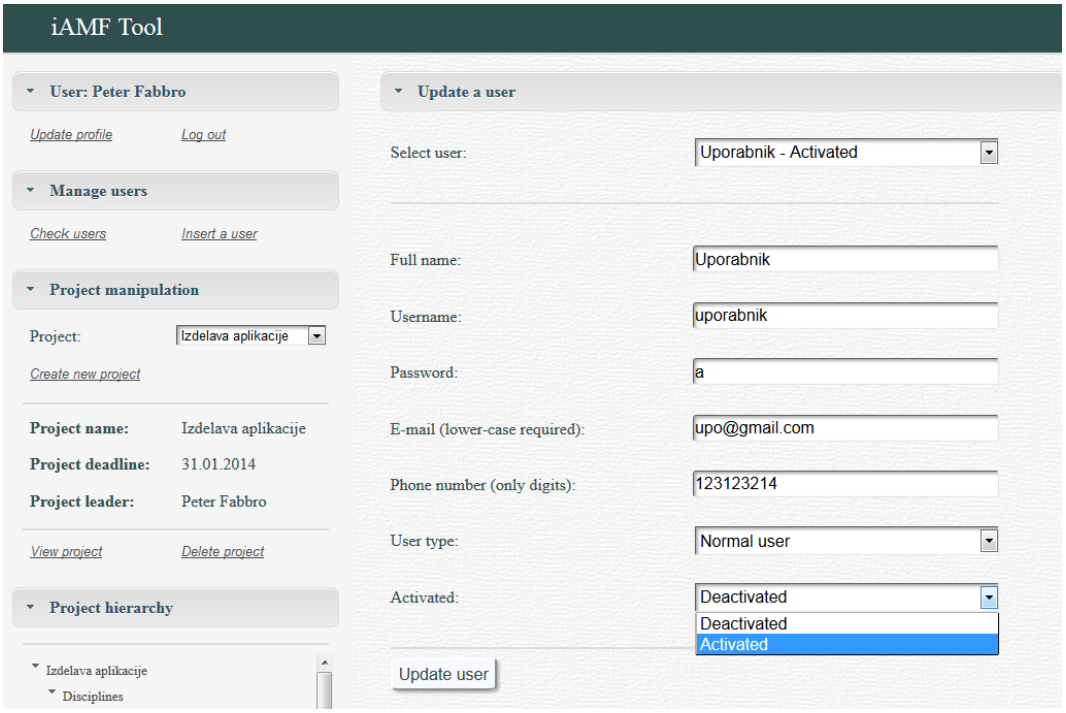
Ob uporabnikovi prijavi v sistem, se odpre začetno okno, ki je sestavljeno iz glavnega menija na levi strani okna (prisotno pri celotni uporabi aplikacije) in sprejemno besedilo, v katerem so uporabniku pojasnjene splošne funkcionalnosti sistema.
Glavni meni je sestavljen iz več segmentov, katerih vidljivost je odvisna od vrste uporabniškega računa. Vsak uporabnik ima ne glede na vrsto računa možnost urejanja svojega profila (urejanje podatkov, ki jih je navedel pri registraciji) in možnost odjave iz sistema. Obe funkcionalnosti sta dosegljivi v

zgornjem delu glavnega menija (prvi segment).

4.2.1 Skrbniški pogled na uporabniške račune

Pod zgoraj navedenima možnostima, je prisoten drugi segment glavnega menija, ki je dosegljiv le skrbniku.

Opcija **Check users** omogoča pregled in urejanje poljubnega uporabnika z izborom preko kombinirane izvlečne liste. Poleg podatkov, ki jih vidi navaden uporabnik pri registraciji, ima skrbnik še dodaten pogled na tip uporabnika (navaden uporabnik ali skrbnik) in aktivacijo.



Slika 4.2: Urejanje obstoječega uporabnika.

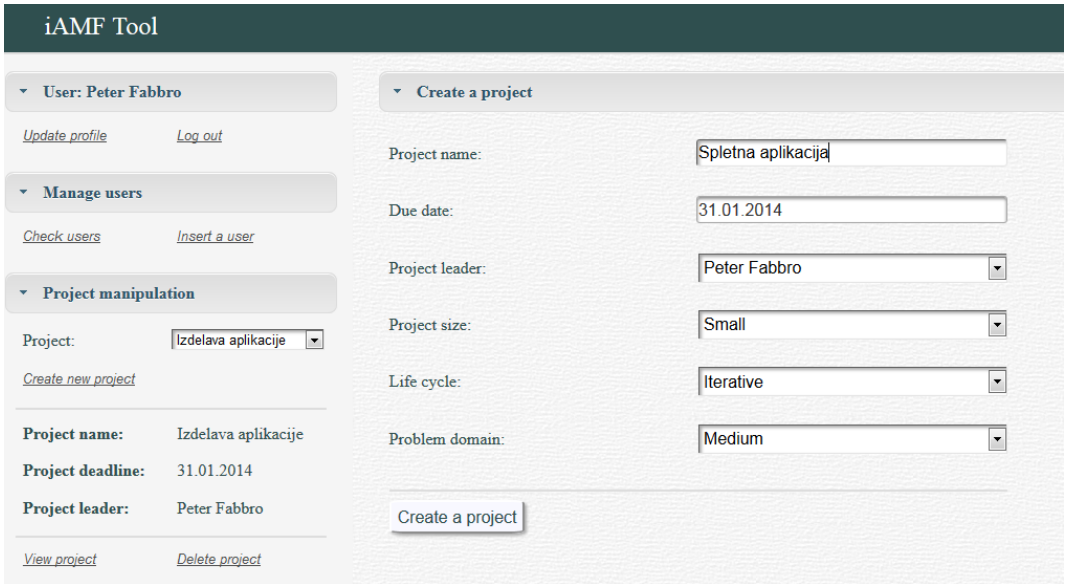
S klikom na gumb **Update user**, se podatki v podatkovni bazi ažurirajo. Poleg posodobitve računov, lahko skrbnik tudi vnaša nove uporabniške račune preko opcije **Insert a user**, pri katerih ima omogočen vnos vseh podatkov, ki jih hranimo o uporabniku.

4.2.2 Vpogled v projekte

Glavna funkcionalnost menija je upravljanje s projekti. V segmentu **Project manipulation** ima uporabnik ponujeno možnost vnosa novega, izbiro in nadaljnji vpogled obstoječega (preko kombinirane izvlečne liste), ter brisanje izbranega projekta. Ob izbiri obstoječega projekta, se v zadnjem segmentu **Project hierarchy** tudi sestavi drevesna struktura, preko katere ima uporabnik dostop do ostalih projektnih elementov. Navedene funkcionalnosti so omejene glede na vrsto in vlogo uporabnika.

4.3 Vnos in urejanje projekta

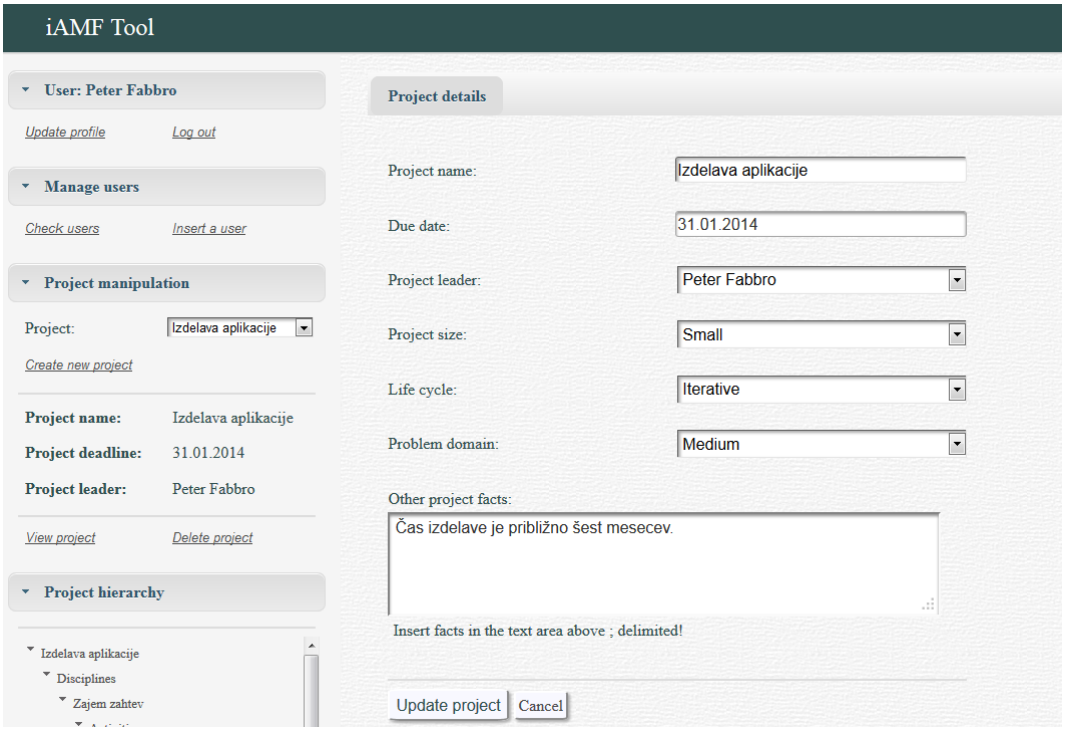
Za vnos novega projekta, ja na voljo gumb **Create new project**, ki je dostopen samo skrbnikom. Ob vnosu vseh osnovnih podatkov, skrbnik zaključi vnos s pritiskom na gumb **Create a project**. S tem je projekt vnešen v sistem. V našem primeru so osnovni podatki, ki jih vnese skrbnik naslednji:



Slika 4.3: Vnos projekta.

Z izbiro projekta v glavnem meniju (opcija, ki je sedaj na voljo projek-

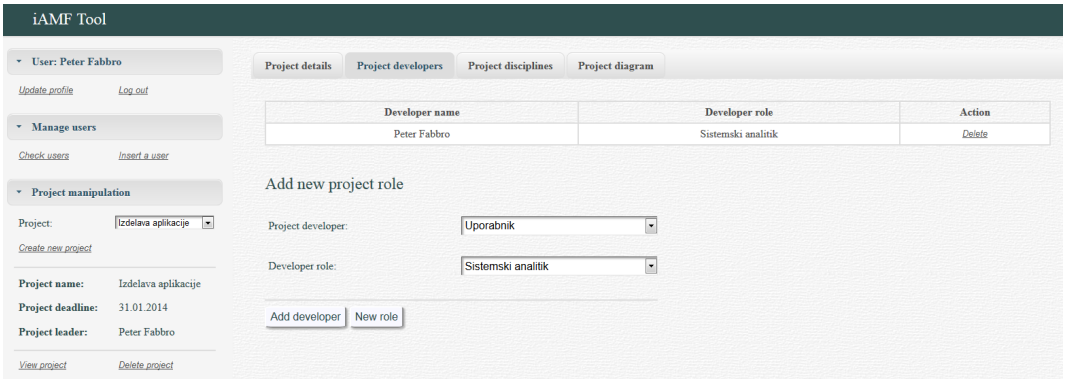
tnemu vodji), se pojavita možnosti **View project** in **Delete project**. Z ogledom projekta, lahko vodja ureja podatke, ki jih je vnesel skrbnik (s klikom na gumb **Edit project**) ter doda še dodatna dejstva, ki so znana o projektu (velikost projekta, življenski cikel in poznavanost problemske domene so že ponujena dejsta). Dejstva se vpisuje v tekstovno polje, ločene s podpičjem.



Slika 4.4: Urejanje osnovnih podatkov projekta in vnos dejstva.

4.3.1 Vnos razvijalcev projekta

Razvijalci projekta so uporabniki, ki lahko v različnih vlogah nastopajo na projektu. Njihov vnos je mogoč v drugem zavihku projektnega pogleda. Predhodno dodani razvijalci so že prisotni v tabeli, dodajanje novih pa je mogoče z izbiro uporabnika in vloge, s katero nastopa na projektu.

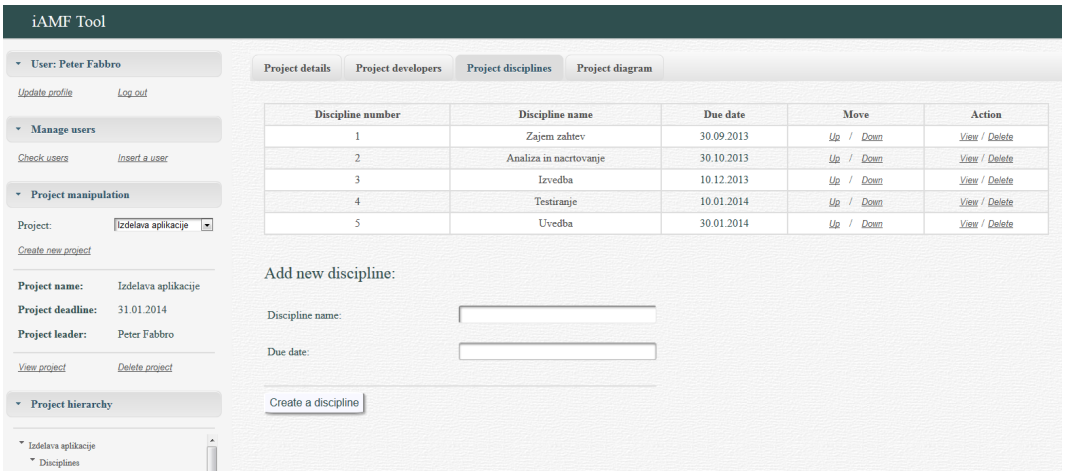


Slika 4.5: Vnos razvijalca projekta.

Po izbiri uporabnika in vloge, se s pritiskom na gumb **Add developer** razvijalec doda k projektu, za kreiranje samo nove vloge pa je na voljo gumb **New role**, ki odpre formo za vnos. Projektu dodane razvijalce, je kasneje mogoče pripisati posamezni aktivnosti, katere podatke in pod elemente lahko urejajo.

4.3.2 Vnos in urejanje discipline

Vnos discipline v projekt, je mogoč preko tretjega zavihka projektnega pogleda. Pri vnosu discipline, je potrebno vnesti **naziv** in **datum dokončanja**. Poleg vnosa discipline, je prisotna v oknu tudi tabela, ki vsebuje že obstoječe projektne discipline. V tabeli z obstoječimi disciplinami ima projektni vodja na voljo spreminjanje vrstnega reda (ki obrne tudi datume dokončanja), brisanje in seveda vpogled. Kot je bilo že zgoraj omenjeno, bo za namen predstavitve v nadaljevanju delo prikazano na disciplini **Zajem zahtev**.

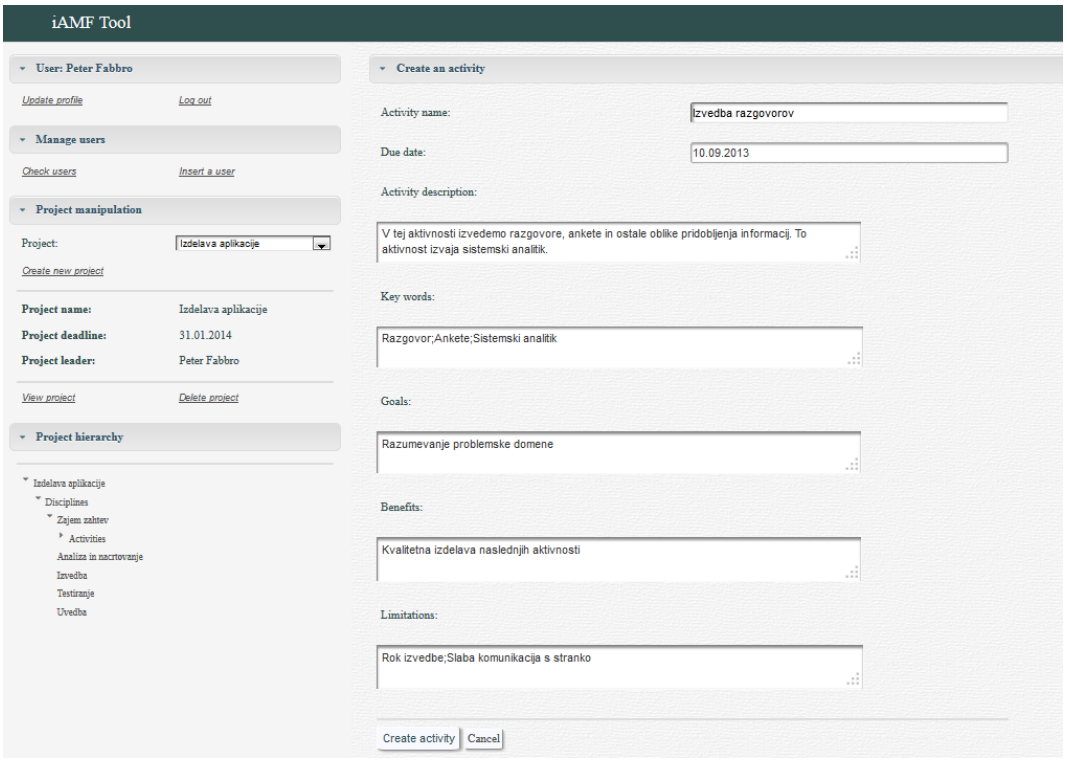


Slika 4.6: Dodajanje discipline projektu.

S pritiskom na opcijo **View** v tabeli disciplin, lahko uporabnik dostopa do pogleda discipline, kjer lahko ob vnosu, navedene podatke ponovno ureja, na voljo pa ima tudi bolj pomembne in zanimive opcije, opisane v nadaljevanju.

4.3.3 Vnos in urejanje aktivnosti

Ogled in dodajanje aktivnosti, je mogoče preko drugega zavihka pogleda discipline. Tukaj je prav tako prisotna tabela obstoječih aktivnosti, katere lahko projektni vodja briše ali ureja. Dodajanje nove aktivnosti pa je mogoče preko gumba **New activity**. Ko se okno za vnos aktivnosti odpre imamo možnost vnesti osnovne podatke aktivnosti. Vnos prve aktivnosti, je prikazan na spodnji sliki:



Slika 4.7: Vnos prve aktivnosti zajema zahtev.

Pri vnosu aktivnosti, lahko projektni vodja vnese tudi cilje (**goals**), pridobitve (**benefits**) in omejitve (**limitations**). Za vsake izmed navedenih elementov je prisotno tekstovno polje, posamezne elemente pa ločimo s podpičjem, kot dodatna dejstva pri projektu. Pri aktivnosti **Izvedba razgovorov** lahko vnesemo naslednje:

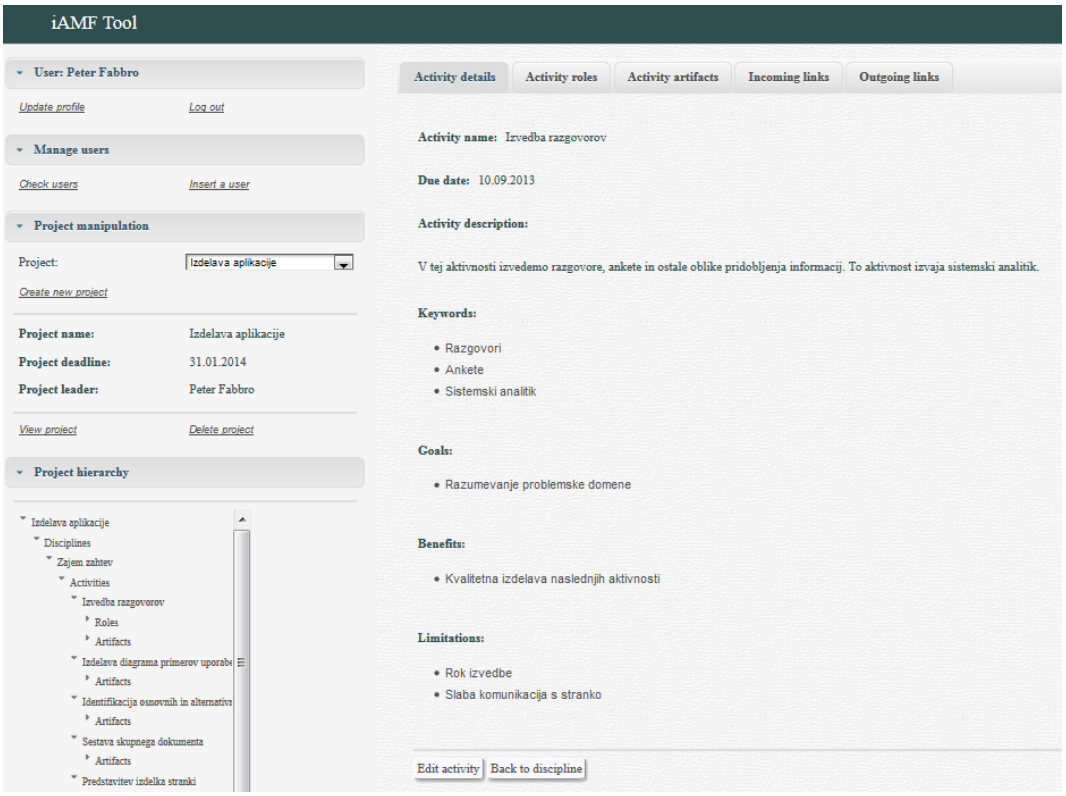
Cilji za našo aktivnost, je lahko primer cilja **dobro razumevanje problemske domene**

Pridobitve ker je cilj dobro razumevanje problemske domene, je pridobitev kvalitetnješa in lažja izdelava naslednjih aktivnosti, ker smo problem dobro identificirali.

Omejitve pri prvi aktivnosti, je lahko problem nepripravljenost stranke in ključnih uporabnikov, za aktivno sodelovanje pri definiranju problema

in zahtev. Druga omejitev bi lahko bila časovna omejitev za dosego zastavljenih rokov.

Ko je aktivnost dodana, s pritiskom na gumb **Create activity**, se program vrne na vpogled discipline. Do aktivnosti lahko projektni vodja dostopa preko tabele ali drevesne strukture v glavnem meniju. Podatki, ki jih je vnesel se izpišejo v tekstovni obliki, kot pri projektu.

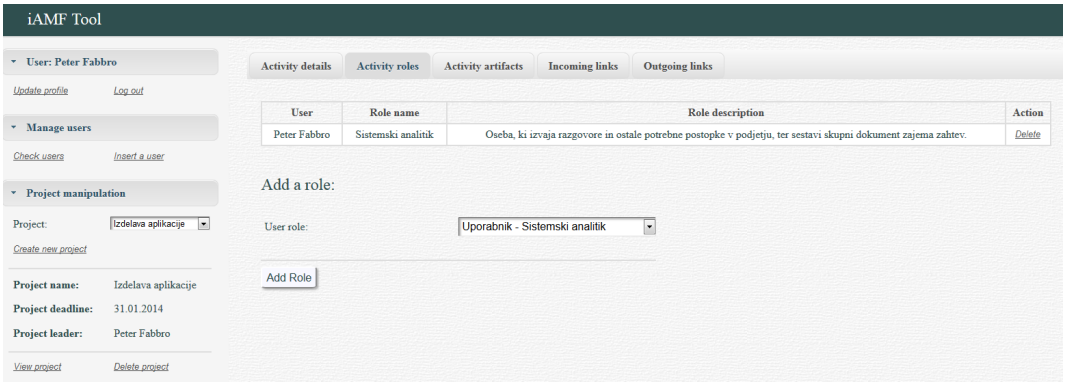


Slika 4.8: Ogled podatkov vnesene aktivnosti.

Uporabnik lahko vnešene podatke še vedno ureja s pritiskom na gumb **Edit activity**.

4.3.4 Dodajanje razvijalskih vlog aktivnosti

Pri vpogledu aktivnosti, je v drugem zavihku na voljo že omenjeno dodajanje razvijalskih vlog. Razvijalce, ki je projektni vodja določil na projektu, lahko sedaj pripiše posamezni aktivnosti in s tem jim omogoči urejanje podatkov aktivnosti. V primeru naše aktivnosti, ki se izvaja znotraj zajema zahtev, moramo navesti vlogo **Sistemski analitik**, katero smo predhodno dodali uporabniku.



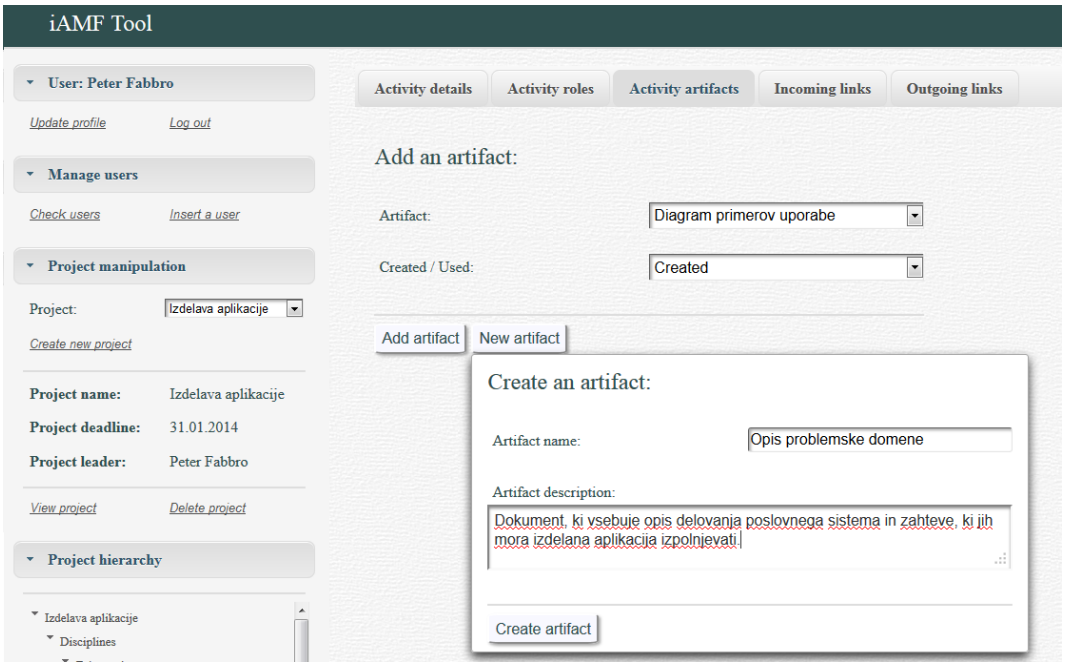
Slika 4.9: Vnos vloge Sistemski analitik aktivnosti.

Ko je vloga uspešno dodana, jo lahko sedaj vidimo v prisotni tabeli pri aktivnosti in drevesni strukturi v meniju. Dodane razvijalske vloge, lahko projektni vodja naknadno tudi odstrani iz aktivnosti in s tem onemogoči nadaljnje urejanje izbranemu razvijalcu (če seveda ta nima še kakšne druge vloge pripisane aktivnosti).

4.3.5 Dodajanje artefaktov

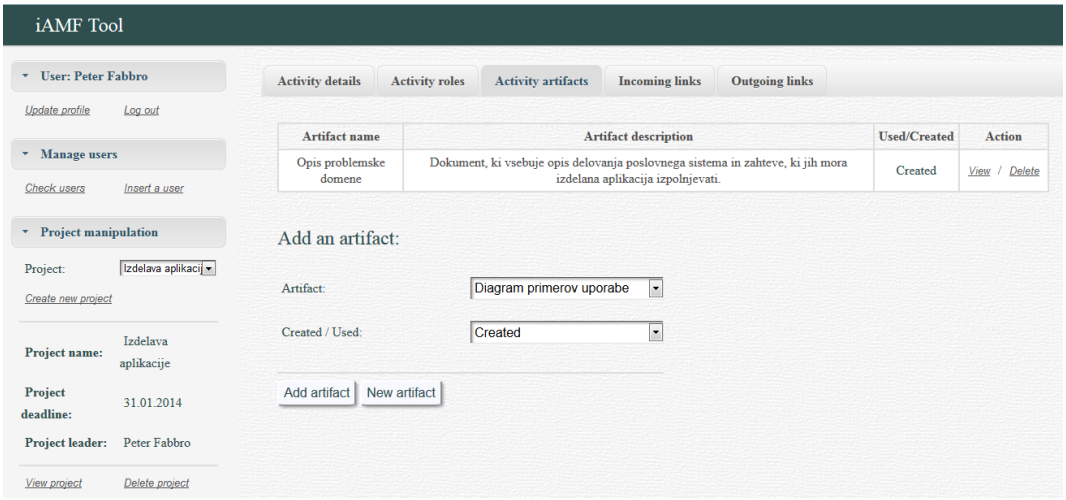
Artefakti predstavljajo izdelek, ki je proizvod izvajanja določene aktivnosti ali temeljni vhod v drugo aktivnost. Tako npr. aktivnost **Izvedba razgovorov** proizvede artefakt **Opis problemske domene**. Vnos artefakta je mogoč v tretjem zavihku vpogleda aktivnosti. Tukaj je prav tako prisotna tabela z dodanimi artefakti. Artefakt je potrebno najprej kreirati. S priti-

skom na **New artifact**, se nam prikaže forma, preko katere vnesemo **ime** in **opis** artefakta.



Slika 4.10: Vnos artefakta.

Ko je artefakt kreiran, vendar ni še povezan z aktivnostjo, ga je mogoče s pomočjo dveh kombiniranih izvlečnih list tudi dodati tako, da izberemo artefakt in določimo ali ga izbrana aktivnost proizvede ali uporablja. S pritiskom na gumb **Add artifact**, izbran artefakt dodamo aktivnosti.



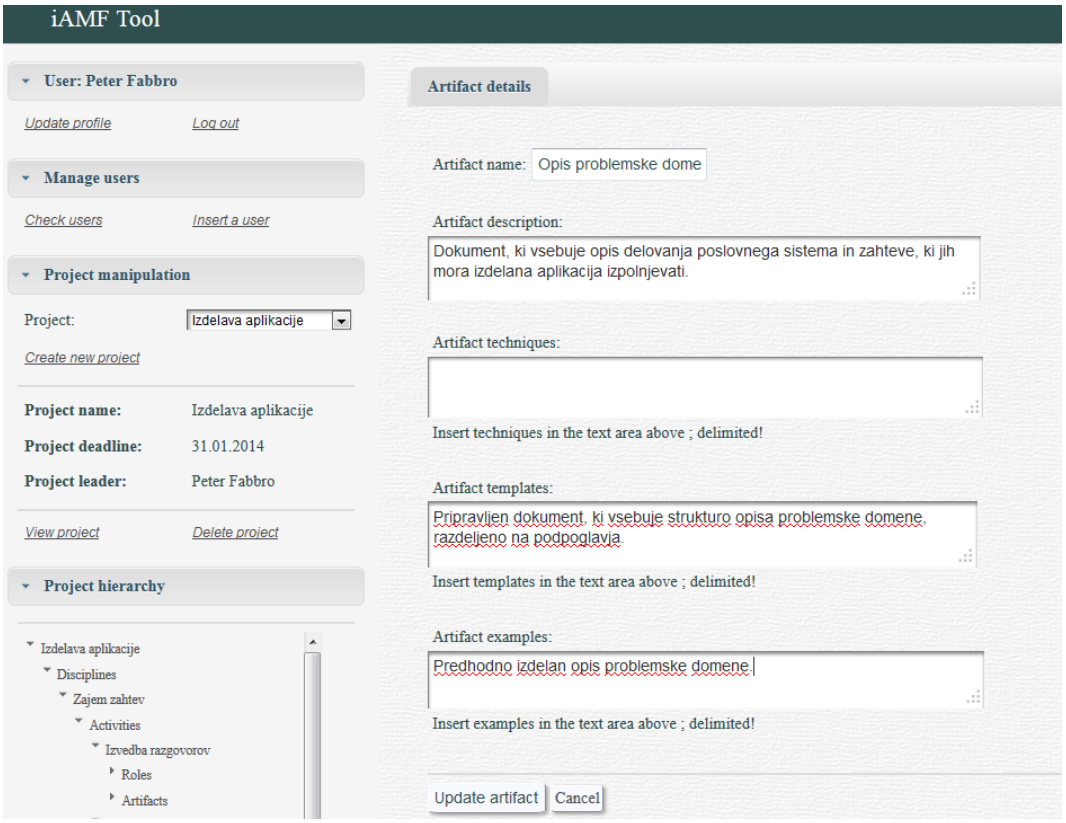
Slika 4.11: Pripis artefakta k aktivnosti, ki ga proizvede.

Ko je artefakt dodan aktivnosti, postane na voljo za urejanje. Do urejanja lahko dostopamo preko tabele ali drevesne strukture menija. Uporabnik lahko sedaj ureja že navedene podatke in doda 4 nove vrste elementov. Vnos **tehnik**, **primerov** in **ogrodij** je mogoč v prvem zavihku, s pritiskom na **Edit artifact**. Način vnosa je enak, kot vnos ciljev, pridobitev in omejitev pri aktivnosti: vnos opisov v tekstovna polja, ločenih s podčrtajem.

V primeru aktivnosti Izvedba razgovorov ne uporabljamo nobene posebne tehnike, je pa ta prisotna pri aktivnosti Izdelava diagrama primerov uporabe, in sicer **Unified moddelling langauage** (UML).

Ogrodje, v primeru naše aktivnosti bi bil pripravljen dokument, ki vsebuje strukturo opisa problemske domene razdeljeno na podpoglavja.

Primeri nam prikazujejo že izdelano verzijo podobnega artefakta, katero lahko prav tako uporabimo, kot pomoč pri izdelavi. Pri naši aktivnosti bi to lahko bil že izdelan opis problemske domene.



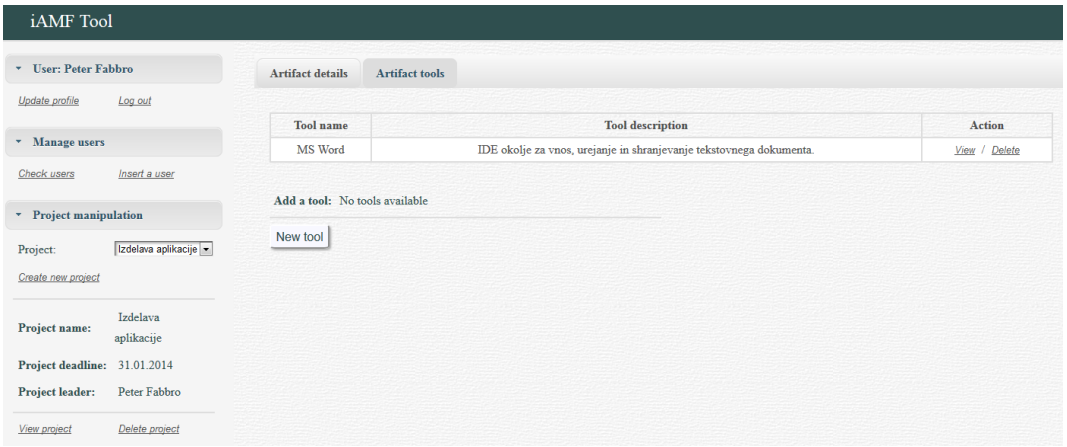
Slika 4.12: Vnos ogrodja in primera pri prvem artefaktu (Izvedba razgovorov).

Po vnosu, so elementi vidni s pomočjo seznamov pri vpogledu artefakta ter v drevesni strukturi menija. Postopek vnosa tehnike pri aktivnosti izdelava diagrama primerov uporabe je enak postopku, ki je prikazan na zgornji sliki. Seveda lahko uporabniki v ta polja vpišejo tekst po želji tako, da bi npr. lahko vnesli imena datotek primerov, ogrodij in tehnik.

Pri artefaktu je mogoče navesti tudi, s pomočjo katerih **orodij** je bil izdelan. Zaradi možnosti ponovne uporabe navedenih orodij pri drugih artefaktih, je vnos le teh nekoliko drugačen od vnosa tehnik, ogrodij in primerov. V primeru aktivnosti **Izvedba razgovorov**, ki proizvede artefakt **Opis problemske domene** orodje, ki ga uporabljamo za zapis, je lahko kar tekstovni

urejevalnik, kot npr. **MS Word**. Dostop do forme za vnos orodja je mogoč preko gumba **New tool** v drugem zavihku vpogleda artefakta.

Ko je orodje uspešno vnešeno, ga lahko uporabnik doda ustreznemu artefaktu na podoben način, kot je aktivnostim dodal vloge: Z izbiro v kombinirani izvlečni listi in pritiskom na gumb **Add tool**. Orodja, ki so bila dodana določenemu artefaktu, se bodo prikazala v ustrezni tabeli orodij.



Slika 4.13: Pripis orodja MS Word artefaktu.

4.3.6 Nadaljnje vnašanje aktivnosti

Poleg prve aktivnosti (Izvedba razgovorov), disciplino **zajem zahtev** sestavljajo še naslednje aktivnosti:

Izdelava diagrama primerov uporabe Proces RUP priporoča izdelavo modela primerov uporabe, s pomočjo UML modelirnega jezika.

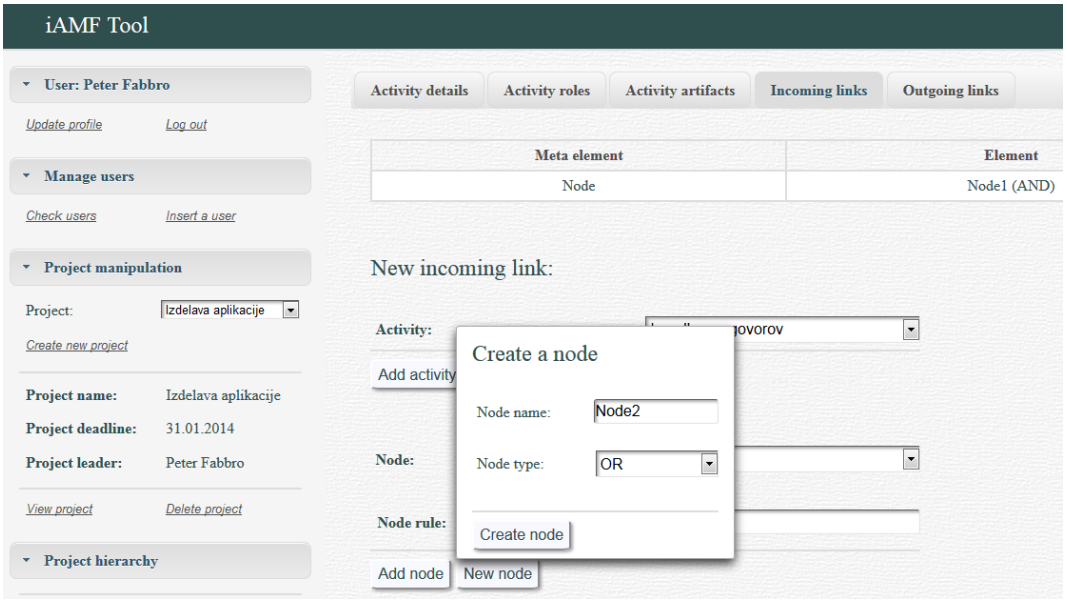
Identifikacija osnovnih in alternativnih tokov Vsak primer uporabe ima svoje toke dogodkov, ki opisujejo dogajanje znotraj posameznega primera uporabe.

Sestava skupnega dokumenta Sestava diagrama primerov uporabe, tokov dogajanja ter izdelava dodatnih specifikacij, opisa problemske domene in slovarja izrazov.

Predstavitev izdelka stranki V tej aktivnosti predstavimo izdelan dokument stranki in skušamo doseči skupno soglasje. To je zelo pomembna aktivnost za nadaljevanje razvoja projekta v pravi smeri.

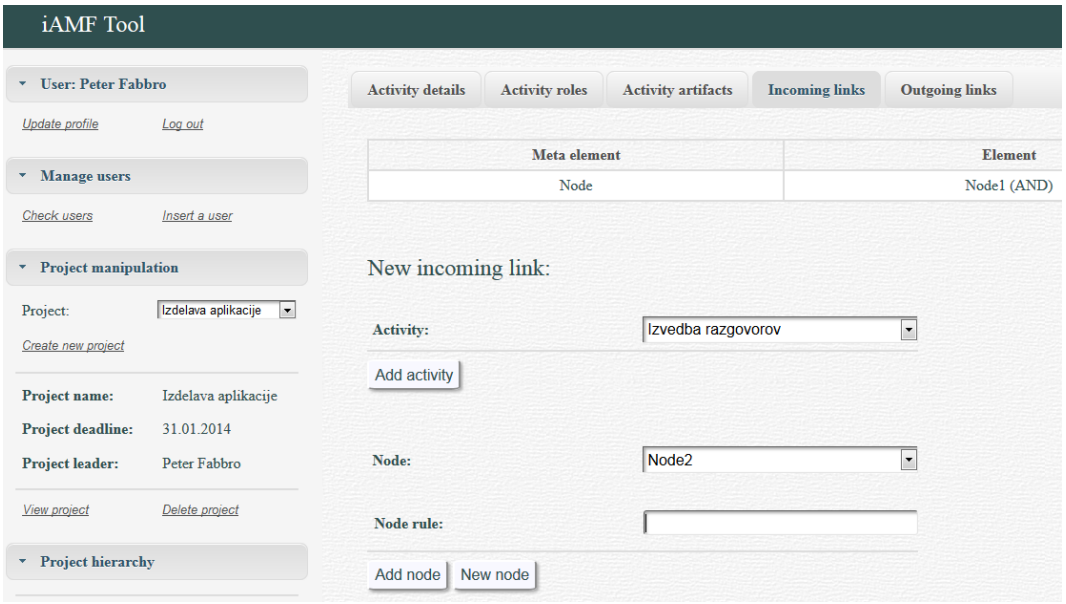
Pri nadaljnjem vnašanju aktivnosti k disciplini, izdelamo tudi sosledje aktivnosti s pomočjo povezav in dveh novih meta elementov: vejitvenih členov in sinhronizacij. To je izvedljivo v četrtem in petem zavihku (**Incoming** in **Outgoing links**) vpogleda aktivnosti, kjer vnašamo vhodne in izhodne povezave. Če se po končanju določene aktivnosti tok loči v dve veji aktivnosti, lahko to realiziramo s pomočjo vejitvenega člena (AND/OR). V našem primeru bi lahko podjetje po izvedbi razgovorov pričelo hkrati z **izdelavo diagrama primerov uporabe** in **identifikacijo osnovnih in alternativnih tokov** (uporabimo torej AND vejitveni člen). Izhodna povezava iz prve aktivnosti bi tako peljala v člen, ki ga najprej kreiramo s pritiskom na **New node**, nato pa dodamo z **Add node**. Člen se prikaže v ustrezni tabeli izhodnih povezav, kjer lahko uporabnik člen tudi odstrani. Pri drugi in tretji aktivnosti pa ta člen navedemo pri vhodnih povezavah s pomočjo kombinirane izvlečne liste. Postopek je enak tudi pri navajanju aktivnosti in sinhronizacijskih elementov.

Kljub temu, da pri naši disciplini ne uporabljamo OR vejitvenega člena, bomo prikazali vnos le tega, saj se z njim lahko prikaže celoten način delovanja vnosa sosledja aktivnosti. Vejitveni člen je potrebno najprej kreirati s pritiskom na **New node** ter vnos imena in izbiro tipa:



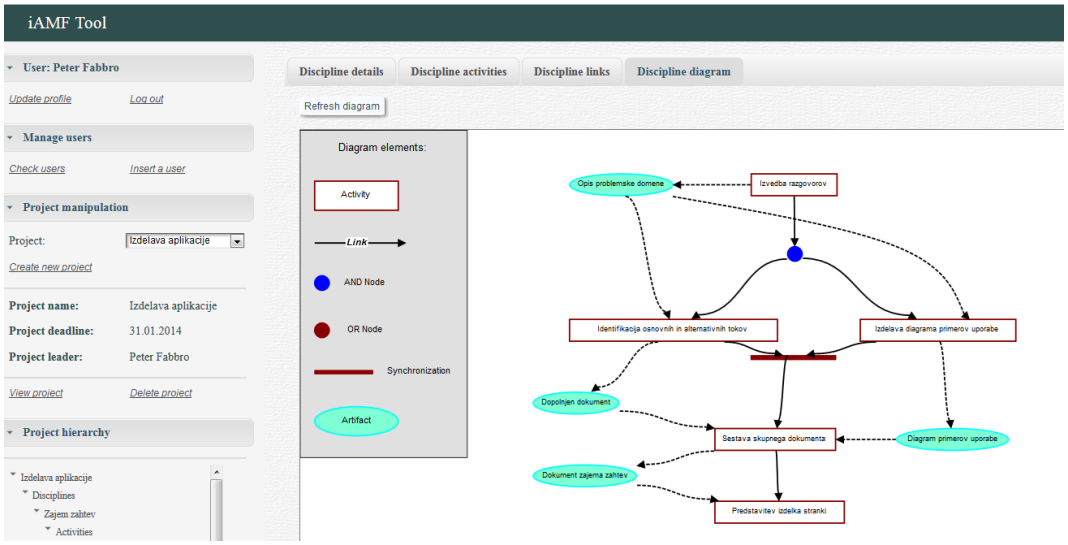
Slika 4.14: Vnos OR vejitvenega člena.

Ko je člen vnešen, ga sedaj navedemo kot vhodni element (v zavihku Incoming links) v izbrano aktivnost, z izbiro v kombinirani izvlečni listi. Kot je razvidno, imamo na voljo tudi vnos pravila, glede na katerega se tok veji v to vejo. S pritiskom na gumb **Add node**, se člen pripiše ter prikaže v diagramu sosledja aktivnosti discipline (element smo tukaj zgolj dodali za prikaz delovanja in ga bomo kasneje odstranili). Urejanje pravila vejitve je sedaj mogoče v zgornji tabeli vhodnih elementov ter v tabeli povezav pri vpogledu discipline. S pritiskom na **Edit rule** imamo na voljo formo, preko katere je pravilo mogoče urediti oz. izbrisati.



Slika 4.15: Pripis člena, kot vhod v izbrano aktivnost.

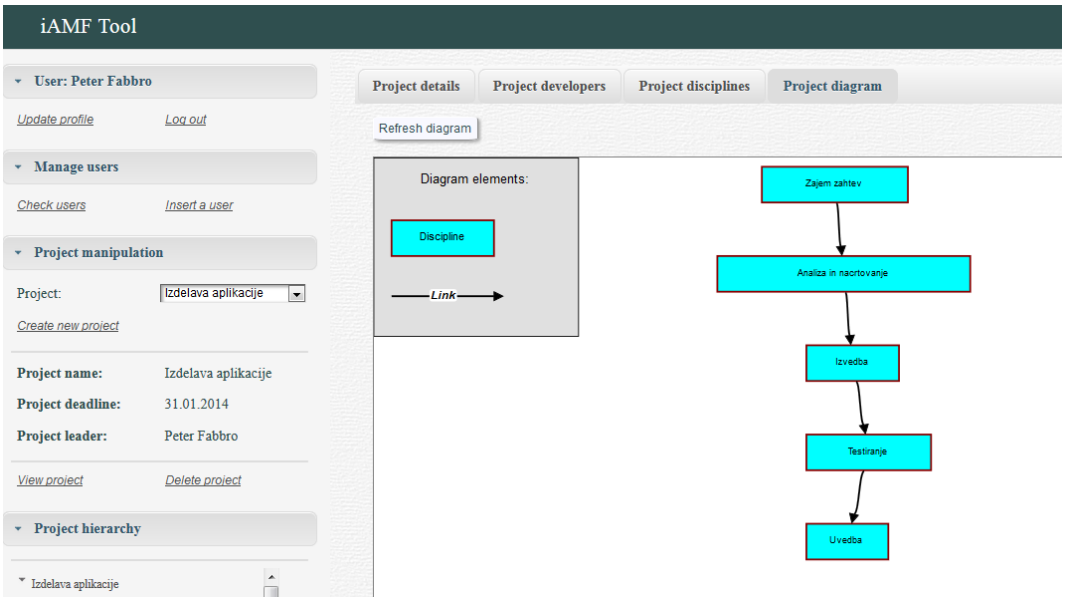
Vnos ostalih povezav poteka na podoben način le, da pri sinhronizaciji nimamo vejitvenega pravila, aktivnosti pa na tem mestu lahko samo pripisujemo, ne moramo pa jih kreirati (na voljo ni gumba **New activity**). Ko smo z izdelavo sosledja zaključili, je pri vpogledu discipline na voljo ogled vseh povezav v tretjem zavihku, v četrtem pa je možen ogled diagrama, ki je generiran na podlagi povezav ob pritisku na gumb Refresh diagram. Izriše se tudi legenda, ki prikazuje pomen posameznih elementov diagrama.



Slika 4.16: Diagram sosledja aktivnosti zajema zahtev.

4.3.7 Nadaljnje vnašanje projekta

Ko je izdelava posamezne discipline zaključena, se lahko prične izdelava naslednje. Kot je bilo že omenjeno, je pri projektnem vpogledu prisotna tabela disciplin projekta. Projektni vodja lahko vrstni red disciplin spreminja, prav tako lahko posamezne discipline izbriše. V naslednjem zavihku pa je na voljo tudi generiran diagram, ki prikazuje zaporedje izvajanja disciplin. Diagram je izdelan po enakem principu kot diagram sosledja aktivnosti.



Slika 4.17: Diagram projektnih disciplin.

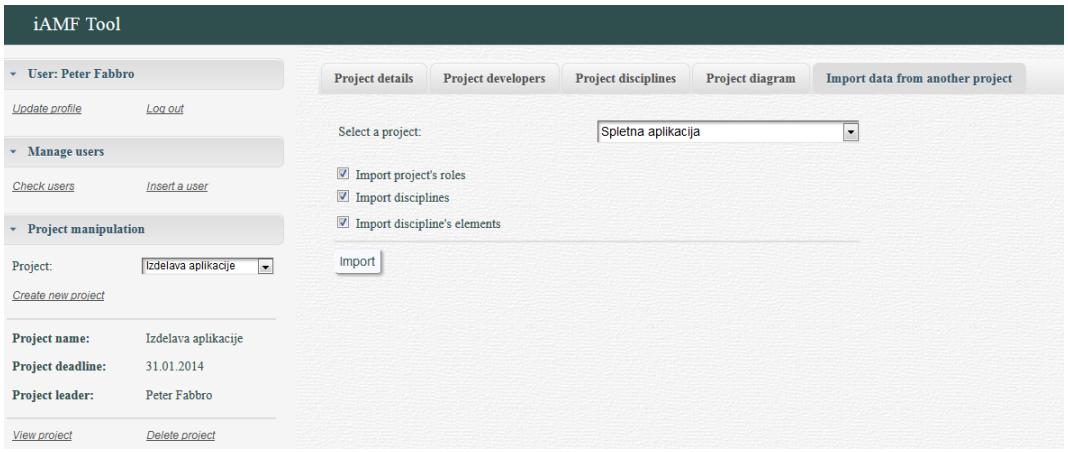
4.3.8 Uvoz podatkov projekta

Aplikacija omogoča tudi uvoz podatkov iz starejših projektov. Ta opcija je na voljo vodji projekta, ki lahko v prisotnosti starejših projektov v zadnjem zavihku izbere ime projekta in naslednje elemente:

Uvoz projektnih razvijalcev: z izbiro te opcije, bodo uvoženi vsi razvijalci projekta, kar je tudi pogoj za uvoz razvijalcev aktivnosti pri tretji opciji.

Uvoz disciplin z izbiro te opcije, bodo uvožene vse discipline starega projekta k novemu. To je tudi predpogoj za možnost izbire tretje opcije.

Uvoz vseh pod elementov discipline z izbiro te opcije, bo poleg disciplin, izveden tudi uvoz aktivnosti in vseh njihovih pod elementov.



Slika 4.18: Uvoz podatkov od starega, k novemu projektu.

Ob pritisku na **Import**, se glede na izbrane opcije, elementi starega prekopirajo k novemu projektu. Uporabnik lahko potem te podatke seveda ureja glede na potrebe svojega projekta. Kot je bilo že omenjeno, je razlog za uporabo te funkcionalnosti predvsem uvoz projektno specifične metode starejšega projekta, kar predstavlja tudi sredstvo za ponovno uporabo uspešno uporabljenih postopkov skozi čas.

Poglavje 5

Sklep in nadalnji razvoj

V diplomskem delu je bila izdelana spletna aplikacija, ki omogoča vnos projektno specifičnih metod, ki jih podjetje uporablja pri razvoju programske opreme. Ker podjetja, ki se ukvarjajo s razvojem programske opreme pogosto nimajo točno definirane metode, aplikacija prinaša možnost beleženja in identifikacije podobnosti uporabljene metode na projektu s podobnimi svetovno uveljavljenimi. Podjetju bi tako bilo omogočeno odkrivanje boljših in kvalitetnejših rešitev, s katerimi bi lahko dosegli boljše vodenje in razvoj programske opreme.

Delo je bilo usmerjeno predvsem na vnos in pregled metod, seveda pa obstajajo tudi možnosti razširitve, ki bi omogočale večjo intuitivnost in dodatne funkcionalnosti. Aplikacija ponuja pogled na projektne metode in sosledje aktivnosti le teh, morebitna razširitev pa bi bili interaktivni diagrami, preko katerih bi lahko razvijalci direktno vnašali elemente projekta in povezave med njimi. Knjižnjica JointJS namreč omogoča izdelavo interaktivnih diagramov. Druga možnost razširitve bi bila omogočeno večje sodelovanje razvijalcev kar preko spletne aplikacije. Izdelava foruma, bi razvijalcem omogočila komunikacijo in iskanje skupnih rešitev med razvojem, z integracijo wiki orodja pa bi bilo omogočeno beleženje definicij in napotkov pri nadaljnjem razvoju projekta.

Literatura

[1] (2014) Ajax. Dostopno na: http://www.w3schools.com/ajax/.

[2] (2014) An introduction to agile software development. Dostopno na: http://www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf.

[3] (2014) Cascading Style Sheets (CSS). Dostopno na: http://en.wikipedia.org/wiki/Cascading_Style_Sheets.

[4] (2014) HTML - XHTML. Dostopno na: http://www.w3schools.com/html/html_xhtml.asp.

[5] (2014) Java EE at a Glance. Dostopno na: http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[6] (2014) JointJS, Javascript diagramming library. Dostopno na: http://jointjs.com/.

[7] (2014) Metodologija informacijskega inženiringa. Dostopno na: http://colos1.fri.uni-lj.si/ERI/RACUNALNISTVO/INFORMATIKA/informacijski_ineniring.html.

[8] (2014) Rational Unified Process, Best Practices for Software Development Teams. Dostopno na: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251bestpractices_TP026B.pdf.

[9] (2014) Software development life-cycle (SDLC) Dostopno na: http://en.wikipedia.org/wiki/Software_development_process.

[10] (2014) Unified Modeling Language (UML). Dostopno na: http://www.uml.org/.

[11] D. Avison G. Fitzgerald. *Information system development methodologies, techniques and tools*. McGraw-Hill, 2006.

[12] M. Jankovic. Semi-automatic improvement of software development methods: Doctoral consortium paper. In *2013 IEEE Seventh International Conference on Research Challenges in Information Science (RCIS)*, pages 1–6, 2013.