UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Anže Časar

PROGRAMSKO ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Viljan Mahnič

Ljubljana, december 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 01770/2011
Datum: 01.09.2011

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

Kandidat: **ANŽE ČASAR**

Naslov: **PROGRAMSKO ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM**

A SOFTWARE TOOL FOR MANAGING SCRUM PROJECTS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite značilnosti agilnih metod za razvoj programske opreme s posebnim poudarkom na metodi Scrum. Analizirajte obstoječa orodja za vodenje projektov po tej metodi in definirajte specifikacije za lastno orodje, ki bo vključevalo najzanimivejše funkcionalnosti, ugotovljene med analizo. Orodje realizirajte, opišite njegovo realizacijo in prikažite njegovo funkcionalnost.

Mentor:

prof. dr. Viljan Mahnič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Anže Časar**,

z vpisno številko **63060048**,

sem avtor diplomskega dela z naslovom:

Programsko orodje za vodenje projektov po metodi Scrum

A Software Tool for Managing Scrum Projects

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Viljana Mahniča

- 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 29.11.2011 Podpis avtorja:

ZAHVALA

Za podporo, usmerjanje in številne dobre nasvete se zahvaljujem svojemu mentorju, prof. dr. Viljanu Mahniču. Prav tako se zahvaljujem tudi svoji družini in prijateljem za podporo, ne samo ob izdelavi diplomske naloge, pač pa tekom celotnega študija.

KAZALO VSEBINE

POVZETEK **1**

ABSTRACT **2**

1 UVOD **3**

2 AGILNE METODOLOGIJE **4**

2.1 POJAV AGILNIH METODOLOGIJ 4

2.2 PRIMERJAVA S KLASIČNIMI PRISTOPI 4

2.3 AGILNE METODOLOGIJE IN NJIHOVE ZNAČILNOSTI 6

2.4 METODA SCRUM 8

2.4.1 Značilnosti metode Scrum *8*

2.4.2 Uporabniške zgodbe *9*

2.4.3 Vloge *10*

2.4.4 Izdelki procesa *12*

2.4.5 Potek procesa Scrum in sestanki *14*

3 ORODJA ZA PODPORO METODI SCRUM **17**

3.1 VLOGA ORODIJ ZA PODPORO RAZVOJU 17

3.2 PREGLED OBSTOJEČIH ORODIJ 17

3.2.1 ScrumWorks *18*

3.2.2 VersionOne *18*

3.2.3 Planbox Agile Tool *19*

3.2.4 Vision Project *19*

3.2.5 TinyPM *20*

3.2.6 Rally Enterprise Edition *20*

3.2.7 Agilo For Scrum 2 ..... *21*

3.2.8 Scrum Desk *21*

3.3 ZANIMIVI KONCEPTI 22

3.3.1 Podpora sprotnemu vodenju napredka na projektu *22*

3.3.2 Podpora za digatalni »Planning Poker« *23*

3.3.3 Podpora oddaljeni komunikaciji med člani *23*

3.3.4 Podpora sprotnemu ustvarjanju dokumentacije *23*

3.3.5 Odziven in preprost uporabniški vmesnik *24*

3.3.6 Primerjalna tabela podprtih konceptov *24*

4 TEHNIČNA ZASNOVA PREDLAGANEGA ORODJA 26

4.1 SPECIFIKACIJA ZAHTEV 26

4.2 PREDSTAVITEV TEHNIČNE ZASNOVE 30

4.2.1 Uporabljene tehnologije 30

4.2.2 Zasnova podatkovne baze 31

4.2.3 Zasnova programske logike 35

5 PREDSTAVITEV UPORABE ORODJA 41

5.1 PREDSTAVITEV VODENJA PROJEKTA 41

5.1.1 Definiranje novega projekta in njegovo časovno umeščanje 41

5.1.2 Urejanje seznama zahtev (Product Backlog) 42

5.1.3 Urejanje seznama nalog (Sprint Backlog) 43

5.1.4 Delo na nalogah 45

5.1.5 Statistični pregled dela na projektu 46

5.2 PREDSTAVITEV OSTALIH FUNKCIONALNOSTI PROGRAMA 47

5.2.1 Podpora komunikaciji med člani 47

5.2.2 Planning Poker 48

5.2.3 Ustvarjanje uporabniške dokumentacije 49

5.2.4 Podpora sprotnemu vodenju dela na projektu 50

6 SKLEP 52

7 VIRI 53

7.1 LITERATURA 53

7.2 INTERNETNI VIRI 54

SEZNAM UPORABLJENIH KRATIC

EMRIS	*Enotna Metodologija Razvoja Informacijskih Sistemov*
HTML	*HyperText Markup Language (Jezik za označevanje nadbesedila)*
CSS	*Cascading Style Sheet (Prekrivni slogi)*
AJAX	*Asinhronous JavaScript And XML (Asinhroni JavaScript in XML)*
PHP	*PHP Hypertext Preprocessor (PHP predprocesor nadbesedila)*
SQL	*Structured Query Language (Strukturirani povpraševalni jezik)*
ROI	*Return Of Investement (donosnost naložbe)*

POVZETEK

Diplomska naloga opisuje razvoj programskega orodja za vodenje projektov po metodi Scrum. V uvodnem delu se osredotoča na relevantne teoretične osnove, ki zajemajo predvsem predstavitev agilnih principov razvoja programske opreme in predstavitev metode Scrum kot najpopularnejše agilne metode. Sledi aktualni pregled trga programskih orodij za podporo vodenju procesov z metodo Scrum, na podlagi katerega so izluščeni nekateri zanimivi koncepti uporabe. Ti koncepti predstavljajo osnovo za izgradnjo nove programske rešitve v okviru te diplomske naloge in med drugim zajemajo: sprotno vodenje napredka na projektu, oddaljeno komunikacijo med člani razvojne skupine, sprotno vodenje uporabniške dokumentacije na spletu in računalniško podporo za ocenjevanje časovne zahtevnosti uporabniških zgodb z metodo »Planning Poker«.

Drugi del diplomskega dela zajema predstavitev realiziranega programskega orodja. Obsega podrobno specifikacijo zahtev v obliki uporabniških zgodb, pregled tehnične zasnove programske rešitve (struktura podatkovne baze in programske logike) in predstavitev ključnih primerov uporabe. Razvito orodje je realizirano v obliki spletne aplikacije z uporabo tehnologije PHP za programsko logiko in tehnologije MySQL za podatkovno bazo. Posebna pozornost je namenjena oblikovanju odzivnega uporabniškega vmesnika, ki temelji na tehnologiji AJAX. Končna rešitev v celoti podpira vodenje procesov po metodi Scrum, poleg tega pa so podprti tudi vsi zgoraj omenjeni koncepti uporabe, kar razviti rešitvi daje neko unikatno vrednost v primerjavi z drugimi tovrstnimi orodji na trgu.

Ključne besede: agilne metode, metoda Scrum, razvojna orodja, vodenje projektov, razvojna skupina

ABSTRACT

The thesis describes the development of a software tool for managing Scrum projects. The first part of the thesis focuses on the relevant theoretical concepts, mainly on the presentation of agile principles in software development and on the Scrum method as the most popular agile method. It is followed by an overview of the currently available software tools for managing Scrum projects, from which a few interesting concepts have been derived. These concepts form the basis for the software solution presented in this thesis and include the support for: an automatic recording of the developement progress, an online documentation management, a remote communication between Scrum Team Members and a computer supported user stories evaluation using the »Planning Poker« method.

The second part of the thesis focuses on the presentation of the realized software tool. It encompasses a detailed specification of requirements in the form of user stories, an overview of the technical structure of the software solution (data base structure and program logic) and the presentation of key use cases. The developed tool is a web based application using PHP technology for the program logic and MySQL technology for the data base. Special emphasis was also given to design a responsive user interface based on AJAX technology. In addition of the finished application fully supporting the management of Scrum projects it also supports all of the above mentioned concepts thus ensuring a positive distinction compared to other similar applications.

Keywords: agile methods, Scrum, software developement tools, project management, developement team

1 UVOD

Današnji razvoj programske opreme postaja vedno bolj dinamičen. Še posebej ga je spodbudil pojav novih naprav, kot so tablični računalniki in pametni mobilni telefoni, ki so sprožili plaz t.i. »mini aplikacij« in iger, z razvojem katerih se ukvarjajo številna mlada podjetja. Bistvene prvine razvoja programske opreme se tako premikajo od rigoroznih postopkov k metodologijam, ki omogočajo glede na vložena sredstva kar najhitrejše rezultate, prav tako pa se tudi hitro odzivajo na spremembe v zahtevah med samim razvojnim procesom. Takšnim metodologijam pravimo agilne metodologije in najpopularnejša med njimi je metoda Scrum [1], ki jo po nekaterih raziskavah uporablja več kot 60% razvijalcev, ki se ukvarjajo z agilnim razvojem [11].

Vodenje projektov z metodo Scrum pa zahteva od razvijalcev veliko mero samodiscipline, tako pri vodenju procesa samega, kot tudi pri vodenju nekaterih izdelkov (artifacts), ki jih metoda zahteva. Pri tem so jim v pomoč številna namenska programska orodja, ki ne nudijo samo podpore procesu, pač pa ponujajo tudi številne dodatne funkcionalnosti, ki razvijalcem po eni strani olajšujejo razvoj, po drugi pa jih pravočasno opozarjajo na morebitne zastoje oziroma težave v procesu ter tako zagotavljajo njihovo zgodnje odkritje in odpravo. S tem pa so nepogrešljiv člen pri vodenju projektov z metodo Scrum.

Tema te diplomske naloge je tako izdelava takšnega orodja, pri čemer postopek izdelave zajema vse od pregleda teoretičnih osnov metode Scrum, pregleda trenutnega stanja na trgu tovrstnih orodij in na podlagi teh ugotovitev, razvoj novega orodja. Pri tem je narejen poudarek na nekaterih zanimivih novih konceptih uporabe tovrstnih orodij, kar nam po eni strani zagotavlja neko edinstvenost razvite rešitve, po drugi strani pa še dodano poenostavlja delo razvojnim skupinam, ki bi orodje uporabljale.

2 AGILNE METODOLOGIJE

2.1 POJAV AGILNIH METODOLOGIJ

Razvoj lahkih (lightweight) metodologij se je začel sredi devetdesetih let prejšnjega stoletja, kot odziv na vedno večje kritike težkih (heavyweight) metodologij, ki so zaradi svoje visoke formalnosti in rigoroznosti pri razvoju programske opreme predstavljale vedno več odvečnega dela, ki bi se mu bilo moč izogniti. Sledil je pojav številnih lahkih metodologij, ki jih danes poznamo pod pojmom **agilne metodologije**. Pojem je prvič uporabila skupina programskih inženirjev februarja 2001, ko so na sestanku v mestu Snowbird (Utah, ZDA) objavili *Manifest agilnega razvoja programske opreme* [2], kasneje pa so tudi oblikovali zvezo imenovano *Agile Alliance* [12]. V manifestu so zapisali štiri načela agilnega razvoja.

1. **Posamezniki in komunikacija** so pomembnejši od samega procesa in orodij
2. **Delujoča programska oprema** je pomembnejša od popolne dokumentacije
3. **Sodelovanje z naročnikom** je pomembnejše od pogajanj na osnovi pogodb
4. **Upoštevanje sprememb** je pomembnejše od sledenja planu.

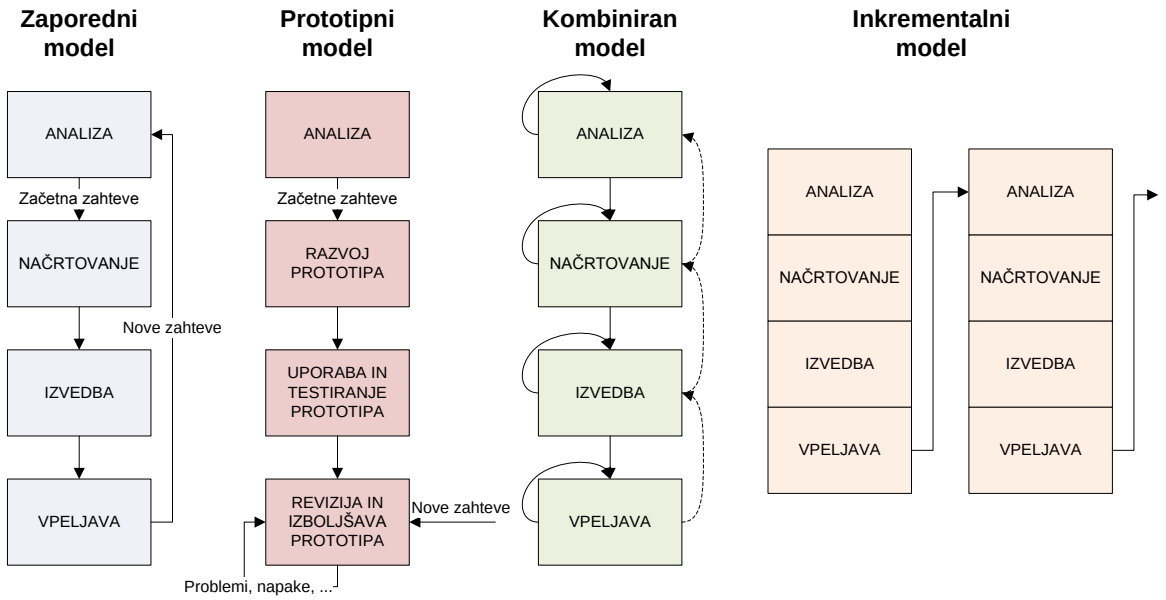
Iz zgornjih načel so izpeljali tudi 12 priporočil, katerim naj bi sledile vse agilne metodologije. Priporočila obsegajo dobre prakse za organizacijo in delovanje razvojnih skupin navznoter, prav tako pa tudi za njeno sodelovanje z naročnikom in uporabniki. Dotikajo pa se tudi samega procesa, še posebej njegove odprtosti za spremembe.

2.2 PRIMERJAVA S KLASIČNIMI PRISTOPI

Ogrodje klasičnih in agilnih metodologij je definirano v dokumentu EMRIS (Enotna Metodologija Razvoja Informacijskih Sistemov) [3].

Klasične metodologije razvoja programske opreme običajno narekujejo razvoj v zaporednih fazah, pri čemer je vsaka faza zaključena celota. Tipično so to: **faza analize**, **faza načrtovanja**, **faza izvedbe** in **faza vpeljave**. Zaporedje faz je nespremenljivo in določeno v naprej, zato vračanje na prejšnje faze praviloma ni mogoče.

Obstaja več variacij klasičnega modela razvoja programske opreme, ki so predstavljene na Sliki 1. Najosnovnejši je zaporedni model (tudi slapovni model), poznamo pa še prototipni model, inkrementalni model in kombinirani model. Razlike med njimi gre iskati predvsem v implementaciji zaporedja posameznih faz in prehodov med njimi.



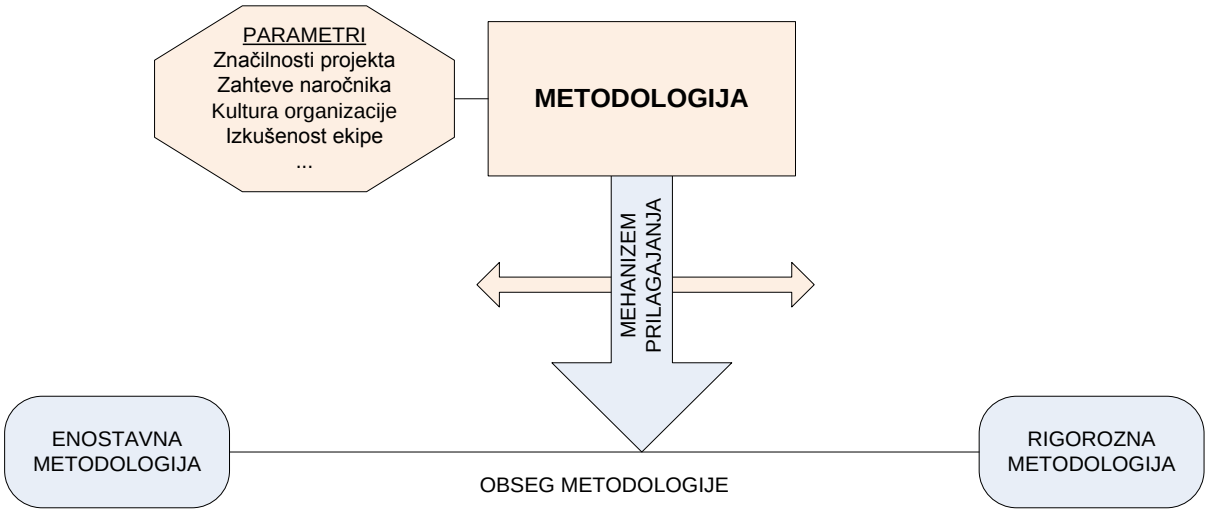
Slika 1: Primerjava klasičnih modelov razvoja programske opreme

Prav zaradi fazne zasnove lahko med največje slabosti klasičnih pristopov štejemo njihovo okornost pri prilagajanju novim zahtevam naročnika, še posebej če te pridejo v kasnejših fazah razvoja. Poleg tega naročnik (oziroma uporabnik) v sam razvojni proces izdelave programske opreme ni dovolj vključen, saj sodeluje le v fazi analize, njegova vloga pa je omejena na podajanje zahtev. Ima tudi zelo omejen vpogled v vmesne izdelke procesa, rešitev pa lahko začne uporabljati šele, ko je ta povsem končana.

Po drugi strani bi med prednosti klasičnih modelov lahko šteli predvsem njihovo primernost za večje projekte s stabilnimi in dobro definiranimi zahtevami in za projekte z višjo stopnjo tveganja. To je posledica formalne in rigorozne zasnove tovrstnih metodologij, kjer je delo povsem jasno definirano in razčlenjeno, dokumentacija pa popolna in podrobna.

Agilne metodologije tako poskušajo po eni strani premostiti slabosti klasičnih metodologij, še posebej pomanjkanje komunikacije z naročnikom in okornost pri sprejemanju novih zahtev, po drugi strani pa obdržati ravnotežje med formalnim in neformalnim delom metodologije na točki, ki je najprimernejša za vsak posamezni projekt. Iskanju tega ravnotežja pravimo

prilagajanje metodologije (method tailoring), pri čemer gre v bistvu za vpeljavo agilnih konceptov v razvojni proces v želeni meri. Postopek je prikazan na Sliki 2.



Slika 2: Prilagajanje metodologije

V primerjavi s klasičnimi metodologijami, so agilni pristopi še najbolj podobni inkrementalnemu modelu razvoja z zelo kratkimi inkrementi (običajno nekaj tednov), pri čemer podrobno planiranje zamenjuje komunikacija med razvijalci in naročnikom. Naročnik ima med procesom razvoja reden vpogled v vmesne izdelke, zato lahko razvijalcem sproti podaja povratne informacije, programerji pa glede na odziv prilagajajo nadaljnji razvojni proces.

2.3 AGILNE METODOLOGIJE IN NJIHOVE ZNAČILNOSTI

Laurie Williams je v svoji študiji [4] opisala 33 praks, značilnih za vodenje procesov z agilnimi metodologijami. Najpomembnejše med njimi so povzete v nadaljevanju poglavja.

Glavna časovna enota agilnih metodologij je ena **iteracija**, ki običajno traja od dveh do štirih tednov. Vsaka iteracija vključuje vse razvojne faze klasičnega pristopa, vključno s predstavitvijo opravljenega dela naročniku. Rezultat iteracije je delujoč izdelek, četudi je za razvoj nekaterih funkcionalnosti potrebnih več iteracij.

Osnovna celica agilnih metodologij je **skupina razvijalcev**, kjer ima vsak član določeno svojo vlogo, v skupini pa je prisotna velika mera samoorganizacije. Ekipa je običajno manjša (5-9 članov), med njimi pa poteka dnevna komunikacija. Na večjih projektih običajno dela več manjših ekip, saj je skupinska dinamika tovrstnih ekip na višjem nivoju.

Vsaka skupina običajno vsebuje tudi **predstavnika naročnika**, ki deluje v interesu končnih uporabnikov in je ves čas na voljo razvijalcem za sprotno razjasnjevanje problemov povezanih z domeno projekta. Prav tako pa ob koncu vsake iteracije opravi tudi sprejemne teste in pomaga pri določanju prioritet nadaljnjega razvoja.

Za zagotovitev dnevne komunikacije agilni koncepti predlagajo **formalne dnevne sestanke**, kjer se člani ekipe seznanijo z opravljenim delom prejšnjega dne in določijo delo za tekoči dan. Predvsem pa se pogovorijo o tekočih problemih in poskušajo najti rešitve, ter jih tako sproti odpraviti.

Opisani proces torej daje za svoj rezultat **delujočo programsko kodo**, ki je tudi glavna mera za njegov uspeh. Posledično ob tem nastane precej manj pisne dokumentacije, kar je značilno za t.i. »value-driven approach«. Pri tem pristopu naročnik na račun formalnosti postopkov in popolnosti dokumentacije dobi v najkrajšem času kar največjo poslovno vrednost.

Razvitih je bilo že več metodologij, ki upoštevajo opisana načela in priporočila agilnosti. Najbolj poznane med njimi so: Scrum, Crystal, Ekstremno programiranje (Extreme Programming), Prilagodljiv razvoj programske opreme (Adaptive Software Developement), Razvoj usmerjen v funkcionalnost (Feature Driven Developement) in Metoda razvoja dinamičnih sistemov (Dynamic Systems Developement Method)

Od naštetih metodologij je daleč najpogosteje uporabljena metoda **Scrum**. Njene korenine segajo v leto 1995 in je tako ena prvih metod, ki so nastale z valom pojava lahkih metodologij. Še posebej v zadnjih letih je veliko pridobila na popularnosti, predvsem zaradi svoje enostavne zasnove, dokazane produktivnosti in velike prilagodljivosti. V nadaljevanju diplomske naloge se bom zato posvetil zgolj metodi Scrum.

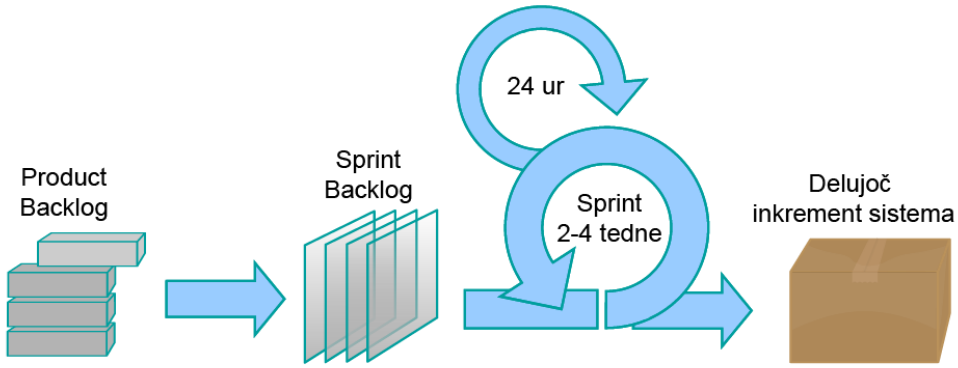
2.4 METODA SCRUM

2.4.1 Značilnosti metode Scrum

Da bi spoznali, kako poteka vodenje procesov z metodo Scrum, moramo najprej na grobo umestiti nekaj osnovne terminologije in principov. Uradna slovenska terminologija za Scrum še ni razvita, zato sem se pri prevodih v določeni meri opiral na obstoječe gradivo [5], nekateri prevodi pa so povsem novi.

Kot že rečeno, poteka razvojni proces večine agilnih metodologij v več manjših iteracijah, ki jih pri metodi Scrum imenujemo **Sprinti**. Tekom vsakega Sprinta projektna skupina razvije naslednji inkrement sistema, z nekaj dodanimi funkcionalnostmi glede na prejšnjo izdajo. Programerji, ki razvijajo sistem, so združeni v eno ali več razvojnih skupin, zato jim pravimo tudi člani razvojne skupine (**Team Members)**. Razvojna skupina seveda potrebuje vodstvo in nadzor, ki ga opravlja oseba z nazivom skrbnik metodologije (**Scrum Master**). Pomembno vlogo v celotni sestavi pa ima še predstavnik naročnika, imenovan produktni vodja (**Product Owner)**, ki med drugim skrbi tudi za podajanje zahtev projekta. Zahteve so pri metodi Scrum podane v obliki kratkih zgodb, ki jih imenujemo uporabniške zgodbe (**User Stories**) [6]. Le te so vedno zbrane v dokumentu imenovanem seznam zahtev (**Product Backlog)**, za vzdrževanje katerega skrbi produktni vodja. Pred začetkom vsakega Sprinta člani projektne skupine izberejo podmnožico zgodb, ki jih bodo realizirali tekom Sprinta, in jih podrobneje definirajo v dokumentu, imenovanem seznam nalog (**Sprint Backlog)**. Ta predstavlja osnovo za delo skupine v tekočem Sprintu, ki običajno traja od 2 do 4 tedne. Naročnik dobi z zaključkom vsakega Sprinta delujoč izdelek, ki vsebuje glede na prejšnjega nekaj dodatnih funkcionalnosti in ga lahko že takoj prične uporabljati. Metoda Scrum je zato izjemno učinkovita in naročniku hitro in redno prinaša dodano vrednost.

Opisani proces, ki je grafično predstavljen na Sliki 3, se ponavlja ves čas trajanja projekta, vse dokler je naročnik še zainteresiran za njegov razvoj.



Slika 3: Prikaz poteka procesa Scrum

2.4.2 Uporabniške zgodbe

Vsak projekt se začne z oblikovanjem zahtev in tudi projekti, vodeni z metodo Scrum, niso izjema. Običajno se zajem zahtev začne z vizijo naročnika, ki je lahko na začetku precej ohlapna in jo je potrebno konkretizirati. Večina agilnih metodologij, tudi Scrum, predlaga, da se izvede zapis vizije v obliki uporabniških zgodb, s čimer se ustvari začetni seznam zahtev. To delo naj bi v idealnem primeru opravil produktni vodja sam, v praksi pa se pogosto oblikuje celotna ekipa uporabnikov oziroma naročnikov, ki nalogo opravijo v sodelovanju.

Uporabniške zgodbe morajo biti **jedrnate** in opisovati zgolj bistvo zahtevane funkcionalnosti končnega produkta. Oblika, v kateri jih zapišemo, je lahko povsem prosta ali pa bolj formalna. Slednja se drži vzorca **»Kot <vloga> si želim <cilj/želja>, da bi <korist>«,** pri čemer navedba koristi ni obvezna, je pa dobrodošla, da razvijalec ve, zakaj je zgodba za uporabnika pomembna. Primer takšnega načina podajanja zgodb je npr.: »Kot administrator si želim vpogleda v sistemski dnevnik, da bi lahko spremljal uporabo sistema«. Če bi enako zgodbo zapisali v neformalni obliki, bi se lahko glasila: »Administrator lahko spremlja uporabo sistema z vpogledom v sistemski dnevnik«. Iz primera vidimo, da mora vsebina zgodbe podajati tako zahtevano **funkcionalnost** (vpogled v sistemski dnevnik) kot tudi **vlogo uporabnika**, ki jo lahko izvede (administrator). V ta namen je priporočljivo, da naročnik identificira vse možne vloge v sistemu že v naprej.

Zgodbe morajo biti med seboj **neodvisne**, kar pomeni, da lahko sorodne ali preveč podrobne zgodbe združujemo med seboj. Po drugi strani pa ena sama zgodba ne sme zajemati prevelikega števila funkcionalnosti. Takšne zgodbe je potrebno razbiti na manjše. K vsaki zgodbi se običajno pripiše še t.i. **sprejemne teste**. Gre za kratek opis omejitev in primerov uporabe, katerih izvedbo mora omogočati pravilno realizirana zgodba. Zgodbe morajo biti tudi **ocenljive**, tako z vidika časovne zahtevnosti, kar opravi razvojna skupina, kot tudi z vidika prioritete izdelave, ki jo oceni produktni vodja. Med samim procesom lahko člani razvojne skupine k zgodbam ob dogovoru s produktnim vodjo dodajajo še svoje **komentarje**, z namenom še večje konkretizacije ali pa boljšega razumevanja besedila zgodbe.

Ocenjevanju časovne zahtevnosti zgodb gre nameniti nekaj več besed. Običajno se kot enoto za ocenjevanje uporabi točke (story points), ki naj bi predstavljale idealne delovne dni, za kar običajno vzamemo 6 delovnih ur. Metod, s katerimi lahko izvedemo ocenjevanje je več, najpogosteje pa se uporablja metoda »**Planning Poker**« [7], imenovana tudi Scrum Poker. Pri tej metodi se med vse člane razvojne skupine razdeli kartice s časovnimi ocenami, ki imajo običajno vrednosti Fibonaccijevega ali podobnega zaporedja, kjer vrednosti naraščajo čedalje hitreje, s čimer se kompenzira večjo negotovost pri višjih ocenah. Člani razvojne skupine nato hkrati ocenjujejo vsako izmed zgodb z dvigom ustrezne kartice, pri čemer postopek ponavljajo vse dotlej, dokler si pri oceni niso enotni. Ob vsakem neenotnem glasovanju poskušajo zbližati stališča na način, da člana, ki sta dala najvišjo in najnižjo oceno, le to utemeljita.

2.4.3 Vloge

Scrum predvideva tri vloge sodelujočih: produktni vodja (Product Owner), skrbnik metodologije (Scrum Master) in član razvojne skupine (Team Member). Vse odgovornosti v projektu so porazdeljene med te tri vloge, njihovi nosilci pa so tudi edini, ki se projektu lahko posvetijo in so zanj odgovorni. Za projekt se lahko zanimajo tudi drugi zainteresirani, vendar se v proces ne morejo vmešavati, niti niso zanj odgovorni. Zato metodologija Scrum jasno ločuje med tema dvema skupinama ljudi, ki jih šaljivo lahko poimenujemo tudi prašiči (sodelujoči na projektu) in piščanci (ostali zainteresirani). Poimenovanje izvira iz zgodbe o

piščancu in prašiču, ki sta odprla restavracijo »Šunka in jajca« in ponazarja njuni vlogi v restavraciji - prašič mora dati šunko, torej je polno udeležen, piščanec pa le jajca.

2.4.3.1 Skrbnik metodologije (Scrum Master)

Skrbnik metodologije zapolni vlogo, katere nosilec je običajno vodja projekta, vendar se od njega bistveno razlikuje. Medtem ko je klasični vodja projekta zadolžen predvsem za definiranje in delegiranje dela, skrbnik metodologije skrbi prvenstveno le za pravilen potek procesa Scrum. Njegova naloga je implementirati določila metodologije Scrum v prakso. Prav zato mora biti njen dober poznavalec, imeti pa mora tudi izkušnje pri njenem vodenju. Skrbeti mora za nemoteno napredovanje procesa in ščititi ekipo pred motečimi dejavniki, da se le ta lahko popolnoma posveti delu na projektu.

2.4.3.2 Produktni vodja (Product Owner)

Produktni vodja predstavlja vez med razvojno skupino in naročnikom oziroma končnimi uporabniki rešitve. Njegova glavna naloga je določitev potrebnih funkcionalnosti projekta v obliki uporabniških zgodb. Imeti mora jasno predstavo, kaj bo rezultat projekta in kako bo izdelana programska oprema koristila njenim uporabnikom. Na podlagi tega lahko določi donosnost naložbe (ROI - Return Of Investment), torej kakšno dodano vrednost prinaša projekt glede na vložena sredstva. Cilj produktnega vodje je za vsak Sprint maksimizirati ROI s pomočjo prioritiziranja uporabniških zgodb projekta, tako da najvišjo prioriteto podeli zgodbam, ki bodo naročniku prinesle največjo dodano vrednost, manj pomembnim zgodbam pa podeli nižjo prioriteto. Njegova naloga je tudi opravljanje sprejemnih testov za že opravljeno delo, s čimer zagotovi njegovo ustreznost. Poleg tega je razvojni skupini vedno na voljo za dodatna pojasnila o podrobnostih v zvezi z realizacijo posameznih zgodb. Seveda pa niti uporabniške zgodbe niti njihove prioritete niso stalne in se tekom procesa neprestano spreminjajo kot odziv na dinamične pogoje poslovnega okolja naročnika, zato je produktni vodja v procesu neprestano udeležen.

2.4.3.3 Član razvojne skupine (Team member)

V nasprotju z običajno prakso vodenja projektov, kjer člani razvojne skupine zgolj izpolnjujejo navodila nadrejenih, metoda Scrum nalaga članom več odgovornosti. Skupina sodeluje že pri določanju dela za posamezen Sprint, znotraj Sprinta pa so popolnoma odvisni od svojih samoorganizacijskih sposobnosti. V svojih rokah nosijo odgovornost tako za delitev dela med posamezne člane, kot tudi za pretvorbo zapisanih zahtev v uporabno funkcionalnost končnega izdelka. Zato je izvedba vsakega posameznega Sprinta popolnoma v rokah članov razvojne skupine. Praktične izkušnje so celo pokazale, da poskusi upravljanja razvojne skupine od zunaj, procesu kvečjemu škodujejo [1].

2.4.4 Izdelki procesa

Med potekom procesa Scrum mora projektna skupina vzdrževati več seznamov, dokumentov in izdelkov, ki jih skupaj imenujemo izdelki procesa Scrum (Scrum Artifacts). Ti izdelki nastajajo in se uporabljajo tekom celotnega procesa in so opisani v nadaljevanju.

2.4.4.1 Seznam zahtev (Product Backlog)

Eden temeljnih projektnih dokumentov je seznam zahtev, saj so v njem v obliki uporabniških zgodb zbrane vse funkcionalnosti, ki jih mora razvita rešitev podpirati. Posodabljanje, prioritiziranje in zagotavljanje dostopnosti seznama zahtev je v domeni produktnega vodje. Seznam zahtev je dinamičen dokument, kar pomeni, da se tekom procesa neprestano dopolnjuje in spreminja ter tako vedno odraža aktualno sliko tega, kakšen naj bi končni izdelek bil. Za ocenjevanje časovne zahtevnosti zgodb v seznamu zahtev so zadolženi člani razvojne skupine, ta ocena pa se uporablja tako pri grobem načrtovanju plana izdaj, kot tudi pri izbiri podmnožice zgodb za tekoči Sprint.

2.4.4.2 Seznam nalog (Sprint Backlog)

In to nas pripelje do naslednjega dokumenta, imenovanega seznam nalog. Sestavljajo ga naloge, ki izhajajo iz izbrane podmnožice uporabniških zgodb za tekoči Sprint in skupaj

predstavljajo naslednji inkrement funkcionalnosti sistema v razvoju. Pri oblikovanju tega projektnega dokumenta mora razvojna skupina izbrane zgodbe razbiti na naloge, pri čemer naj posamezna naloga enemu programerju ne bi vzela več kot 4 do 16 ur dela. Razvojna skupina je zadolžena tudi za vzdrževanje dodatnih atributov za vsako posamezno nalogo. Med drugim so to podatki o članu, ki je zadolžen za njeno izvedbo, predvidenem času za dokončanje naloge in po želji tudi o času, ki je že bil vložen v delo na nalogi. S tem seznam nalog vedno predstavlja ažurni posnetek dela na projektu, prav tako pa tudi ponuja vpogled v delo, ki ga je skupina že opravila in ga še namerava opraviti v tekočem Sprintu.

2.4.4.3 Delujoča programska koda

Za naročnika bistven izdelek vsake iteracije procesa je seveda delujoča programska koda. Ta predstavlja naslednji inkrement funkcionalnosti sistema v razvoju. Delujočo kodo sestavljajo vse uspešno realizirane uporabniške zgodbe preteklih iteracij. Te zgodbe običajno označimo kot zaključene (done), kar pomeni, da so uspešno prestale vse sprejemne teste in je to potrdil tudi produktni vodja. Tu so določila metode Scrum zelo striktna, saj so iz naslednjega inkrementa sistema izločene tudi vse zgodbe, ki so realizirane zgolj delno – koncept zaključene zgodbe namreč brezpogojno zahteva njeno pravilno delovanje v celoti, brez znanih pomanjkljivosti.

2.4.4.4 Ostali izdelki

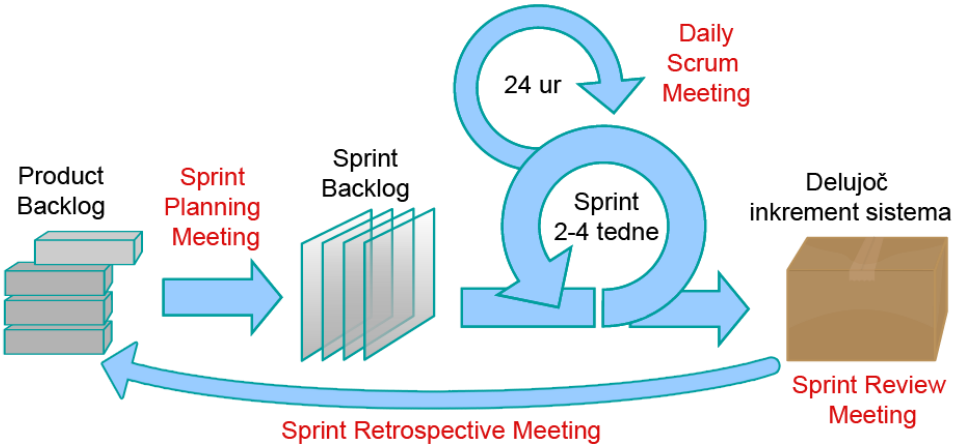
Med procesom lahko nastajajo še nekateri drugi izdelki. Pogosto razvojna skupina vzporedno z delom na projektu piše tudi **uporabniško dokumentacijo**, kjer je uporaba rešitve razložena s stališča uporabnika. Ta dokument je pomemben, saj končnim uporabnikom precej poenostavi postopek uvajanja v nov sistem in pomaga pri razreševanju njihovih vprašanj brez posredovanja razvojne skupine. S tem pa predstavlja enega redkih primerov dejanske dokumentacije, ki se mu niti agilne metodologije ne morejo ogniti.

Običajno se tekom razvoja pojavi tudi potreba po dokumentu, ki bi nudil hiter vpogled v stanje na projektu. Eden takih je grafikon o preostalem delu, imenovan tudi »**Burn Down Chart**«. V osnovi gre za standardni graf, kjer na abscisno os nanašamo čas trajanja projekta, na ordinatno os pa preostalo količino dela (v dnevih, urah ali točkah). Graf nam poda

informacijo o preostalem delu za vsak trenutek trajanja projekta, prav tako pa je z njega moč razbrati nekatere trende. Običajno se graf vodi na dveh nivojih – na nivoju Sprinta, kjer opazujemo preostalo delo na trenutnem Sprintu (Sprint Burn Down) in na nivoju celotnega projekta (Release Burn Down), ki ga med drugim uporabljamo za analizo hitrosti dela in za oceno predvidenega zaključka projekta.

2.4.5 Potek procesa Scrum in sestanki

Metodologija Scrum sledi smernicam agilnih principov tudi pri obliki redne komunikacije med člani skupine. V ta namen je zahtevano periodično vodenje sestankov celotne ekipe, kjer se njeni člani pogovorijo o preteklem in prihodnjem delu na projektu. Sestankov je več vrst in nastopajo ob točno določenih trenutkih v procesu, kot je razvidno na Sliki 4.



Slika 4: Umestitev sestankov članov skupine v potek procesa Scrum

2.4.5.1 Sestanek za načrtovanje Sprinta (Sprint Planning Meeting)

Ta sestanek se zgodi pred začetkom vsakega Sprinta in običajno traja en delovni dan, oziroma 8 ur. Razdeljen pa je na dva dela.

V prvih štirih urah produktni vodja predstavi preostalim članom razvojne skupine seznam zahtev, pri čemer se osredotoči predvsem na uporabniške zgodbe z višjo prioriteto, ekipa pa mu zastavlja vprašanja, dokler jim namen in vsebina zgodb ni popolnoma jasna. Zgodbe nato časovno ocenijo in še pred koncem prvega dela sestanka dosežejo dogovor, katere bodo

vključili v naslednji Sprint. Pri tem morajo upoštevati tako prioritete posameznih zgodb, kot tudi časovno omejitev, ki je običajno podana v številu točk, ki jih lahko programerska skupina realizira v obdobju enega Sprinta. Ta količina se imenuje tudi hitrost Sprinta (**Sprint velocity**).

V drugem delu sestanka sledi planiranje Sprinta. Ekipa vsako izbrano zgodbo razbije na posamezne naloge, ki ob uspešni realizaciji skupaj tvorijo celotno zahtevano funkcionalnost, lahko pa si tudi že v grobem razdelijo delo. Proces oblikovanja nalog in delitve dela sicer poteka tekom celotnega Sprinta.

2.4.5.2 Vsakodnevni pregledni sestanek (Daily Scrum Meeting)

Vsakodnevni pregledni sestanek (**Daily Scrum Meeting**) se zgodi vsak delovni dan pred začetkom dela, ko se člani razvojne skupine zberejo za kratek 15 minutni razgovor. Pri tem mora vsak član skupine odgovoriti na tri vprašanja:

1. Kaj je naredil na projektu od prejšnjega sestanka
2. Kaj namerava narediti na projektu do naslednjega sestanka
3. Katere težave so se mu pojavile pri delu

Namen tega sestanka je sinhronizirati delo vseh članov skupine in se dogovoriti za morebitne dodatne sestanke, če so ti potrebni za nemoteno nadaljevanje procesa. Omogoča tudi sprotno identifikacijo in reševanje problemov.

2.4.5.3 Sestanek za recenzijo Sprinta (Sprint Review Meeting)

Ob koncu vsakega Sprinta se vsi člani projektne skupine zopet zberejo na štiriurnem neformalnem sestanku, ki je namenjen predstavitvi rezultatov preteklega Sprinta. V tem času ekipa predstavi produktnemu vodji (in morebitnim ostalim zainteresiranim) rezultat njihovega dela v zadnjem Sprintu, pri čemer se osredotočijo predvsem na sprejemne teste. Produktni vodja ob tem kritično oceni, če je realizacija uporabniških zgodb ustrezna in jo potrdi ali zavrne. Eden od namenov tega sestanka je tudi pripeljati skupaj vse zainteresirane za projekt, da se lahko seznanijo z opravljenim delom in se okvirno dogovorijo, kaj naj bi projektna skupina naredila v prihodnje.

2.4.5.4 Sestanek za revizijo Sprinta (Sprint Retrospective Meeting)

Še pred začetkom novega Sprinta skliče skrbnik metodologije člane projektne skupine na še en sestanek, ki naj bi trajal približno tri ure. Skupina na njem revidira zadnji Sprint, pri čemer se osredotoči predvsem na sam proces, njegov potek in uporabljene prakse. Pri tem skušajo ugotoviti, kaj je bilo v procesu dobrega in kaj bi bilo še mogoče izboljšati v prihodnosti in s tem optimizirati celotni proces. Čeprav se ta sestanek ne osredotoča neposredno na projekt v razvoju, je kljub temu zelo pomemben, saj zagotavlja neprestano izpopolnjevanje uporabljenih praks in povečuje produktivnost skupine.

3 ORODJA ZA PODPORO METODI SCRUM

3.1 VLOGA ORODIJ ZA PODPORO RAZVOJU

Zasnova metode Scrum zahteva od njenih uporabnikov vzdrževanje nekaterih dokumentov, kot so seznam zahtev in seznam nalog, prav tako pa tudi evidenco napredovanja projekta. Najpreprostejše projekte je mogoče voditi že brez računalniške podpore. To storimo tako, da uporabniške zgodbe in naloge napišemo na posebne kartice (Story Cards) [8], z razvrščanjem katerih lahko vodimo zahtevane sezname. Na kartice lahko poleg zgodb beležimo tudi njihove prioritete in časovne ocene. Četudi s tem že popolnoma zadostimo zahtevam metode Scrum, je pogosto učinkoviteje uporabiti eno od cele vrste namenskih programskih orodij, ki vodenje procesa poenostavijo, poleg tega pa je z njimi moč voditi tudi zahtevnejše projekte. Prav tako pa ponujajo tudi kar nekaj naprednih funkcionalnosti.

Glavna prednost namenskih programov je predvsem zmožnost, da lahko sodelujoči kadarkoli vpogledajo v trenutno dogajanje na projektu in lahko posledično pravočasno ukrepajo ob morebitnih zastojih ali prevelikih odstopanjih od plana. Večinoma nam je ponujena tudi statistična obdelava podatkov dela na projektu, kjer nam programska orodja med drugim nudijo izračun povprečne hitrosti dela, hitrosti posameznih Sprintov, primerjavo med planiranim in dejanskim delom, izris številnih grafikonov in poročil, ter mnoge druge statistične izračune.

3.2 PREGLED OBSTOJEČIH ORODIJ

Tovrstnih orodij je na trgu že mnogo [9], od katerih jih je precej na voljo tudi v domeni prostega programja (freeware). V nadaljevanju bom nekoliko podrobneje predstavil osem izmed njih. Za začetek bom predstavil orodji, ki sta na tem področju med najbolj znanimi, saj imata od vseh namenskih orodij največjo bazo uporabnikov - to sta ScrumWorks in VersionOne. Za pregled sem nato izbral še šest orodij, ki so na neodvisnem portalu UserStories.com [13] najbolje ocenjena s strani uporabnikov. Vsa orodja sem preizkusil tudi v praksi. Namen tega pregleda ni kritično ocenjevanje posameznih programov, še posebej ne

zato, ker so razvoj na tem področju hiter in že v nekaj mesecih tovrstni opis ne bi bil več aktualen. Zato sem želel le predstaviti nekatere izbrane funkcionalnosti, ki jih ta orodja ponujajo v trenutku pisanja te diplomske naloge in tako narediti pregled za uporabnike zanimivih konceptov uporabe. Le ti bodo služili predvsem kot osnova za izgradnjo nove rešitve, ki je tudi osrednja tema te diplomske naloge.

3.2.1 ScrumWorks

Orodje ScrumWorks ima bazo uporabnikov, ki šteje kar 65.000 projektnih skupin [14]. Njegove zmogljivosti lahko kategoriziramo v šest področij: upravljanje s programi (program = skupina sorodnih projektov), upravljanje s projekti, upravljanje z iteracijami oz. Sprinti, izdelava poročil z analizo, zagotavljanje dostopa do podatkov in upravljanje z uporabniki. Program je realiziran kot samostojna aplikacija, ponuja pa tudi številne spletne vmesnike za dostop do sistema. Veliko je možnosti za organizacijo zgodb v seznamu zahtev; po prioriteti jih lahko razvrščamo s tehniko »povleci in spusti«, na enak način jih dodeljujemo tudi Sprintom. Poleg tega lahko sorodne zgodbe združujemo v t.i. teme. Program nam ponuja še možnost samodejnega izračuna ROI za posamezno zgodbo glede na vnesene poslovne vrednosti. Znotraj Sprinta lahko člani skupine spreminjajo status posameznih nalog z njihovim razvrščanjem v štiri sezname (še nezačete, ovirane, v delu in dokončane zgodbe). Prav tako lahko naloge tudi filtriramo po različnih kriterijih, s čimer je zagotovljena velika preglednost seznama nalog.

3.2.2 VersionOne

Orodje VersionOne je med uporabniki, ki jih je več kot 30.000 [15], prav tako zelo priljubljeno. Je zelo sistematično in ureja delo na projektu v časovnico petih stopenj: planiranje izdelka, planiranje izdaj, planiranje Sprintov, sledenje Sprintom in revizija procesa. Prav tako pa orodje nudi tudi odlično podporo statistični analizi, izdelavi poročil, testiranju izdelka, komunikaciji med člani skupine in integraciji z odprtokodnimi razvojnimi orodji. Kot posebnost programa velja omeniti predvsem številne možnosti pri časovnem planiranju, kjer že vnaprej razdelimo uporabniške zgodbe med izdaje, nato pa v okviru posamezne izdaje med

Sprinte. Pri časovnem umeščanju zgodb nam je na voljo mnogo orodij tako za analizo hitrosti (glede na preteklo delo), kot tudi za »kaj če« analizo. Program dodatno omogoča še izdelavo številnih naprednih poročil, ki so na voljo za različne namene (sledenje poteku procesa, sledenje spremembam), za različne dele procesa (celoten projekt, posamezna izdaja, posamezen Sprint, …) in za različna časovna obdobja.

3.2.3 Planbox Agile Tool

Orodje Planbox Agile Tool [16] nastaja pod okriljem podjetja Planbox, odlikuje pa ga predvsem preprosta zasnova in hiter uporabniški vmesnik, ki je neverjetno preprost. Glavni del programa sestavljajo zahteve projekta (oz. iniciative, kot projekt poimenujejo razvijalci orodja), zbrane v štirih zavihkih: zgodbe v delu (current), zgodbe naslednje iteracije (next), nekategorizirane zahteve (backlog) in zgodbe preteklih iteracij (last). Planiranje dela je preprosto – skupina iz seznama nekategoriziranih zahtev izbere zgodbe, ki bodo sestavljale naslednjo izdajo produkta in jih razporedi med iteracije. Med samim delom orodje ponuja številne možnosti za pregledovanje vloženega in preostalega časa na projektu. Le tega lahko opazujemo po posameznih nalogah, po projektnih skupinah in tudi po posameznih članih. Med delom program omogoča uporabnikom tudi merjenje časa (štopanje) dela na posamezni nalogi in posledično bolj zanesljive informacije o vloženem delu.

3.2.4 Vision Project

Podjetje Visionera AB je razvilo sistem Vision Project [17]. Orodje je namenjeno delu z raznovrstnimi agilnimi metodologijami in je zato zasnovano precej robustno. Program sestavljajo številni moduli, med drugim tudi sistem za upravljanje z dokumenti, modul za upravljanje z zahtevki, sistem za urejanje baze znanja, forum za komunikacijo med člani skupine in celo sistem za podporo uporabnikom. Slednji med drugim podpira vodenje zahtevkov (tickets), ki so jih posredovali uporabniki preko E-pošte, telefonskih klicev ali pa preko vgrajenega uporabniškega spletnega portala. Funkcije dela na posameznem projektu program sicer razvršča med šest zavihkov: projektna stran (project page), izdaje (issues), Sprinti (Sprints), forumi (forums), dokumenti (documents) in nastavitve (settings). Večino

dela poteka prek zavihka »izdaje«, kjer so zbrane tekoče naloge projekta. Orodje vsebuje tudi dober sistem za upravljanje z razpoložljivostjo človeških virov, še posebej pa se izkaže pri časovnem planiranju izdaj in iteracij (zavihek »Sprints«).

3.2.5 TinyPM

Program TinyPM [18] družbe Agilers je preprosto orodje namenjeno prvenstveno vodenju projektov po metodi Scrum. Sestavlja ga osem razdelkov: upravljalna plošča (Dashboard), peskovnik (Sandbox), seznam zgodb (Backlog), seznam nalog (Taskboard), časovni plan (Timesheet), sprejemno testiranje (Acceptance), modul »Wiki« in zgodovina procesa (History). Orodje odlikuje preprosto planiranje iteracij, kjer zgodbe razvrščamo med Sprinte s tehniko »povleci in spusti«, ki je prisotna tudi drugje po programu. Zanimiva funkcionalnost je t.i. peskovnik, katerega namen je ponuditi alternativo kredi in tabli pri urejanju različnih idej o projektu in predstavlja dobro dopolnilo rednemu sestankovanju. Prav tako je sestavni del orodja tudi modul »Wiki«, kjer lahko v stilu znane spletne enciklopedije urejamo dokumentacijo projekta. Sistem ponuja tudi odličen pregled zgodovine razvoja, kjer so na pregleden način zbrane vse aktivnosti povezane s projektom.

3.2.6 Rally Enterprise Edition

Orodje Rally Enterprise edition [19] je namenjeno predvsem informacijski povezavi med različnimi interesnimi skupinami v podjetju. To je tudi njegova največja prednost in razširja možnost uporabe orodja izven ozkega kroga razvijalcev. Program tako ločuje med razvijalci, vodstvenim kadrom, preizkuševalci in ostalimi deležniki (stakeholders) v projektu. Vsak član ima tako na voljo le zanj relevantne informacije o poteku projektov, na katere lahko vpliva v okviru svoje vloge. Orodje vsebuje pet glavnih razdelkov: 1.) domača stran uporabnika, kjer je vmesnik popolnoma prilagodljiv posamezniku in mu nudi le zanj relevantne informacije, 2.) razdelek za načrtovanje, namenjen oblikovanju zahtev projekta, 3.) stran za vodenje projekta, kjer je moč pregledovati in vnašati podatke o trenutnem poteku projekta, 4.) razdelek za kontrolo kvalitete, s katerim upravljajo predvsem nosilci testiranja in še 5.) razdelek za izpis poročil. Vsak razdelek vsebuje še številne podrazdelke, kjer je moč

nastavljati tudi najmanjše podrobnosti v zvezi s procesom, zahtevami ali udeleženci, s čimer orodje po eni strani gotovo ustreza tudi najzahtevnejšim uporabnikom, po drugi strani pa pridobiva na kompleksnosti in zahtevnosti rabe.

3.2.7 Agilo For Scrum 2

Program Agilo [20], ki nastaja pod okriljem Agile42 GmbH, je ravno v času pisanja te naloge prešel v različico 2, ki ima s prejšnjo le malo skupnega. Novo orodje povsem sledi zahtevam metode Scrum, kar se odraža tako v uporabljeni terminologiji, kot v samem poteku dela s programom. Vodenje projektov je razdeljeno v preprosto zaporedje faz, ki si po vrsti sledijo: urejanje seznama zgodb, planiranje 1 (uvrščanje zgodb med Sprinte), planiranje 2 (razbijanje zgodb na naloge), delo na Sprintu, pregled opravljenega dela in opažanja. Delo na Sprintu je izjemno preprosto. Člani razvojne skupine s tehniko »povleci in spusti« razvrščajo naloge v tri sezname: planirane naloge, naloge v teku in zaključene naloge, pri tem pa k vsaki nalogi dopisujejo, koliko časa so vanjo vložili. V fazi pregleda opravljenega dela produktni vodja označi uspešno opravljene zgodbe kot končane (done) ali pa jih zavrne (reject) in s tem vrne v seznam zahtev. Zanimiv je tudi pogled »opažanja« (observations), kjer lahko člani na zanimiv način pišejo svoje ideje in opažanja na projektu, jih kategorizirajo z ustrezno ikono smeška in s tem obvestijo druge člane skupine o svojih razmišljanjih, kar lahko služi tudi kot način oddaljene komunikacije med njimi.

3.2.8 Scrum Desk

Program Scrum Desk [21] je eden redkih, ki ne teče v spletnem brskalniku niti ne nudi spletnih vmesnikov za dostop, zato je tudi njegov izgled in občutek ob rabi precej drugačen. Na vrhu okna je uporabnikom na voljo časovni trak, ki nudi hiter pregled časovnih mejnikov projekta, večino okna pa zaseda virtualna tabla, ki predstavlja zanimiv koncept uporabniškega vmesnika in je prisotna povsod po programu. Funkcionalnosti orodja so razdeljene med več razdelkov: projekt, seznam zahtev, planiranje, Sprint, retrospektiva in poročila. Orodje ima še poseben poudarek na časovnem planiranju, kjer je vgrajena celo podpora za digitalni »Planning Poker«, kjer uporabniki glasujejo vsak na svojem računalniku s klikom na kartico z

ustrezno oceno. Tudi zgodbe in naloge imajo v programu obliko kartic, ki jih v seznamu zahtev in nalog glede na želene kriterije razvrščamo na virtualno tablo. Na vsako kartico lahko pripisujemo številne podrobnosti, med drugim tudi oceno časovne zahtevnosti in količino vloženega dela. Na voljo nam je tudi virtualna tabla »retrospektiva«, kamor lahko uporabniki pripenjajo svoje ideje in jih tako posredujejo tudi drugim.

3.3 ZANIMIVI KONCEPTI

Iz zgoraj opisanih rešitev lahko izpostavimo več zanimivih konceptov, ki jih ti sistemi implementirajo. Nekatera opisana orodja so narejena precej robustno in ponujajo ogromno količino funkcionalnosti. Program, ki je tema te diplomske naloge, pa je omejen s časom odmerjenim za njeno izdelavo, zato sem se osredotočil predvsem na tiste koncepte, ki s stališča razvijalca k vodenju projektov z metodo Scrum prispevajo kar največ, hkrati pa niso preobsežni za realizacijo. V nadaljevanju naštevam tiste, ki sem jih v implementaciji lastnega orodja tudi uporabil.

3.3.1 Podpora sprotnemu vodenju napredka na projektu

Eden takih konceptov je podpora sprotnemu vodenju dela na projektu. Večina programov namreč zahteva od članov skupine ročno vnašanje delovnih ur v program, kar lahko povzroči nerealnost podatkov, saj člani skupine ure pogosto vnašajo naknadno in zato samo približno. Posledično pa je težko oceniti dejansko hitrost Sprintov in natančnost članov pri časovnem ocenjevanju nalog in zgodb, s tem pa je oteženo planiranje naslednjih Sprintov. Zato je smiselno v rešitev vključiti sistem, ki bo člane skupine spodbujal k sprotnemu vodenju dela na projektu na način, ki zanje ne bo moteč in ne bo predstavljal veliko odvečnega dela. Eden takšnih načinov je, da uporabnik sproti označuje, na katerih nalogah trenutno dela, program pa samodejno beleži vloženi čas. Seveda pa možnost ročnega vnosa delovnih ur zaradi tega ne sme biti izključena.

3.3.2 Podpora za digatalni »Planning Poker«

Ocenjevanje časovne zahtevnosti uporabniških zgodb z metodo »Planning Poker« je proces, ki ga projektna skupina običajno opravlja pred vsakim Sprintom, tradicionalno kar z uporabo kartonastih kartic, rezultate glasovanj pa beležijo bodisi na papir bodisi jih neposredno vnašajo v program. Računalniška podpora procesu glasovanja bi morala biti izvedena celovito, ob predpostavki, da vsak član skupine lahko glasuje na svojem računalniku. To je pomembno, saj posamezen član ne sme videti ocen drugih dokler ni oddal svoje. Ocena uspešnega glasovanja pa se mora seveda avtomatsko zabeležiti kot končna ocena zgodbe.

3.3.3 Podpora oddaljeni komunikaciji med člani

Mnogo projektov razvijajo skupine, katerih člani poleg rednih sestankov nimajo vedno možnosti osebne komunikacije ali pa je ta nepraktična. Zato je dobro, da v program vključimo tudi nek sistem za podporo oddaljeni komunikaciji med člani skupine. Ta sistem ni namenjen zamenjavi rednih sestankov, pač pa jim je v dopolnilo. Nekatera orodja v ta namen implementirajo kar celoten spletni forum, spet druga pa podpirajo zgolj izmenjavo preprostih sporočil. Najprimernejši pa bi bil sistem, prilagojen metodologiji Scrum, kjer bi si člani skupine med rednimi sestanki in tudi med delom lahko zabeležili sprejete sklepe ali nove ideje, kasneje pa bi jih lahko še dodatno komentirali s pripombami in idejami, ki so se pojavile med delovnim procesom.

3.3.4 Podpora sprotnemu ustvarjanju dokumentacije

Dokumentacija je zelo pomemben del končnega produkta, saj uporabnikom razloži, kako z novim sistemom pravilno rokovati. Zato je smotrno v sistem vključiti tudi podporo za pisanje dokumentacije. V pregledu obstoječih orodij je moč ugotoviti, da nekateri v ta namen implementirajo kar celoten modul »Wiki«, ki je zelo primeren za spletno pregledovanje dokumentacije, manj pa za tisk na papir. Druga rešitev je implementacija nekakšnega urejevalnika besedila, ki pa mora biti na nek način povezan s projektom, tako da je sestavljavcu dokumentacije delo olajšano. Zato si želimo podpreti možnost uvoza

uporabniških zgodb v dokumentacijo, saj je s tem ogrodje dokumentacije že postavljeno, urejevalec dokumentacije pa ga nato le razširi.

3.3.5 Odziven in preprost uporabniški vmesnik

Nepregleden ali slabo odziven uporabniški vmesnik lahko uporabo orodja spremeni iz uporabnega pripomočka v odvečno delo, ki ga je potrebno opravljati poleg samega dela na projektu. Ker je večina orodij implementiranih v obliki spletne aplikacije, lahko postane neprestano osveževanje vmesnika ob tudi najmanjših spremembah zelo moteče. K sreči obstaja mnogo tehnologij (predvsem AJAX - več v poglavju 4.2.1), ki naredijo tudi spletne aplikacije hitre in odzivne, saj se od strežnika vedno pretoči le zahtevana vsebina, ne pa celotna stran.

Preglednost vmesnika pa je mogoče izpopolniti tako, da napredne funkcije umikamo v t.i. dialoge, ki se odprejo le na ukaz. Tako uporabnikom omogočimo, da imajo pred sabo vedno le tisto vsebino, ki jo trenutno potrebujejo. Prav preprostost rabe pa je tudi glavno vodilo, ki sem ga uporabil pri snovanju lastne rešitve za to diplomsko nalogo.

3.3.6 Primerjalna tabela podprtih konceptov

Tabela 1: Primerjava podpore opisanih konceptov s strani preizkušenih orodij

Podprti koncepti:	**Scrum Works**	**Version One**	**Planbox**	**Vision Project**	**Tiny PM**	**Rally**	**Agilo 2**	**Scrum Desk**
avt. merjenje časa dela	NE	NE	**DA**	NE	NE	NE	NE	NE
vodenje dokumentacije	NE	NE	NE	**DA**	**DA**	NE	NE	NE
oddaljena komunikacija	NE	**DA**	NE	**DA**	**DA**	**DA**	**DA**	NE
digitalni »Planning Poker«	NE	NE	NE	NE	NE	NE	NE	**DA**
Deluje kot spletna aplikacija	**DA**	**DA**	**DA**	**DA**	**DA**	**DA**	**DA**	NE

V Tabeli 1 se nahaja povzetek funkcionalnosti posameznih orodij. Ker vsa preizkušena orodja v celoti podpirajo vse osnovne koncepte vodenja projektov po metodi Scrum (planiranje izdaj in Sprintov, vodenje seznamov zgodb in nalog, vodenje napredka na projektu, vodenje preostanka dela, statistična obdelava podatkov, …) je smiselno v primerjalno tabelo vključiti le koncepte opisane v tem poglavju, ki so za namen te diplomske naloge tudi najbolj relevantni. Opazimo lahko, da še nobeno od preizkušenih orodij ne implementira vseh konceptov, kar zagotavlja programu razvitemu v sklopu te diplomske naloge določeno unikatno vrednost.

4 TEHNIČNA ZASNOVA PREDLAGANEGA ORODJA

4.1 SPECIFIKACIJA ZAHTEV

Specifikacija zahtev mora zajemati vse funkcionalnosti, ki jih od končnega izdelka pričakujemo. V duhu osrednje teme tega diplomskega dela, jih lahko navedemo kar v obliki uporabniških zgodb. Zgodbe so zaradi omejenega prostora in s težnjo po večji preglednosti zapisane v prosti obliki, brez navajanja sprejemnih testov in v zaporedju, kot ga narekuje običajna raba tovrstnega orodja.

Za začetek je potrebno določiti temeljne uporabniške zgodbe, ki so predpogoj za delovanje programa. Sem prištevamo prvo namestitev programa in določanje prvih uporabnikov, prav tako pa tudi vzdrževanje njihovih podatkov:

- *Administrator sistema lahko namesti program. Ustvari podatkovno bazo in svoj uporabniški račun.*
- *Administrator sistema lahko vnaša nove uporabnike v sistem. Določi jim uporabniško ime in geslo, osebne podatke (ime, priimek, e-pošta), ter sistemske pravice (administrator sistema, uporabnik sistema).*
- *Administrator sistema lahko ureja in briše obstoječe uporabnike sistema. Spremeni jim lahko uporabniško ime, osebne podatke, geslo in sistemske pravice.*
- *Uporabnik sistema lahko za svoj uporabniški račun spreminja uporabniško ime, geslo in druge osebne podatke.*

Ko so uporabniki sistema določeni, lahko začnemo razmišljati o projektih, ki jih bomo s pomočjo programa vodili. Obdelati je potrebno tako možnost kreiranja projekta z določitvijo njegovih lastnosti in nosilcev, kot tudi možnost njegovega poznejšega urejanja:

- *Član z ustreznimi sistemskimi pravicami lahko začne nov projekt. Postane skrbnik metodologije za ta projekt. Določi mu naziv, opis ter ostale sodelujoče na projektu in njihove vloge (produktni vodja, skrbnik metodologije, član razvojne skupine).*

- *Administrator sistema in skrbnik metodologije lahko urejata lastnosti že obstoječega projekta, dodajata nove člane v projekt, urejata vloge obstoječih članov v projektu in odstranjujeta člane iz projekta.*

Ko je projekt ustvarjen, ga moramo še časovno opredeliti. Skrbnik metodologije mora določiti število Sprintov in njihovo trajanje, ter jim jasno določiti datum začetka in konca.

- *Skrbnik metodologije lahko ustvari nov Sprint. Določi mu začetni in končni datum ter pričakovano hitrost.*
- *Skrbnik metodologije lahko ureja ali briše Sprinte, ki se še niso pričeli.*

Pred pričetkom dela na projektu je potrebno definirati še seznam zahtev v obliki uporabniških zgodb. Omogočiti moramo tudi njihovo kasnejše spreminjanje in dopolnjevanje:

- *Produktni vodja in skrbnik metodologije lahko vnašata nove uporabniške zgodbe v že obstoječ projekt. Za vsako zgodbo lahko določita njeno ime, besedilo, sprejemne teste, prioriteto (must have, could have, sholud have) in poslovno vrednost.*
- *Produktni vodja in skrbnik metodologije lahko urejata in brišeta tiste uporabniške zgodbe v projektu, ki še niso realizirane in niso dodeljene nobenemu Sprintu.*
- *Člani razvojne skupine lahko k uporabniškim zgodbam v dogovoru s produktnim vodjo ali skrbnikom metodologije dopisujejo svoje opombe.*

Uporabniške zgodbe moramo tudi časovno oceniti. To lahko naredimo z metodo »Planning Poker«, izvedbo katere naj program v celoti podpira. Prav tako mora seveda obstajati tudi opcija ročnega vnosa časovne ocene zgodbe:

- *Skrbnik metodologije lahko nedodeljeni zgodbi v seznamu zahtev določi (oziroma spremeni) oceno časovne zahtevnosti.*
- *Skrbnik metodologije lahko za nedodeljeno uporabniško zgodbo začne »Planning Poker«, v katerem lahko sodelujejo vsi člani projekta, prijavljeni v tistem trenutku.*
- *Člani skupine in skrbnik metodologije lahko oddajo svojo časovno oceno v trenutno aktivni igri »Planning Pokra«.*
- *Skrbnik metodologije lahko zaključi trenutno aktivno igro »Planning Pokra« in po presoji rezultat igre shrani kot veljavno oceno za to zgodbo.*

Ko so vse zgodbe v seznamu zahtev časovno ocenjene, lahko skupina dejansko prične pripravljati delo za naslednji Sprint. Izbrati morajo zgodbe, ki jih bodo v tekočem Sprintu realizirali, nato pa jih morajo smiselno razbiti na naloge in tudi te časovno oceniti:

- *Skrbnik metodologije lahko dodaja nove zgodbe v Sprint. To stori tako, da izbere dogovorjeno podmnožico zgodb v seznamu zahtev in jih dodeli Sprintu. Izbere lahko le tiste zgodbe, ki že imajo ocenjeno časovno zahtevnost in še niso bile realizirane.*
- *Skrbnik metodologije in člani razvojne skupine lahko dodajajo nove naloge (tasks) k posamezni uporabniški zgodbi znotraj aktivnega Sprinta. Določijo opis naloge, oceno časa za dokončanje naloge in po želji tudi člana ekipe za njeno realizacijo (naloga s tem še ni dodeljena, saj jo mora član ekipe prej še sprejeti).*
- *Skrbnik metodologije in člani razvojne skupine lahko urejajo vse parametre in brišejo obstoječe naloge (tasks) pri posameznih uporabniških zgodbah.*

Z določitvijo nalog za vse zgodbe v aktivnem Sprintu so izpolnjeni vsi pogoji, da lahko člani programerske skupine pričnejo z delom. Član mora nalogo, na kateri bo delal, najprej sprejeti, da se mu le ta tudi uradno dodeli:

- *Član programerske skupine lahko sprejme še nedodeljeno nalogo aktivnega Sprinta v delo. Naloga se označi kot dodeljena, s tem pa se prepreči, da bi jo lahko sprejel še kak drug član.*
- *Član programerske skupine lahko že sprejeto nalogo vrne nazaj in s tem omogoči, da delo na njej nadaljuje kak drug član.*

Med potekom dela naj se v bazi podatkov samodejno beležijo podatki o tem, kateri član je delal na kateri nalogi in koliko časa je v delo vložil. Članu moramo dati možnost, da te podatke tudi ročno spreminja:

- *Član razvojne skupine lahko prične z delom na nalogi, ki jo je sprejel. Ob tem se zažene števec časa, ki beleži čas njegovega dela. Naloga se označi kot aktivna.*
- *Član razvojne skupine lahko konča delo na trenutno aktivni nalogi. Ob tem se števec časa zaključi, vloženo delo (število ur) pa se zabeleži v bazo. V primeru, da je ta član v istem dnevu že delal na tej nalogi, se število ur le prišteje k že obstoječemu vpisu.*
- *Član skupine lahko pregleduje in dopolnjuje preglednico svojega dela na nalogah v tekočem dnevu in v preteklih dnevih. Lahko popravlja število vloženih ur na posamezni*

nalogi za posamezni dan. Prav tako lahko po svoji presoji določa potrebno število ur za dokončanje naloge.

- *Član skupine lahko označi svojo nalogo kot zaključeno. To stori tako, da vnese 0 kot oceno časa za dokončanje naloge.*

Vzporedno z izdelavo projekta naj bi v skladu s smernicami metodologije Scrum potekala tudi izdelava uporabniške dokumentacije. Program mora to podpirati:

- *Vsi sodelujoči na projektu lahko urejajo uporabniško dokumentacijo za projekt.*
- *Pri urejanju dokumentacije lahko urejevalec uvozi uporabniške zgodbe v dokumentacijo, s čimer si pomaga pri postavitvi njenega ogrodja.*
- *Vsi člani projekta lahko dokumentacijo izpišejo oziroma izvozijo.*

Podpreti želimo tudi možnost komunikacije med člani skupine v stilu »projektnega zidu«, kot ga poznajo nekatera socialna omrežja. Nanj lahko sodelujoči na projektu pripenjajo lastne objave in jih tudi komentirajo:

- *Vsi sodelujoči na projektu lahko na projektni zid pripenjajo nove objave.*
- *Vsi sodelujoči na projektu lahko komentirajo objave na projektnem zidu.*
- *Scrum Master lahko briše objave in komentarje na projektnem zidu.*

Ob koncu Sprinta je v domeni produktnega vodje, da oceni ustreznost realizacije posameznih uporabniških zgodb:

- *Produktni vodja lahko ob koncu Sprinta označi uporabniške zgodbe, ki so prestale potrditveni test, kot realizirane.*
- *Produktni vodja lahko ob koncu Sprinta uporabniške zgodbe, ki niso prestale potrditvenega testa ali še niso dokončane, vrne v seznam zahtev kot nerealizirane, pri čemer lahko k zgodbi doda tudi svoj komentar.*

Za konec moramo podpreti še možnost pregleda vseh glavnih dokumentov metode Scrum. Nekateri so delno zajeti že v ostalih zgodbah, vendar jih zaradi večje preglednosti velja izpostaviti:

- *Vsi člani projekta si lahko ogledajo seznam zahtev (Product Backlog), kjer so zbrane vse uporabniške zgodbe projekta. Zgodbe so jasno razdeljene v dve kategoriji –*

realizirane (zgodba je realizirana in sprejemni test je bil uspešno opravljen), in nerealizirane (vse ostale zgodbe). Nerealizirane zgodbe so še naprej ločene na dodeljene (pripadajo aktivnemu Sprintu) in nedodeljene (vse ostale).

- *Vsi člani projekta si lahko ogledajo seznam nalog (Sprint Backlog), kjer so zbrane uporabniške zgodbe in naloge aktivnega sprinta. Naloge so razdeljene v štiri kategorije – nedodeljene, dodeljene, zaključene in aktivne (t.j. na njih poteka delo v tem trenutku).*
- *Vsi člani projekta si lahko ogledajo grafikon »Burndown Chart« za celotni projekt, kjer je tabelarično in grafično prikazana primerjava vloženega dela s preostalim delom na projektu.*

4.2 PREDSTAVITEV TEHNIČNE ZASNOVE

Naslednji korak na poti k realizaciji orodja je določitev njegovih tehničnih značilnosti. Orodje želimo realizirati kot spletno aplikacijo. S tem olajšamo njegovo rabo v večračunalniškem sistemu in na preprost način omogočimo, da ima vsak uporabnik, ne glede na njegovo fizično lokacijo, dostop do programa. Kljub spletni zasnovi pa želimo ohraniti dobro odzivnost uporabniškega vmesnika, ki je sicer značilna za lokalne aplikacije. Ta cilj pa terja uporabo vrste različnih tehnologij. V grobem jih lahko umestimo v dve kategoriji:

1. tehnologije na strani strežnika (server-side technologies)
2. tehnologije na strani uporabnika (client-side technologies)

4.2.1 Uporabljene tehnologije

Tehnologije **na strani strežnika**, ki so potrebne za razvoj takšne aplikacije, morajo pokrivati dve področji. Prvo področje je realizacija programske logike, ki bo izvedena v enem od številnih skriptnih jezikov. V ta namen sem uporabil jezik PHP [10], ki je tudi sicer med vsemi sorodnimi jeziki najbolj popularen. Drugo področje pa zajema sistem za upravljanje s podatkovno bazo, ki obsega hranjenje, posodabljanje in posredovanje podatkov. V ta namen

sem uporabil tehnologijo MySQL [22]. Obe tehnologiji imata še eno prednost – sta odprtokodni, zato je njuna uporaba za vsakršne namene povsem prosta.

Na strani uporabnika pa je izbira tehnologij precej bolj omejena. Ker želimo imeti spletno aplikacijo, smo omejeni na tehnologije, ki jih poznajo spletni brskalniki. To pa so: HTML (Hyper Text Markup Language) [23] za prenos vsebine strani in CSS (Cascading Style Sheet) [24] za njeno oblikovanje. Tudi izbira skriptnega jezika na strani uporabnika je preprosta. Uporabil sem JavaScript, ki ga znajo interpretirati praktično vsi moderni spletni brskalniki.

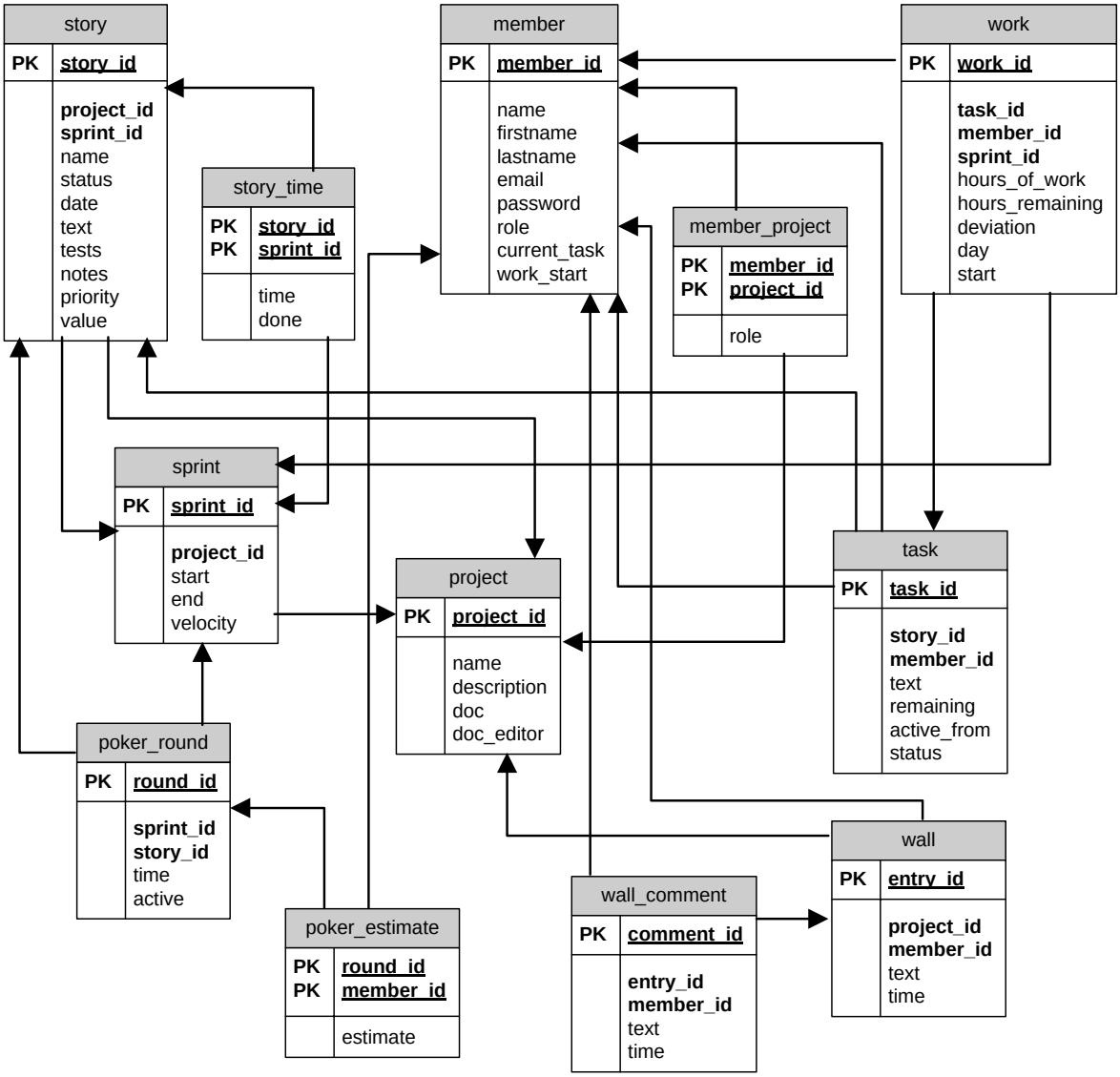
Posebno pozornost velja nameniti tehnologiji AJAX (Asinhronous JavaScript And XML) [25]. Gre za koncept povezovanja uporabnika s strežnikom v ozadju, brez nepotrebnega osveževanja celotnega uporabniškega vmesnika. Zahtevke AJAX vedno sproži aplikacija na uporabnikovi strani preko JavaScript funkcije in nato običajno pričakuje strežnikov odgovor. Ko je ta dostavljen, ga aplikacija ustrezno obdela. Tovrstna komunikacija omogoča, da uporabnik lahko aplikacijo uporablja tudi v času čakanja na strežnikov odgovor. Rezultat pa je hitrejši in bolj odziven uporabniški vmesnik.

Pri implementaciji kode JavaScript sem izkoristil tudi knjižnico jQuery [26], ki kodiranje v tem skriptnem jeziku znatno pohitri in poenostavi, predvsem pa zabriše nekatere sintaktične razlike, ki so prisotne med različnimi brskalniki. Prav tako sem uporabil tudi dodatek jQueryUI [27] za implementacijo naprednih grafičnih elementov programa. Sem spada predvsem implementacija dialogov, zavihkov, datumskih izbirnikov in še nekaterih drugih elementov. Uporabil sem še besedilni urejevalnik tinyEditor [28] za izdelavo urejevalnika dokumentacije projekta.

4.2.2 Zasnova podatkovne baze

Zasnova podatkovne baze programa je prikazana na spodnji skici (Slika 5) in obsega 10 entitetnih tipov ter 2 povezovalna entitetna tipa.

Entitetni tipi v strukturi podatkovne baze so: zgodba (story), član (member), delo (work), Sprint (sprint), projekt (project), naloga (task), runda pokra (poker_round), ocena pri pokru (poker_estimate), zid (wall) in komentar na zidu (wall_comment). Povezovalni entiteti pa povezujeta zgodbo in sprint (story_time) ter člana in projekt (member_project).



Slika 5: Logični model podatkovne baze

Vlogo posameznih entitetnih tipov najlažje objasnimo skozi postopek rabe programa pri vodenju novega projekta. V nadaljevanju poglavja so imena entitetnih tipov oziroma tabel zapisana v navednicah, imena atributov pa v oglatih oklepajih.

Administrator sistema najprej vanj vpiše nove uporabnike. S tem napolni tabelo »member«. Določi jim uporabniško ime in geslo [name, password], ime in priimek [firstname, lastname], e-poštni naslov [email] in sistemske pravice [role]. Preostala polja za enkrat ostajajo prazna.

Uporabnik z ustreznimi pravicami [role] lahko ustvari nov projekt. Tabelo »project« napolni z imenom [name] in opisom [description] projekta (ostala polja za enkrat ostanejo prazna), prav tako pa napolni še vmesno tabelo »member_project« s sodelujočimi člani na projektu [member_id, project_id], pri čemer vsakemu članu določi še vlogo [role] znotraj projekta (1 = Scrum Master, 2 = Product Owner, 0 = Team Member).

Scrum Master mora nato določiti Sprinte. V tabelo »sprint« se zabeležita začetni [start] in končni [end] datum Sprinta, zraven pa še pričakovana hitrost Sprinta [velocity]. Vsak Sprint je seveda vezan na določen projekt [project_id].

Vzporedno lahko lastnik projekta že začne z vnašanjem uporabniških zgodb v tabelo »story«. Zgodba je ob prvem vnosu vezana samo na projekt [project_id], na Sprint [sprint_id] pa bo vezana šele takrat, ko mu bo dodeljena. K zgodbi se zabeležijo informacije o datumu vpisa [date], nazivu [name], besedilu [text], prioriteti [priority] in poslovni vrednosti [value], prav tako pa lahko shranimo še morebitne sprejemne teste [tests] in opombe [notes]. Zgodba je sprva označena kot nedokončana [status = 0], šele ko bodo sprejemni testi uspešno opravljeni, bo postala dokončana [status = 1].

Sledi postopek časovnega ocenjevanja zgodb. Ocene se shranjujejo v vmesno tabelo »story_time«, ki povezuje zgodbo [story_id] s Sprintom [sprint_id]. Namen te povezave je v tem, da lahko v vsakem Sprintu zgodbo ocenimo znova in kljub temu ohranimo vse prejšnje ocene. V tabeli »story_time« se torej hrani še časovna ocena zgodbe [time] v določenem Sprintu, poleg tega pa tudi čas, ki je bil v tekočem Sprintu vložen v delo na tej zgodbi [done] in je na začetku enak nič.

Ocenjevanje lahko izvedemo tudi z igro »Planning Poker«. Ko skrbnik metodologije prične igro »Planning Pokra« za določeno zgodbo, se v tabelo »poker_round« za izbrano zgodbo [story_id] in za trenutni sprint [sprint_id] shrani čas začetka runde [time], prav tako pa se runda označi kot aktivna [active = 1]. Ko je runda aktivna, lahko ostali člani oddajo svojo oceno [estimate], ki se shrani v tabelo »poker_estimate«. Prav tako se zabeleži tudi, kateri član je glasoval [member_id] in v kateri rundi je bil glas oddan [round_id]. Scrum Master nato rundo konča [active = 0] in po presoji povprečno oceno glasovanja vpiše v tabelo »story_time«.

Ocenjene zgodbe lahko v tekočem Sprintu vzamemo v delo (v tabeli »story« se nastavi [sprint_id]). Takšne zgodbe je nato potrebno razbiti na naloge, pri čemer napolnimo tabelo »task«. Določimo vsebino naloge [text], njeno časovno zahtevnost [remaining] in po želji tudi člana za njeno realizacijo [member_id] ter jo označimo za nedodeljeno [status=0]. Ko član zgodbo prevzame, se le ta označi kot dodeljena [status=1] članu, ki jo je prevzel [member_id].

Član nato lahko prične z delom na nalogi. S tem v tabelo »member« vnese nalogo, na kateri trenutno dela [current_task], prav tako pa tudi, kdaj je z delom pričel [work_start]. Ko s svojim delom konča, se trenutna naloga [current_task] nastavi nazaj na 0, s podatki o njegovem delu pa se posodobi tabela »work«. Ta povezuje člana [member_id] z nalogo [task_id] in Sprintom [sprint_id] v katerem je bilo delo opravljeno. Zabeleži se, koliko ur dela [hours_of_work] je član opravil v določenem dnevu [day]. Če za neki dan vnos že obstaja, se ure dela le prištejejo k že opravljenim uram. Član lahko vnese še pričakovano število ur do dokončanja naloge [remaining]. Če se ta vnos (V) razlikuje od razlike med oceno prejšnjega dne in delom trenutnega dne (P – T), se nastavi še deviacija [deviation] (D), tako da velja V = P – T + D, oziroma D = V – P + T. To je pomembno za poenostavitev kasnejše analize natančnosti časovnih ocen.

Ob koncu Sprinta mora produktni vodja sprejeti zgodbe, ki so bile uspešno realizirane. To lahko stori le za tiste zgodbe, pri katerih je za vsako njeno zgodbo zadnji vnos pričakovanega števila ur do dokončanja [remaining] enak 0. To je namreč znak, da je naloga zaključena. Če je produktni vodja zgodbo sprejel, se le ta označi kot dokončana [status=1], v nasprotnem primeru pa se vrne med nedodeljene zgodbe [sprint_id=0, status=0].

Program podpira tudi urejanje dokumentacije projekta [doc]. Ta se nahaja v tabeli »project«, hrani pa se še uporabnika, ki trenutno ureja dokumentacijo [doc_editor], s čimer se prepreči interferenca pri hkratnem urejanju s strani več uporabnikov.

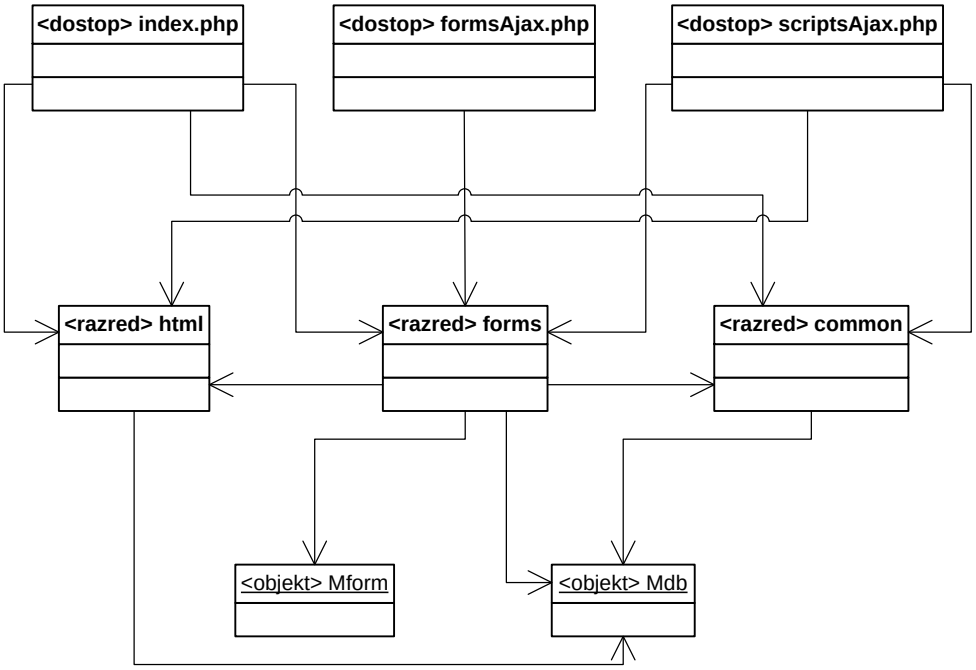
Za konec je potrebno opisati še delovanje komunikacije med člani, oziroma projektni zid. Ko član projektne ekipe objavi sporočilo na projektni zid, se le to zapiše v tabelo »wall«, ki je vezana na projekt [project_id] in člana [member_id], hrani pa čas oddaje [time] in vsebino sporočila [text]. Komentarji na to sporočilo se hranijo v tabeli »wall_comment«, poleg reference na sporočilo [entry_id] pa vsebujejo še referenco na člana [member_id], čas oddaje [time] in vsebino komentarja [text].

4.2.3 Zasnova programske logike

Programska logika orodja je, kot že povedano, realizirana v skriptnem jeziku PHP, ob podpori sistema za upravljanje s podatkovno bazo MySQL. PHP je bil razvit že leta 1995, njegov oče pa je Rasmus Lerdorf. Namenjen je bil predvsem ustvarjanju dinamičnih HTML strani, zato se njegova koda vpisuje neposredno v HTML dokumente in se interpretira na zahtevo. Danes je PHP že v različici 5.4, njegov razvoj pa so prevzeli pri PHP Group [29], kjer sprejemajo tudi standarde na področju razvoja jezika. Najpomembnejši napredek v razvoju je PHP doživel v tretji različici, ko je bila dodana podpora objektom, čeprav je bila le ta zadovoljivo izvedena šele v verziji 5. Podporo objektnemu programiranju sem do neke mere izkoristil tudi pri izdelavi orodja v okviru te diplomske naloge.

Ko uporabnik zahteva dostop do programa, mora to vedno storiti prek ene same datoteke - index.php, ki glede na vhodne parametre izriše ustrezno vsebino. Program ima še dve vstopni točki, namenjeni AJAX zahtevkom; to sta formsAjax.php in scriptsAjax.php. Vse ostale datoteke s programsko logiko so zaklenjene za neposredni dostop in se lahko kličejo le posredno prek teh treh datotek.

Jedro programske logike je sicer izdelano po principu objektnega programiranja. Vsebovano je v treh razredih (forms, common, html) in dveh objektih (Mdb, Mform). Poleg tega sta za namen izrisa grafikonov uporabljena še objekta pChart in pData, ki sta del knjižnice pChart [30]. Grafična ponazoritev strukture programa je podana na Sliki 6 z nekoliko poenostavljenim in prirejenim razrednim diagramom, kjer so navedene tudi dostopne točke.



Slika 6: Poenostavljeni razredni diagram rešitve

4.2.3.1 Dostopne točke do programa

Datoteka **index.php** predstavlja glavne vhodne dveri programa. Tu se ustvari HTML dokument uporabniškega vmesnika. Ta med drugim poskrbi, da se pri uporabniku sproži nalaganje ustreznih stilnih dokumentov CSS in datotek s kodo JavaScript. Vsebina znotraj uporabniškega vmesnika pa se ustvari že na strežniku in je odvisna od več spremenljivk. Za začetek, ali je uporabnik prijavljen v sistem - če ni, se bo vedno izrisal obrazec za prijavo - in kot drugo, kateri razdelek programa je uporabnik zahteval. Sistem je namreč razdeljen na šest razdelkov: Nadzorna plošča (Control Panel), Projektni zid (Project Wall), Seznam zgodb (Product Backlog), Seznam nalog (Sprint Backlog), Moje naloge (My Tasks) in Burn down chart. Kateri razdelek se bo naložil določa t.i. GET parameter, ki se pripiše URL naslovu strani (npr: do nadzorne plošče pridemo z naslovom »index.php?p=panel«). Ko je stran naložena lahko uporabnik s kliki na ustrezne povezave sproži nalaganje novih vsebin prek AJAX-a.

Vsi AJAX zahtevki so usmerjeni na enega od dveh naslovov: scriptsAjax.php ali pa formsAjax.php. ScriptsAjax.php poskrbi za ponovno nalaganje različnih elementov

uporabniškega vmesnika, ki so bili prvotno že naloženi prek index.php. Kot primer lahko navedem dodajanje novega projekta; ko je le ta dodan, je potrebno osvežiti seznam vseh projektov v menijskem izbirniku, kar se zgodi z AJAX zahtevkom na scriptsAjax.php. Po drugi strani pa formsAjax.php skrbi za delo z obrazci. Podprto je nalaganje obrazcev v elemente uporabniškega vmesnika (npr. v dialoge) in pošiljanje izpolnjenih obrazcev na strežnik. Ta na poslan obrazec odgovori bodisi s sporočilom o napaki bodisi s sporočilom o pravilno poslanem in shranjenem obrazcu. Kaj točno se ob vsem tem dogaja na strežniku, pa v nadaljevanju.

4.2.3.2 Razredi in objekti

Objekt Mform skrbi za ustvarjanje HTML kode spletnih obrazcev in za preverjanje ustreznosti uporabnikovih vnosov vanje. Čeprav je na spletu za ta namen moč dobiti številne že narejene PHP knjižnice, sem se odločil za lastno realizacijo, saj sem s tem pridobil na fleksibilnosti. Vsaka instanca objekta Mform tako predstavlja en spletni obrazec. V nadaljevanju je naštetih le nekaj najpomembnejših metod objekta Mform.

- **__construct**(imeObrazca, akcija, stil)
  Metoda je konstruktor objekta Mform in ustvari nov spletni obrazec z izbranim imenom in stilom, podatke pa pošlje na URL naslov »akcija«.
- **formElement**(tipElementa, imeElementa, opisElementa, nastavitve)
  Metoda ustvari element obrazca izbranega tipa z izbranim imenom in opisom. Na voljo je več različnih tipov elementov, kot so: tekstovno vnosno polje (textBox), polje za vnos gesla (passwordBox), spustni meni (simpleList) itd. Ob vsem tem lahko v polju (array) navedemo še dodatne nastavitve, kot so: minimalna dolžina vnosa, regularni izraz za preverjanje pravilnosti vnosa, polje vrednosti za spustni meni itd. Za primer naj navedem klic metode ob ustvarjanju tekstovnega vnosnega polja za vpis poštne številke:
  ```
  formElement ( 'textBox', 'postna_stevilka', 'Vpišite poštno
  številko', array('regEx' => '^[0-9]{4}$', 'required' => true));
  ```
- **endForm**(prikažiGumb, imeGumba, nastavitve)
  Metoda konča trenutni obrazec in v primeru, da je »prikažiGumb« nastavljen, ustvari

gumb za pošiljanje obrazca z izbranim imenom. S poljem nastavitve lahko po potrebi na konec obrazca dodamo še dodatne napise in druge gumbe.

- **__toString**()
  Metoda redefinira privzeto metodo za izpis objekta v obliki niza (String) in tako vrne HTML kodo izdelanega spletnega obrazca.
- **isPosted**()
  Metoda vrne »true« le v primeru, če je bil obrazec pravilno izdelan, pravilno poslan in pravilno izpolnjen. Sicer je njen rezultat »false«.

Objekt Mdb je namenjen povezavi programske logike s podatkovno bazo MySQL. Vsebuje metode za povezavo z bazo, za branje iz nje in za pisanje vanjo.

- **__construct**(naslovStrežnika, nazivBaze, uporabnik, geslo)
  Metoda je konstruktor objekta Mdb in ustvari novo povezavo z MySQL podatkovno bazo z izbranim nazivom na izbranem spletnem naslovu. V bazo se prijavi z uporabniškim imenom »uporabnik« in geslom »geslo«.
- **select**(sql)
  Metoda vrne polje (array) podatkov, ki so rezultat poizvedbe »sql« v bazi. Vsak element rezultata vsebuje eno vrstico iz tabele rezultatov.
- **selectT**(sql)
  Metoda je podobna prejšnji, le da je tabela rezultatov transponirana. Torej vsak element rezultata vsebuje en stolpec iz tabele rezultatov.
- **execute**(sql)
  Metoda izvede stavek »sql« v bazi podatkov. Če je bil le ta uspešno izveden, vrne »true«, sicer pa »false«.

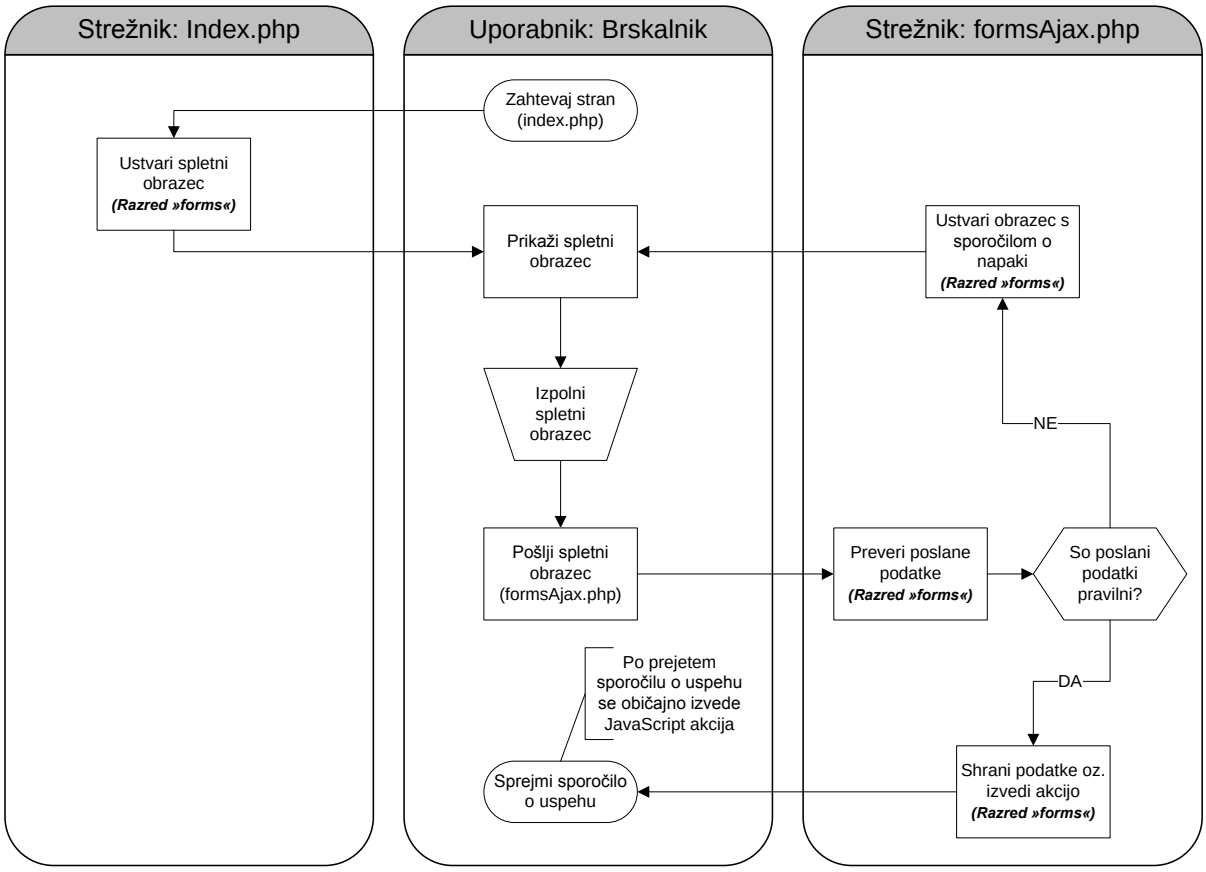
Razred forms predstavlja srce programa, saj realizira logiko za veliko večino interakcij uporabnika z orodjem. Vsaka funkcija razreda forms predstavlja realizacijo enega spletnega obrazca, skupaj pa jih je kar 27 in so z jedrnatim opisom navedene v nadaljevanju:

- **setup** – Obrazec za prvo namestitev programa (generiranje podatkovne baze in administratorskega računa)
- **login** – Obrazec za prijavo uporabnika v sistem.
- **logout** – Realizira odjavo uporabnika iz sistema.

- **addMember** – Obrazec za vnos novega uporabnika v sistem.
- **editMembers** – Seznam uporabnikov sistema s povezavami na možne akcije.
- **editMember** – Obrazec za urejanje profila uporabnika.
- **newProject** – Obrazec za ustvarjanje novega projekta.
- **editProjects** – Seznam vseh projektov s povezavami na možne akcije.
- **editProject** – Obrazec za urejanje podrobnosti projekta.
- **unfinishedStories** – Seznam nedokončanih zgodb v seznamu zahtev s povezavami na možne akcije.
- **finishedStories** – Seznam dokončanih zgodb v seznamu zahtev.
- **newStory** – Obrazec za vnašanje nove uporabniške zgodbe.
- **editStory** – Obrazec za urejanje podrobnosti že obstoječe uporabniške zgodbe.
- **editNotes** – Obrazec za urejanje lastnih opomb pri izbrani uporabniški zgodbi.
- **newSprint** – Seznam obstoječih Sprintov s pripadajočimi akcijami in obrazec za določanje novega Sprinta.
- **sprintBacklog** – Seznam nalog s povezavami na možne akcije.
- **editTasks** – Obrazec za ustvarjanje nove naloge pri izbrani zgodbi skupaj s seznamom že definiranih nalog s pripadajočimi akcijami i.
- **myTasks** – Seznam sprejetih nalog za prijavljenega uporabnika s povezavami na možne akcije.
- **work** – Tabela preteklega dela za izbrano nalogo skupaj z možnostjo urejanja.
- **todaysWork** – Seznam nalog, na katerih je uporabnik delal v zadnjem dnevu, skupaj z možnostjo urejanja podrobnosti o delu.
- **rejectStory** – Obrazec za komentiranje razlogov za zavrnitev realizacije zgodbe.
- **postToWall** – Projektni zid skupaj z obrazcem za dodajanje objav.
- **newWallComment** – Obrazec za dodajanje komentarjev na objavo projektnega zidu.
- **showWallComments** – prikaz komentarjev za izbrano objavo projektnega zidu.
- **editDoc** – Obrazec za urejanje projektne dokumentacije.
- **importStories** – Obrazec za uvažanje uporabniških zgodb v dokumentacijo.
- **planningPoker** – Celotni vmesnik za igranje igre »Planning Poker«.

Klicanje funkcij v razredu »forms« poteka na dva načina; bodisi direktno iz datoteke index.php bodisi z zahtevki AJAX prek datoteke formsAjax.php. Ta princip delovanja

omogoča obdelavo spletnih obrazcev v ozadju, ne da bi uporabnik moral osvežiti spletno stran. Princip delovanja je grafično objasnjen na Sliki 7.



Slika 7: Potek generiranja, izpolnjevanja in pošiljanja spletnega obrazca

Za konec naj omenim še **razreda html in common**. Njun namen je predvsem organizirano hranjenje številnih splošnih funkcij, ki se kličejo iz različnih mest v programu. Funkcije razreda html so namenjene generiranju in izpisu pogosto rabljenih html blokov, kot so npr. dialogi, vmesniki z zavihki, seznami uporabniških zgodb in nalog itd. Razred common pa vsebuje vse druge splošne funkcije, večinoma takšne, ki so namenjene vračanju in izračunu podrobnosti v zvezi s trenutno prijavljenim uporabnikom in trenutnim projektom. Med drugim je podprto vračanje trenutnega uporabnika, njegove vloge, trenutno izbranega projekta, tekočega Sprinta, prihodnjih Sprintov itd.

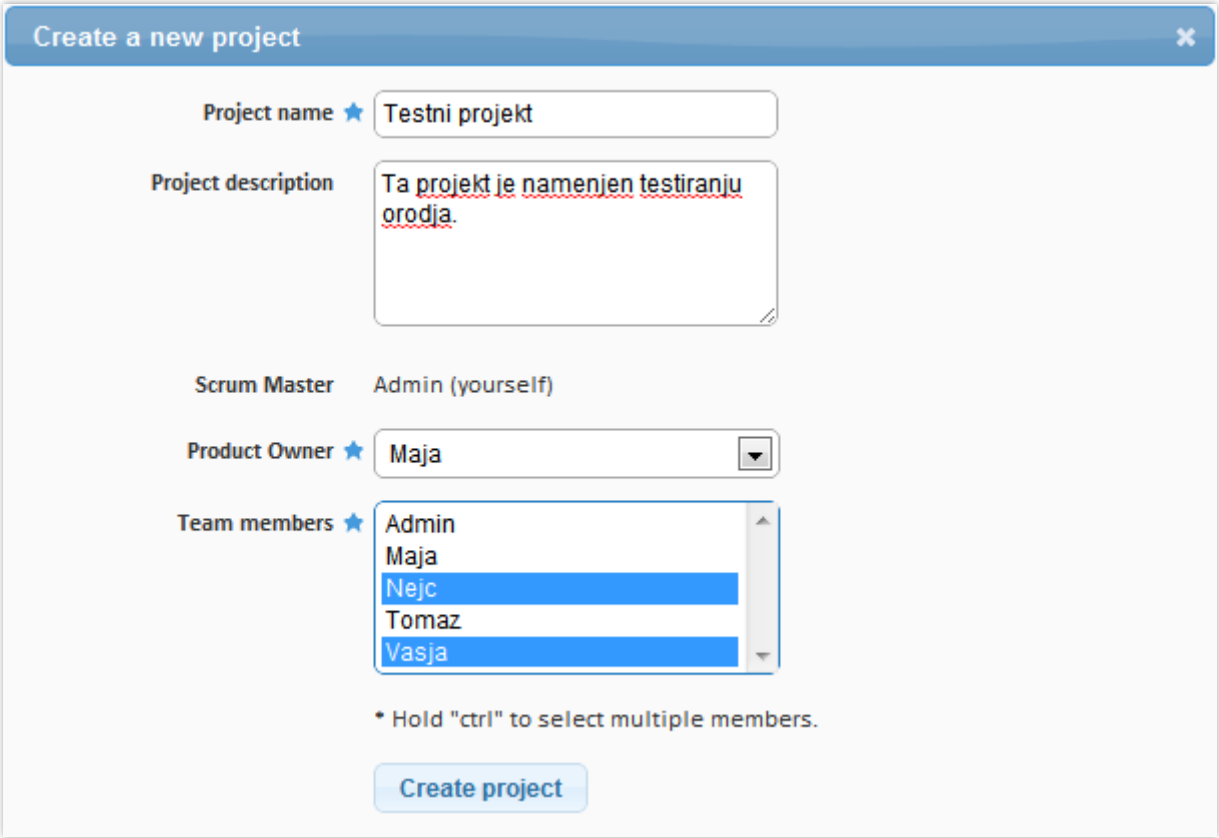
5 PREDSTAVITEV UPORABE ORODJA

Namen tega poglavja je predstaviti delovanje realizirane programske rešitve. Razdelek 5.1 zaobsega predstavitev funkcionalnosti, ki jih smatramo kot osnovne pri vodenju projektov z metodo Scrum, predstavitev vseh drugih funkcionalnosti realiziranega orodja pa je moč najti v razdelku 5.2. Funkcionalnosti so ponovno predstavljene kronološko po zaporedju rabe.

5.1 PREDSTAVITEV VODENJA PROJEKTA

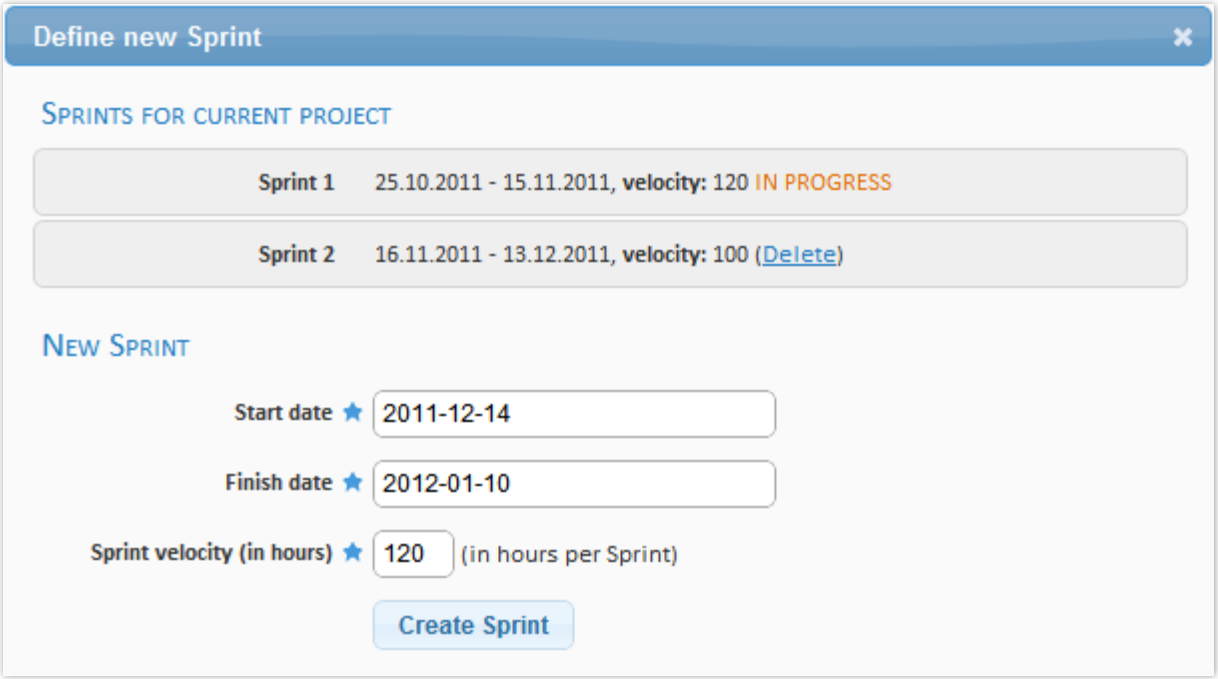
5.1.1 Definiranje novega projekta in njegovo časovno umeščanje

Pri kreiranju novega projekta je uporabniku orodja na voljo obrazec za vnos osnovnih informacij projekta in izbiro sodelujočih na njem, vključno z njihovimi vlogami (Slika 8).



Slika 8: Kreiranje novega projekta

Uporabnik lahko nato projekt še časovno umesti z vnosom začetnih in končnih datumov za posamezne Sprinte (Slika 9). Vse tiste Sprinte, ki se še niso pričeli, lahko uporabnik izbriše in definira ponovno, s čimer je na enostaven način realizirano njihovo urejanje.



Slika 9: Časovno planiranje projekta

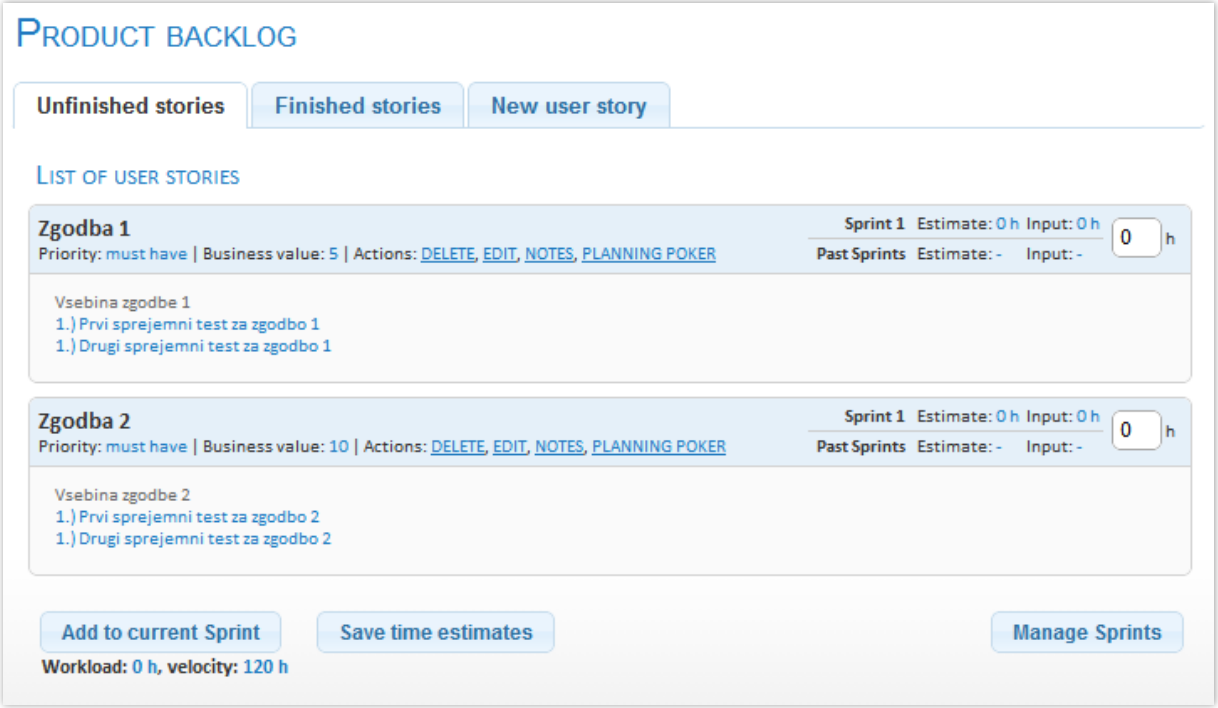
5.1.2 Urejanje seznama zahtev (Product Backlog)

Za vzdrževanje uporabniških zgodb v seznamu zahtev je v največji meri odgovoren produktni vodja. Ponujene so mu vse možnosti za njihovo urejanje, dodajanje in odstranjevanje [Slika 10]. Vsem članom projekta pa je na voljo tudi dodajanje lastnih komentarjev na zgodbe.

Znotraj vmesnika seznama zahtev poteka tudi časovno ocenjevanje zgodb. Ocene lahko skrbnik metodologije vpiše neposredno v okence v zgornjem desnem kotu vsake zgodbe, njihova enota pa so ure. Uporablja se običajna pretvorba: 1 točka = 6 ur. Ocene se lahko pridobi tudi z igro »Planning Poker«, več o tem v razdelku 5.2.2.

Ko je Sprint v teku, se pred imenom vsake zgodbe pojavijo okenca, ki jih skrbnik metodologije lahko odkljuka in s tem podmnožico zgodb dodeli aktivnemu Sprintu. Izbrane

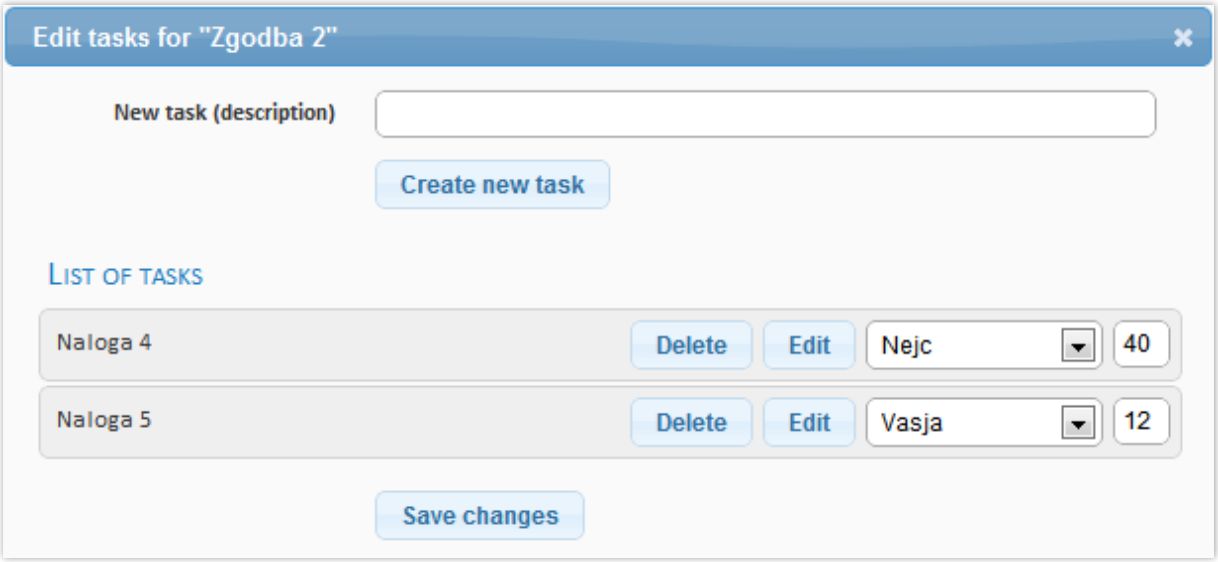
zgodbe se pojavijo tudi v seznamu nalog (Sprint Backlog). Zgodbe je tako mogoče dodeliti le trenutno aktivnemu Sprintu, ne pa tudi prihodnjim, kar je zavestno vgrajena omejitev.



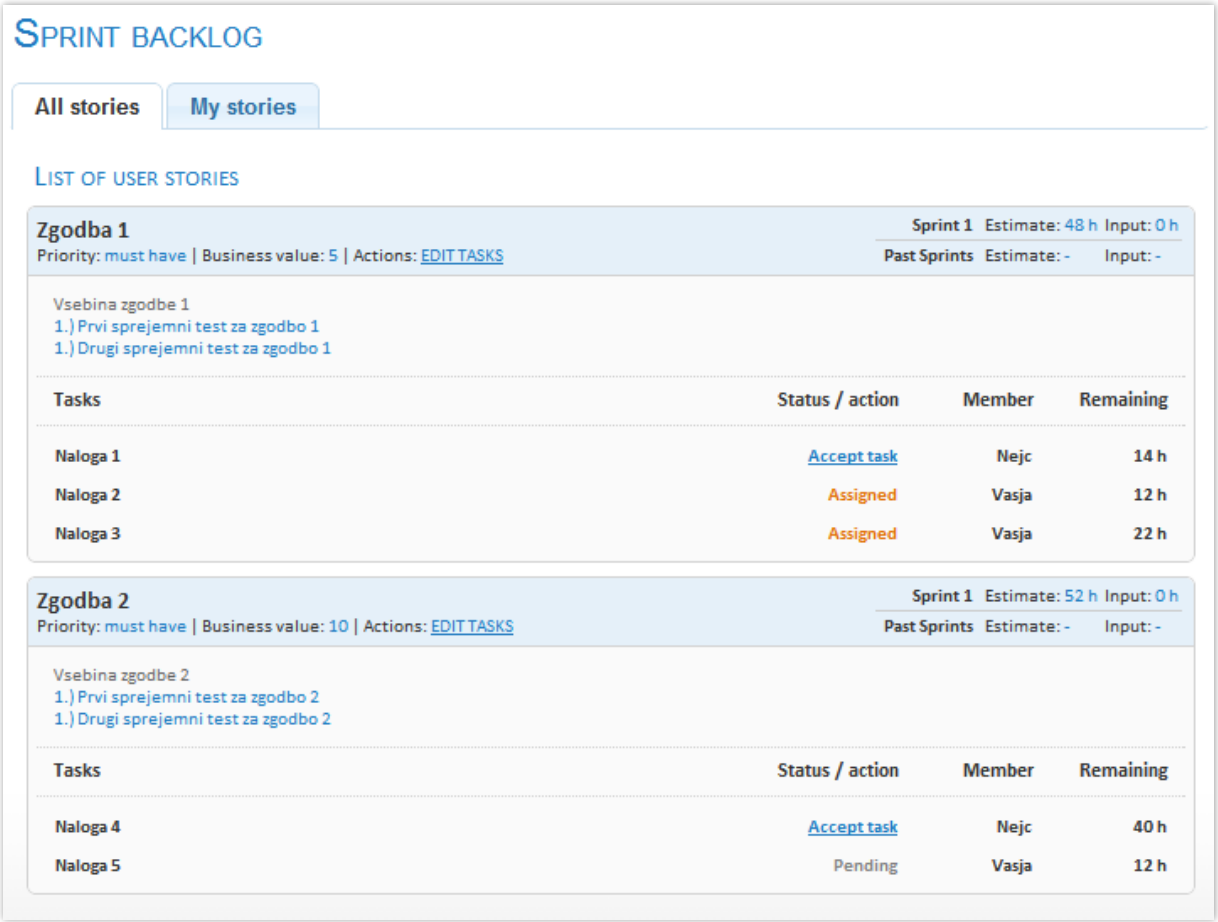
Slika 10: Seznam zahtev (Product Backlog)

5.1.3 Urejanje seznama nalog (Sprint Backlog)

Seznam nalog (Slika 12) se nahaja v svojem razdelku programa. Vedno vsebuje seznam zgolj tistih zgodb, ki so dodeljene trenutno aktivnemu Sprintu. Te zgodbe morajo člani skupine razbiti na posamezne naloge, kar lahko storijo preko za to namenjenega dialoga (Slika 11). V njem lahko nastavijo ime naloge, določijo njeno časovno oceno in predlagajo člana za njeno realizacijo. Naloga s tem sicer še ni dodeljena. V seznamu nalog mora vsak član razvojne skupine s klikom na povezavo »Sprejmi nalogo« (Accept task) le to še dejansko prevzeti v delo. Naloga dobi status »Dodeljena« (Assigned) in se prikaže v razdelku »Moje naloge« (My tasks). Še nesprejete naloge imajo v seznamu nalog status »Čakanje« (Pending).



Slika 11: Urejanje nalog

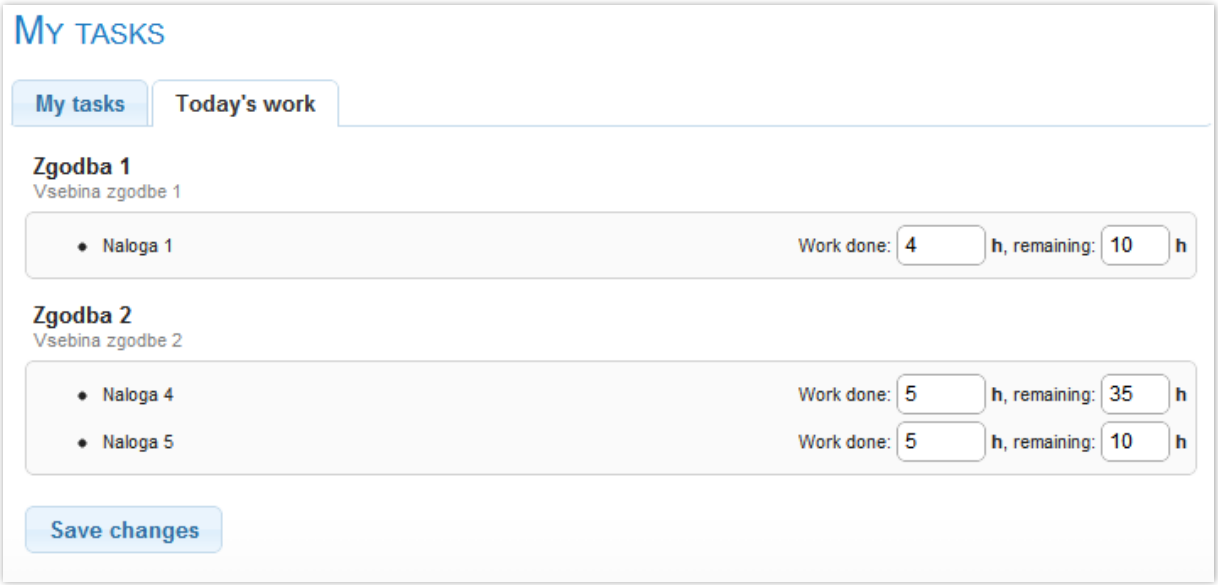


Slika 12: Seznam nalog (Sprint Backlog)

5.1.4 Delo na nalogah

Razdelek »Moje naloge« ponuja skupini programerjev kar najboljši pregled njihovega dela. Na enem mestu so zbrane vse naloge, ki jih je posamezni član v tekočem Sprintu sprejel v delo (Slika 13), prav tako pa ima tudi hiter vpogled v delo trenutnega dne, kjer lahko dopolnjuje in popravlja podatke o vloženem in preostalem delu na tekočih nalogah (Slika 14).

Slika 13: Pregled sprejetih nalog

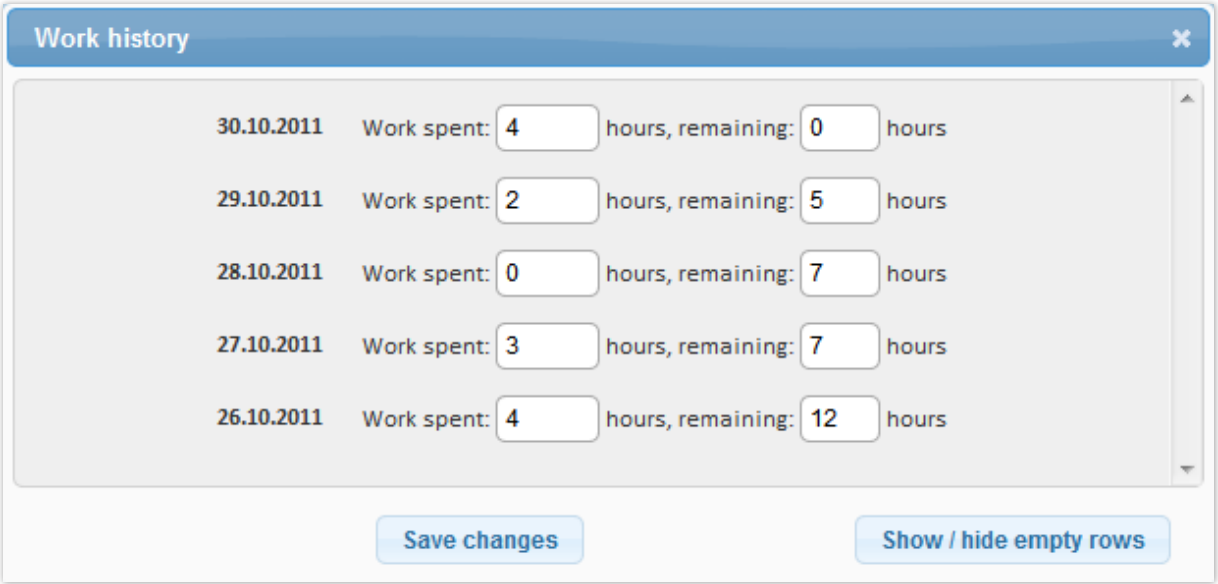


Slika 14: Pregled dela za tekoči dan

Preostale operacije, ki jih lahko v tem razdelku še izvaja uporabnik, vključujejo:

- začetek dela na nalogi (Start work) - več o tem v razdelku 5.2.4
- pregled zgodovine dela (Work history)
- možnost prepustitve posamezne naloge drugim članom (Release task)

Pregled zgodovine dela (Slika 15) velja še dodatno izpostaviti. Uporabniki lahko v tem dialogu vplivajo na zabeležene količine vloženega in preostalega dela, a le za obdobje trenutnega Sprinta. Ko se le ta konča, ocene postanejo trajno shranjene. Za večjo preglednost je uporabniku na voljo še opcija »Prikaži/skrij prazne vrstice« (Show/hide empty rows), ki po želji skrije podatke o delu za tiste dni, kjer je količina vloženega dela enaka nič.



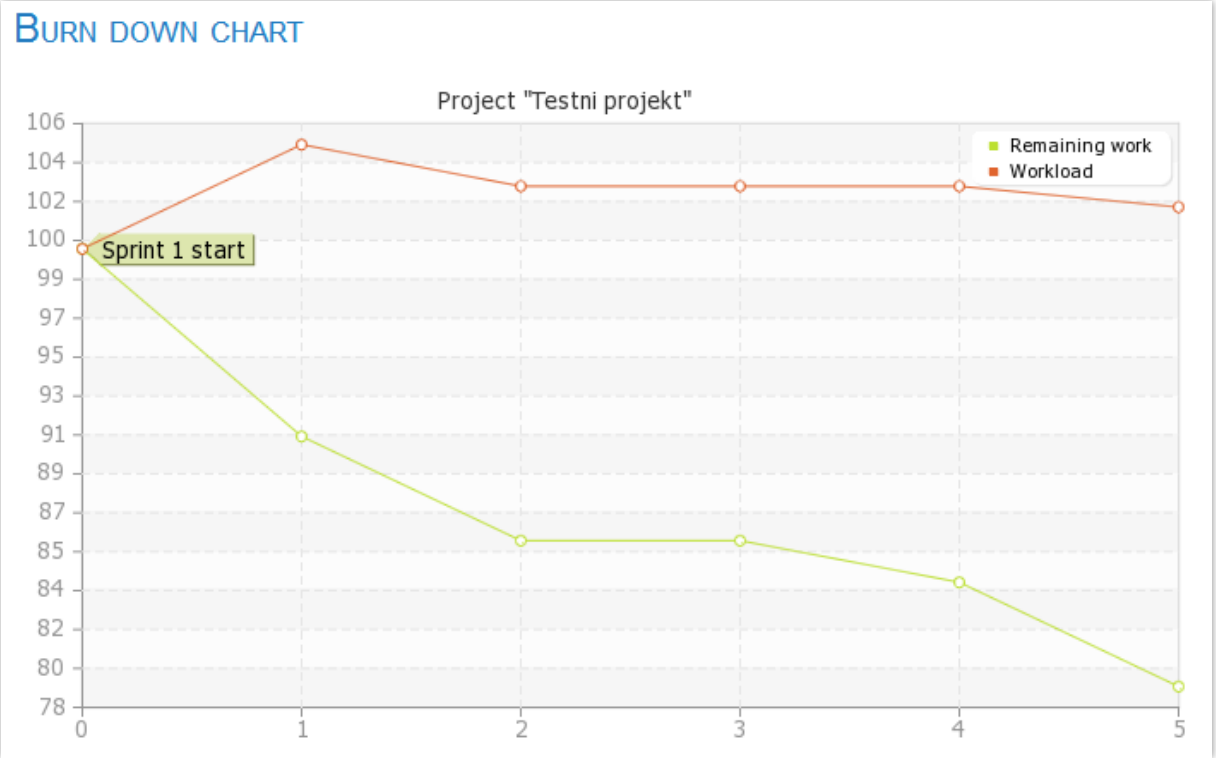
Slika 15: Pregled zgodovine dela

5.1.5 Statistični pregled dela na projektu

Še zadnji razdelek programa, ki ga obravnavamo v tem poglavju, je prikaz grafikona »Burn-down chart« in drugih statističnih podatkov (Slika 16). Na grafu, ki je v celoti izveden z uporabo knjižnice pChart, se poleg običajne krivulje preostalega dela (zelena barva) izriše tudi krivulja skupne količine dela (rdeča barva), ki predstavlja časovni posnetek vsote že opravljenega dela in dela, ki ga je še potrebno opraviti. Ta krivulja narašča, ko se v seznam zahtev dodajajo nove zgodbe ali pa se povečujejo ocene za že obstoječe, pada pa v primerih,

ko kakšno od zgodb izbrišemo oziroma se katera izmed časovnih ocen zniža. Razdalja med krivuljama v smeri y tako predstavlja opravljeno delo na projektu.

Ostali statistični podatki tega razdelka vključujejo tabelo primerjave med planirano in dejansko hitrostjo Sprintov in tabelo primerjave med časovnimi ocenami zgodb in dejansko porabljenim časom. Ti podatki so še posebej uporabni pri planiranju prihodnjih Sprintov, saj se takrat lahko naslonimo na informacije o natančnosti časovnih ocen v preteklosti.



Slika 16: Grafikon "Burn-down"

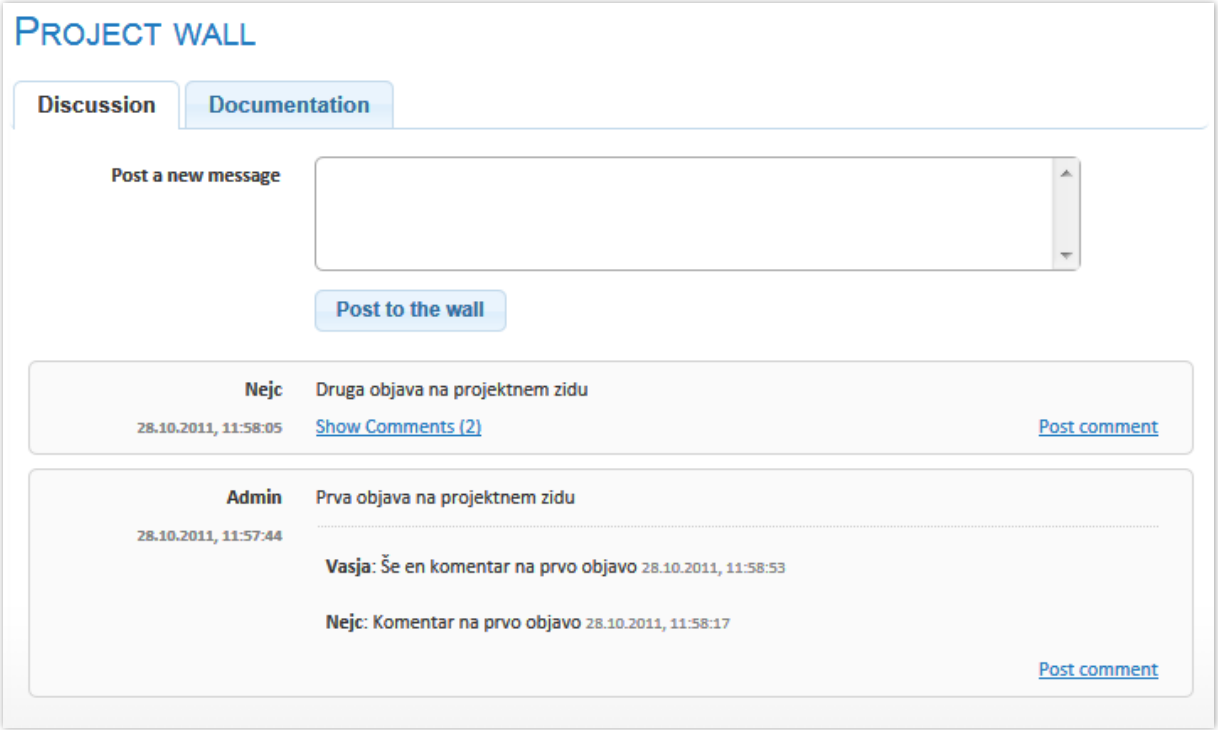
5.2 PREDSTAVITEV OSTALIH FUNKCIONALNOSTI PROGRAMA

5.2.1 Podpora komunikaciji med člani

Za komunikacijo je v realiziranem programu vsem članom na voljo projektni zid (Slika 17), ki se nahaja v enem od razdelkov orodja. Že takoj po prihodu na stran je mogoče v okence na zidu vpisovati nove objave, ki se dodajajo na vrh obstoječih objav. S klikom na »Pošlji

komentar« (Post comment) je podprto tudi komentiranje objav na zidu, s čimer imajo člani skupine možnost podajanja svojega mnenja na objave drugih.

Takšen način komuniciranja se je pri socialnih omrežjih že izkazal kot izvrsten za informiranje ciljne skupine ljudi o aktualnih dogodkih. Prav zato ga lahko smatramo tudi kot dobro osnovo za hitro širjenje pomembnih informacij pri projektnem delu z metodo Scrum, še posebej izven uradnih sestankov.



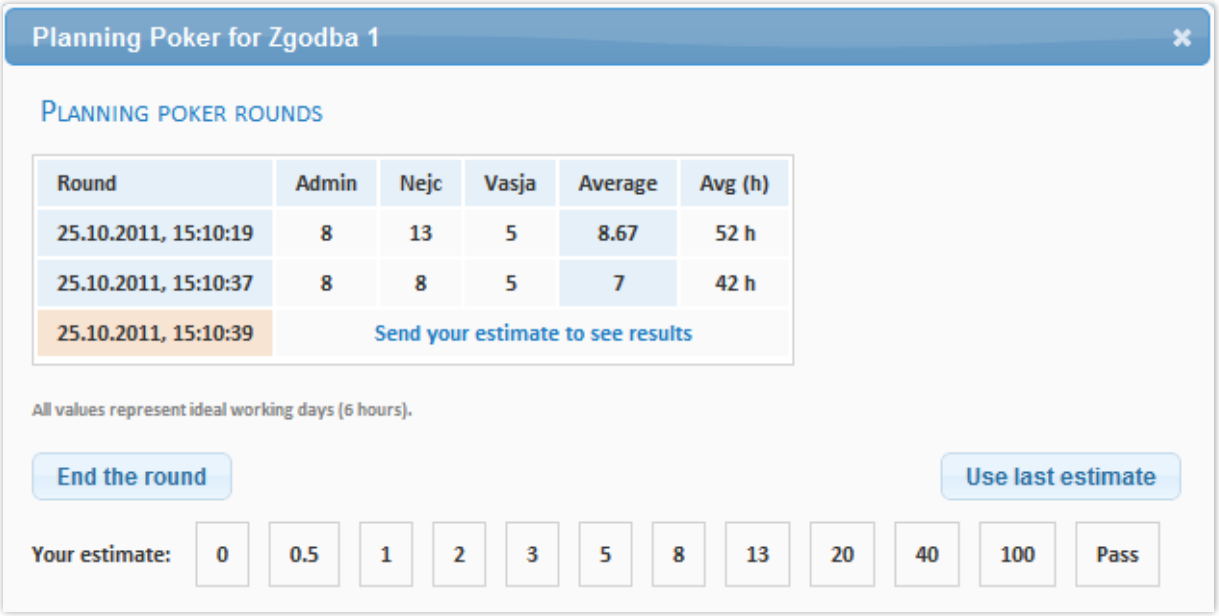
Slika 17: Projektni zid

5.2.2 Planning Poker

Igro »Planning Poker« smo omenjali že večkrat. Na sliki v nadaljevanju (Slika 18) je prikazana njena realizacija v končni rešitvi. Igro za posamezno zgodbo sproži skrbnik metodologije, s čimer preostalim članom odpre možnost glasovanja, ti pa lahko nato oddajo svoj glas s klikom na »kartico« z ustrezno oceno. Uporabnikom so med igro na voljo tudi informacije o vseh preteklih igrah, ki jim lahko služijo kot osnova pri podajanju njihove nove ocene. Program tudi preprečuje, da bi uporabniki videli ocene sočlanov še preden bi oddali svojo oceno, saj bi ta informacija preveč vplivala na njihovo odločitev. Ko so vse ocene

oddane, lahko skrbnik metodologije igro zaključi, povprečje vseh ocen pa bodisi uporabi kot končno oceno zgodbe, bodisi ga zavrže in začne novo rundo Pokra.

Program tekom igre samodejno skrbi za vedno aktualno stanje uporabniškega vmesnika, saj se periodično (na 2 sekundi) povezuje s strežnikom z metodo AJAX in tako zagotavlja sprotno osveževanje vmesnika v ozadju. Tudi sicer potekajo vse operacije (npr: oddajanje ocen, začenjanje nove runde, končanje tekoče runde) prek zahtevkov AJAX, kar pomeni, da uporabniku tekom igre ni nikoli potrebno osvežiti celotnega uporabniškega vmesnika.



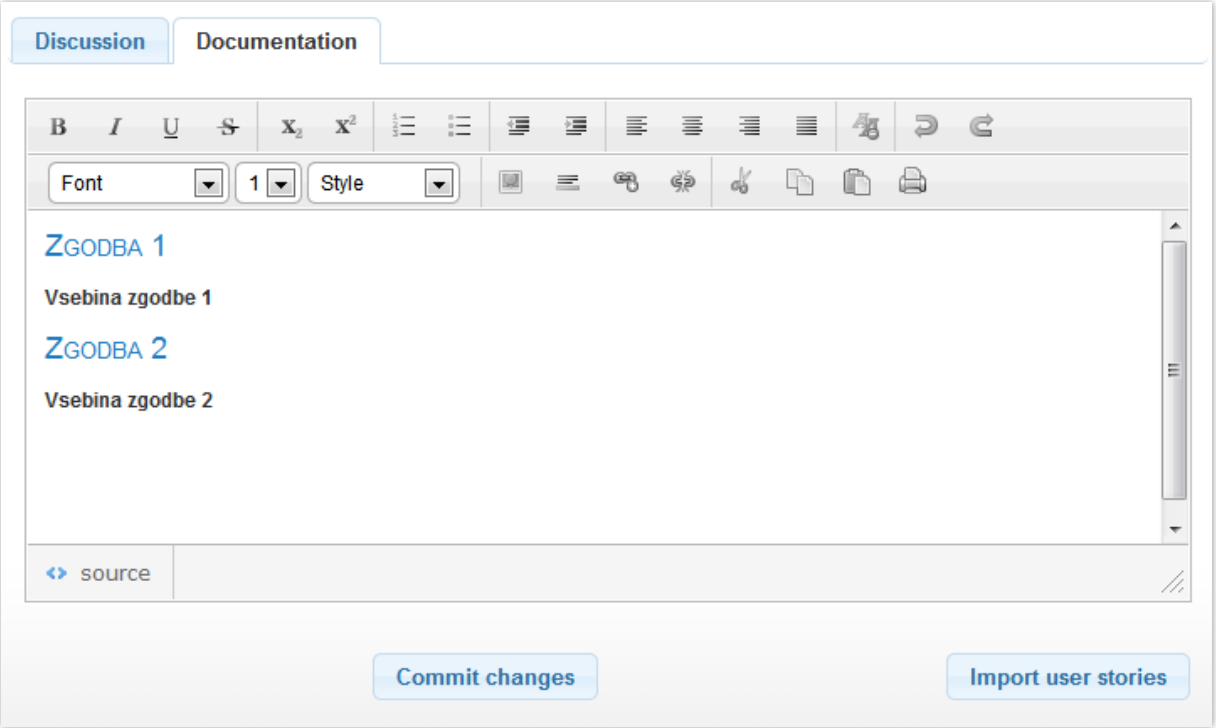
Slika 18: Igra "Planning Poker"

5.2.3 Ustvarjanje uporabniške dokumentacije

Uporabniška dokumentacija je pomemben projektni dokument za končne uporabnike produkta, v katerem je opisan postopek uporabe razvijanega programa. In čeprav lahko dokumentacijo urejamo s katerimkoli urejevalnikom besedil, s tem izgubimo njeno dosegljivost na spletu, poleg tega pa lahko nastane zmeda z različnimi verzijami dokumentacije pri različnih urejevalcih, ki bi jih bilo potrebno združiti.

Realizirana rešitev omogoča urejanje dokumentacije kar na spletu z uporabo preprostega tekstovnega urejevalnika tinyEditor (Slika 19), ki temelji na tehnologiji JavaScript. Urejevalcu je ponujena tudi možnost uvoza uporabniških zgodb v dokumentacijo, s čimer si

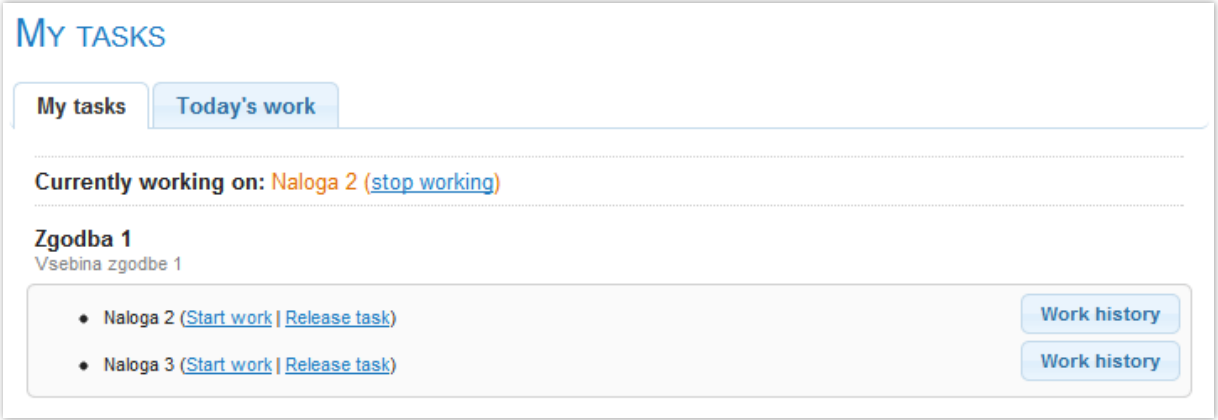
lahko pomaga pri njeni zasnovi. Poleg tega je podprto tudi enostavno zaklepanje urejanja dokumentacije na način, da jo lahko hkrati vedno ureja le en uporabnik. S tem se prepreči, da bi si urejevalci dokumentacije »skakali v zelje«.



Slika 19: Urejevalnik dokumentacije

5.2.4 Podpora sprotnemu vodenju dela na projektu

Člani programerske skupine si lahko z orodjem pomagajo tudi pri beleženju delovnih ur (Slika 20). V razdelku »Moje naloge« (My tasks) se poleg vsake naloge nahaja povezava »Začni delo« (Start work), ki v podatkovni bazi označi izbrano nalogo kot aktivno, uporabnik pa nato začne na njej opravljati delo. Med delom se lahko iz programa tudi odjavi, ne da bi to vplivalo na merjenje časa, saj se le to opravlja na strežniku. Ko uporabnik z delom konča, mora o tem obvesti program s klikom na »Končaj delo« (Stop working), ta pa avtomatsko izračuna čas dela in ga zabeleži v bazo. Ob koncu dneva lahko uporabnik v razdelku »Današnje delo« (Today's work) pregleduje in popravlja podatke o delu za vse naloge, na katerih je tisti dan delal. Prav tako lahko vnaša tudi ocene časa za njihovo dokončanje.



Slika 20: Aktivno delo na nalogi

Z opisanim postopkom po eni strani razbremenimo programerja ročnega vnašanja delovnih ur v program, po drugi strani pa dosežemo večjo natančnost vnesenih vrednosti in posledično natančnejšo statistiko dela. Ta pa je uporabna predvsem pri planiranju prihodnjega dela na projektu in pri morebitni izdelavi poročil o poteku procesa.

6 SKLEP

Orodje, ki je bilo razvito v sklopu te diplomske naloge, v ospredje postavlja člane razvojne skupine. Glavni namen rešitve je bil ponuditi razvijalcem kar največjo podporo pri razvojnem procesu ob kar najmanjšem dodatnem delu z upravljanjem programa. V ta namen končna rešitev podpira avtomatsko vodenje vloženega časa, sprotno ustvarjanje uporabniške dokumentacije, enostavno komunikacijo med člani razvojne skupine in preprosto ocenjevanje časovne zahtevnosti uporabniških zgodb. Ob vsem tem pa celoten proces poteka prek odzivnega uporabniškega vmesnika, ki večino komunikacije s strežnikom opravlja z uporabo tehnologije AJAX. Delo s programom pri razvoju po metodi Scrum je tako predvsem za člane razvojne skupine še dodatno poenostavljeno.

V morebitni prihodnji različici programa bi bilo smiselno dodajati še funkcionalnosti, ki bi bile v pomoč tudi drugim udeležencem v projektu. Orodje že sedaj vsebuje vse potrebne zasnove za izboljšavo njegovih statističnih zmogljivosti, ki bi bile v interes predvsem produktnemu vodji in skrbniku metodologije. Ponudili bi lahko samodejni izračun nekaterih pomembnih poslovnih in procesnih spremenljivk, kot so izračun faktorja donosnosti naložbe, priporočene hitrosti Sprintov in faktorja tveganja glede na zgodovinski potek procesa. Prav tako bi lahko dodali še možnost za neposredno sodelovanje s končnimi uporabniki prek modula za tehnično podporo in sistema za sporočanje hroščev. Preostali deležniki projekta (piščanci) pa bi cenili predvsem možnost izdelave in izpisa poročil o poteku razvojnega procesa.

Možnosti za nadgradnjo je torej še dovolj, o čemer priča tudi hiter razvoj ostalih tovrstnih orodij na trgu, saj so samo v obdobju pisanja te diplomske naloge številna orodja opisana v poglavju 3.2 prešla v novo različico. To pa nas napeljuje na sklep, da je razvoj programske opreme z agilnimi metodami, še posebej pa z metodo Scrum, v izrednem porastu.

7 VIRI

7.1 LITERATURA

[1] Ken Schwaber: Agile Project Management with Scrum, Microsoft Press - Microsoft Corporation, 2004.

[2] Kent Beck, Mike Beedle, James Grenning in drugi: Manifesto for Agile Software Developement, http://agilemanifesto.org/, 2001

[3] Krisper Marjan, Rupnik Rok, Bajec Marko in drugi: Enotna metodologija razvoja informacijskih sistemov [Zv. 1] Uvod, Center vlade RS za informatiko, 2003.

[4] Laurie Williams: Agile Software Developement Methodologies and Practices, Advances in Computers, vol. 80, 2010, str. 1-44.

[5] Viljan Mahnič, Problemi in rešitve pri uvajanju metode Scrum v proces razvoja programske opreme, Zbornik 18. konference "Dnevi slovenske informatike", Portorož, april 2011.

[6] Mike Cohn: User Stories Applied For Agile Software Developement, Addison – Wesley, 2008.

[7] James Grenning, Planning Poker or How to avoid analysis paralysis while release planning, Renaissance Software Consulting, 2002.

[8] Michael J. Rees, A Feasible User Story Tool for Agile Software Development, Ninth Asia-Pacific Software Engineering Conference, 2002

[9] Gayane Azizyan, Miganoush K. Magarian, Mira Kajko-Mattson, Survey of Agile Tool Usage and Needs, Agile Conference, 2011

[10] Matt Zandstra, Sams Teach Yourself PHP in 24 hours, Sams, 2003

7.2 INTERNETNI VIRI

[11] http://www.versionone.com/state_of_agile_development_survey/10

[12] http://www.agilealliance.org/

[13] http://www.userstories.com

[14] http://www.open.collab.net/products/scrumworks

[15] http://www.versionone.com/

[16] http://www.planbox.com

[17] http://www.visionproject.se/

[18] http://www.tinypm.com/

[19] http://www.rallydev.com/

[20] http://www.agiloforscrum.com

[21] http://www.scrumdesk.com/

[22] http://www.mysql.com

[23] http://www.w3.org/standards/techs/html

[24] http://www.w3.org/standards/techs/css

[25] http://www.w3schools.com/ajax/

[26] http://jquery.com/

[27] http://jqueryui.com/

[28] http://www.scriptiny.com/2010/02/javascript-wysiwyg-editor/

[29] http://www.php.net/

[30] http://pchart.sourceforge.net/