UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

DAMIR LALIĆ

ORODJE ZA VODENJE IN MERJENJE UČINKOVITOSTI RAZVOJA PROGRAMSKE OPREME PO METODI SCRUM

MAGISTRSKO DELO

Mentor prof. dr. Viljan Mahnič

Ljubljana, 2014

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.: 120-MAG-ISO/2011
Datum: 24. 11. 2011

Damir LALIĆ, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Orodje za vodenje in merjenje učinkovitosti razvoja programske opreme po metodi Scrum**

A software tool for managing and measuring performance of a Scrum-based software development process

Tematika naloge:
Agilne metode za razvoj programske opreme so se pojavile kot alternativa tradicionalnim discipliniranim metodam. Temeljijo predvsem na komunikaciji med ljudmi, ki sodelujejo na določenem projektu, hitrem odzivu na spremembe uporabniških zahtev ter zadovoljstvu strank. Poglavitni namen agilnih metod je odpravljanje problemov in tveganj, ki so posledica nefleksibilnosti in počasne odzivnosti na spremembe pri tradicionalnih discipliniranih pristopih za razvoj programske opreme. Ena izmed danes najbolj razširjenih metod je metoda Scrum.

V magistrski nalogi predstavite osnovne koncepte agilnih metod za razvoj programske opreme, podrobno pa opišite metodo Scrum, izpostavite njeno uporabnost, trenutno razširjenost in preučite možnosti za uvedbo meritev za spremljanje in izboljšanje učinkovitosti razvojnega procesa. V okviru merjenja učinkovitosti predlagajte in opišite metrike, ki bodo le-to omogočale, s posebnim poudarkom na indeksih SPI in CPI, ki ju predlaga metoda prislužene vrednosti. V nadaljevanju implementirajte programsko orodje, ki bo omogočalo podporo vodenju procesa razvoja programske opreme po metodi Scrum, hkrati pa bo na osnovi vhodnih podatkov možno meriti in spremljati učinkovitost razvoja. V sklopu implementacije uporabite več nivojsko arhitekturo in nazorno predstavite odločitev o izbiri tehnologij za realizacijo orodja. Orodje uporabite in preizkusite na primeru iz prakse.

Mentor:

prof. dr. Viljan Mahnič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

magistrskega dela

Spodaj podpisani Damir Lalić, z vpisno številko 63970088, sem avtor magistrskega dela z naslovom »Orodje za vodenje in merjenje učinkovitosti razvoja programske opreme po metodi Scrum«.

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal/-a samostojno pod vodstvom mentorja prof. dr. Viljana Mahniča,
- so elektronska oblika magistrskega dela, naslova (slov., angl.), povzetka (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela
- in soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 9.5.2014 Podpis avtorja: _____

Zahvala

Zahvaljujem se mentorju prof. dr. Viljanu Mahniču za vodenje, strokovno usmeritev in pomoč pri pripravi magistrskega dela.

Svoji soprogi Alissi se zahvaljujem za veliko mero potrpežljivosti, spodbujanja, motiviranja in soustvarjanje družinskega okolja v katerem je izvedba magistrske naloge bila mogoča.

Zahvaljujem se tudi svojim staršem za moralno podporo in bratu Dinu za pomoč pri slovnični, pravopisni in jezikovni obdelavi besedila.

KAZALO

Kazalo slik

Kazalo tabel

Seznam kratic

AJAX *Asynchronous JavaScript and XML*
Asinhrona JavaScript in XML

API *Application Programming Interface*
Programski vmesnik

ASP *Active Server Pages*
Aktivne strežniške strani

CGI *Common Gateway Interface*
Skupni prehodni vmesnik

CLR *Common Language Runtime*
Skupno izvajalno okolje za jezike

CPI *Cost Performance Index*
Stroškovni indeks

CSS *Cascading Style Sheets*
Prekrivni slogi

DOM *Document Object Model*
Dokumentno objektni model

EVM *Earned Value Method*
Metoda prislužene vrednosti

FTP *File Transfer Protocol*
Protokol za izmenjavo datotek preko omrežja

(X)HTML **(***Extensible) Hypertext Markup Language*
(Razširljivi) Označevalni jezik za prenos hiperteksta

HTTP *Hypertext Transfer Protocol*
Protokol za prenos hiperteksta med odjemalcem in strežnikom na spletu

ISAPI *Internet Server Application Programming Interface*
Programski vmesnik spletnega strežnika

JSON *JavaScript Object Notation*
Notacija objektov jezika JavaScript

JSP *Java Server Pages*
Strežniške strani pisane v programskem jeziku Java

NSAPI *Netscape Server Application Programming Interface*
Programski vmesnik spletnega strežnika proizvajalca Netscape

PHP *PHP: Hypertext Preprocessor*
Splošno uporaben skriptni programski jezik, ki ga tolmači strežnik

REST *Representational State Transfer*
Predstavitveno stanje prenosa

RPC *Remote Procedure Call*
Klic oddaljene procedure

SDK *Software Development Kit*
Komplet funkcij za razvoj

SOAP *Simple Object Access Protocol*
Protokol enostavnega dostopa do objekta

SPI *Schedule Performance Index*
Terminski indeks

SQL *Structured Query Language*
Strukturiran povpraševalni jezik za delo s podatkovnimi bazami

UDDI *Universal Description, Discovery and Integration*
Imenik spletnih storitev

WSDL *Web Services Design Language*
Jezik za dizajniranje spletnih storitev

XML *Extensible Markup Language*
Razširljivi označevalni jezik

XSLT *Extensible Stylesheet Language Transforms*
Razširljivi jezik za preoblikovanje dokumentov XML

Povzetek

V magistrski nalogi so predstavljeni osnovni koncepti agilnih metod za razvoj programske opreme, podrobno pa je opisana metoda Scrum. Izpostavljeni sta njena uporabnost in trenutna razširjenost, obenem so preučene možnosti za uvedbo meritev za spremljanje in izboljšanje učinkovitosti razvojnega procesa. V okviru merjenja učinkovitosti so predlagane in opisane metrike, ki le-to omogočajo, s posebnim poudarkom na merjenju terminskega indeksa SPI (ang. *Schedule Performance Index*) in stroškovnega indeksa CPI (ang. *Cost Performance Index*), ki ju predlaga metoda prislužene vrednosti (ang. *Earned Value Method*, skrajšano EVM).

Posebno poglavje je namenjeno detajlnemu prikazu postopka izdelave programskega orodja, ki omogoča podporo vodenju procesa razvoja programske opreme po metodi Scrum. V okviru implementacije so na začetku izpostavljene elementarne zahteve za podporo vodenju procesa razvoja, kot tudi zahteve za podporo merjenju učinkovitosti. Predstavljen je visokonivojski arhitekturni načrt programskega orodja in odločitev o izbiri tehnologij, ki so bile uporabljene za implementacijo. Pomemben del naloge predstavljajo tudi v nadaljevanju opisani tehnični detajli implementacije. V sklopu tega so opisane bistvene datoteke, ki vsebujejo izvršilno programsko kodo. Logično razdeljena datotečna struktura programskega projekta je prav tako predstavljena in ustrezno opisana. Za boljše razumevanje so tako izpostavljeni in poudarjeni deli kode, ki se na eni strani izvršujejo na strežniku, na drugi pa na odjemalcu. V okviru ustrezno načrtovanega in postavljenega podatkovnega modela, ki ga programska koda uporablja za shranjevanje vseh zapisov za ustrezno vodenje razvoja programske opreme po metodi Scrum, so opisane najbolj pomembne tabele, polja ter povezave med njimi.

Tehničnemu opisu sledi predstavitev uporabniškega vmesnika, v katerem so zajete vse bistvene funkcije, ki jih orodje podpira, njihova praktična uporaba in rezultati, ki jih le-te dajejo. Na ta način je bralec seznanjen tudi s praktično uporabo programskega orodja.

Uporaba orodja je nadalje prikazana in preizkušena na primeru iz prakse, za zaključek pa so predlagane smernice za izboljšave in dodatne funkcionalnosti, ki bi jih orodje lahko podpiralo v okviru nadaljnjega razvoja.

Ključne besede: Scrum, SPI, CPI, programsko orodje, praktična uporaba.

Abstract

In this Master's thesis the basic concepts of agile software development methods are presented in general. Scrum method is described in detail, its usefulness exposed, the current prevalence and the possibilities of introducing measurements to monitor and improve the efficiency of the development process are explored. In the scope of measuring the effectiveness, the metrics that allow it are proposed and described, with special emphasis on the measurement of Schedule Performance Index (abbreviated SPI) and Cost Performance Index (abbreviated CPI), proposed by the Earned Value Method (abbreviated EVM).

A special chapter is devoted to a detail view of the development of software tool that supports the governance of Scrum software development method. Initially, elemental requirements to support the conduct of the process of development are exposed, as well as requirements to support the performance measurement. A high-level architectural software design and decisions regarding the technologies, which have been used for implementation, are presented. An important part of the thesis is also further presented technical details of implementation. In this context the most important files containing executional software code are presented. Logically divided folder and file structure of the software project is also presented and adequately described. For better understanding the parts of the software code that are executed on one side on the server, and on the other on the client are also emphasized. In the scope of appropriately planned, built and set database model that is used for storing all entries for appropriate governance of Scrum software development method, the most important tables, fields and relationships between them are described.

The technical description is followed by the graphical user interface description that includes the essentional functions that the software tool supports as well as their practical usage and results given by these functions. This way the reader is acquainted with the practical usage of the tool.

The tool is further shown and proven in the case in practice and for completion improvements and additional functionalities that can be supported by the tool in possible further software development are proposed.

Keywords: Scrum, SPI, CPI, software tool, practical usage.

1 Uvod

1.1 Predstavitev problematike obravnavanega področja

V zadnjih petnajstih letih so nastale številne agilne metode za razvoj programske opreme, ki skušajo odpraviti togost in neprilagodljivost tradicionalnih metodologij, tako da omogočajo sprotno obvladovanje čedalje pogostejših sprememb v uporabniških zahtevah in pričakovanjih strank [11]. Nekatere bolj znane agilne metode so: Ekstremno programiranje (ang. *Extreme Programming*, skrajšano XP), Agilno modeliranje (ang. *Agile Modeling*), Metoda dinamičnega razvoja sistemov (ang. *Dynamic Systems Development Method*, skrajšano DSDM), Razvoj na podlagi lastnosti (ang. *Feature Driven Development*, skrajšano FDD), Agilen enoten proces (ang. *Agile Unified Process*, skrajšano AUP) in Scrum.

Ena najbolj razširjenih agilnih metod je metoda Scrum [8, 9]. Korenine metode izvirajo iz proizvodnih dejavnosti, na področju razvoja programske opreme pa je bila nekoliko kasneje sprejeta in ustrezno prilagojena [12].

Nedavna raziskava [13] je pokazala, da 35% organizacij, ki se ukvarjajo z razvojem programske opreme, pri svojem delu primarno uporablja agilne metode. Pri tem največ organizacij, in sicer kar 11%, uporablja metodo Scrum, kar jo postavlja na sam vrh najbolj uporabljanih metod za razvoj programske opreme.

Razlogi za takšno razširjenost ležijo predvsem v prednostih metode, ki so logična posledica osredotočenosti manjših projektnih skupin na krajše razvojne cikle. V literaturi lahko zasledimo naslednje prednosti [8]:

- izdelek (projekt) lahko razdelimo na zaporedje obvladljivih delov,
- projekt napreduje tudi, ko zahteve niso stabilne,
- vsi udeleženci imajo popoln vpogled v potek dela (ang. *everything is visible to everyone*),
- izboljša se komunikacija med člani razvojne skupine,
- člani razvojne skupine so zaslužni za uspeh v času razvoja in ob zaključku projekta,
- naročnik (uporabniki) dobiva(jo) novo funkcionalnost (ang. *increments*) v dogovorjenih časovnih intervalih,
- naročnik (uporabniki) lahko sproti preverja(jo), kako izdelana programska oprema v resnici deluje,

- metoda prispeva k izgradnji boljših odnosov med naročnikom (uporabniki) in razvijalci: poveča se medsebojno zaupanje in presek skupnega znanja,
- metoda prispeva k izgradnji pozitivnega vzdušja, v katerem vsi pričakujejo uspeh projekta.

Metoda Scrum predpostavlja, da je razvoj programske opreme nepredvidljiv in ga zato ni mogoče vnaprej natančno načrtovati. Iz tega razloga je predvidena uvedba empiričnega nadzora nad samim procesom, ki zagotavlja zadostno visoko stopnjo preglednosti in prilagodljivosti. To je doseženo skozi iterativni in inkrementalni razvojni proces, ki ga bomo bolj podrobno obdelali v poglavju *3 Metoda Scrum* [9].

Za uspešno izvedbo projekta metoda predvideva tudi empirično spremljanje razvojnega procesa. Predvidena je uporaba ene metrike, in sicer količine preostalega dela, ki se meri na dveh ravneh: na ravni iteracije v urah in na ravni celotnega projekta s številom točk nedokončanih zgodb. Na osnovi teh podatkov je s pomočjo ustreznih diagramov možno (ang. *burndown chart*) sproti spremljati, koliko dela je še ostalo do konca iteracije oziroma do konca projekta.

V zadnjih letih so raziskovalci in praktiki prepoznali potrebo po uvedbi dodatnih metrik, ki bodo omogočale boljši vpogled v proces razvoja programske opreme [4]. Kot eno izmed možnosti nekateri avtorji [10] predlagajo uporabo ključnih komponent metode prislužene vrednosti (ang. *Earned Value Method*, skrajšano EVM) [1]. Na tej podlagi je bil razvit model za spremljanje učinkovitosti razvojnega procesa po metodi Scrum [6, 7], ki poleg količine preostalega dela predvideva tudi merjenje terminskega indeksa SPI (ang. *Schedule Performance Index*) in stroškovnega indeksa CPI (ang. *Cost Performance Index*). Terminski indeks SPI predstavlja razmerje med prisluženo in planirano vrednostjo, stroškovni indeks CPI pa razmerje med prisluženo vrednostjo (merjeno v enotah izbrane valute) in dejanskimi stroški. Ciljna vrednost obeh indeksov je ena ali več.

Za učinkovito beleženje in analizo podatkov, ki so pomembni za spremljanje učinkovitosti razvojnega procesa, je pomembno, da imajo razvijalci na voljo ustrezno orodje za vodenje projektov po metodi Scrum.

1.2 Rezultati magistrskega dela

Rezultat magistrskega dela je izdelano programsko orodje, ki na eleganten način omogoča vodenje procesa razvoja programske opreme po metodi Scrum. V primerjavi s trenutno razpoložljivimi orodji razvito orodje dodatno vključuje podporo izbranim metrikam za

spremljanje učinkovitosti procesa razvoja, ki so opredeljene v [6, 7], ter možnost dinamičnega vnosa dodatnih, v tem trenutku še nedefiniranih metrik, brez posegov v programsko kodo. Pomemben vidik dela je tudi izbira med razpoložljivimi tehnologijami za implementacijo in tehnični detajli izdelave programske opreme.

Magistrsko delo je razdeljeno na sedem poglavij. Uvodu sledita splošna predstavitev agilnih metod (poglavje 2) in podrobnejša predstavitev metode Scrum (poglavje 3). V četrtem poglavju je opisan postopek merjenja učinkovitosti razvoja programske opreme, ki ga podpira v magistrski nalogi razvito orodje. Osrednji del naloge predstavljata peto in šesto poglavje. V petem poglavju je podrobno predstavljen pristop k izgradnji orodja (zajem osnovnih zahtev za izgradnjo orodja, izbira ustreznih tehnologij) in podani so najpomembnejši detajli implementacije orodja (sistemske nastavitve, podatkovni model ter direktorijska in datotečna struktura). V šestem poglavju je predstavljen grafični uporabniški vmesnik in najpomembnejše funkcije, ki jih orodje podpira, na koncu poglavja pa je prikazana uporaba orodja na preprostem primeru. Sedmo poglavje predstavlja sklep, v katerem so povzeti najpomembnejši rezultati magistrskega dela in predlogi za nadaljnji razvoj.

2 Agilne metode za razvoj programske opreme

2.1 Uvod

Agilne metode so v zadnjem času na področju razvoja programske opreme postale precej priljubljene. Korenine vlečejo iz t.i. »vitkega vodenja« (ang. *lean management*), ki se je pojavil na Japonskem v 80. letih prejšnjega stoletja. Te metode postavljajo načela za pridobitev uporabniških zahtev, načrtovanje in razvoj aplikacij ter informacijskih sistemov na način, ki vključuje končnega uporabnika v veliko večji meri, kot pa s tradicionalnimi metodami. Poleg tega agilne metode predlagajo nov, bolj fleksibilen način razmišljanja. Gre za bolj »sproščen« odnos do celotnega projekta, ki stremi k odpravljanju nepotrebne dokumentacije, zmanjševanju nesporazumov in povečevanja učinkovitosti projektov razvoja programske opreme [5].

Zaradi nezadovoljstva, ki ga je povzročil velik odstotek neuspešnih projektov na področju razvoja programske opreme, z uporabo takrat razpoložljive klasične (ang. *plan-driven*) razvojne metode, so se sredi devetdesetih let prejšnjega stoletja pojavile agilne metode in prakse kot poskus bolj formalno in izrecno zajeti čedalje višje stopnje sprememb v programskih zahtevah in pričakovanjih strank [11].

2.2 Nastanek, osnovna načela in priporočila agilnih metod

Februarja leta 2001 se je skupina sedemnajstih strokovnjakov na področju razvoja programske opreme sestala v mestu Snowbird (Utah, ZDA) z namenom najti rešitev za probleme obstoječih metod programskega inženirstva [5, 11]. Rezultat njihovega srečanja je bil Agilni Manifest (ang. *Agile Manifesto*), dokument ki opisuje temeljna določila agilnega razvoja programske opreme. Agilni Manifest poudarja štiri osnovne vrednote [25]:

1. Posamezniki in interakcije pred procesi in orodji
2. Delujoča programska oprema pred vseobsežno dokumentacijo
3. Sodelovanje s stranko pred pogodbenimi pogajanji
4. Odziv na spremembe pred togim sledenjem načrtom

Omenimo še, da je iz osnovnih načel bilo izpeljanih 12 temeljnih priporočil, kateremu naj bi sledile agilne metode za razvoj programske opreme [25]:

1. Naša najvišja prioriteta je zadovoljiti stranko z zgodnjim in nepretrganim izdajanjem vredne programske opreme.

2. Sprejemamo spremembe zahtev, celo v poznih fazah razvoja. Agilni procesi vprežejo tovrstne spremembe v prid konkurenčnosti naše stranke.
3. Delujočo programsko opremo izdajamo pogosto, znotraj obdobja nekaj tednov, do nekaj mesecev, s preferenco po krajšem časovnem okvirju.
4. Poslovneži in razvijalci morajo skozi celoten projekt dnevno sodelovati.
5. Projekte gradimo okrog motiviranih posameznikov. Omogočimo jim delovno okolje, nudimo podporo in jim zaupamo, da bodo svoje delo opravili.
6. Najboljša in najučinkovitejša metoda posredovanja informacij razvojni ekipi in znotraj ekipe same, je pogovor iz oči v oči.
7. Delujoča programska oprema je primarno merilo napredka.
8. Agilni procesi promovirajo trajnostni razvoj. Sponzorji, razvijalci in uporabniki morajo biti zmožni konstantnega tempa za nedoločen čas.
9. Nenehna težnja k tehnični odličnosti in k dobremu načrtovanju izboljša agilnost.
10. Preprostost -- umetnost zmanjševanja količine nepotrebnega dela -- je bistvena.
11. Najboljše arhitekture, zahteve in načrti izhajajo iz tistih ekip, ki so samoorganizirane.
12. V rednih časovnih razdobjih ekipa išče načine, kako postati učinkovitejša ob rednem prilagajanju svojega delovanja.

2.3 Osnovne značilnosti agilnih metod

Agilne metode so podmnožica iterativnih in evolucijskih metod, ki temeljijo na iterativnih izboljšavah in oportunističnih razvojnih procesih. Vsaka iteracija agilnih metod je samostojen mini-projekt, z aktivnostmi, ki obsegajo analizo zahtev, načrtovanje, izvajanje, preskušanje in sprejem s srani naročnika. Vsaka iteracija vodi k izdaji (lahko je tudi notranja izdaja), ki integrira vso programsko opremo razvojne skupine in predstavlja naraščajočo in razvijajočo podmnožico končne programske opreme. Agilne metode predlagajo/potrebujejo kratek čas razvoja iteracij, ker pridobitev povratnih informacij pred in po neki iteraciji lahko vodi do izpopolnitev in prilagoditev zahtev za naslednje iteracije. Naročnik raje sproti določa in prilagaja zahteve za naslednjo izdajo glede na opazovanje delovanja razvijajočega produkta, kot pa da »špekulira« preden se je projekt sploh začel. Pogosto določeni roki zmanjšujejo nihanja v razvojnem procesu in povečujejo predvidljivost ter učinkovitost. Vnaprej določena dolžina iteracij služi kot časovni okvir za celoten tim. Za vsako iteracijo je točno določen obseg dela oz. določitev zahtev, ki bodo v iteraciji izpolnjene. Pri tem se dolžina iteracije ne prilagaja tako, da ustreza časovnemu okviru za izpolnitev izbranih zahtev, temveč obratno: izbere se takšen obseg zahtev, da se prilega časovni dolžini iteracije. Ključna razlika med agilnimi metodami ter zgodnjimi iterativnimi metodami je vnaprej določena dolžina iteracije. Iteracije prejšnjih metod so bile lahko dolge od 3 do 6 mesecev. Z agilnimi metodami iteracije

varirajo med 1 in 4 tedni in namenoma ne presegajo 30 dni. Raziskave so pokazale, da krajše iteracije imajo nižjo kompleksnot ter posledično nižje tveganje, hkrati pa boljše povratne informacije, višjo produktivnost, učinkovitost in stopnjo uspešnosti [11].

3 Metoda Scrum

3.1 Uvod

Izraz Scrum izvira iz športa ragbi. Gre za skupino osmih oseb. Vsakdo v skupini deluje skupaj z vsemi ostalimi, zato da premaknejo žogo čimbolj na nasprotnikovo stran igrišča. Za tiste, ki poznajo pravila športa je slika jasna. Skupina deluje kot tesna, integrirana enota pri čemer vsak posamezen član skupine ima natančno določeno vlogo, celotna skupina pa je osredotočena na en sam cilj [8].

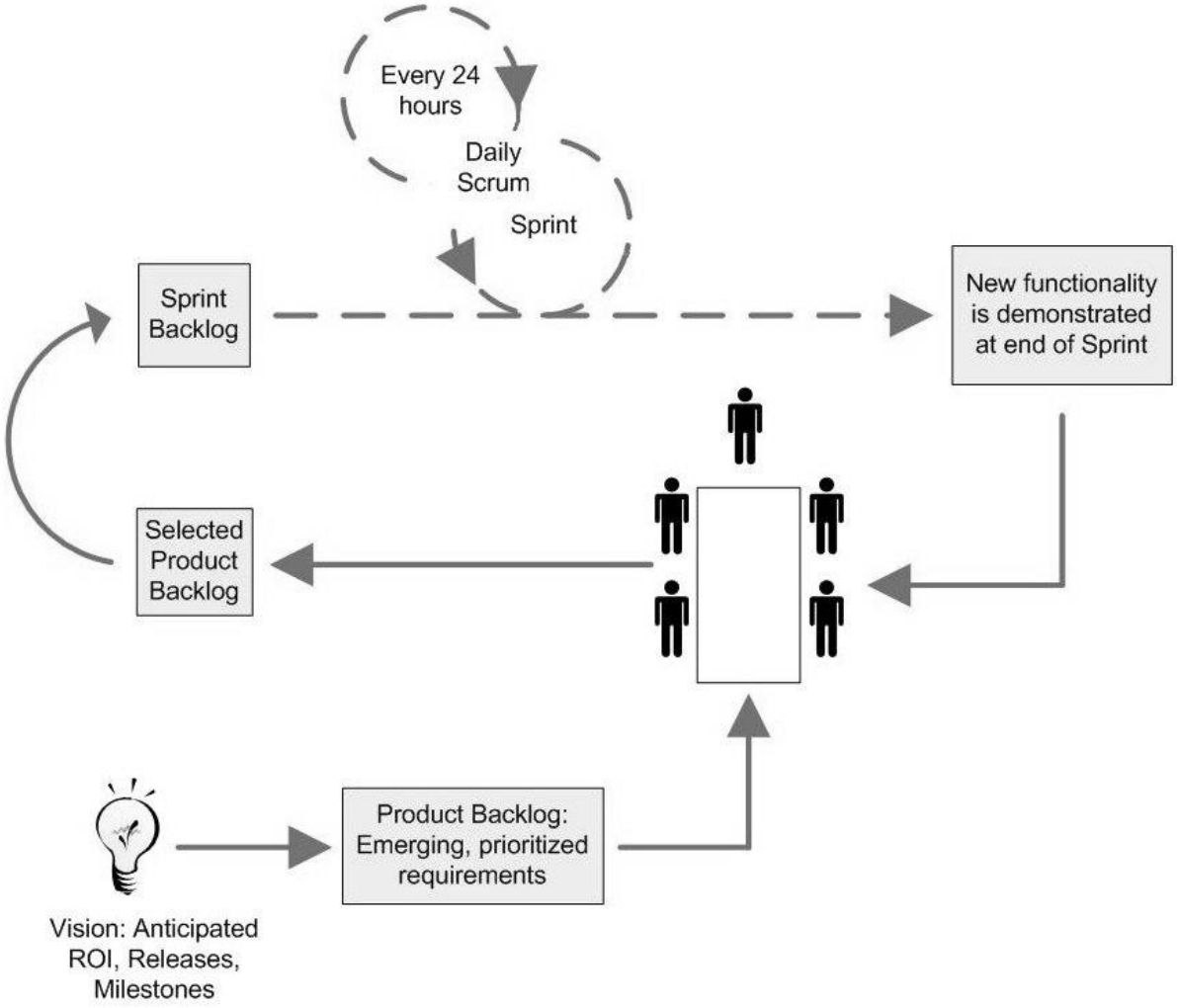
V skupinah, kjer govorimo o razvoju programske opreme, mora vsak član skupine dobro poznati svojo vlogo in naloge, ki jih mora izpolniti v okviru vsake iteracije razvojnega cikla. Celotna razvojna skupina mora biti osredotočena na dosego istega cilja. Prioritete zastavljenih nalog morajo biti jasne. Proces razvoja programske opreme po metodi Scrum omogoča postaviti in ohranjati osredotočenost razvojne skupine za dosego tega skupnega cilja [8].

V nadaljevanju poglavja bomo opisali postopek razvoja po metodi in bistvene vloge, ki nastopajo v procesu. Nekaj besed bo namenjenih pomembnim sestankom, ki so del samega procesa, ter izdelkom metode.

3.2 Opis procesa

Metoda Scrum (potek procesa je prikazan na sliki 1) temelji na seznamu zahtev (ang. *Product Backlog*), sestavljenem iz množice vseh aktivnih uporabniških zgodb (ang. *user stories*) [2]. Seznam zahtev vzdržuje produktni vodja (ang. *Product Owner*), ki je edina oseba, pooblaščena za spreminjanje prioritete posameznih zgodb. Razvoj poteka v več zaporednih iteracijah (ang. *Sprint*). Vsaka iteracija traja do 30 koledarskih dni in se začne s sestankom za načrtovanje Sprinta (ang. *Sprint Planning Meeting*), kjer se doseže dogovor o tem, katere uporabniške zgodbe iz celotnega seznama bodo realizirane v tej iteraciji. Razvojna skupina (ang. *Scrum Team*), ki je odgovorna za realizacijo, vsako zahtevo še naprej razbije na posamezne naloge, ki trajajo od 4 do največ 16 ur in sestavljajo seznam nalog (ang. *Sprint Backlog*). Razvojna skupina je pri svojem delu samostojna, deluje po načelu samoorganizacije in je kolektivno odgovorna za uspeh vsake iteracije in celotnega projekta. Skrbnik metodologije (ang. *ScrumMaster*) je odgovoren za pravilen potek razvojnega procesa, tako da se ta vklaplja v kulturo organizacije in prinaša pričakovane koristi, hkrati pa zagotavlja, da vsak član projektne skupine sledi praksam in pravilom metode Scrum. Skrbnik metodologije vsak dan vodi 15-minutni sestanek (ang. *Daily Scrum meeting*), kjer vsak član projektne

skupine odgovarja na tri osnovna vprašanja: »Kaj si naredil na projektu od zadnjega dnevnega sestanka?«, »Kaj boš naredil do naslednjega sestanka?« in »Ali imaš kakšne ovire?«. Skrbnik metodologije je prav tako odgovoren za reševanje ovir, ki so nastale med iteracijo, da zagotovi nemoten potek razvojnega procesa. Na koncu vsake iteracije razvojna skupina organizira sestanek (ang. *Sprint Review Meeting*), na katerem produktnemu vodji in zainteresiranim uporabnikom predstavi rezultate dela v tej iteraciji. Po tem sestanku in pred sestankom za načrtovanje naslednje iteracije skrbnik metodologije organizira še sestanek z namenom zagotavljanja stalnega izboljševanja toka procesa (ang. *Sprint Retrospective Meeting*) [6, 7, 9].



Slika 1: Potek procesa metode Scrum [6]

3.3 Sprint

Vsak Sprint je samostojen »mini-projekt«, sestavljen iz aktivnosti, kot so ukvarjanje z zahtevami, načrtovanje, programiranje in preizkušanje [11]. Delo v okviru Sprinta se začne, ko se doseže dogovor o tem, katere uporabniške zgodbe iz celotnega seznama bodo realizirane v tej iteraciji ter, ko se določijo prioritete in odgovornosti za posamezne naloge znotraj iteracije (to je po sestanku za načrtovanje). Sprint traja tipično do 30 koledarskih dni (nekateri viri [8, 11] navajajo od enega do štirih tednov). Bistveno za Sprint je izdelava vidnega, uporabnega ne nujno končnega produkta, ki podpira eno ali več uporabniških interakcij s sistemom. Ključna ideja leži v dostavi delujoče programske kode, ki predstavlja naslednji inkrement funkcionalnosti produkta hkrati pa temelji na prejšnjem. Cilj je zaključiti vse naloge iteracije do načrtovanega datuma zaključka Sprinta. Ker gre za časovno natančno načrtovan potek, datuma zaključka Sprinta ni moč spreminjati, prav tako pa nihče izven razvojne skupine ne sme spreminjati dogovorjene vsebina Sprinta. Omenimo, da po drugi strani razvojna skupina lahko zmanjša obseg funkcionalnosti, ki jo bo implementirala med samim Sprintom, vendar le v primeru, da ji za prvotno načrtovan obseg zmanjka časa za implementacijo [8].

3.4 Vloge v procesu

Vloge, ki nastopajo v procesu metode Scrum so sledeče [6]: produktni vodja, razvojna skupina in skrbnik metodologije.

Produktni vodja je odgovoren za zagotavljanje ciljev vseh interesnih skupin projekta in končnega produkta. Predstavlja glavno vez med razvojno skupino in naročnikom. Vzdržuje seznam uporabniških zgodb ter jim določa ustrezne priorite glede na dodano vrednost, ki jo izpolnitev posamezne uporabniške zgodbe prinaša naročniku. Ker je produktni vodja udeležen v procesu, mora bti članom razvojne skupine praktično vedno na razpolago za morebitne dodatne informacije v zvezi s samimi uporabniškimi zgodbami. Produktni vodja je prav tako odgovoren za izvršitev sprejemnih testov (ang. *acceptance tests*).

Razvojna skupina je odgovorna za razvoj / implementacijo funkcionalnosti. Pri svojem delu je popolnoma samostojna, deluje po načelu samoorganizacije in je kolektivno odgovorna za pretvorbo seznama uporabniških zgodb in pripadajočih nalog v inkrement funkcionalnosti produkta oziroma dostavo delujoče programske kode. Člani razvojne skupine so kolektivno odgovorni za uspeh vsakega posameznega Sprinta in projekta v celoti.

Skrbnik metodologije zapolnjuje vlogo, ki jo ponavadi zaseda vodja projekta, vendar pa se od njega nekoliko razlikuje. V prvi vrsti je odgovoren za vodenje procesa po metodi Scrum in ne za definiranje ali celo delegiranje dela. Skrbeti mora, da vsak član projektne skupine sledi praksam in pravilom metode Scrum. Pomembno je še omeniti, da je njegova naloga tudi odstranitev kakršnihkoli ovir (ang. *impediments*) v samem procesu zato, da razvojna skupina dela čimbolj učinkovito.

3.5 Sestanki

3.5.1 Sestanek za načrtovanje Sprinta

Pred začetkom dela na Sprintu se organizira sestanek za načrtovanje Sprinta (ang. *Sprint Planning Meeting*), ki ponavadi traja en delovni dan in je sestavljen iz dveh vsebinsko različnih delov. V prvem delu produktni vodja članom razvojne skupine predstavi uporabniške zgodbe in zastavljene prioritete. Člani razvojne skupine med predstavitvijo aktivno sodelujejo, saj jim mora biti popolnoma jasen vsak detajl posamezne uporabniške zgodbe. Uporabniške zgodbe se nato časovno ocenijo ponavadi s številom točk, ang. *user story points* [2] in se izberejo tiste, ki bodo implementirane v tem Sprintu. Pri tem seštevek točk izbranih zgodb ne sme presegati ocenjene hitrosti dela razvojne skupine (ang. *velocity* [2]). V drugem delu sestanka se izbrane uporabniške zgodbe razbijejo na posamezne naloge (vsaka naj bi trajala približno 4 do 16 ur) in se na koncu dogovori odgovornost za izpolnitev vsake.

3.5.2 Vsakodnevni sestanki

Metoda Scrum predvideva kratke vsakodnevne 15-30 minut trajajoče sestanke (ang. *Daily Scrum meeting*), ki jih vodi skrbnik metodologije. Kot že prej nakazano, vsak član razvojne skupine mora odgovoriti na 3 osnovna vprašanja: »Kaj si naredil na projektu od zadnjega dnevnega sestanka?«, »Kaj boš naredil do naslednjega sestanka?« in »Ali imaš kakšne ovire?«. Kljub temu, da so vsakodnevni sestanki precej kratki, je vseeno zagotovljeno, da se omenjajo tudi ovire, ki nastanejo med samim razvojem, vendar pa ni dovolj časa za njihovo obravnavo. Odgovori posameznega člana razvojen skupine morajo biti kratki in jedrnati. Vsake daljše razprave je zato potrebno prestaviti na kakšen drug kasnejši sestanek, ki naj bi vključeval le tiste člane, ki se jih izpostavljeni problem tiče. Cilji vsakodnevnih sestankov so sledeči [8]:

- Osredotočenost in prizadevanje članov razvojne skupine za pravočasno izpolnitev zadanih nalog.
- Redno obveščanje razvojne skupine o napredku razvoja in morebitnih ovirah.

- Reševanje ovir kolikor hitro je to mogoče.
- Spremljanje napredka razvoja programske opreme in pravočasna dostava inkrementa funkcionalnosti.
- Obravnavanje in zmanjševanje tveganja celotnega projekta.

3.5.3 Sestanek za pregled rezultatov Sprinta

Po koncu dela na Sprintu se organizira sestanek za pregled rezultatov Sprinta (ang. *Sprint Review Meeting*). Na tem sestanku razvojna skupina in skrbnik metodologije predstavita novo funkcionalnost, ki je bila razvita v pravkar končanem Sprintu (oziroma delujoč inkrement produkta) produktnemu vodji in vsem zainteresiranim deležnikom. Pri tem se vsi udeleženci sestanka (pod vodstvom produktnega vodje) osredotočajo na oceno dostavljenega inkrementa produkta (npr. z izvajanjem sprejemnih testov, ang. *acceptance tests*) in se odločijo o naslednjih aktivnostih. Med sestankom je možno predlagati nove zahteve in celo spremeniti smer razvoja produkta [9].

3.5.4 Sestanek za oceno dela v preteklem Sprintu

Po sestanku za pregled rezultatov Sprinta, skrbnik metodologije organizira še en sestanek katerega primarni namen je izboljševanja toka procesa (ang. *Sprint Retrospective Meeting*) [9]. Za razliko od sestanka za pregled Sprinta, se na tem sestanku skrbnik metodologije in vsi člani razvojne skupine osredotočajo predvsem na potek procesa, uporabljene prakse in pravila metode Scrum [6]. Razprava omogoča, da se naslovijo in obravnavajo pozitivne in negativne zadeve v zvezi s samim procesom, kar omogoča neprekinjeno izboljševanje in optimiziranje procesa, posledično pa povečevanje učinkovitost razvojne skupine in verjetnost uspešnosti celotnega projekta [9].

3.6 Izdelki metode Scrum

Metoda Scrum predvideva vzdrževanje dveh že omenjenih osnovnih seznamov skozi celoten razvojni cikel. To sta seznam zahtev ali uporabniških zgodb (ang. *Product Backlog*) in seznam nalog tekočega Sprinta (ang. *Sprint Backlog*). Gre za dva dinamična in med seboj odvisna dokumenta, ki se sproti spreminjata in v vsakem trenutku odražata trenutno stanje razvoja programske opreme. Kot že prej omenjeno, vsaka uporabniška zgodba se razbije na posamezne naloge. Izpolnitev posamezne zgodbe je dosežena, ko se vse naloge izpolnijo in se uspešno izvedejo sprejemni testi. Na prvi pogled je vzdrževanje obeh seznamov relativno enostavno. V nadaljevanju bomo videli, da temu le ni tako, posebej s stališča implementacije orodja, ki to podpira. Problem se pojavi, kadar določene uporabniške zgodbe ni bilo možno

zaključiti v tekočem Sprintu. Takrat je omenjeno uporabniško zgodbo in njene nedokončane naloge potrebno prestaviti v naslednji Sprint, pri tem pa ohranjati informacije iz tekočega. V magistrski nalogi bo omenjeni problem detajlno obrazložen in predstavljena bo tudi rešitev, ki jo izdelano orodje ponuja.

4 Merjenje učinkovitosti razvoja programske opreme po metodi Scrum

4.1 Uvod

Scrum je ena izmed najbolj razširjenih agilnih metodologij, ki se osredotoča predvsem na upravljanje projektov razvoja programske opreme. V literaturi je v zadnjih letih moč zaslediti precej uspešnih izvedb projektov po metodi Scrum. Izkušnje so pokazale, da sta se s sprejetjem agilnih metod izboljšali upravljanje razvojnega procesa in odnos s strankami. Zmanjšala naj bi se tudi količina nadur in hkrati povečalo zadovoljstvo strank [6].

Kot že prej omenjeno, raziskovalci in praktiki so v zadnjih letih prepoznali potrebo po uvedbi dodatnih metrik, ki bodo omogočale boljši vpogled v proces razvoja programske opreme [4]. Z dodatnimi metrikami oziroma z uporabo bolj natančnih pristopov merjenja je možno precej dobro pokazati, kako učinkovit je razvoj programske opreme. To omogoča kritično oceno negativnih kot pozitivnih reči, ki so se oziroma se pojavljajo med razvojem, pri čemer se ustvari temeljna podlaga za izboljšanje procesa razvoja s pravočasnim ukrepanjem tako med razvojem kot tudi v prihodnje.

4.2 Opredelitev ciljnih metrik in umestitev v ogrodje Scrum

Najprej je potrebno definirati metrike, ki bodo omogočile izvedbo ustreznega pristopa merjenja učinkovitosti. Na podlagi ključnih komponent metode prislužene vrednosti je bil razvit model za spremljanje učinkovitosti razvojnega procesa po metodi Scrum [6, 7].

Opredelitev metrik temelji predvsem na upoštevanju stališč različnih interesnih skupin, ki sodelujejo v procesu razvoja [6]: vodstvo IT, člani razvojne skupine in naročnik oziroma stranka.

Medtem ko je za člane razvojne skupine ter naročnika pomembna predvsem ocena njihovega zadovoljstva med razvojem, se vodstvo IT v glavnem ukvarja s tradicionalnimi vidiki uspešnosti razvoja programske opreme glede na čas, stroške in kakovost. Stremi k uresničevanju naslednjih ciljev [6]:

- Cilj 1: Pravočasno obveščanje o uspešnosti projektov s poudarkom na projektih, ki se odmikajo od prvotno začrtanih postavk glede časa zaključka in proračuna projekta.
- Cilj 2: Izboljšanje kakovosti.

V magistrski nalogi se bom osredotočil predvsem na merjenje učinkovitosti razvoja programske opreme, ki je usmerjena le na eno ciljno skupino: vodstvo IT ter uresničevanje že prej omenjenega cilja 1.

Sprva je metoda Scrum imela le eno osnovno metriko: količino preostalega dela, ki je potrebno za dokončanje zahteve in naloge. Na nivoju seznama zahtev se ocena o količini preostalega dela za vsako zahtevo posebej zbere na začetku vsakega Sprinta. Na nivoju seznama nalog pa se omenjena metrika zbira vsak dan za vsako nalogo posebej. S pravilnimi vnosi metrike je tako na obeh nivojih možno razviti graf (ang. *Burndown chart*), ki prikazuje preostalo delo skozi čas na dnevni bazi in na nivoju Sprinta [7].

Da bi lahko merili cilj 1 vodstva IT, predlagana in definirana je dodatna metrika [7]: količina porabljenega dela, ki jo lahko zajamemo na vsakodnevnih Daily Scrum sestankih istočasno s količino preostalega dela.

4.3 Terminski in stroškovni indeks

S pomočjo definiranih metrik (količine preostalega in količine porabljenega dela) je možno za vsak dan v Sprintu sproti izračunati terminski indeks SPI in stroškovni indeks CPI.

Terminski indeks SPI [6,7] predstavlja razmerje med prisluženo vrednostjo (tj. dejansko vrednostjo vseh nalog, ki smo jih zaključili do tistega dne) in načrtovano vrednostjo (tj. vrednostjo vseh nalog v Sprintu, ki bi – upoštevajoč enakomeren tempo razvoja – morale biti dokončane do tistega dne). Ciljna vrednost za SPI je 1. SPI večji kot 1 pomeni, da se projekt odvija hitreje kot je načrtovano, vrednost manjša kot 1 pa pomeni obratno: glede na načrt projekt zamuja. Na splošno lahko rečemo, da je vrednost SPI boljša, če je večja kot 1, vendar pa se je potrebno zavedati, da vrednost, ki je veliko večja od 1, kaže na to, da smo Sprint slabo načrtovali.

Stroškovni indeks CPI [6,7] predstavlja razmerje med prisluženo vrednostjo (merjeno v enotah izbrane valute) in dejasnkimi stroški. Ciljna vrednost CPI je 1. CPI večji kot 1 pomeni, da so stroški dela za zaključek manjši, kot je bilo načrtovano, vrednost manjša kot 1 pa pomeni, da so stroški dela presegli načrtovano vrednost.

Celoten model izračuna predpostavlja, da je količina nalog, ki se morajo zaključiti do neke časovne točke znotraj Sprinta sorazmerna času, ki je pretekel od začetka Sprinta. S količino preostalega in porabljenega dela je možno natančno izračunati delež dejansko prislužene vrednosti $ER_{d,j}$ za vsako nalogo j iz seznama nalog na dan d Sprinta. Izračunamo ga kot

razmerje med količino že porabljenega dela in vsoto količine porabljenega in preostalega dela, potrebnega za dokončanje naloge, kot je razvidno iz formule (1) [6,7]:

$$ER_{d,j} = \frac{\sum_{i-1}^{d-1} WS_{i,j}}{\sum_{i-1}^{d-1} WS_{i,j} + WR_{d,j}} \quad (1)$$

$WS_{i,j}$ pomeni količino porabljenega dela za nalogo *j* na dan *i, i=1,2,...,d-1*, $WS_{i,j}$ pa pomeni količino preostalega dela za nalogo *j* na dan *d*. Z uporabo formule (1) pridemo do izračuna SPI na dan *d*, kot je razvidno iz formule (2) [6,7]:

$$SPI_d = \frac{\sum_{j-1}^{n} ER_{d,j} * WR_{init,j}}{\sum_{j-1}^{n} WR_{init,j}} * \frac{SL}{DE} \quad (2)$$

$WR_{init,j}$ pomeni prvotno oceno količine preostalega dela za nalogo *j*, *SL* dolžino Sprinta (ang. *Sprint Length*), DE (ang. *Days Elapsed*) pa koliko dni v Sprintu je že preteklo. Formula za izračun SPI dovoljuje merjenje prislužene vrednosti v katerikoli enoti (v magistrski nalogi je za količino porabljenega in preostalega dela uporabljena enota »ura«, kot bo razvidno v nadaljevanju), medtem ko pa izračun CPI zahteva izražanje prislužene vrednosti in dejanskih stroškov dela v izbrani valuti. Z uporabo metrike količine preostalega dela, lahko izračunamo dejanski strošek dela kot zmnožek med številom porabljenih ur in stroškom ene inženirske ure (ang, *Cost of Engineering Hour*, skrajšano CEH) za vse naloge v seznamu. Podobno lahko izračunamo prisluženo vrednost kot zmnožek med »prisluženimi urami« in CEH_j, ki predstavlja strošek inženirske ure za nalogo j. Na koncu pridemo do izračuna CPI, ki predstavlja razmerje med prisluženo vrednostjo in dejanskimi stroški, kot je razvidno iz formule (3) [6, 7]:

$$CPI_d = \frac{\sum_{j-1}^{n} ER_{d,j} * WR_{init,j} * CEH_j}{\sum_{i-1}^{DE} \sum_{j-1}^{n} WS_{i,j} * CEH_j} \quad (3)$$

5 Implementacija orodja za vodenje in merjenje učinkovitosti razvoja po metodi Scrum

Trenutno je na trgu veliko razpoložljivih orodij, ki podpirajo vodenje programske opreme po metodi Scrum. Razlog je predvsem v razširjenosti metode, ki je predstavljena bolj detajlno v poglavju 3. Povsem logično je, da vsa razpoložljiva orodja stremijo k istem cilju, a vendar ed njimi obstajajo določene razlike, zaradi katerih so ena orodja bolj uporabna in druga manj. Potrebno je omeniti, da programsko orodje, ki podpira veliko funkcionalnosti, ni nujno tudi boljše. Preprostejša orodja z manj podprtimi funkcionalnostmi so običajno lažja za uporabo, saj pred samo uporabo ponavadi ne zahtevajo dodatnega časa za izobraževanje, branje dokumentacije in podobnih časovno zamudnih opravil.

Ideja o razvoju programskega orodja, ki bi omogočalo podporo vodenju procesa razvoja programske opreme po metodi Scrum, izvira predvsem iz podpore ključnih elementov merjenja učinkovitosti razvoja programske opreme - terminskega (SPI) in stroškovnega (CPI) indeksa, ter možnost dinamičnega vnosa dodatnih, v tem trenutku še nedefiniranih metrik, brez posegov v programsko kodo. Glede na trenutno razpoložljive informacije v sicer omejenem obsegu (pregledana literatura in spletni viri) preračunov obeh indeksov ni moč najti v sklopu razpoložljivih funkcij kakšnega drugega orodja na trgu. Drugače rečeno: tudi če bi obstajalo orodje, ki bi omenjene funkcionalnosti podpiralo, lahko s precejšnjo gotovostjo trdimo, da le-to ne bi bilo prosto dostopno.

Pred začetkom razvoja je skorajda obvezno postaviti osnovna in ključna vodila, ki jim je potrebno slediti skozi potek celotnega razvojnega cikla (celo pred zajetjem osnovnih funkcionalnih zahtev, ki naj bi jih to orodje izpolnjevalo). Prvoten namen vodil je, da razvijalcu na enostaven način povedo, da orodje ne sme izpolnjevati zgolj golo podane funkcionalne zahteve, temveč je obenem potrebno paziti, na kakšen način se te zahteve izpolnijo. Nekoliko slabše zasnovani uporabniški vmesniki, (s katerimi določenih podatkov ni mogoče urejati enostavno, kot bi si uporabnik to morda želel) lahko tako orodje denimo postavijo v nezavidljiv položaj, ne glede na to, koliko in katere funkcionalnosti podpira. Na osnovi vodil lahko ustvarimo tudi nove funkcionalne zahteve, ki prispevajo k izboljšanju kakovosti programskega orodja.

Med razvojem orodja je tako bilo upoštevano sledeče:

- čim krajša poraba časa s strani uporabnika glede uporabe orodja in seznanjenosti s podprtimi funkcionalnostmi,
- preprostost in intuitivnost uporabe vseh podprtih funkcij,

- izločitev nepotrebnih korakov za izvedbo določene akcije s strani uporabnika,
- podpora funkcionalnostim, ki razvito orodje vidno loči od orodij, ki so trenutno razpoložljiva na trgu (merjenje učinkovitosti razvoja programske opreme) na enostaven in intuitiven način.

Načrtovanje in implementacija orodja za podporo metodi Scrum je precej kompleksna naloga in zahteva sledenje nekaterim elementarnim korakom pred dejanskim »pisanjem kode« in po njem. Osnovni in izvedeni koraki razvoja orodja so bili sledeči:

- zajetje osnovnih zahtev za izdelavo orodja,
- visokonivojski arhitekturni načrt orodja,
- preučitev in izbira ustreznih tehnologij na strani odjemalca in strežnika kot tudi na strani shranjevanja podatkov,
- sistemske nastavitve in implementacija,
- preizkušanje programskega orodja na primeru.

V nadaljevanju bodo vsi našteti koraki detajlno predstavljeni.

5.1 Zajem zahtev za izdelavo orodja

V koraku zajema zahtev, katerim razvito orodje zadošča, je bilo potrebno odgovoriti na sledeča ključna vprašanja:

- Kateri akterji nastopajo v procesu oz. kdo so uporabniki?
- Kateri elementi še nastopajo v procesu?
- Katere podatke je potrebno urejati in vzdrževati?
- Kateri so izhodni podatki?

Ob dobrem poznavanju metode Scrum so odgovori na omenjena vprašanja relativno enostavni. Akterji, ki nastopajo v procesu, so produktni vodja, skrbnik metodologije in razvijalci. Ostali bistveni elementi procesa so produkt, ki ga razvijamo (razdeljen na izdaje in iteracije), metrike in nazadnje ovire. Izhodni podatki so seznam uporabniških zgodb, seznam nalog ter ključna indeksa merjenja učinkovitosti: SPI in CPI.

OSNOVNE FUNKCIONALNE ZAHTEVE
Vzdrževanje podatkov o projektih (vnos/posodabljanje/brisanje)
Vzdrževanje podatkov o izdajah (vnos/posodabljanje/brisanje)
Vzdrževanje podatkov o iteracijah (vnos/posodabljanje/brisanje)
Vzdrževanje podatkov o uporabnikih (vnos/posodabljanje/brisanje)
Vzdrževanje podatkov o uporabniških zgodbah (vnos/posodabljanje/brisanje)
Vzdrževanje podatkov o nalogah (vnos/posodabljanje/brisanje)
Vzdrževanje tabele metrik (vnos/posodabljanje/brisanje)
Vzdrževanje podatkov o ovirah (vnos/posodabljanje/brisanje)
Zajem metrik na začetku vsakega cikla (ang. *Sprint planning meeting*)
Zajem metrik na vsakodnevnih sestankih (ang. *Daily Scrum meeting*)
Zajem metrik na sestankih na koncu vsake iteracije (ang. *Sprint review meeting)*
Zajem metrik na sestankih zagotavljanja stalnega izboljševanja toka procesa (ang. *Sprint retrospective meeting)*
Izpis seznama uporabniških zgodb (ang. *Product Backlog)*
Izpis seznama nalog (ang. *Sprint Backlog)*
Izračun in izpis terminskega (SPI) in stroškovnega (CPI) indeksa

Tabela 1: Ključne funkcionalne zahteve programskega orodja

V tabeli 1 so zajete vse osnovne funkcionalne zahteve, ki jih orodje mora izpolnjevati. Razvidno je, da orodje mora omogočati vnos, posodabljanje in brisanje vseh akterjev procesa ter vseh ostalih elementov, ki nastopajo v procesu. Redno zajemanje metrik je pri tem potrebno še posebej poudariti. Vpeljemo lahko različne metrike, ki se nanašajo na različne elemente metode Scrum. Vsaka metrika je tako vezana na določen element in za vsako metriko je točno določeno, na katerem sestanku se bo zajemala – vsakodnevnih, sestankih za načrtovanje Sprinta, sestankih na koncu iteracij in/ali sestankih zagotavljanja stalnega izboljševanja toka procesa.

V naštetih zahtevah niso razvidni detajli, ki se skrivajo za vsako posamezno zahtevo. Tako denimo ni razvidno, kateri podatki so sestavni del projektov, iteracij, ovir itn. Prav tako ni jasno, kako so elementi procesa povezani med seboj - ena izdaja lahko vsebuje več iteracij, ena uporabniška zgodba lahko pripada več iteracijam itn. V zahtevah ni opredeljeno, katere metrike je potrebno vzdrževati, saj je predviden dinamičen vnos poljubnih metrik brez poseganja v programsko kodo. V zahtevah za izpis seznama nalog, terminskega in stroškovnega indeksa pa se skrivata dve osnovni metriki, ki ju je potrebno programsko podpreti: število porabljenih ur na nalogi in število preostalih ur za dokončanje naloge.

Izpostavljeni detajli ne sodijo v osnovne zahteve in se bodo upoštevali med samim razvojem. Razlog je predvsem v pristopu, ki je bil uporabljen za razvoj omenjenega orodja. Ta sledi osnovnim agilnim principom in zato zahteve na začetku razvoja niso detajlno razdelane. Z detajli se bo potrebno ukvarjati med samim postavljanjem podatkovnega modela in pisanjem programske kode. V tem trenutku je bistvenega pomena, kako in na kakšen način je zajete osnovne zahteve mogoče najučinkoviteje izpolniti z upoštevanjem osnovnih vodil med razvojem.

5.2 Pristop k izgradnji orodja

V visoko razvitih informacijskih družbah sta danes izjemnega pomena ažurnost in dostopnost podatkov. Ključnega pomena predstavljata porazdeljenost in povezljivost aplikacij. Visoka stopnja porazdeljenosti nam omogoča dostop do več podatkovnih virov, večjo dosegljivost aplikacij in učinkovito uporabo sistemskih virov. Visoka stopnja povezljivosti istočasno omogoča medsebojno sodelovanje aplikacij in močno poveča pretok podatkov in informacij [14].

V grobem aplikacije ločimo na spletne in namizne. Spletne aplikacije imajo veliko prednosti pred običajnimi namiznimi aplikacijami. Omogočajo delo na daljavo, dostop do oddaljenih ter porazdeljenih virov, dokaj enostavno povezljivost, enostavnejše vzdrževanje itn. Slabost

spletnih aplikacij se kaže predvsem v varnosti, za katero je še posebej potrebno poskrbeti, in nekoliko manjši hitrosti obdelave podatkov v primerjavi z namiznimi aplikacijami. Spletne aplikacije so postale mogoče šele s tehnologijami dinamičnih spletnih strani kot na primer: skupni prehodni vmesnik (ang. *Common Gateway Interface*, skrajšano CGI), programski vmesnik spletnega strežnika (ang. *Internet Server Application Programming Interface*, skrajšano ISAPI), programski vmesnik spletnega strežnika proizvajalca Netscape (ang. *Netscape Server Application Programming Interface*, skrajšano NSAPI), aktivne strežniške strani (ang. *Active Server Pages*, skrajšano ASP), PHP (skrajšano od ang. *PHP: Hypertext Preprocessor*), strežniške strani pisane v jeziku Java (ang. *Java Server Pages*, skrajšano JSP) [14]. Te tehnologije za vsako odjemalčevo zahtevo ustvarijo novo stran, ki se nato vrne odjemalcu kot odgovor. To povzroča pretok dodatne in nepotrebne količine podatkov od strežnika do odjemalca, za vsako zahtevo pa je odjemalcu potrebno osvežiti stran, kljub temu, da je sprememba celotne strani relativno majhna.

Razvoj spletnih aplikacij se pri tem ni ustavil. Le nekaj let nazaj se je pojavil nov arhitekturni pristop asinhrona JavaScript in XML (ang. *Asynchronous JavaScript and XML*, skrajšano AJAX), ki predpisuje drugačen koncept. Gre za način izmenjave podatkov med odjemalcem in strežnikom, pri čemer strežnik za vsako zahtevo ne pošilja vsebine celotne strani, temveč le tistih podatkov, ki so bili zahtevani. To na odjemalčevi strani pomeni, da osveževanje celotne strani ni potrebno, razen tistih podatkov, ki so bili zahtevani.

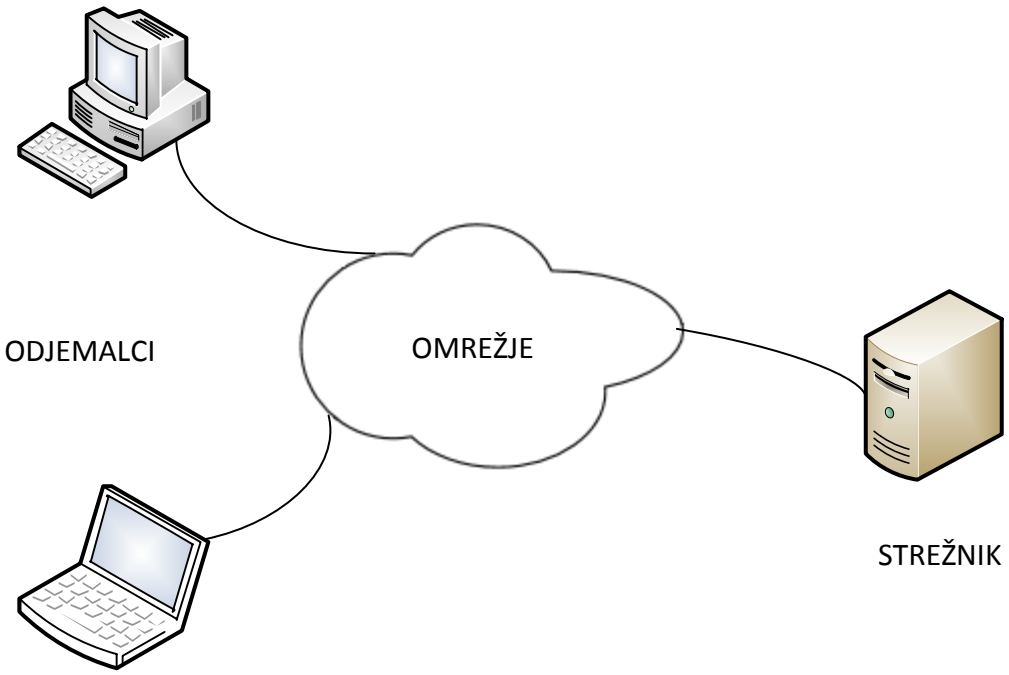
Pomembno je omeniti, da so v preteklosti spletne aplikacije implementirali s tehnologijo CGI. Ta je v osnovi povzročala visoke obremenitve strežnika, saj se je za vsako zahtevo na strežniku ustvaril nov proces. Novejše tehnologije (na primer ASP, PHP, JSP) medtem omogočajo izvajanje v nitih, ki se lahko predpomnijo in tako močno razbremenijo strežnik.

Iz naštetih razlogov je odločitev o izbiri vrste aplikacije za podporo vodenju razvoja programske opreme po metodi Scrum povsem logična. Ker je odločitev spletna aplikacija, bo v nadaljevanju potrebno določiti arhitekturo spletne aplikacije in izbrati ustrezne tehnologije za implementacijo.

5.2.1 Arhitekture spletnih aplikacij

Spletne aplikacije v osnovi temeljijo na arhitekturi odjemalec/strežnik (prikazano na sliki 2). Osnovna značilnost te arhitekture je delitev procesiranja in podatkov med enim ali več odjemalčevimi računalniki, ki izvajajo aplikacijo, in strežnikom, ki nudi storitve vsakemu izmed odjemalcev. Računalniki so lahko med seboj povezani v omrežje, pri tem pa sta strojna in programska oprema odjemalca in strežnika lahko tudi različni. Komunikacija med

odjemalcem in strežnikom poteka na osnovi zahtev in odgovorov. Tako strežnik ne pošilja nikakršnih podatkov brez zahteve s strani odjemalca.



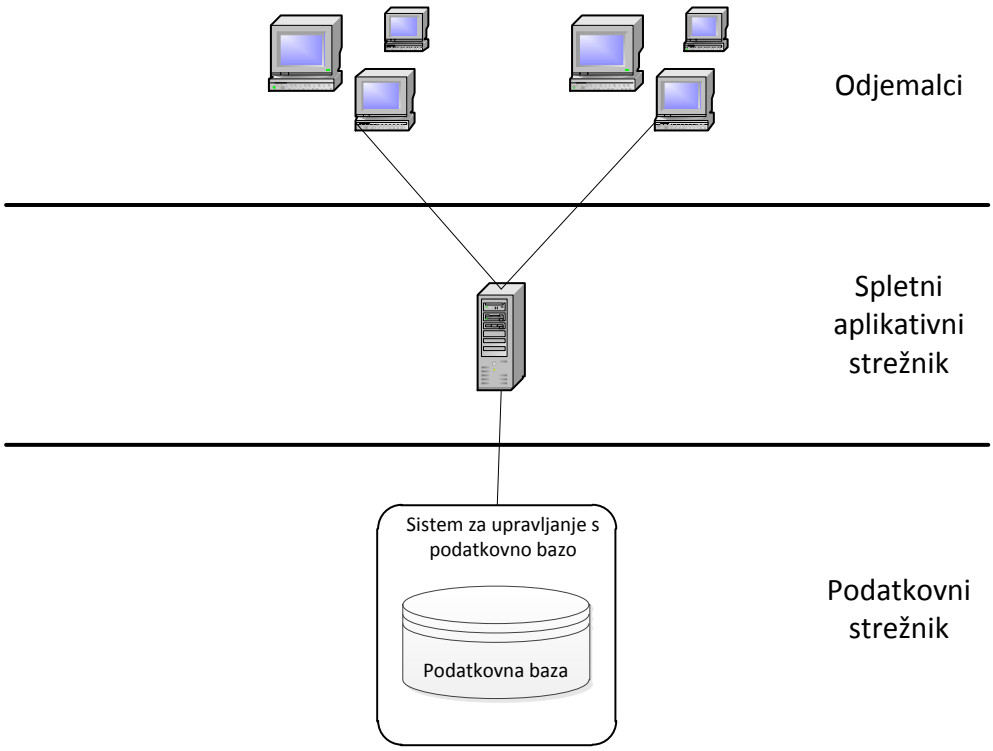
Slika 2: Arhitektura odjemalec/strežnik

Spletne aplikacije so razdeljene na več manjših delov, ki jim rečemo nivoji. Čeprav nekateri viri, dajejo posebno pozornost fizični lokaciji posameznih nivojev, se bomo pri opredelitvi tipa arhitekture nivojev osredotočali predvsem na vlogo, ki jo posamezni nivo mora opravljati ter na kakšen način ga je možno implementirati. Pri tem fizični lokaciji ne bomo posvečali veliko pozornosti.

Vsak posamezen nivo ima svojo vlogo, ki jo opravlja. Tako ločimo med dvonivojsko, trinivojsko in večnivojsko arhitekturo spletnih aplikacij.

Danes je večina spletnih aplikacij tri- ali večnivojskih. Med dvonivojske spletne aplikacije uvrščamo predvsem statične spletne strani. Odjemalcu (spletni brskalnik) streže spletni aplikativni strežnik, ki kot odgovor na zahteve pošilja vnaprej shranjene spletne strani.

V trinivojski arhitekturi spletnih aplikacij je odjemalec prvi nivo, spletni aplikativni strežnik drugi in podatkovni strežnik tretji nivo. Odjemalec skrbi predvsem za grafični vmesnik in deloma izvaja določena preprostejša poslovna pravila. Spletni aplikativni strežnik skrbi za izvajanje večine poslovne logike, podatkovni strežnik pa za streženje podatkov in lahko tudi deloma izvaja poslovna pravila skozi t.i. bazne procedure (ang. *stored procedures*). Na sliki 3 je ponazorjena trinivojska arhitektura spletnih aplikacij.



Slika 3: Trinivojska arhitektura spletnih aplikacij

Večnivojska arhitektura dodatno razdeli nivo, na katerem se nahaja spletni aplikativni strežnik: to poenostavi porazdelitev aplikacij in virov, lahko se uporabijo različni varnostni mehanizmi, zagotovi boljša razpoložljivost ob izrednih dogodkih itn.

Z uporabo dvonivojske arhitekture nikakor moremo doseči funkcionalnosti, ki so bile izpostavljene, ker nujno potrebujemo podatkovni strežnik. Po drugi strani pa se večnivojska arhitektura spletnih aplikacij osredotoča predvsem na zadeve, ki niso predmet pričujoče

magistrske naloge. Logična in pravilna izbira za rešitev problematike implementacije orodja za podporo vodenju razvoja programske opreme po metodi Scrum je trinivojska arhitektura.

V svoji doktorski disertaciji [3] je Roy Fielding leta 2000 predstavil arhitekturni stil omrežnih sistemov, predstavitveni prenos stanja (ang. *Representational State Transfer*, skrajšano REST). Temelji na trditvi, da je splet sestavljen iz virov, ki predstavljajo elementarne predmete zanimanja. Po sprožitvi zahteve po določenem resursu, ki je enolično označen, je le-ta dostavljen odjemalcu v neki predstavitveni obliki. To postavi odjemalca v neko določeno stanje. Ko je odjemalec pripravljen za prehod v drugo stanje, lahko sproži zahtevo po novem resursu, kar ga po prejemu postavi v neko novo stanje. Ker gre za arhitekturni stil in ne za standard, je implementacija posameznih komponent spletne aplikacije povsem svobodna. Potrebno je le slediti omejitvam, ki jih arhitekturni stil predlaga:

- **Odjemalec/strežnik** (ang. c*lient/server*) – ločitev funkcionalnosti med odjemalci in strežniki. To pomeni, da na primer odjemalci ne skrbijo za shranjevanje podatkov, kar ostaja temeljna skrb strežnikov, pri čemer je prenosljivost odjemalčeve programske kode izboljšana. Po drugi stran pa strežniki ne skrbijo za grafični vmesnik in tudi ne za stanja, v katerih se odjemalec lahko nahaja.
- **Brez stanja** (ang. *stateless*) – komunikacija med odjemalcem in strežnikom zahteva, da med zahtevami odjemalčevo stanje ni shranjeno na strežniku. Vsaka zahteva mora vsebovati vse potrebne informacije za uspešno obdelavo na strani strežnika, kakršnokoli stanje seje pa je lahko shranjeno le na strani odjemalca. Poudariti je potrebno, da to ne pomeni, da strežnik ne sme imeti stanja. Strežnik se lahko postavi v neko določeno stanje, vendar to lahko povzroči odjemalec v svoji zahtevi, ki naslavlja stanje strežnika kot resurs (na primer vzpostavitev seje s piškotki, ang. *cookie session*).
- **Predpomnjenje** (ang. *cachable*) – s pomočjo predpomnjenja strežniških odgovorov je mogoče interakcijo med odjemalcem in strežnikom minimizirati.
- **Sistem, ločen na nivoje** (ang. *layered system*) – to pomeni, da odjemalec običajno ne zna povedati, ali je povezan na strežnik ali ne. Ne gre torej za klasično vzpostavitev in vzdrževanje povezave, kot je to v primeru protokola za izmenjavo datotek preko omrežja (ang. *File Transfer Protocol*, skrajšano FTP).
- **Programska koda na zahtevo** (ang. *code on demand*) – omejitev omogoča strežnikom razširitev in prilagoditev funkcionalnosti odjemalcev s prenosom programske kode (na primer programski jezik JavaScript, ki je predstavljen v nadaljevanju).
- **Enoten vmesnik** (ang. *uniform interface*) – poenotena komunikacija med strežnikom in odjemalcem močno olajša in razdeli arhitekturo, s čimer omogoči neodvisen razvoj.

V delu se ne bomo spuščali v nadaljnje podrobnosti arhitekturnega stila REST, omeniti pa je potrebno, da razvita aplikacija sledi tem omejitvam, kar omogočajo tudi izbrane tehnologije za implementacijo, kot bomo videli v nadaljevanju.

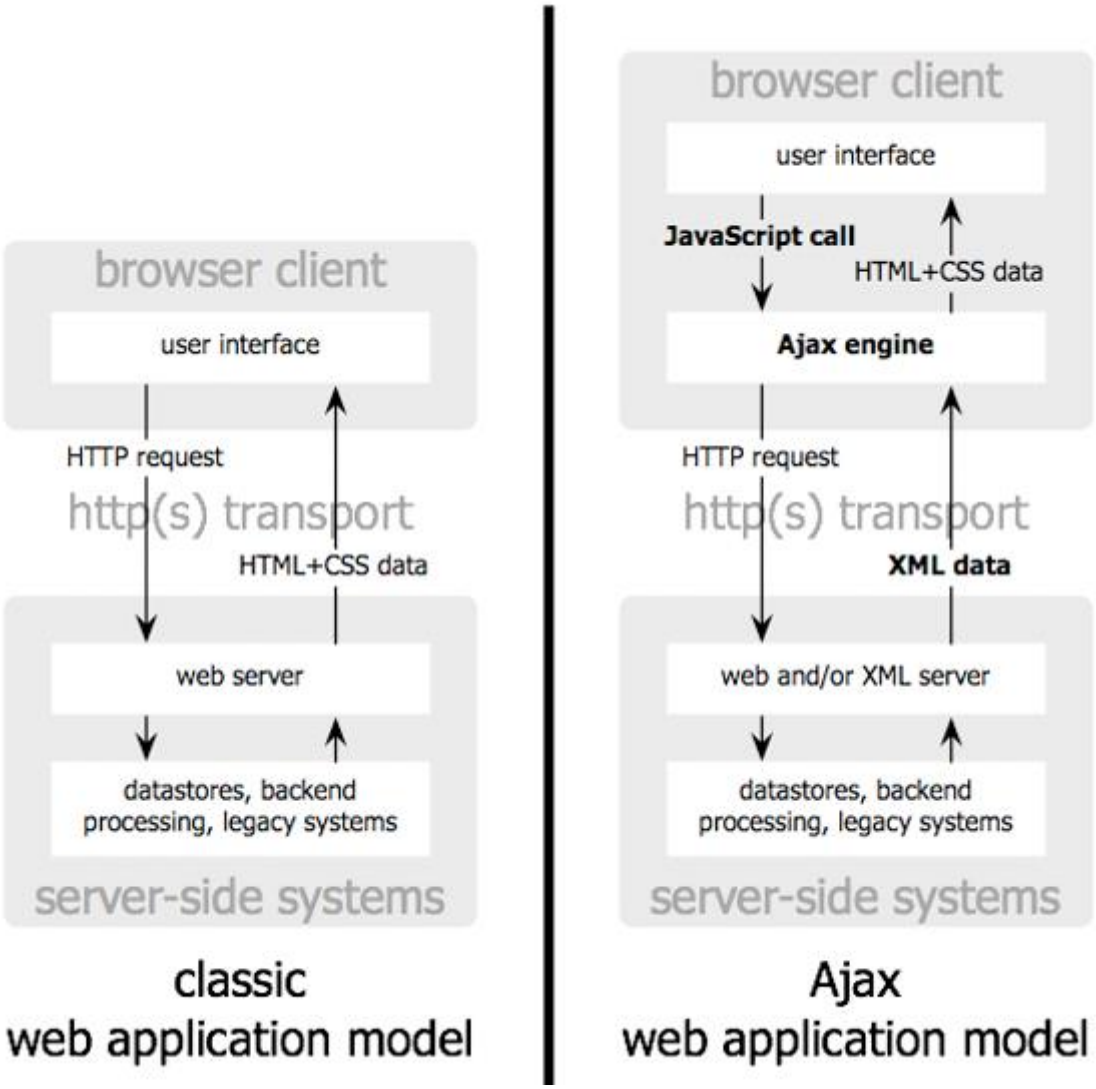
5.2.2 Kratek pregled tehnologij, jezikov, orodij in pristopov k izgradnji spletnih aplikacij

AJAX

AJAX je definiral Jesse James Garett [15] leta 2005. Ne predstavlja tehnologije, temveč arhitekturno platformo, oziroma »nov« pristop v smeri izkoriščanja in združevanja številnih obstoječih spletnih tehnologij, kot so:

- predstavitev s pomočjo razširljivega označevalnega jezika za prenos hiperteksta (ang. *Extensible Hypertext Markup Language*, skrajšano XHTML) in prekrivnih slogov (ang. *Cascading Style Sheets*, skrajšano CSS),
- dinamični prikaz in interakcija z uporabo dokumentno objektnega modela (ang. *Document Object Model*, skrajšano DOM),
- izmenjava in manipulacija s podatki z uporabo razširljivega označevalnega jezika (ang. *Extensible Markup Language*, skrajšano XML) in razširljivega jezika za preoblikovanje dokumentov XML (ang. *Extensible Stylesheet Language Transforms*, skrajšano XSLT),
- asinhrona pridobitev podatkov preko objekta XMLHttpRequest,
- združitev s pomočjo programskega jezika JavaScript.

S pristopom AJAX spletne aplikacije lahko pošiljajo in prejemajo podatke s strežnika asinhrono in v ozadju. Pri tem osveževanje strani ni potrebno, saj so prejeti in/ali poslani podatki le tisti podatki, ki so potrebni in/ali zahtevani. Podatki se pridobijo s pomočjo objekta XMLHttpRequest, katerega funkcije se lahko pokličejo iz skriptnega jezika JavaScript (obrazloženo v nadaljevanju). Kljub temu, da je v imenu in v prvotnih navedbah obstoječih tehnologij poudarjena uporaba jezika XML, pa to ni potrebno. Danes se za izmenjavo in manipulacijo s podatki čedalje bolj uporablja notacija objektov skriptnega jezika JavaScript (ang. *JavaScript Object Notation*, skrajšano JSON).



Slika 4: Primerjava med klasičnim in modelom spletnih aplikacij AJAX [15]

Na sliki 4 je ponazorjen pristop AJAX v primerjavi s klasičnim pristopom spletnih aplikacij. Razlika je očitna: pri klasičnem modelu gre za neposredno sprožitev zahteve na strežnik preko protokola za prenos hiperteksta (ang. *Hypertext Transfer Protocol*, skrajšano HTTP), medtem ko pa pristop AJAX vključuje dodaten nivo (na sliki 4 kot »Ajax engine«), ki na osnovi klica funkcij JavaScript objekta XMLHttpRequest sproži zahtevo HTTP. Na sliki 4 je prikazano, da so podatki v formatu jezika XML, vendar kot že prej povedano, je uporaba sporočil v formatu JSON prav tako mogoča.

XML

Jezik XML se uporablja predvsem za namene prenosa podatkov. Uporaben je posebej za komunikacijo med različnimi tipi informacijskih sistemov, ki med seboj niso združljivi. Dobra lastnost jezika XML je ta, da je za ljudi berljiv in razumljiv. Za »kodiranje« podatkov, jezik uporablja oznake (ang. *tag*). Oznake so besede ali fraze, ki opisujejo določene podatke, in sicer v naslednji obliki: `<ime oznake>Podatki…</ime oznake>`. Uporaba imen oznak ni vnaprej določena in je prosta izbira vsakega programerja. Edina omejitev je, da je za podobne vrste podatkov potrebno uporabljati ista imena oznak. Če za kodiranje podatka o imenu tako uporabimo denimo oznako `<ime>`, jo moramo uporabljati v celotnem dokumentu za ta namen. Možno je tudi imeti oznake, ki so znotraj neke druge oznake, kot na primer:

```
<stranka>
      <ime>Tomaž</ime>
      <priimek>Rojc</priimek>
      <emso>1509976550342</emso>
</stranka>
```

Za nadaljnjo izmenjavo podatkov med različnimi sistemi mora vsak izmed njih poznati strukturo podatkov v obliki jezika XML. To se lahko doseže z uporabo sheme XML, ki je nekoliko bolj kompleksna in tudi manj berljiva kot sam jezik. Podrobnosti sheme XML za namen izdelave magistrske naloge niso relevantne, za boljše razumevanje pa je potrebno omeniti, da služijo predvsem za opis vseh etiket in relacij med njimi v strukturi datoteke XML, ki se prenaša.

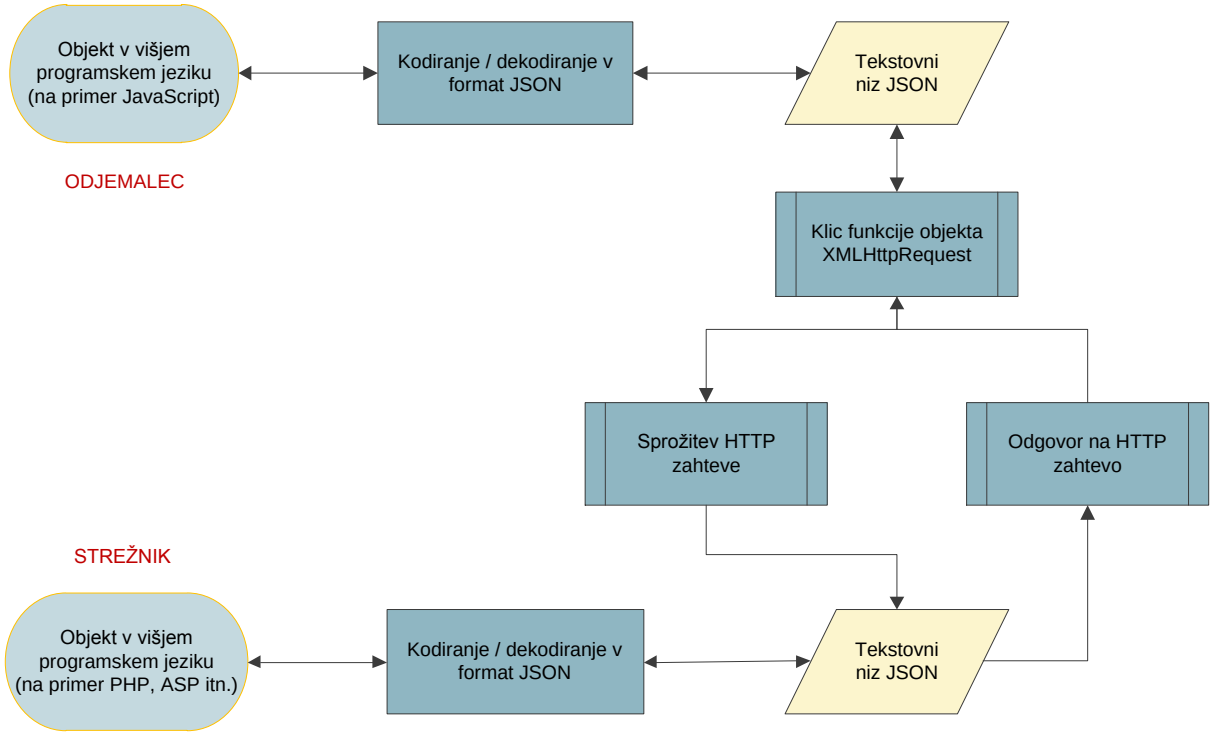
JSON

Notacija objektov v jeziku JavaScript JSON [17] predstavlja relativno enostaven format za izmenjavo podatkov. Podobno kot jezik XML je za ljudi berljiv in razumljiv. Največja prednost je v tem, da ga zaradi svojega formata računalnik enostavno razčleni in/ali generira. Temelji na skriptnem jeziku JavaScript oziroma standardu ECMA-262 [16]. Gre za jezikovno neodvisen tekstualni format sporočil, ki pa uporablja konvencije, ki so podobne programskim jezikom, kot na primer C, C++, C#, Java, JavaScript, Perl, Python, PHP in drugim. Glede na te osnovne lastnosti lahko rečemo, da predstavlja dobro izbiro formata za izmenjavo podatkov. JSON temelji na dveh strukturah:

- zbirka parov ime/vrednost, ki v različnih programskih jezikih pomeni objekt (ang. *object*), zapis (ang. *record*), struktura (ang. *struct*), slovar (ang. *dictionary*), razpršena tabela (ang. *hash table*), seznam na osnovi ključev (ang. *keyed list*) ali asociativna tabela (ang. *associative array*),

- urejen seznam vrednosti, ki v različnih jezikih pomeni polje (ang. *array*), vektor (ang. *vector*), seznam (ang. *list*) ali zaporedje (ang. *sequence*).

Naštete podatkovne strukture so »univerzalne« in jih praktično vsi sodobni programski jeziki podpirajo v eni ali drugi obliki. Zato je smiselno imeti format za izmenjavo podatkov, ki podpira vse omenjene podatkovne strukture.

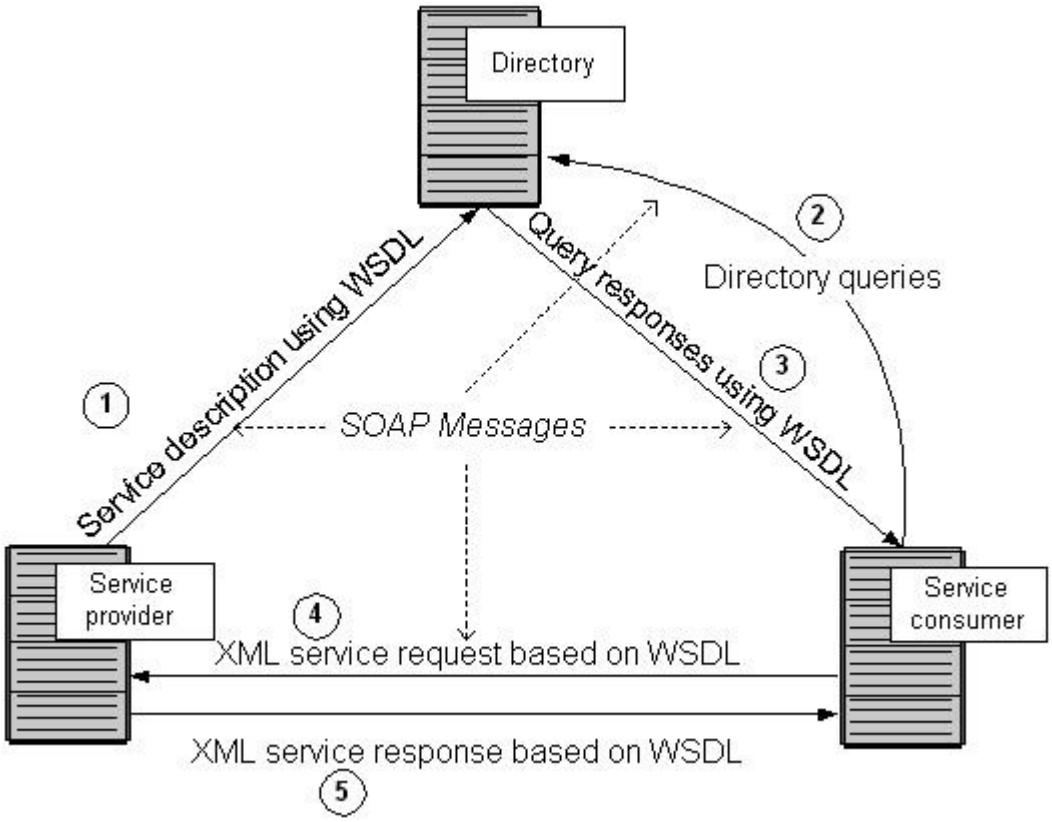


Slika 5: Princip uporabe formata JSON

V osnovi JSON deluje tako, da se na strani odjemalca objekt kodira v tekstovni niz v formatu JSON (prikazano na sliki 5). Z uporabo funkcij objekta XMLHttpRequest (arhitekturni pristop AJAX) se ta niz pošlje strežniku, kjer je ustrezno dekodiran in obdelan. Kot odgovor na zahtevo strežnik lahko na enak način pošlje odjemalcu zahtevane podatke nazaj.

WSDL

Jezik za opis spletnih storitev WSDL [18] (ang. *Web Services Description Language*, skrajšano WSDL) predstavlja osnovo spletnih storitev. Na sliki 6 je prikazana uporaba WSDL. Na levi strani se nahaja ponudnik storitev, na desni pa odjemalec le-teh.



Slika 6: Uporaba jezika za opis spletnih storitev [18]

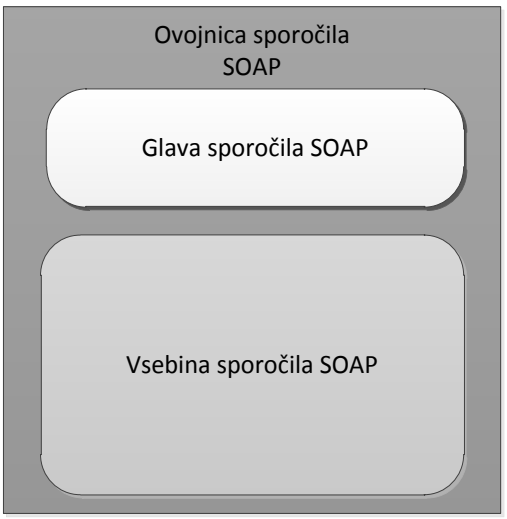
Koraki, ki opisujejo uporabo spletnih storitev, so sledeči:

- Ponudnik storitev svojo storitev opiše z uporabo jezika WSDL. Definicija storitve je objavljena v imeniku storitev, kot na primer UDDI (ang. *Universal Description, Discovery and Integration*, skrajšano UDDI).
- Odjemalec storitve sproži eno ali več poizvedb imeniku storitev, zato da storitev locira in ugotovi, na kakšen način lahko komunicira s to storitvijo.
- Del vsebine WSDL je s strani ponudnika storitev posredovana odjemalcu storitev. Ta pove odjemalcu storitev, kakšne so lahko zahteve in odgovori ponudniku storitev.

- Odjemalec storitev uporablja jezik WSDL za sprožitev zahteve ponudniku storitev.
- Ponudnik storitev se na zahtevo odzove z ustreznim odgovorom.

SOAP

Protokol enostavnega dostopa do objektov (ang. *Simple Object Acess Protocol*, skrajšano SOAP) je komunikacijski protokol za izmenjavo podatkov med različnimi tipi informacijskih sistemov. Oblikovan je za komunikacijo preko spleta in se uporablja predvsem v povezavi s spletnimi storitvami (prikazano na sliki 6 - WSDL). Temelji na jeziku XML in je platformsko ter jezikovno neodvisen. Z uporabo protokola SOAP je možno prenašati cele dokumente ali povzročiti klic oddaljene procedure (ang. *Remote Procedure Call*, skrajšano RPC). Sporočila SOAP (prikazano na sliki 7) so lahko dostavljena z uporabo različnih transportnih protokolov, vendar se protokol primarno osredotoča na klice oddaljenih procedur prek uporabe protokola HTTP. Protokol SOAP omogoča odjemalčevim aplikacijam enostavno povezljivost z oddaljenimi spletnimi storitvami in sprožitev oddaljenih procedur.




```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"

SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  ...
Vsebina sporočila
 ...
</SOAP-ENV:Envelope>
```

Slika 7: Struktura sporočila SOAP

ASP in ASP.NET

Aktivne strežniške strani (ang. Active Server Pages, skrajšano ASP) predstavljajo strežniško skriptno okolje, ki se lahko uporablja za ustvarjanje in izvajanje interaktivnih spletnih strežniških aplikacij. Z uporabo tehnologije ASP je možno kombinirati HTML strani, skriptne

ukaze in objekte COM (ang. *Component Object Model*, skrajšano COM) za namen kreiranja spletnih strani in zmogljivih spletnih aplikacij, ki jih je enostavno razvijati in spreminjati [20].

ASP tehnologija je bila razvita s strani podjetja Microsoft leta 1996. Temelji na skriptnih jezikih VBScript (zasnovan na programskem jeziku Visual Basic, ki ga je razvilo podjetje Microsoft) in JScript (zasnovan na programskem jeziku Java, ki ga je razvilo podjetje Sun). Januarja 2002 je podjetje Microsoft izdalo prvo verzijo ASP.NET, ki predstavlja nadgradnjo prvotne ASP. Temelji na ogrodju .NET, ki predstavlja skupek tehnologij in izdelkov podjetja Microsoft [19]. ASP.NET aplikacije lahko dostopajo do razredov ogrodja .NET in so lahko pisane v katerem koli programskem jeziku, ki je združljiv s CLR (ang. *Common Language Runtime*, skrajšano CLR), kot sta to na primer Microsoft Visual Basic in C# [21].

ASP.NET aplikacije lahko za vsako odjemalčevo zahtevo dinamično ustvarijo spletno stran HTML, ki se vrne odjemalcu kot odgovor. Na takšen način je odjemalcu posredovana vsebina, ki je pridobljena iz različnih virov na strežniški strani (na primer iz podatkovne baze, nekega drugega oddaljenega računalniškega sistema, ki komunicira s ciljnim strežnikom itn.) in glede na odjemalčevo zahtevo dinamično generirana. Naj omenimo še, da omenjena tehnologija ni omejena samo na ustvarjanje in posredovanje dinamičnih spletnih strani. Tehnologija ASP.NET podpira tudi arhitekturni pristop AJAX, kar omogoča prenos le potrebnih podatkov med strežnikom in odjemalcem in ne celotnih dinamično ustvarjenih spletnih strani. Tako je na strani odjemalca možno naložiti dinamično vsebino v ozadju, brez osveževanja in ponovnega izrisovanja celotne spletne strani.

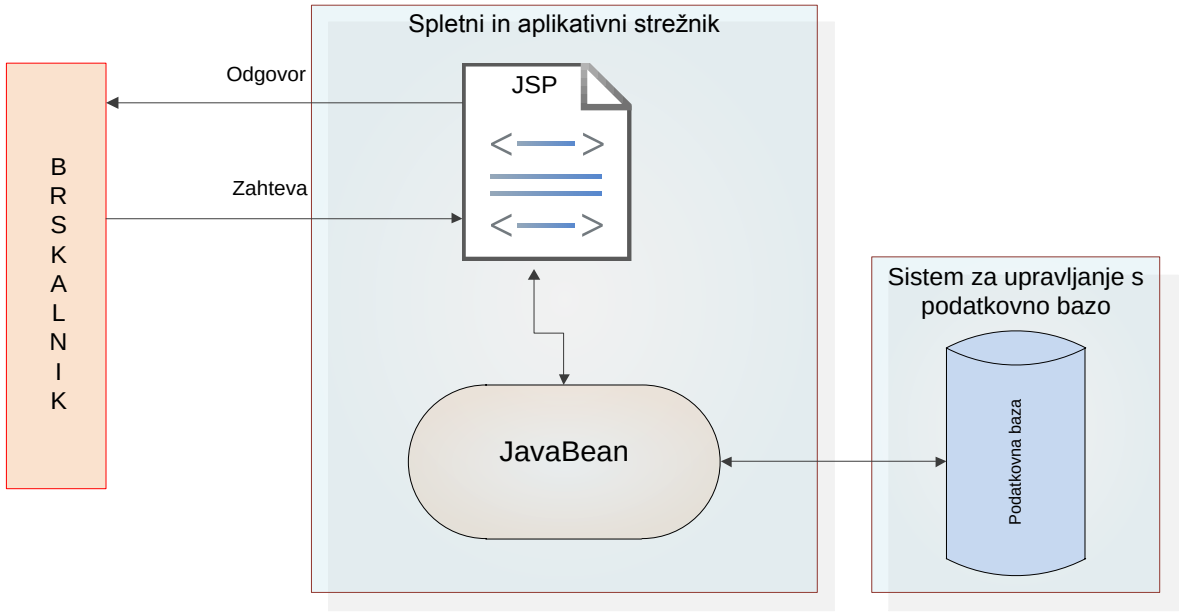
Za izvajanje kode ASP.NET je potrebno imeti postavljen spletni strežnik – najboljši je IIS (ang. *Internet Information Services*, skrajšano IIS) podjetja Microsoft, čeprav so možni tudi nekateri drugi (na primer Apache), vendar ne omogočajo vseh funkcionalnosti kot IIS.

JSP

Tehnologija strežniških strani JSP [22] (ang. *Java Server Pages*, skrajšano JSP), pisanih v programskem jeziku Java, omogoča podobne funkcionalnosti kot na primer ASP.NET, le da gre za drugačen pristop, ki temelji na družini tehnologij Java. Razvoj specifikacije JSP je voden s strani proizvajalca Sun Microsystems in definira interakcijo med strežnikom in stranmi JSP ter opisuje format ter sintakso strani.

Tehnologija JSP omogoča ustvarjanje strežniških strani, ki prikazujejo dinamično generirano vsebino. Prevedene so v *servlete* in z njimi je mogoče klicati različne komponente iz družine drugih tehnologij Java (na primer komponente JavaBeans) za namen procesiranja podatkov na strežniški strani. Teoretično je možno napisati *servlete* za podporo spletnim aplikacijam,

vendar pa je tehnologija JSP bila razvita zato, da poenostavi proces ustvarjanja strani, ki ločuje spletno predstavitev od spletne vsebine. Na sliki 8 je prikazan možen način delovanja spletne aplikacije z uporabo tehnologije JSP.



Slika 8: Eden izmed možnih načinov delovanja spletne aplikacije z uporabo tehnologije JSP

Spletne aplikacije Java je možno namestiti na veliko različnih aplikativnih in spletnih strežnikov, ki vključujejo podporo tehnologiji JSP, kot na primer BEA WebLogic, IBM WebSphere, Sun Java System in drugi.

Iz obrazloženega je razvidno, da je tehnologija JSP ključna komponenta v arhitekturnem pristopu, ki spletnim aplikacijam omogoča visoko razširljivost in platformno neodvisnost.

PHP

PHP [23] je široko uporaben odprtokodni skriptni programski jezik, ki je posebej primeren za razvoj spletnih aplikacij. Koda PHP se izvršuje na strežniku, pri čemer se po navadi generira HTML, ki je nato poslan nazaj odjemalcu. V grobem je delovanje PHP podobno delovanju JSP in/ali ASP.NET, le da gre ponovno za drugačno tehnologijo, ki zahteva nekoliko drugačen pristop.

```
<?php
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */


function logout()
{
    //here we destroy all data associated with the session
    // Initialize the session.
    // If you are using session_name("something"), don't forget it now!

    // Unset all of the session variables including cookies
    $_SESSION = array();
    $_COOKIE = array();

    // If it's desired to kill the session, also delete the session cookie.
    // Note: This will destroy the session, and not just the session data!
    if (ini_get("session.use_cookies"))
    {
        $params = session_get_cookie_params();
        setcookie(session_name(), '', time() - 42000, $params["path"],
                        $params["domain"], $params["secure"], $params["httponly"]);
        setcookie("SCRUMSESSIONID", '', time() - 42000);
        setcookie("SCRUMUSER", '', time() - 42000);
        setcookie("SCRUMUSERROLE", '', time() - 42000);
        setcookie("SCRUMEMPLOYEEID", '', time() - 42000);
    }
    // Finally, destroy the session.
    session_destroy();

    $par = "<p>Thank you for using SCRUM project management tool.</a></br>";
    echo $par . "<a href=" . "'" . "." . "'>" . "Click here if you want to login again</a>";

}

session_start();
logout();

?>
```

Slika 9: Primer funkcije v PHP, ki uporabnika odjavi iz sistema in izbriše sejo – datoteka logout.php orodja

Obstajajo tri glavna področja, v katerih so uporabljene skripte, pisane v programskem jeziku PHP:

- Skripte na strani strežnika (ang. *server side scripting*) so tradicionalni in poglavitni cilj programskega jezika PHP (primer je prikazan na sliki 9). Za normalno delovanje je potrebno imeti postavljeno naslednjo programsko opremo: razčlenjevalnik (ang. *parser*) PHP, spletni strežnik in spletni brskalnik. Spletni strežnik in razčlenjevalnik morata biti združljiva in ju je potrebno ustrezno nastaviti. S pomočjo spletnega brskalnika je možno sprožiti zahtevo na strežnik, ki gosti skripto PHP, in prikazati izhod oziroma odgovor skripte.

- Skripte v ukazni vrstici (ang. *Command line scripting*) se lahko aktivirajo brez kakršnega koli strežnika ali brskalnika. Potrebna je samo namestitev razčlenjevalnika PHP. Takšna vrsta uporabe je po navadi dobra za manj zahtevna opravila (na primer procesiranje tekstovnih datotek), ki se sprožajo ob določenih časovnih intervalih (na primer v operacijskem sistemu Linux s pomočjo programa cron ali v Windows s pomočjo programa Task Scheduler).
- Namizne aplikacije je sicer možno pisati v programskem jeziku PHP, vendar je splošno znano dejstvo, da to ni ravno najboljša izbira. V podrobnosti se niti ne bomo spuščali, saj je predmet implementacije spletna in ne namizna aplikacija.

PHP je možno uporabiti v več operacijskih sistemih: Linux, Microsoft Windows, Mac OS X, RISC OS itn. Prav tako je omogočena podpora PHP s strani kar nekaj spletnih strežnikov, od katerih je najbolj pomembno omeniti strežnika Apache in Microsoft IIS.

Uporaba PHP prinaša svobodo pri izbiri operacijskega sistema in spletnega strežnika. Poleg tega je možno izbrati uporabo proceduralnega ali objektno orientiranega načina programiranja, ali celo mešanico obeh, kar je v osnovi značilno za programski jezik PHP.

Ena izmed najmočnejših in najpomembnejših značilnosti PHP je podpora širokemu naboru podatkovnih baz skozi različne razširitve (na primer razširitev mysqli za podatkovno bazo MySQL in druge) in podpora komunikaciji z različnimi storitvami, ki uporabljajo protokole kot na primer HTTP, COM in veliko drugih.

PHP že v osnovi vsebuje veliko uporabnih funkcij za procesiranje teksta. Ta vključuje podporo regularnim izrazom, obstaja pa tudi veliko razširitev in orodij, ki omogočajo dostop in razčlenjevanje dokumentov XML.

Kljub tako razširjeni podpori bi bila uporaba PHP nesmiselna brez možnosti razhroščevanja programske kode. To je posebej pomembno, če gre za implementacijo kakšnega kompleksnejšega orodja, kot je ta, ki je predstavljeno v delu. PHP tako podpira razširitev XDebug, ki jo je možno naknadno namestiti v sklopu nastavitev razvojnega okolja.

JavaScript

Gre za enostaven programski jezik, ki se ga lahko naučimo relativno hitro. Jezik JavaScript je mogoče uporabiti le na odjemalčevi strani. V kombinaciji s HTML lahko implementiramo grafični vmesnik, ki bo služil za interakcijo z uporabnikom. Omogoča pa tudi izvedbo nekaterih ne toliko bistvenih preverjanj predvsem v smislu vnosov podatkov s strani uporabnika (na primer uporabnik vnese črko, a je zahtevan vnos številka, kar koda JavaScript

lahko detektira), pošiljanja določenih podatkov strežniku, sprožanja zahtev po podatkih s strani strežnika, sprožitev izvajanja določenih akcij na strežniku itn.

```
function GenerateSprintBacklogCSVFile(selectedMeasuresCriteria)
{
    //here we have to send a request to generate a CSV file and to link it to it
    var SprintBacklogCSV_DS = {
        ID:"SprintBacklogCSV",
        dataFormat: "json",
        dataURL: "rest.php",
        fields:[
            {name:"link", length: 65535, type:"text"}
        ]
    };
    isc.RestDataSource.create(SprintBacklogCSV_DS);

    var recSprint = listSprintsGrid.getSelectedRecord();
    if (recSprint)
    {
        SprintBacklogCSV.fetchData({procedure: "Create_Backlog_CSV_File",
                                            Sprint_ID: recSprint.Sprint_ID,
                                            Selected_Measures:
                                            selectedMeasuresCriteria},
                                    function (dsResponse, data)
                                    {
                                        var record = data.get(0);
                                        if(record && record.link)
                                        {
                                            window.open(record.link);
                                        }
                                        if(dsResponse.status == -4)
                                        {
                                            isc.warn("Operation could not be
                                                                completed!<br>
                                                                Please verify that you have
                                                                Sufficient privileges!");
                                        }
                                    }
        );
    }
    else
    {
        isc.warn("Please select Sprint record first !");
    }
}
```

Slika 10: Primer funkcije v jeziku JavaScript – datoteka SprintBacklog_gui.js orodja

Jezik JavaScript je podprt je s strani večjega števila podjetij (npr. Netscape, Microsoft, Google itn.) in je odprt jezik, ki ga lahko uporablja vsak brez kakšnega posebnega plačila. Interpreter jezika JavaScript je vgrajen v spletne brskalnike (npr. Netscape, Internet Explorer, Mozilla Firefox, Opera, Chrome itn.), tako da posebnih orodij ni potrebno imeti. JavaScript lahko mešamo s HTML kodo, kar dovoljuje spletnim programerjem popestritev strani z dinamično vsebino.

JavaScript kodo napišemo med oznake `<SCRIPT>` in `</SCRIPT>` kjerkoli v glavi ali telesu našega HTML dokumenta, možno pa je kodo shraniti v posebno datoteko in jo vključiti v našo spletno stran, kot na primer:

```
<script type="text/javascript" src="GUI/SprintBacklog_gui.js"></script>
```

Na sliki 10 je prikazan primer funkcije, ki je napisana v jeziku JavaScript in je del programskega orodja, ki je bilo razvito v okviru pričujoče magistrske naloge. Čeprav se v tem trenutku v podrobnosti kode orodja ne moremo spuščati, je potrebno omeniti le, da se funkcija sproži ob zahtevi po izpisu seznama nalog.

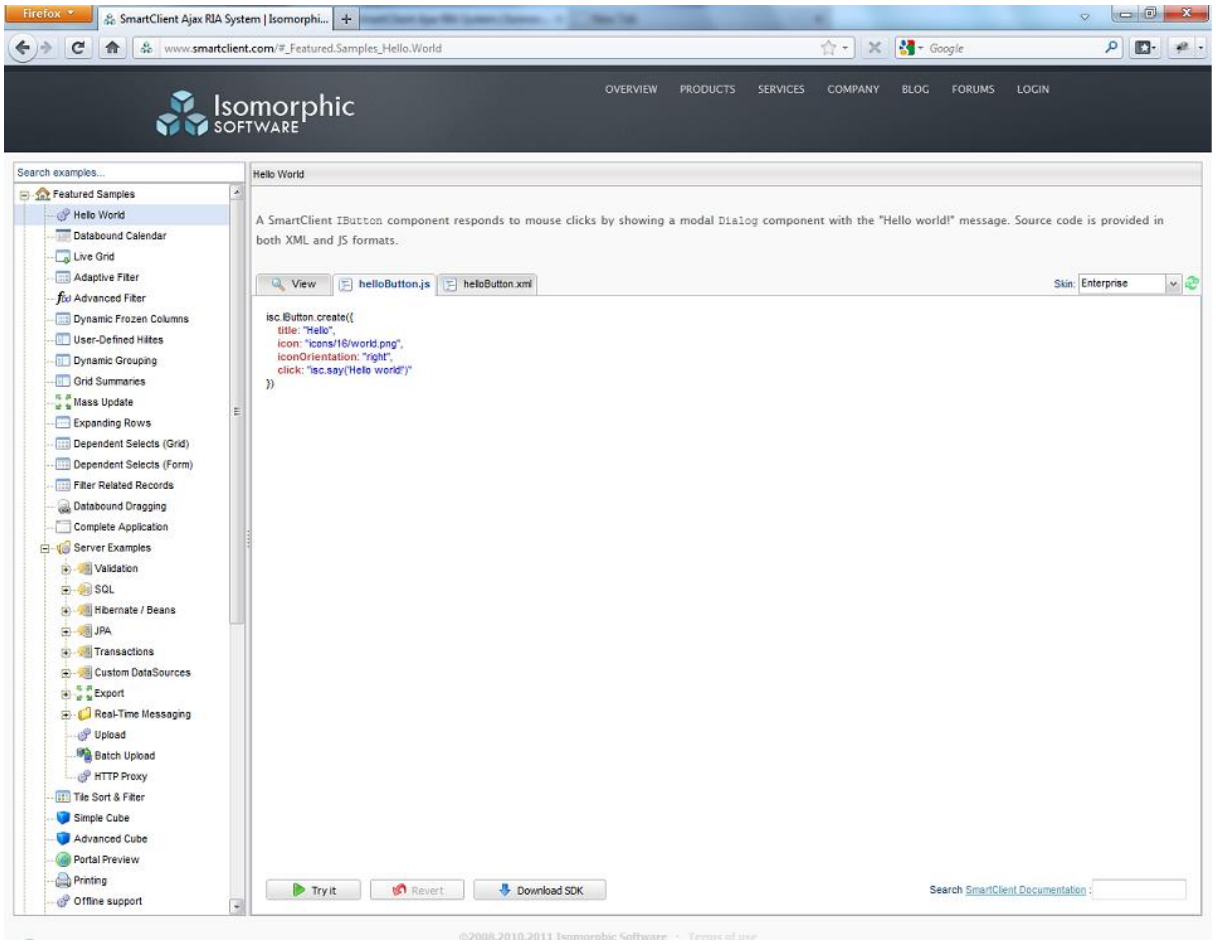
Na tej točki se pojavita dva problema glede implementacije grafičnega vmesnika: Ali bo potrebno »na roke« pisati kodo grafičnega vmesnika v smislu kodiranja kombinacije HTML in JavaScript in Ali bo za vsako zahtevo potrebno stran osveževati na odjemalčevi strani. Odgovor na obe vprašanji reši uporaba orodja SmartClient proizvajalca Isomorphic Software, ki v osnovi uporablja kombinacijo tehnologij HTML5 (peta generacija HTML, ki prinaša kar nekaj sintaktičnih novosti, v katere se ne bomo spuščali, ker presega obseg naloge) in AJAX.

SmartClient

Smartclient [24] je tehnologija, ki podpira AJAX/HTML5 izvajalno okolje (ang. *runtime*). Razvita je bila s strani proizvajalca Isomorphic Software in vsebuje sledeče:

- stroj na strani odjemalca AJAX/HTML5 (ang. *AJAX/HTML5 client engine*), ki ne zahteva posebne namestitve,
- bogat nabor komponent in storitev, ki omogočajo enostavno izgradnjo uporabniškega vmesnika,
- podporo funkcionalnostim za enostavnejšo komunikacijo med strežnikom in odjemalcem.

Z uporabo tehnologije SmartClient so aplikacije pisane v objektno usmerjenem programskem jeziku JavaScript. Vsa predstavitvena opravila in generiranje HTML dokumentov se izvršujejo v spletnem brskalniku. Na strežniški strani ni potrebno skrbeti za generiranje HTML dokumentov ali kakršnakoli opravila v povezavi s predstavitvijo podatkov. Ko se koda aplikacije enkrat naloži v spletni brskalnik, je potrebno prenašati le podatke med spletnim brskalnikom in strežnikom. S takšnim pristopom je frekvenca pošiljanja zahtev na strežnik precej zmanjšana. Ker strežniku ni potrebno skrbeti za predstavitveni del spletne aplikacije, je koda na strežniški strani prav tako minimizirana. S takšnim arhitekturnim pristopom tehnologija SmartClient zagotovo izboljšuje odzivnost spletne aplikacije in omogoča boljšo razširljivost, zmanjšana količina kode pa posledično pomeni boljšo stabilnost in zanesljivost.



Slika 11: Testna spletna stran Isomorphic SmartClient (http://www.smartclient.com/#Welcome)

SmartClient podpira bogat nabor razširljivih komponent za izgradnjo uporabniških vmesnikov, ki so podprte v večini danes najbolj razširjenih spletnih brskalnikov (na primer Mozilla Firefox, Internet Explorer, Opera, Google Chrome idr.). Te komponente je mogoče razširiti in prilagoditi potrebam aplikacije.

Tehnologija SmartClient je fokusirana predvsem na poslovne aplikacije. Skuša se približati konceptu namiznih aplikacij s perspektive uporabnika s to razliko, da je njena uporaba znotraj spletnega brskalnika. Tako obstajajo uporabniške komponente (na primer komponente »*grid*«), ki omogočajo skorajda večino funkcionalnosti, ki jih podpirajo komponente s podobnimi lastnostmi namiznih aplikacij - prikaz, shranjevanje in urejanje. Na sliki 11 je prikazana povezava na proizvajalčevo spletno stran, kjer je mogoče preveriti način delovanja najbolj uporabljenih uporabniških komponent.

Pomembno je omeniti, da je SmartClient možno integrirati s katero koli strežniško platformo, ki podpira in uporablja bodisi arhitekturni stil REST bodisi WSDL v povezavi s spletnimi storitvami.

5.2.3 Izbira ustreznih tehnologij in orodij za implementacijo

Na začetku tega poglavja je bilo nakazano, da bo programsko orodje, ki bo podpiralo metodo Scrum, trinivojska spletna aplikacija. Za vsak nivo je potrebno določiti ustrezen pristop k izgradnji, ki pomeni izbiro med razpoložljivimi tehnologijami, programskimi jeziki in orodji, s katerimi bo mogoče orodje realizirati. Po pregledu in nekoliko grobi preučitvi kaj le-te omogočajo in česa ne, je bila izbira posledica odgovorov na nekaj preprostih vprašanj, ki se nanašajo predvsem na tehnologije, orodja in jezike:

- Ali je mogoče rešiti zastavljen problem?
- Ali obstaja možnost, da bo rešitev kompleksnejša in časovno zamudna?
- Ali je mogoče slediti osnovnim vodilom, ki skrbijo za kakovostno izdelavo programskega orodja?
- Katere tehnologije, orodja in jezike je možno med seboj združiti na vsakem nivoju in brez večjih težav?
- Ali je izbrana tehnologija platformsko odvisna?
- Kako je z dostopnostjo (prosto dostopno, plačljivo)?
- Kakšna je podpora (dokumentiranost)?
- Kako je z razširljivostjo določenih rešitev problemov na spletu?
- Kakšno je predznanje?
- Kako je z dostopnostjo podatkovnih strežnikov?

Takoj na začetku je potrebno poudariti, da kakšne posebne izbire na strani odjemalca ni. Izbira je uporaba tehnologije SmartClient, jezik za implementacijo pa JavaScript. Razlogi so predvsem v tem, da so odgovori na večino zastavljenih vprašanj v prid uporabe, razen predznanja, ki ga praktično ni, kar pomeni, da bo potrebno nekaj časa vložiti v samo učenje tehnologije. SmartClient je platformsko neodvisen, razmeroma dobro dokumentiran in združljiv s katero koli strežniško platformo, ki uporablja REST ali WSDL. Obstaja verzija, ki je prosto dostopna in hkrati podpira funkcionalnosti, ki so zadostne za rešitev zadanega problema.

Nekoliko večji problem predstavlja izbira tehnologije na strani strežnika z ustrezno izbiro spletnega aplikativnega strežnika. Vse tri tehnologije: ASP.NET, JSP in PHP, podpirajo

funkcionalnosti, ki so zadostne za ustrezno implementacijo in so prav tako združljive s tehnologijo SmartClient.

V tabeli 2 je prikazana preprosta primerjava med tehnologijami, ki prispeva k ustrezni izbiri. Iz prikazanega je razvidno, da je ASP.NET najslabša izbira za rešitev problema. Kljub najboljši dokumentiranosti in podpori gre za tehnologijo, ki je odvisna od operacijskega sistema Windows in kjer orodja, ki omogočajo implementacijo, niso prosto dostopna oziroma so prosto dostopne verzije precej omejene v podprtih funkcionalnostih (na primer podatkovni strežnik Microsoft SQL Server Express). Poleg omenjenega izkušnje s tehnologijo ASP.NET in ustreznim razvojnim okoljem kažejo na nižje predznanje, kar pomeni dodaten časovni vložek v samo učenje. Glede na upoštevane kriterije sta JSP in PHP relativno enakovredni. Loči ju samo predznanje in subjektivna ocena razpoložljivosti rešitev specifičnih problemov na spletu, ki je nekoliko višja na strani PHP. Zato je končna izbira uporaba PHP.

PHP na strežniški strani za seboj potegne izbiro spletnega aplikativnega strežnika Apache in sistema za upravljanje podatkovne baze MySQL. Razvita aplikacija je bila sicer razvita v okolju operacijskega sistema Windows 7 Professional, vendar pa jo je, z ustreznimi sistemskimi nastavitvami in brez spreminjanja programske kode, možno relativno enostavno prenesti v drug operacijski sistem kot na primer Linux. Detajlen opis okolja, v katerem je spletna aplikacija bila razvita:

- Operacijski sistem: Windows 7 Professional, 64-bitna različica.
- Spletni aplikativni strežnik: Apache 2.2.19.
- Jezik in tehnologija na strani strežnika: PHP 5.3.6.
- Jezik in tehnologija na strani odjemalca: JavaScript na osnovi SmartClient_70rc2_LGPL.
- Podatkovna baza: MySQL 5.5, 64-bitna različica.
- Sistem za upravljanje s podatkovno bazo: MySQL Workbench 5.2.34 CE.
- Razvojno okolje: Netbeans 7.0.1.
- Razhroščevalnik: php_xdebug-2.1.1-5.3-vc9.dll.
- Prenos sporočil: format JSON.

	ASP.NET	JSP	PHP
Ocena učinkovitosti rešljivosti problema (hitro in brez večjih težav)	DA	DA	DA
Združljivost s tehnologijo SmartClient	DA	DA	DA
Dostopnost razvojnih okolij	1. Microsoft Visual Studio ni prosto dostopen 2. Microsoft Web Matrix je prosto dostopen	Prosto dostopno (Netbeans, Eclipse)	Prosto dostopno (Netbeans, Eclipse)
Dostopnost kompleta funkcij za razvoj (ang. *Software development Kit*, skrajšano SDK)	Prosto dostopno (del .NET)	Prosto dostopna	Prosto dostopna
Dokumentiranost in podpora tehnologijam	Zelo visoka	Visoka	Visoka
Razpoložljive rešitve na spletu	Srednje	Srednje	Veliko
Predznanje tehnologij	Nizko	Zadostno	Napredno
Predznanje razvojnih okolij	Nizko	Zadostno	Napredno
Platforme	Windows	Neodvisno	Neodvisno
Spletni aplikativni strežniki	IIS je prosto dostopen oz. je del operacijskega sistema Windows Vista Business	Apache Tomcat prosto dostopen	Apache prosto dostopen
Podatkovni strežniki	Najbolj primeren in priporočljiv je Microsoft SQL Server, ki ni prosto dostopen	Primeren je lahko MySQL, ki je prosto dostopen	Primeren je lahko MySQL, ki je prosto dostopen

Tabela 2: Primerjava med tehnologijami ASP.NET, JSP in PHP za ustrezno izbiro

Velja omeniti še izbiro formata za prenos sporočil med XML in JSON. Kljub temu, da SmartClient in PHP podpirata funkcionalnosti, ki omogočajo sestavljanje in razčlenitev podatkov v obeh formatih, je izbira na strani JSON. V podrobne razlike med obema formatoma se ne bomo spuščali, ker to presega obseg dela, a poudariti je potrebno, da je format JSON veliko bolj enostaven kot XML, za ljudi je lažje berljiv, notacija je že vgrajena v programski jezik JavaScript (to pomeni, da ni potrebe po dodatni programski opremi) in optimiziran je za prenos »golih podatkov« (ne zvoka, slike ali videa, ki v orodju sploh ne bo potrebno). Iz tega razloga uporaba protokola SOAP odpade, kot tudi uporaba jezika spletnih storitev WSDL.

5.3 Osnovne sistemske nastavitve

Za delovanje spletne aplikacije je potrebno ustrezno postaviti strežniško okolje. To vključuje namestitev spletnega aplikativnega strežnika Apache, podatkovne baze MySQL ter okolja, ki bo omogočalo izvajanje kode PHP.

Spletni aplikativni strežnik Apache omogoča precej nastavitev, s katerimi je mogoče nadzirati delovanje strežnika. Omenimo le najpomembnejše nastavitve, ki se lahko spreminjajo v datoteki »httpd.conf« (nahaja se v podmapi korenske mape namestitve strežnika Apache, po navadi je to `C:\Program Files\Apache Software Foundation\Apache2.2\conf\`):

- Vrata, na katerih bo strežnik poslušal oziroma sprejemal zahteve, se nastavijo z direktivo `Listen`.
- Direktiva `DocumentRoot` omogoča nastavitev korenske mape spletnih aplikacij, katere bo Apache stregel (po navadi je to v mapi `C:\Program Files\Apache Software Foundation\Apache2.2\htdocs`).
- Z direktivo `<Directory/>…<Directory>` se nastavijo stopnje dostopa do vseh in/ali le specifičnih map, ki so na razpolago strežniku Apache.
- Strežniku Apache je potrebno povedati, kje se nahajajo nastavitve izvajalnega okolja PHP (z direktivo `PHPIniDir`) in da je ob zagonu potrebno naložiti poseben modul, ki bo omogočal izvajanje kode PHP (z direktivo `LoadModule`).

```
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#Listen 12.34.56.78:80
Listen 80
######################
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
DocumentRoot "C:/Program Files (x86)/Apache Software Foundation/Apache2.2/htdocs"
# Each directory to which Apache has access can be configured with respect
# to which services and features are allowed and/or disabled in that
# directory (and its subdirectories).
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>
# Note that from this point forward you must specifically allow
# particular features to be enabled - so if something's not working as
# you might expect, make sure that you have specifically enabled it
# below.
# This should be changed to whatever you set DocumentRoot to.
<Directory "C:/Program Files (x86)/Apache Software Foundation/Apache2.2/htdocs">
    #
    # Possible values for the Options directive are "None", "All",
    # or any combination of:
    #   Indexes Includes FollowSymLinks SymLinksifOwnerMatch ExecCGI MultiViews
    #
    # Note that "MultiViews" must be named *explicitly* --- "Options All"
    # doesn't give it to you.
    #
    # The Options directive is both complicated and important.  Please see
    # http://httpd.apache.org/docs/2.2/mod/core.html#options
    # for more information.
    #
    Options -Indexes FollowSymLinks
    #
    # AllowOverride controls what directives may be placed in .htaccess files.
    # It can be "All", "None", or any combination of the keywords:
    #   Options FileInfo AuthConfig Limit
    #
    AllowOverride None
    #
    # Controls who can get stuff from this server.
    #
    Order allow,deny
    Allow from all
</Directory>
######################
LoadModule php5_module "C:/Program Files (x86)/PHP/php5apache2_2.dll"
AddType application/x-httpd-php .php
AcceptPathInfo on
PHPIniDir "C:/Program Files (x86)/PHP"
```

Slika 12: Bistvene nastavitve strežnika Apache (del datoteke `httpd.conf`)

Na sliki 12 se nahaja del datoteke `httpd.conf`, v kateri je nastavljeno, da strežnik Apache posluša na vratih številka 80 in da je korenska mapa spletnih aplikacij `C:\Program Files\Apache Software Foundation\Apache2.2\htdocs`. Dostop do korenske mape je dovoljen, vendar omejeno - potrebno je poznati ime mape spletne aplikacije, v kateri se nahaja datoteka, ki omogoča začetek delovanja spletne aplikacije (na primer `index.php`). Iz

slike 12 je razvidno tudi to, da se nastavitvena datoteka okolja PHP `php.ini` in ustrezen modul oziroma dinamična knjižnica, ki jo Apache mora naložiti ob zagonu (datoteka `php5apache2_2.dll`), nahajata v korenski mapi namestitve izvajalnega okolja PHP (po navadi v `C:\Program Files\PHP`).

Nastavitve izvajalnega okolja PHP se nahajajo v datoteki `php.ini`, ki se nahaja v korenski mapi same namestitve (po navadi v `C:\Program Files\PHP`). Bistvene so predvsem sledeče nastavitve, ki so ponazorjene na sliki 13:

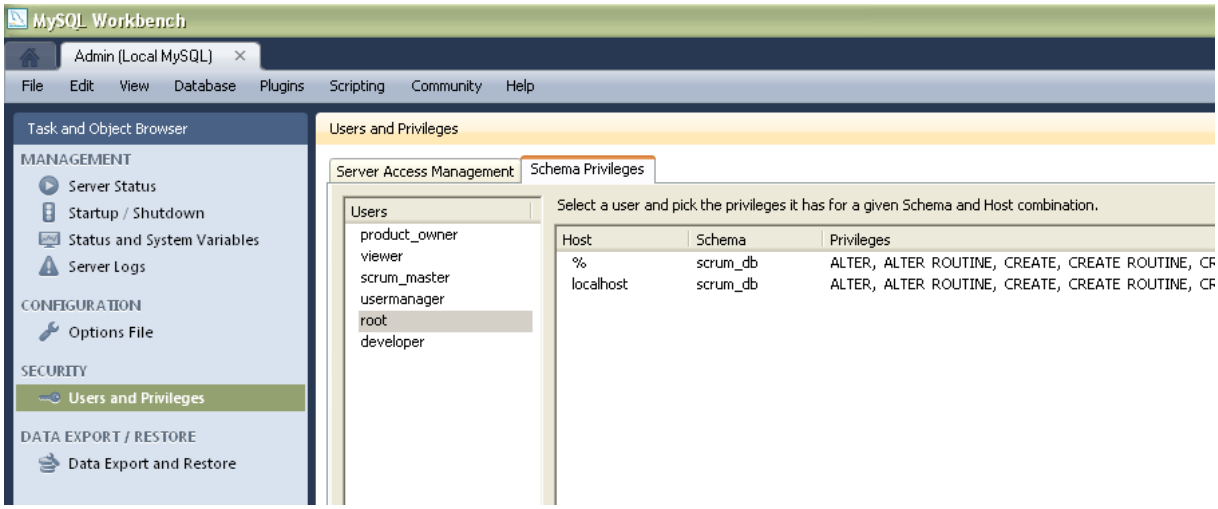
- Vklop ali izklop posameznih razširitev, ki podpirajo različne funkcionalnosti, kot na primer manipulacija z nizi, povezljivost s podatkovno bazo MySQL ipd. s pomočjo direktive `extension`.
- Nastavitev razhroščevalnika, ki nima vloge za izvajanje spletne aplikacije, a je bistvena za podporo implementaciji zaradi nujno potrebne funkcionalnosti odkrivanja napak v povezavi z razvojnim okoljem NetBeans.

```
[PHP_GETTEXT]
extension=php_gettext.dll
 [PHP_MBSTRING]
extension=php_mbstring.dll
[PHP_MYSQL]
extension=php_mysql.dll
[PHP_MYSQLI]
extension=php_mysqli.dll
 [PHP_PDO_MYSQL]
extension=php_pdo_mysql.dll

;XDebugger
zend_extension = "C:\Program Files (x86)\PHP\ext\php_xdebug-2.1.1-5.3-vc9.dll"
xdebug.remote_enable=on
xdebug.remote_handler=dbgp
xdebug.remote_host=localhost
xdebug.remote_port=9000
```

Slika 13: Bistvene nastavitve PHP (del datoteke `php.ini`)

Poleg osnovne namestitve podatkovne baze MySQL je pomembno, da se ustrezno nastavijo uporabniki in nivoji dostopa do podatkovne baze, ki vsebuje vse podatke aplikacije. Z uporabo orodja MySQL Workbench 5.2.34 to lahko relativno enostavno naredimo (na slika 14).



Slika 14: Uporabniki in nivoji dostopa do podatkovne baze – orodje MySQL Workbench

Bistvenega pomena je, da se nastavijo sledeči uporabniki:

- root,
- product_owner,
- scrum_master,
- usermanager,
- developer,
- viewer.

Za pravilno delovanje orodja je potrebno nastaviti gesla omenjenih uporabnikov na isto vrednost, kot so njihova uporabniška imena. To je seveda mogoče in za dejansko uporabo orodja potrebno spremeniti, vendar pa se v podrobnosti tega ne bomo spuščali. Pomembno je omeniti, da osnovna ideja leži v vlogah, ki jih uporabniki orodja lahko imajo. Omenjeni uporabniki v resnici niso uporabniki orodja, temveč predstavljajo le vloge, ki jih uporabniki lahko imajo. Zato je osnovni namen omenjene funkcionalnosti vgradnja dodatnega varnostnega mehanizma za pravice dostopa, vpogleda ali spreminjanja podatkov v podatkovni bazi.

Kot bomo videli v nadaljevanju, se do podatkovne baze dostopa z različnimi klici funkcij razširitve MySQL v kodi PHP. Te omogočajo neposredno izvedbo stavkov strukturiranega povpraševalnega jezika za delo s podatkovnimi bazami (ang. *Structured Query Language*, skrajšano SQL) ali izvedbo baznih procedur (ang. *stored procedures*). Pred izvedbo operacije nad podatkovno bazo je potrebno ustvariti povezavo z ustreznim vpisom. Če vpis ni opravljen ali je zavrnjen, potem povezava ni ustvarjena in je izvedba kakršne koli operacije nad

podatkovno bazo zavrnjena. Omenimo še, da je podatkovno bazo MySQL možno namestiti na drugi fizični lokaciji, saj funkcija za ustvarjanje povezave vsebuje dodaten parameter, ki je naslov računalnika, na kateri je podatkovna baza nameščena.

5.4 Podatkovni model orodja

5.4.1 Uvod

Podatkovni model za podporo metode Scrum je bil narejen na podlagi zajetih zahtev. Uporabljeno je bilo orodje MySQL Workbench 5.2.34 (slika 14), ki poleg grafičnega načrtovanja podatkovnega modela omogoča tudi administracijo podatkovnih baz MySQL.

Na sliki 15 je predstavljena celotna struktura podatkovnega modela. Zaradi obsežnosti bodo v nadaljevanju podrobneje opisani podatkovni model, ključne tabele in polja kot tudi najbolj pomembne povezave med njimi. Izpostavljeni bodo tudi nekateri problemi in ustrezne rešitve predvsem v povezavi z merjenjem/analiziranjem aktivnosti znotraj časovno in/ali funkcionalno zaključene enote. Nekaj besed namenimo baznim proceduram in podatkovnim pogledom (ang. *database view*), ki so bile posebej narejene/implementirane v sklopu podatkovne baze. Namen je bil predvsem delni prenos funkcionalnosti na stran podatkovne baze za lažjo in manj obsežno implementacijo kode na strani strežnika. Zaradi obsežnosti se v podrobnosti baznih procedur in podatkovnih pogledov ne bomo spuščali.

5.4.2 Opis osnovnih značilnosti podatkovnega modela

Najprej se je potrebno seznaniti z dejstvom, da orodje omogoča vodenje več projektov (tabela *Project*). Za vsak projekt lahko planiramo več izdaj (tabela *Release*). Vsako izdajo izdelamo v več iteracijah/Sprintih (tabela *Sprint*). Zahteve uporabnika so predstavljene v obliki uporabniških zgodb (tabela *UserStory*). Za vsako zgodbo je potrebno določiti prioriteto (tabela *UserStory_Priority*). Vsebino vsake izdaje določimo tako, da zgodbe razporedimo po posameznih iteracijah/Sprintih upoštevajoč njihovo prioriteto. Zgodbe, ki niso končane v enem Sprintu, lahko prenesemo v naslednji Sprint, kar pomeni, da ista zgodba lahko nastopa v več Sprintih (tabela *Sprint_UserStory*). Vsako uporabniško zgodbo znotraj Sprinta razdelimo na naloge, ki jih je treba opraviti za njeno realizacijo (tabela *Task*). Skupaj z nedokončanimi zgodbami se v naslednji Sprint prenesejo tudi nedokončane naloge. Zato lahko ista naloga nastopa v več Sprintih. Za vsako nalogo je treba določiti njenega izvajalca (tabela *Employee*) in tip naloge (tabela *Task_Type*) ter zasledovati trenutni status (tabela *Task_Status*). Tip naloge pove, za katero vrsto dela gre (npr. analiza, načrtovanje, kodiranje, testiranje ipd.). Na ta način omogočimo beleženje (in analizo) vloženega dela za vsako od navedenih vrst dela posebej.

Tabela *Measure* predstavlja šifrant vseh možnih metrik, ki jih nameravamo uporabiti za merjenje učinkovitosti razvojnega procesa. Vsebina tabele ni predpisana vnaprej, ampak jo lahko določi vsak uporabnik v skladu s svojimi potrebami.

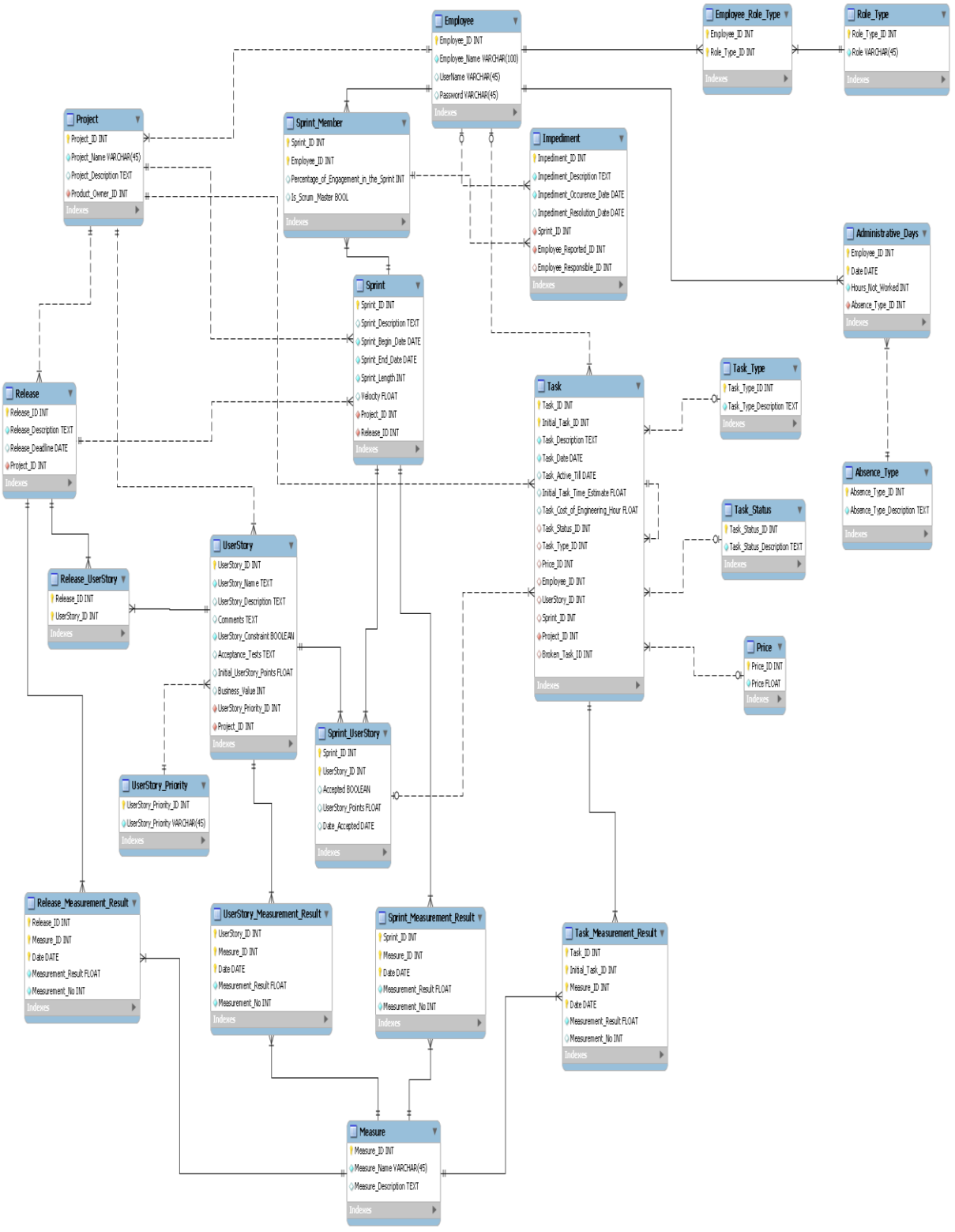
Podatkovni model predpostavlja, da metrike zahtevajo izvajanje meritev na različnih ravneh (na ravni izdaje, Sprinta, uporabniške zgodbe in posamezne naloge), zato je za vsako raven predvidena posebna tabela za hranjenje rezultatov posameznih meritev (tabele *Release_Measurement_Result*, *Sprint_Measurement_Result*, *UserStory_Measurement_Result* in *Task_Measuerement_Result*).

Zaradi boljšega razumevanja podatkovnega modela in samega delovanja orodja omenimo še nekaj detajlov.

Podatkovni model (in tudi samo orodje) omogoča, da se ustvarijo zapisi nalog, ki ne pripadajo nobeni uporabniški zgodbi in s tem posledično niso razvrščene v nobeno izdajo ali Sprint. Gre za dodatno funkcionalnost orodja, saj lahko definiramo »pomožne« naloge, ki so vezane le na določen projekt (povezava med tabelama *Project* in *Task* na sliki 16).

Možno je tudi definirati uporabniške zgodbe le znotraj določenega projekta. Nato jih je lahko razvrstimo v izdaje (tabela *Release_UserStory*) in šele na koncu v Sprinte (tabela *Sprint_UserStory*). Pri tem opozorimo, da je uporabniško zgodbo možno razvrstiti v več izdaj in tudi več Sprintov (predvsem uporabno kadar gre za nedokončanje uporabniške zgodbe v tekočem Sprintu, kar je bolj podrobno obrazloženo v poglavju *5.4.4 Merjenje/analiziranje aktivnosti*). Pri razvrščanju uporabniških zgodb je potrebno paziti: če zelimo zgodbo razvrstiti v Sprint, ki pripada naslednji oziroma neki drugi izdaji, jo je potrebno najprej razvrstiti v to izdajo in šele nato v Sprint znotraj te izdaje.

Poleg opisanega je potrebno še opozoriti, da lahko nalogo, ki hočemo, da je vezana na uporabniško zgodbo (torej ne pomožno nalogo, ki je vezana le na projekt), lahko ustvarimo šele, ko je uporabniška zgodba razvrščena v izdajo in tudi v Sprint (zapisa uporabniške zgodbe v tabelah *Release_UserStory* in *Sprint_UserStory*).



Slika 15: Podatkovni model orodja

5.4.3 Bistvene tabele in polja podatkovne baze

V nadaljevanju bodo nekoliko detajlneje predstavljene sledeče tabele podatkovne baze:

- *Project*,
- *Release*,
- *Release_UserStory*,
- *Sprint*,
- *Sprint_UserStory*,
- *UserStory*,
- *UserStory_Priority*,
- *Task*,
- *Task_Type*,
- *Task_Status*,
- *Task_Measurement_Result* in
- *Measure*.

Tabela *Project*

Tabela *Project* (slika 16) vsebuje zapise projektov. Brez prvotnega vnosa zapisa projekta ni mogoče vnašati zapisov izdaj, Sprintov nalog ali meritev, saj so le-te vezane neposredno ali posredno na tabelo *Project*. Polja tabele so sledeča:

- Polje *Project_ID* je tipa *INT* in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- *Project_Name* je polje tipa *TEXT* in predstavlja tekstualni opis ali ime projekta. Ob vnosu zapisa vrednost polja ne sme biti prazno.

- *Project_Description* je polje tipa *TEXT* in predstavlja tekstualni opis projekta.

- Polje *Product_Owner_ID* je tipa *INT* in enolično določa zapis v tabeli *Employee*, ki vsebuje seznam vseh zaposlencev. S tem poljem je določen lastnik produkta (ang. *Product Owner*) oziroma projekta.

Tabela *Release*

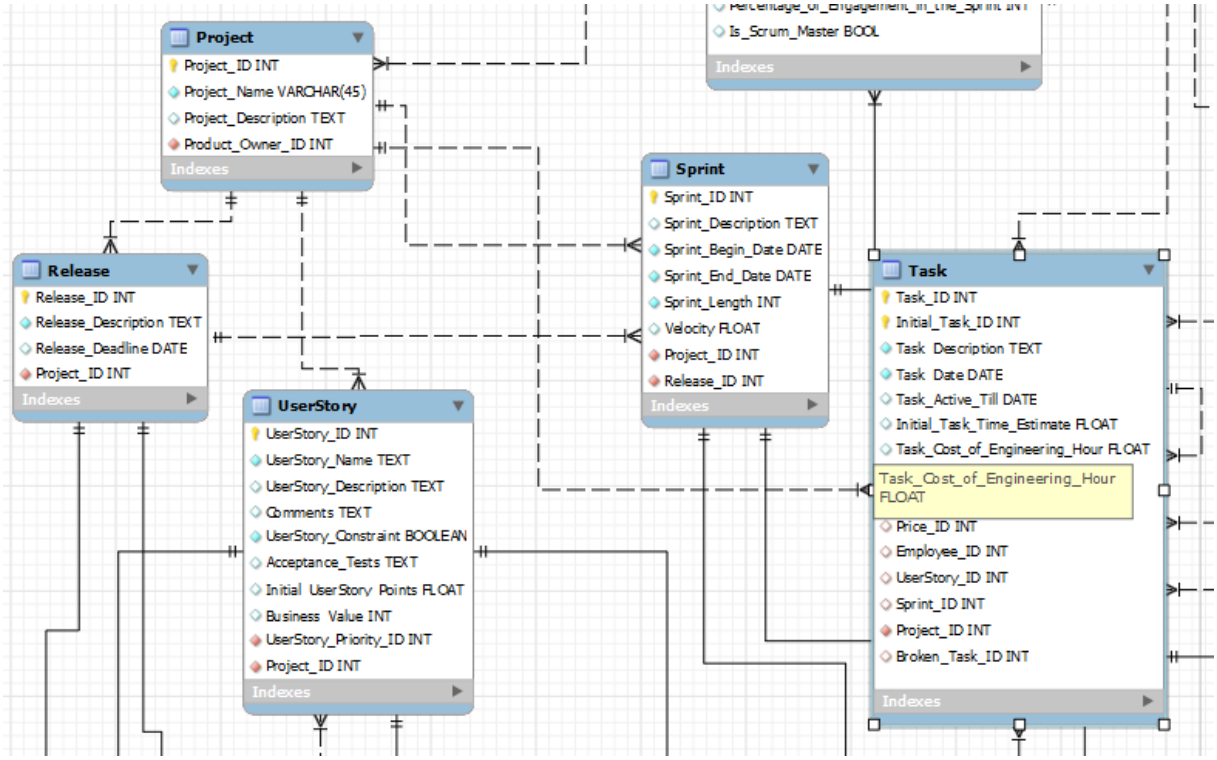
Tabela *Release* (prikazano na sliki 16) vsebuje vse zapise izdaj. Vsak zapis izdaje je vezan na posamezen zapis projekta. Polja tabele so sledeča:

- *Release_ID* je polje tipa *INT* in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- *Release_Description* je polje tipa *TEXT* in predstavlja tekstualni opis ali ime izdaje. Ob vnosu zapisa vrednost polja ne sme biti prazno.

- *Release_Deadline* je polje tipa *DATE* in predstavlja predviden datum zaključka izdaje.

- *Project_ID* je polje tipa *INT* in enolično določa zapis v tabeli *Project*. S tem poljem je določeno, kateremu zapisu projekta pripada posamezen zapis izdaje (slika 16). Ob vnosu zapisa vrednost polja ne sme biti prazno.



Slika 16: Povezava tabele *Project* s tabelami *Release*, *Sprint*, *UserStory* in *Task*

Tabela *Release_UserStory*

Tabela *Release_UserStor*y (slika 19) je pomožna tabela, ki omogoča, da je uporabniško zgodbo mogoče vezati na več izdaj, vsaka izdaja pa lahko ima več uporabniških zgodb. Polja tabele so sledeča:

- Polji *Release_ID* in *UserStory_ID* predstavljata sestavljeni primarni ključ tabele, hkrati pa (vsak zase) določata pripadajoča zapisa v tabelah *Release* oziroma *UserStory*.

Tabela *Sprint*

Tabela *Sprint* (slika 16) vsebuje vse zapise Sprintov. Vsak zapis Sprinta mora pripadati določenemu projektu in izdaji. Bistvena polja tabele so sledeča:

- *Sprint_ID* je polje tipa *INT* in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- *Sprint_Description* je polje tipa *TEXT* in predstavlja opis Sprinta.

- *Sprint_Begin_Date* je polje tipa *DATE* in predstavlja datum začetka Sprinta. Vrednost polja v zapisu ne sme biti prazno.

- *Sprint_End_Date* je polje tipa *DATE* in predstavlja datum konca Sprinta. Vrednost polja v zapisu ne sme biti prazno.

- *Project_ID* in *Release_ID* polji sta tuja ključa, ki kažeta (vsak posebej) na zapis v tabelah *Project* in *Release*. Z njima je določeno, kateremu projektu in izdaji Sprint pripada. Vrednost obeh polj v zapisu ne sme biti prazno.

Tabela *Sprint_UserStory*

Tabela *Sprint_UserStor*y (slika 19) je pomožna tabela, ki omogoča, da en Sprint lahko vsebuje več uporabniških zgodb, hkrati pa ista uporabniška zgodba lahko pripada več Sprintom. V poglavju *5.4.4 Merjenje/analiziranje aktivnosti* je »razširjena« uporabnost tabele

v podatkovnem modelu, predvsem v povezavi z meritvami, bolj podrobno predstavljena. Bistvena polja tabele so sledeča:

- Polji *Sprint_ID* in *UserStory_ID* predstavljata sestavljen primarni ključ tabele, hkrati pa (vsak zase) določata pripadajoča zapisa v tabelah *Sprint* oziroma *UserStory*.

Tabela *UserStory*

Tabela *UserStory* (slika 16) vsebuje vse zapise uporabniških zgodb. Omenimo, da uporabniška zgodba mora »pripadati« projektu, ni pa nujno, da je tudi razvrščena v izdaje (preko tabele *Release_UserStory*) in Sprinte (preko tabele *Sprint_UserStory*), vsaj ne ob prvem vnosu zapisa. Bistvena polja tabele so sledeča:

- *UserStory_ID* je polje tipa INT in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- *UserStory_Name* je polje tipa *TEXT* in predstavlja ime uporabniške zgodbe. Ob vnosu zapisa vrednost polja ne sme biti prazno.

- *UserStory_Description* je polje tipa *TEXT* in predstavlja tekstualni opis uporabniške zgodbe.

- *Comments* je polje tipa *TEXT* in predstavlja prostor za dodaten tekstualni opis (poseben komentar, opomba in podobno) uporabniške zgodbe.

- Polje *Initial_UserStory_Points* je tipa *FLOAT* in predstavlja začetno oceno kompleksnosti oziroma začetno oceno števila točk posamezne uporabniške zgodbe. Namen je predvsem določitev, katere uporabniške zgodbe vključiti v posamezen Sprint, saj obstaja zgornja meja (seštevek teh vrednosti), ki jo lahko nek tim v enem Sprintu obvladuje.

- *Business_Value* je tipa *INT* in predstavlja oceno poslovne vrednosti (oziroma vrednost za stranko) posamezne uporabniške zgodbe.

- Polje *UserStory_Priority_ID* je tipa *INT* in predstavlja tuji ključ, ki kaže na zapis v tabeli *UserStory_Priority*. Z vrednostjo tega polja, ki ne sme biti prazno, je določena prioriteta posamezne uporabniške zgodbe (na primer: »Must have«, »Should have« in podobno).

- Polje *Project_ID* je tipa *INT* in predstavlja tuji ključ, ki kaže na zapis v tabeli *Project*. Vrednost polja v zapisu naloge ne sme biti prazna, saj je z njo enolično določeno, kateremu projektu uporabniška zgodba pripada.

Tabela *UserStory_Priority*

Tabela *UserStory_Priority* vsebuje vse zapise vrst prioritet, ki jih posamezna uporabniška zgodba lahko zavzame.

- *UserStory_Priority_ID* je polje tipa *INT* in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- UserStory_Priority je polje tipa *TEXT* in predstavlja dejansko ime prioritete. Vrednost tega polja ne sme biti prazno. Zapisi s privzetimi vrednostmi v tej tabeli so sledeči (po padajoči prioriteti):

    - *Must have*,
    - *Would have*,
    - *Should have* in
    - *Won't have this time.*

Tabela Task

Tabela *Task* (slika 16) vsebuje vse zapise nalog, ki pripadajo določenemu projektu, uporabniški zgodbi ali Sprintu. Vsak zapis v tabeli podatkovne baze vsebuje dodatne informacije, ki bolj podrobno opisujejo posamezno nalogo. Bistvena polja tabele so sledeča:

- *Task_ID* – polje je tipa *INT* (32 bitno število) in predstavlja del sestavljenega primarnega ključa (skupaj s poljem *Initial_Task_ID*) tabele, ki enolično označuje zapis naloge v podatkovni bazi. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena. Vrednost tega polja v nobenem primeru ne sme biti prazno (ang. *not null*).

- *Initial_Task_ID* – predstavlja del sestavljenega primarnega ključa (skupaj s poljem *Task_ID*). Polje je tipa *INT*, namen pa je na najelegantnejši in najpreprostejši način

rešiti problem prestavljanja nalog v druge izdaje in/ali Sprinte. Ta problem nastopi, če nekatere naloge ne dokončamo v enem samem Sprintu, temveč se mora le-ta nadaljevati v naslednjem Sprintu ali celo izdaji. Privzeta vrednost polja *Initial_Task_ID* je enaka vrednosti v polju *Task_ID*, kar pomeni, da gre za običajen vnos naloge. V primeru, da se ti dve vrednosti razlikujeta, pa to pomeni, da je naloga, na katero kaže polje *Initial_Task_ID,* bila prestavljena oziroma kopirana. Podrobnosti problema prestavljanja nalog v druge izdaje in/ali Sprinte so opisane v poglavju *Merjenje/analiziranje aktivnosti*.

- Polje *Broken_Task_ID* je tipa *INT,* njen primarni namen je ohraniti referenco na nalogo, ki je bila preobsežna in jo je zato bilo smiselno razdeliti na več manjših, hkrati pa je bilo vloženega nekaj dela na njej. Problem razdelitve nalog na več manjših je bolj podrobno obrazložen v poglavju *5.4.4 Merjenje/analiziranje aktivnosti*.

- Polje *Task_Description* je tipa *TEXT* (običajen tekst) in predstavlja tekstualni opis naloge. Ob vnosu zapisa to polje ne sme biti prazno.

- Polji *Task_Date in Task_Active_Till* sta tipa *DATE* (datum) in označujeta začetek in konec veljavnosti določene naloge. Program mora poskrbeti, da sta ta datuma znotraj veljavnosti projekta, izdaje in Sprinta, katerim naloga pripada (pripadnost je določena z vrednostmi polj *Project_ID, UserStory_ID* in *Sprint_ID*). Polje *Task_Date* ob vnosu zapisa ne sme biti prazno.

- Polje *Initial_Task_Time_Estimate* je tipa *FLOAT* in predstavlja prvotno oceno truda v urah, ki ga je potrebno vložiti za dokončanje naloge.

- Polje *Task_Status_ID* je tipa *INT* in predstavlja tuji ključ, ki kaže na zapis v tabeli *Task_Status* (ponazorjeno na sliki 18). Z vrednostjo polja je označen trenutni status določene naloge.

- Polje *Task_Type_ID* je tipa *INT* in predstavlja tuji ključ, ki kaže na zapis v tabeli *Task_Type* (ponazorjeno na sliki 18). Z vrednostjo polja je označen trenutni tip določene naloge.

- *Employee_ID je polje tipa INT* in predstavlja tuji ključ, ki kaže na zapis v tabeli, v kateri se nahaja seznam zaposlencev (tabela *Employee*). Z vrednostjo polja se določi odgovorna oseba za dokončanje naloge.

- Polji *UserStory_ID* in *Sprint_ID* sta tipa *INT* in predstavljata tuji ključ, ki kaže na zapis v tabeli *Sprint_UserStory*. Vrednosti obeh polj sta hkrati lahko ali izpolnjeni (kadar naloga pripada uporabniški zgodbi in Sprintu) ali prazni (kadar gre za pomožno

vrsto naloge vezano le na projekt, ki ni del nobene uporabniške zgodbe, posledično pa tudi ne izdaje ali Sprinta). Z vrednostmi v obeh poljih je enolično določeno, kateri izdaji in Sprintu naloga pripada.

- Polje *Project_ID* je tipa *INT* in predstavlja tuji ključ, ki kaže na zapis v tabeli *Project*. Vrednost polja v zapisu naloge ne sme biti prazno, saj je z njo enolično določeno, kateremu projektu naloga pripada. Omenimo, da naloga mora »pripadati« projektu, medtem ko pa ni nujno, da je tudi razvrščena v izdajo in Sprint, vendar le v primeru kadar gre za pomožno vrsto naloge vezano le na projekt, ki ni del nobene uporabniške zgodbe, posledično pa tudi ne izdaje ali Sprinta.

Tabela *Task_Type*

Tabela *Task_Type* vsebuje zapise, ki opredeljujejo vrsto naloge, kot na primer »Coding«, »Testing« ali celo »Documentation«. Zapisi v tej tabeli služijo predvsem kot izbira med vrstami nalog ob vnosu zapisa naloge (v tabeli *Task*). Polja tabele so sledeča:

- Polje *Task_Type_ID* je tipa *INT* in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- *Task_Type_Description* je polje tipa TEXT in predstavlja dejanski opis naloge, kot na primer: »Coding«, »Testing«, »Documentation« in podobno. Vrednost tega polja mora biti izpolnjeno.

Tabela *Task_Status*

Tabela *Task_Status* vsebuje zapise, ki opredeljujejo trenutni status nalog. Zapisi v tej tabeli služijo predvsem kot izbira med statusi nalog ob vnosu ali spreminjanju zapisa naloge (v tabeli *Task*). Polja tabele so sledeča:

- *Task_Status_ID* je polje tipa INT in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezene zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- *Task_Status_Description* je polje tipa TEXT in predstavlja opis trenutnega statusa določene naloge. Vrednost tega polja mora biti izpolnjeno. Zapisi s privzetimi

vrednostmi v tej tabeli so sledeči:

- *Not Started* – delo na nalogi se še ni začelo.
- *In Progress* – delo na nalogi trenutno poteka.
- *Completed* – naloga je končana.
- *Omitted* – naloga je opuščena.
- *Moved Into Next Sprint* – naloga je prestavljena v naslednji Sprint (ali morda tudi izdajo).
- *Broken Into Smaller Tasks* – naloga je razdeljena na več manjših.

Tabela *Task_Measurement_Result*

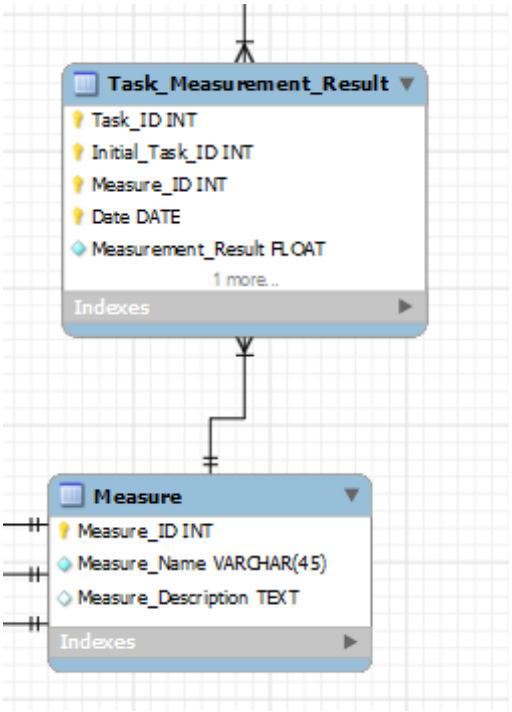
Tabela *Task_Measurement_Result* (slika 17) vsebuje vse zapise meritev, ki se nanašajo na posamezno nalogo. V poglavju Merjenje/analiziranje aktivnosti je vloga in uporabnost tabele v podatkovnem modelu podrobneje predstavljena. Bistvena polja tabele so sledeča:

- Polji *Task_ID* in *Initial_Task_ID* sta tipa *INT* in enolično določata zapis v tabeli *Task* (ponazorjeno na sliki 17). Namen teh polj je določiti, kateri nalogi pripada posamezen zapis meritve. Vrednosti obeh polj morata biti izpolnjeni.

- Polje *Measure_ID* je tipa *INT* in enolično določa zapis v tabeli *Measure*, kot je prikazano na sliki 17. S tem poljem je označeno, za katero vrsto meritve gre, saj so v tabeli *Measure* shranjene vse vrste meritev – kot na primer »Hours Spent« (število ur vloženega dela) in »Hours Remaining« (število ur, ki je še preostalo za zaključek dela na nalogi).

- Polje *Date* je tipa *DATE* in predstavlja datum posamezne meritve. Vrednost tega polja mora biti izpolnjeno.

- Polje *Measurement_Result* je tipa *FLOAT* in predstavlja dejansko vrednost meritve v odvisnoti od njene vrste.



Slika 17: Povezava med tabelami *Measure* in *Task_Measurement_Result*

Tabela *Measure*

Tabela *Measure* (ponazorjeno na sliki 17) vsebuje vse zapise, ki predstavljajo vrste meritev, ki jih je možno vnesti v podatkovno bazo. Polja tabele so sledeča:

- *Measure_ID* je polje tipa INT in predstavlja primarni ključ tabele, ki enolično označuje vsak posamezen zapis v tabeli. Ob vnosu zapisa v tabelo to polje dobi vrednost, ki je od zadnjega zapisa večje za ena.

- Polje *Measure_Name* je tipa *TEXT* in vsebuje ime posamezne meritve. Vrednost tega polja mora biti izpolnjeno. Zapisi s privzetimi vrednostmi v tej tabeli so sledeči:

    - *Hours Remaining* – označuje vrsto meritve, pod katero se v zapisu v tabeli *Task_Measurement_Result* vpiše število ur, ki je še preostalo za zaključek naloge.
    - *Hours Spent* - označuje vrsto meritve, pod katero se v zapisu v tabeli *Task_Measurement_Result* vpiše število ur vloženega dela na izbrani nalogi.

- Polje *Measure_Description* je tipa *TEXT* in predstavlja tekstualni opis posamezne vrste meritve.

5.4.4 Merjenje/analiziranje aktivnosti

V tabeli 1 so predstavljene ključne funkcionalne zahteve, katerim mora programsko orodje zadoščati. Jasno je, da podatkovni model mora izpolnjevati zastavljene funkcionalne zahteve, vendar pa je potrebno omeniti, da se v »ozadju« le-teh skrivajo dodatne zahteve, ki jih na prvi pogled mogoče ni moč zaznati. Iz tega razloga je bilo potrebno dodatno in natančneje definirati zahteve, ki so morale biti vsebovane v okviru osnovnih zahtev. Te se osredotočajo predvsem na zajem in vzdrževanje metrik ter njihovo spremljanje in uporabnost pri določenih izračunih. V nadaljevanju so definirana osnovna izhodišča oziroma zahteve.

Podatkovni model tako mora omogočati najmanj dvoje:

1. Merjenje/analiziranje aktivnosti znotraj neke časovno zaključene enote (na primer izdaje ali sprinta). Primer: Koliko ur dela je bilo opravljenega v nekem Sprintu ali izdaji?

2. Merjenje/analiziranje aktivnosti, ki je vezana na neko funkcionalno zaključeno enoto (na primer naloga, uporabniška zgodba). Primer: Koliko ur dela je bilo potrebnih za realizacijo neke uporabniške zgodbe ali naloge?

S stališča točke 1 moramo vedeti, katere uporabniške zgodbe in naloge spadajo v neko izdajo ali Sprint; izmed meritev, ki pripadajo neki nalogi ali uporabniški zgodbi, pa moramo upoštevati samo tiste, ki se nanašajo na izbrano izdajo ali Sprint.

S stališča točke 2 nastopi problem, če nekatere uporabniške zgodbe ali naloge ne dokončamo v enem samem Sprintu, ampak se mora ta uporabniška zgodba ali naloga nadaljevati v naslednjem Sprintu ali izdaji. V tem primeru mora podatkovni model omogočati, da hranimo podatke o rezultatih meritev tako za uporabniško zgodbo (ali nalogo) kot celoto, kot tudi za njene dele, ki so bili realizirani v posameznih Sprintih.

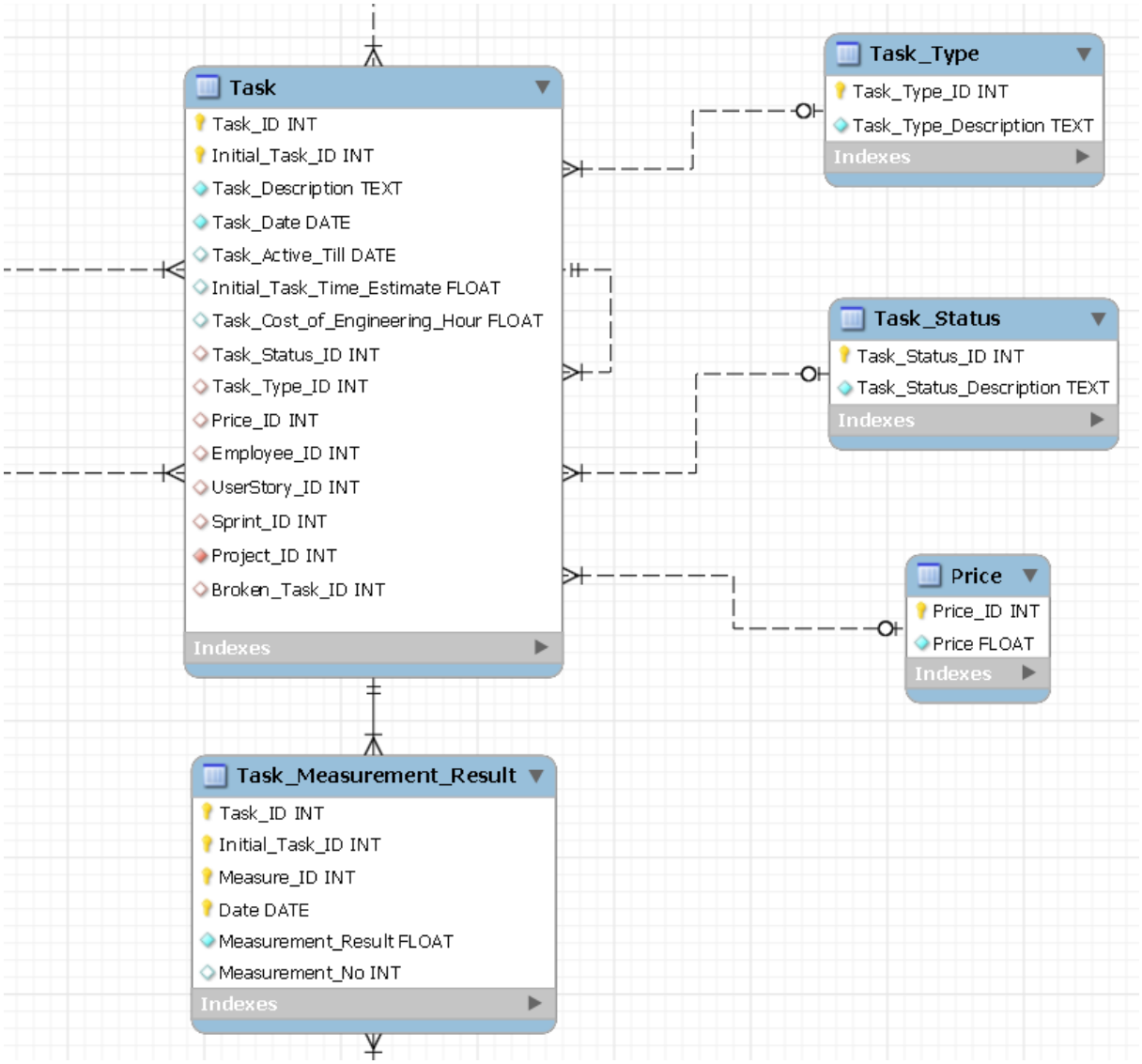
Z drugimi besedami: podatkovni model mora biti zasnovan tako, da omogoča razmejevanje podatkov, ki pridejo v poštev pri analizah, na dva načina:

- glede na to, kateremu Sprintu oziroma izdaji pripadajo in
- glede na to, kateri uporabniški zgodbi in nalogi pripadajo.

Tako je sprejemljiva vsaka rešitev, ki izpolnjuje zgoraj navedene zahteve.

Rešitev omenjenega problema ali drugače povedano izpolnitev omenjenih zahtev je dosežena na naslednji način.

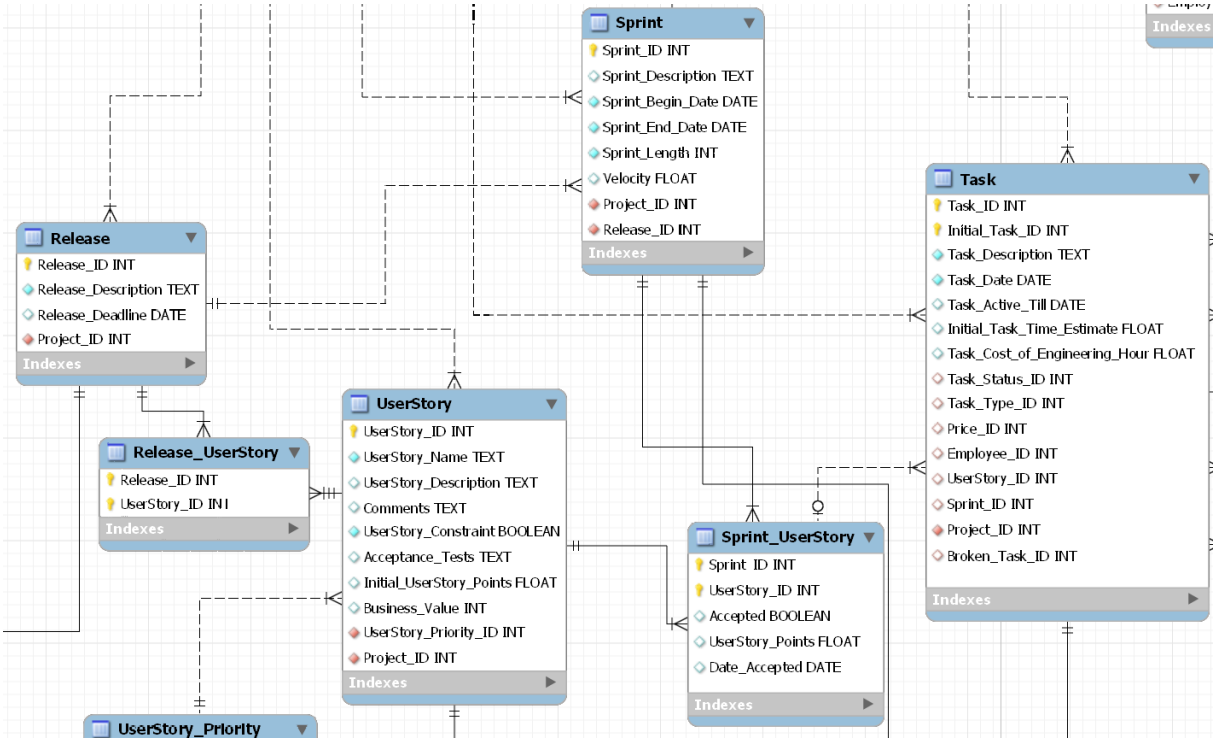
V podatkovnem modelu se uporabi polje *Date* v tabeli *Task_Measurement_Result* (slika 18), s katerim se dejansko loči, kateremu Sprintu ali izdaji pripada določena meritev, ki se nanaša na nalogo. Pri tem je pomembno, da so zapisi datumov pravilni oziroma se vnašajo sproti, sicer analiza ne more zajeti oziroma ne bo zajela vseh meritev pravilno (v program je zato vpeljana tudi dodatna varovalka, tako da se datuma tudi ne da vpisati v podatkovno bazo, če je izven obsega Sprinta ali izdaje). Merjenje/analiziranje znotraj neke časovno zaključene enote (na primer izdaje ali sprinta) je s tem relativno enostavno rešeno, posledično pa je tudi sama implementacija kode enostavna. Potrebno se je le »sprehajati« po seznamu meritev, vrednost zapisana v polju *Date* tabele *Task_Measurement_Result* pa dejansko pove, kateri izdaji in/ali Sprintu meritev pripada. Na ta način je relativno enostavno rešen prvi izpostavljen problem.



Slika 18: Povezava med tabelami *Task* in *Task_Measurement_Result*

Točka 2 zahteva precej več razmisleka, saj je potrebno omogočiti prestavljanje nalog v naslednje izdaje ali Sprinte. To za seboj potegne tudi prestavljanje pripadajoče uporabniške zgodbe. Problem v resnici ni v sami implementaciji prestavljanja, saj bi v tem primeru bilo potrebno le zamenjati ustrezno polje, ki kaže na Sprint ali izdajo. V primeru naloge je to polje *Sprint_ID* v tabeli *Task*, ki predstavlja tuji ključ, ki kaže na zapis v tabeli *Sprint_UserStory*. V primeru uporabniške zgodbe pa je to polje *Sprint_ID*, ki je sestavni del primarnega ključa tabele *Sprint_UserStory*. V tem primeru je pomembno poudariti, da bi bilo najprej potrebno zamenjati ustrezno vrednost polja *Sprint_ID* tabele *Sprint_UserStory* in šele nato vrednost polja *Sprint_ID* v tabeli *Task*. Kakršen koli drugačen pristop bi pomenil vnos napačnih zapisov in tudi napačno obravnavo podatkov s strani programske kode. Kakor koli omenjena rešitev ni sprejemljiva, saj v tem primeru ne bi imeli pravilnega pregleda nad meritvami

predvsem iz stališča uporabnika in tudi ne nad pregledom zapisov, ki so bili prestavljeni. Na primer naloge, ki so bile opravljene v enem Sprintu, v njem ne bi bile več vidne, ker bi celotna uporabniška zgodba bila prestavljena. To pa seveda ni pravilno.



Slika 19: Povezave med tabelami *Release, Release_UserStory, UserStory, Sprint_UserStory, Sprint in Task*

Iz tega razloga je implementirana naslednja rešitev omenjenega problema. Program mora omogočati, da se naloga, ki jo je potrebno prestaviti, v resnici kopira z vsemi ustreznimi vrednostmi v tabelo *Task*. Pri tem je pomembno ohraniti kazalec oziroma referenco na prvotni zapis. Iz tega razloga se uvede dodatno polje *Initial_Task_ID*, ki skupaj s poljem *Task_ID* predstavlja primarni ključ v tabeli Task. V tabeli *Task_Measurement_Result* pa ti dve polji predstavljata del sestavljenega primarnega ključa in hkrati določata pripadajoči zapis v tabeli *Task*. V primeru, da sta vrednosti zapisa naloge v poljih *Task_ID* in *Initial_Task_ID* enaki, je razvidno, da naloga ni bila prestavljena. V primeru, da se ti dve vrednosti razlikujeta, pa je jasno, katera naloga je bila prestavljena oziroma kopirana. Na ta način dosežemo spremljanje nalog skozi zgodovino oziroma izpolnitev prej omenjene zahteve po merjenju/analiziranju

aktivnosti, ki so vezane na neko funkcionalno zaključeno enoto (na primer naloga, uporabniška zgodba).

Prej smo se osredotočili le na problem prestavljanja naloge, vendar nismo omenili, kako lahko prestavimo uporabniško zgodbo. Ta del problema je relativno enostaven zaradi obstoja tabele *Sprint_UserStory*. Gre za vmesno tabelo med tabelami *UserStory* in *Sprint* (razvidno na sliki 19). Polji *Sprint_ID* in *UserStory_ID* v tabeli *Sprint_UserStory* predstavljata sestavljeni primarni ključ in hkrati (vsak zase) določata pripadajoča zapisa v tabelah *Sprint* oziroma *UserStory*. Z zapisi v tabeli *Sprint_UserStory* je omogočeno, da en Sprint lahko vsebuje več uporabniških zgodb, hkrati pa ista uporabniška zgodba lahko pripada več Sprintom. Program torej mora le omogočati, da se v primeru prestavljanja/kopiranja naloge ustvari ustrezen dodaten zapis v tabeli *Sprint_UserStory*, ki veže uporabniško zgodbo na drugi ciljni Sprint.

Omenimo še tabelo *Release_UserStory* s polji *Release_ID* in *UserStory_ID*, ki predstavljata sestavljeni primarni ključ tabele, hkrati pa (vsak zase) določata pripadajoča zapisa v tabelah Release oziroma UserStory. Z njo je uporabniško zgodbo mogoče vezati na več izdaj, vsaka izdaja pa lahko ima več uporabniških zgodb. Podobno kot že prej opisano za Sprint je prestavljanje uporabniških zgodb omogočeno tudi v druge izdaje.

Če povzamememo celotno rešitev prestavljanja uporabniških zgodb ter kopiranja zapisov nalog glede na opisani model podatkovne baze, mora program v celoti poskrbeti, da se pravilno vnesejo ustrezni zapisi v tabelah *Task, Sprint_UserStory in Release_UserStory*.

Poleg omenjenega podatkovni model omogoča, da uporabniške zgodbe ob vnosu zapisa ni potrebno razvrstiti v izdaje in/ali Sprinte. Najprej jih je možno le vnesti v orodje pod izbranim projektom in šele nato razvrstiti v izdaje in Sprinte. To seveda morajo omogočati posebne funkcije orodja.

Podatkovni model orodja omogoča tudi razbitje naloge na več manjših (ang. *Break Into Smaller Tasks*). Uporabi se ponavadi v primerih, kadar se med razvojem ugotovi, da je naloga preobsežna in jo je smiselno razdeliti na več manjših, hkrati pa je bilo vloženega nekaj dela na njej. Za ta namen je uporabljeno polje *Broken_Task_ID*, ki predstavlja enolično referenco na »razbito« nalogo.

V nadaljevanju so v poglavju *6 Grafični uporabniški vmesnik, funkcije orodja in uporaba na primeru* nekoliko detajlneje predstavljene funkcije, ki izkoriščajo omenjene in ponujene možnosti za implementacijo funkcij orodja na pravilen način.

5.5 Direktorijska in datotečna struktura izvršilne kode orodja

5.5.1 Uvod

Izvršilna koda celotnega orodja se nahaja znotraj glavnega direktorija, ki je razdeljen na več poddirektorijev. V vsakem poddirektoriju se nahajajo datoteke, ki vsebujejo kodo, ki je med seboj bodisi logično povezana bodisi povezana glede na druge skupne lastnosti, kot na primer ali se izvršuje na strežniku ali pa na odjemalcu in podobno. Za pravilno delovanje orodja je potrebno glavni direktorij s celotno strukturo prenesti v korenski direktorij spletnega aplikativnega strežnika Apache, kot je nastavljeno v datoteki `httpd.conf`, ponavadi pa je to v `C:\Program Files\Apache Software Foundation\Apache2.2\htdocs`.

5.5.2 Glavni direktorij

Glavni direktorij celotnega projekta vsebuje sledeče bistvene poddirektorije in datoteke z izvršilno kodo:

- `Backlogs`,
- `Data`,
- `DBConnection`,
- `GUI`,
- `Icons`,
- `Isomorphic`,
- `WriteBacklogs`,
- Datoteka `index.php`,
- Datoteka `logout.php`,
- Datoteka `rest.php`.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script type="text/javascript">var isomorphicDir="isomorphic/"</script>
        <script type="text/javascript" src="isomorphic/system/modules/ISC_Core.js"></script>
        <script type="text/javascript" src="isomorphic/system/modules/ISC_Foundation.js"></script>
        <script type="text/javascript" src="isomorphic/system/modules/ISC_Containers.js"></script>
        <script type="text/javascript" src="isomorphic/system/modules/ISC_Grids.js"></script>
        <script type="text/javascript" src="isomorphic/system/modules/ISC_Forms.js"></script>
        <script type="text/javascript" src="isomorphic/system/modules/ISC_DataBinding.js"></script>
        <script type="text/javascript" src="isomorphic/skins/SilverWave/load_skin.js"></script>
        <script type="text/javascript" src="Data/DataSources.js"></script>
        <script type="text/javascript" src="GUI/ScrumPRSHelper_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleSprintMembers_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleDataGrids_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleUsers_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleAdministrativeDays_gui.js"></script>
        <script type="text/javascript" src="GUI/UserStoriesBacklog_gui.js"></script>
        <script type="text/javascript" src="GUI/SprintBacklog_gui.js"></script>
        <script type="text/javascript" src="GUI/SPIandCPIBacklog_gui.js"></script>
        <script type="text/javascript" src="GUI/FillInMissingMeasurements_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleAbsenceTypes_gui.js"></script>
        <script type="text/javascript" src="GUI/HandlePrices_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleTaskStatuses_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleTaskTypes_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleUserStoryPriorities_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleMeasures_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleReleaseMeasurements_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleSprintMeasurements_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleTaskMeasurements_gui.js"></script>
        <script type="text/javascript" src="GUI/HandleUserStoryMeasurements_gui.js"></script>
    </head>
    <body>
    <?php
        //start / resume the session
        session_start();
        //check if we have a username set - if not destroy all session variables and cookies for just
in case
        if(! $_SESSION['role'])
        {
            $_SESSION = array();
            $_COOKIE = array();
            setcookie("SCRUMSESSIONID", '', time() - 42000);
            setcookie("SCRUMUSER", '', time() - 42000);
            setcookie("SCRUMUSERROLE", '', time() - 42000);
            setcookie("SCRUMEMPLOYEEID", '', time() - 42000);
        }
        //end of section start the session
        $start = '<script type' . '=' . '"' . 'text/javascript' . '"' . ' src=' . '"' .
'GUI/ScrumFramework_gui.js' . '"' . '></script>';
        echo $start;
    ?>

    </body>
</html>
```

Slika 20: Vsebina datoteke `index.php`

Najprej se osredotočimo na datoteke znotraj glavnega direktorija `index.php`, `logout.php` in `rest.php`.

Datoteka `index.php` je napisana v kombinaciji programskih jezikov PHP in HTML. V resnici celotna datoteka predstavlja spletno stran HTML, ki ima kot vsaka druga običajna stran HTML glavo (ang. *head*) in telo (ang. *body*). Kot je razvidno iz slike 20, so v razdelku glava (oznaka `<head>`) zapisane vse datoteke s končnico *jsp*, ki se prenesejo v uporabnikov spletni brskalnik. To so datoteke, ki se nahajajo v poddirektorijih `isomorphic`, `Data` in `GUI`, so napisane v skriptnem jeziku JavaScript in se izvršujejo na strani odjemalca. V telesu strani HTML se nahaja koda PHP, ki se izvrši takoj po prenosu vsebin datotek (te so navedene v glavi strani HTML). Z njo se ustvari nova seja, če že ne obstaja, in sicer s shranjevanjem določenih piškotkov (ang. *cookies*), ki se kasneje uporabijo v programu. Na koncu se spletnemu brskalniku z ukazom `echo` pošlje koda, vsebovana v datoteki `ScrumFramework_gui.js`, ki je dejansko prva, ki se izvrši. Poleg veliko drugih predopravil, ki jih ta koda izvrši, se uporabniku prikaže okno za vpis v orodje.

Datoteka `logout.php` je v celoti napisana v programskem jeziku PHP in vsebuje le funkcije, ki uporabnika izpišejo iz orodja. Pri tem je poskrbljeno, da je vzpostavljena seja izbrisana. V primeru, da se uporabnik bodisi izpiše iz orodja bodisi zapre spletni brskalnik, je uporabniku za ponovni dostop do orodja potrebno vpisati uporabniško ime in geslo. V primeru, da se uporabnik vpiše v orodje in zapre le okno spletnega brskalnika, se mu pri ponovnem dostopu v drugem oknu istega spletnega brskalnika ni potrebno ponovno vpisovati, saj seja v tem primeru ni izbrisana.

Koda datoteke `rest.php` predstavlja osrednji del celotnega projekta. Dejansko gre za neke vrste »usmerjevalnik«, saj so praktično vse zahteve, ki pridejo s strani odjelmalca, naslovljene prav na njo. Datoteka tako podpira veliko funkcionalnosti, ker mora glede na prejeto zahtevo sprožiti ustrezno akcijo (na primer branje ali pisanje v podatkovno bazo) in podati ustrezen odgovor odjemalcu (na primer seznam Sprintov znotraj neke izdaje). Najprej omenimo, da se celotna zahteva odjemalca zapiše v spremenljivko na naslednji preprost način:

```
$request = (empty($_GET) ? $_POST : $_GET);
```

Pomembno je povedati, da v tem primeru spremenljivka `$request` mora vsebovati podatke v formatu JSON, saj se za njeno razčlenjevanje uporabljajo že obstoječe funkcionalnosti PHP. V nasprotnem primeru bi razčlenjevanje sporočila bilo napačno, kot tudi interpretacija akcij, ki se morajo izvršiti. Tako na primer na enostaven način v spremenljivko `$UserName`

shranimo uporabniško ime, ki mora biti del poslanega sporočila (oziroma zahteve) s strani odjemalca v primeru, ko se uporabnik želi vpisati v orodje:

```
$UserName = $request['username'];
```

Na podoben način dostopamo tudi do katerega koli drugega podatka v odjemalčevem sporočilu. Poznati je potrebno le oznako podatka (v omenjenem primeru `'username'`), samo razčlenjevanje pa je, kot že prikazano v jeziku PHP zelo enostavno.

```
public static function createResponse($status, $data, $dataHeader = null)
     {
        $res = array('status' => $status,($status != -4) ? 'data' : 'errors' => $data);

        if (is_array($dataHeader))
        {
            $res['totalRows'] = $dataHeader['totalRows'];
            $res['startRow'] = $dataHeader['startRow'];
            $res['endRow'] = $dataHeader['endRow'];
        }
        return array('response' => $res);
    }
```

Slika 21: Funkcija za pripravo odgovora

Podobno poteka sestavljanje odgovora odjemalcu, ki mora biti prav tako sestavljeno v formatu JSON. To je lahko sporočilo, ki vsebuje informacije, da je prišlo do napake, potrditev, da je bila akcija uspešno izvedena (na primer vnos podatkov v podatkovno bazo) ali pa dejanski podatki, ki so bili prebrani iz podatkovne baze. Na sliki 21 je prikazana funkcija, ki sestavi odgovor, vendar pa je po vsakem klicu omenjene funkcije potrebno poklicati funkcijo `json_encode`, ki je sestavni del funkcij PHP (slika 22).

```
$response = self::createResponse(0, $data, $dataHeader);

$response = json_encode($response);

//return response

echo $response;
```

Slika 22: Preprost primer posredovanje odgovora odjemalcu

S pravim formatom in znanimi ključnimi besedami na obeh straneh dejansko poteka komunikacija med strežnikom in odjemalcem.

Funkcionalno gledano se koda datoteke `rest.php` izvaja v naslednjem vrstnem redu:

- Najprej se shrani sporočilo v določeno spremenljivko, ki se nato razčlenjuje skozi različne veje osnovne funkcije.
- Preverijo se podatki uporabnika (slika 23) – morda gre za vpis v orodje ali pa je seja med odjemalcem in strežnikom že vzpostavljena.
- V primeru, da ne gre za vpis uporabnika (torej seja že obstaja), se začne bolj detajlno razčlenjevanje zahteve. Vzemimo za primer izračun indeksov SPI in CPI. Na sliki 24 je prikazan del kode, ki preveri ključno besedo '`procedure`' in v primeru, da le-ta pod seboj v sporočilu vsebuje ključno besedo '`Create_SPIandCPI_CSV_File`', je to indikator, da je uporabnik sprožil zahtevo za izračun obeh indeksov in ustvarjanje datoteke z ustreznimi podatki. Posledično se pokliče funkcija `WriteSPIandCPI()`, ki je implementirana v datoteki `WriteBacklogs.php` in se nahaja v poddirektoriju `WriteBacklogs`. Ta funkcija bo nekoliko podrobneje obrazložena v nadaljevanju.

```
public function RestMain()
    {
        //require_once 'DBConnection/DBConnection.php';
        require_once 'WriteBacklogs/WriteBacklogs.php';

        //get the request
        $request = (empty($_GET) ? $_POST : $_GET);

        //create log
        $dump = new Dump('dump.txt', false /*true*/);
        $dump->write('open() - ' . $_SERVER['REQUEST_URI']);

        //if we received a login request check credentials
        $roles = array();
        if(($UserName = $request['username']) && ($Password = $request['password']))
        {
            $mysqliConn = InitializeDBAccess('usermanager');
            if(! ($roles = VerifyUserCredentials($UserName, $Password, $mysqliConn)))
            {
                $response = self::loginError();
                $response = json_encode($response);
                return $response;
            }
```

Slika 23: Del kode datoteke `rest.php`, ki preverja uporabnika

```
if($_SESSION['role'])
        {
            $ScrumDBConn = InitializeDBAccess($_SESSION['role']);
            if(strcmp($request['procedure'], "Create_Backlog_CSV_File") == 0)
            {
                $writeBacklog     =     new     WriteBacklogs($ScrumDBConn,     $request['Project_ID'],
$request['Release_ID'], $request['Sprint_ID'],
                                                    json_decode($request['Selected_Measures'], true));
                $writeBacklog->DeleteBacklog();
                $link = $writeBacklog->WriteBacklog();
                $data = null;
                $data = array('link' => $link);
            }
            elseif(strcmp($request['procedure'], "Create_SPIandCPI_CSV_File") == 0)
            {
                $writeBacklog     =     new     WriteBacklogs($ScrumDBConn,     $request['Project_ID'],
$request['Release_ID'], $request['Sprint_ID']);
                $writeBacklog->DeleteBacklog();
                $link = $writeBacklog->WriteSPIandCPI();
                $data = null;
                $data = array('link' => $link);
            }
            elseif(strcmp($request['procedure'], "Fill_In_Missing_Measurements") == 0)
            {
                $writeBacklogPointer   =   new   WriteBacklogs($ScrumDBConn,   $request['Project_ID'],
$request['Release_ID'], $request['Sprint_ID']);
                $writeBacklogPointer->FillInMissingMeasurementsDB();
            }
            elseif(!$request['procedure'])
            {
```

Slika 24: Del kode datoteke `rest.php`, ki povzroči izvršitev nekaterih funkcij na strežniku

5.5.3 Poddirektoriji s kodo, ki se izvršuje na strani odjemalca

V nadaljevanju omenimo poddirektorije, ki vsebujejo datoteke s kodo, napisano v jeziku JavaScript in ki se izvršuje na strani odjemalca – `isomorphic`, `Data` in `GUI`.

V poddirektoriju `isomorphic` se nahajajo datoteke, ki vsebujejo komplet funkcij orodja SmartClient, ki služijo za razvoj (ang. *Software Development Kit*, skrajšano SDK). Kombinacije klicev teh funkcij z določenimi vhodnimi parametri lahko povzročijo avtomatično generiranje ter željeno razmestitev, obliko, velikost, obliko in druge lastnosti posameznih objektov grafičnega uporabniškega vmesnika (glej pogavje *Grafični uporabniški vmesnik, funkcije orodja in uporaba na primeru*).

Poleg omenjenega SmartClientov komplet funkcij za razvoj podpira še uporabne funkcije, ki avtomatično zapolnijo predstavljene objekte s podatki, ki se jim dostavijo. Da bi to lahko relativno enostavno dosegli, je potrebno podrobneje predstaviti obliko podatkov, ki jih želimo predstaviti na grafičnem objektu ali drugače rečeno »kontroli«. Tukaj ne gre za format sporočil, ki v vsakem primeru mora biti JSON, temveč za »razumevanje«, kako oziroma kje in katere podatke naj dotični grafični objekt predstavi. Za ta namen je implementirana datoteka `DataSources.js`, ki se nahaja v poddirektoriju `Data`. Datoteka v resnici odraža

podatkovno bazo, vendar na SmartClientu razumljiv način. Za boljše razumevanje je na sliki 25 prikazan del kode datoteke `DataSources.js`, ki vsebuje definiciji dveh podatkovnih izvorov (ang. *data sources*) `Sprint_UserStory_DS` in `UserStory_DS`, ki se ju s pomočjo klica funkcije `isc.RestDataSource.create(datasource parameter)` iz SmartClientovega kompleta funkcij za razvoj tudi »ustvari«. Razvidno je, da je definicija podatkovnega izvora `UserStory_DS` podobna definiciji tabele `UserStory` v sami podatkovni bazi (glej poglavje *Podatkovni model orodja*), saj vsebuje vse potrebne elemente za pravilno razumevanje podatkovnega modela – primarni in tuji ključi, imena polj podatkov in ustrezni tipi. Potrebno je le opozoriti na nekatere posebnosti. Na primer podatkovni izvor `UserStory_DS` vsebuje dodatno polje `Accepted`, ki ni del tabele `UserStory` v podatkovni bazi. Omenjeno polje je potrebno, ker v sami kodi obstaja implementiran podatkovni pogled (ang. *database view*), ki ustvari prikaz podatkov iz kombinacije več tabel, navidezno pa za uporabnika izgleda, da gre za prikaz podatkov ene same tabele iz podatkovne baze. V tem primeru gre za prikaz podatkov iz dveh podatkovnih tabel: `UserStory` in `Sprint_UserStory`. To izjemo je potrebno poudariti, saj na strani odjemalca ne gre za definicijo podatkovnega modela, temveč le za pravilen prikaz podatkov.

Obrazloženi pristop prikaza podatkov seveda ni edini način, je pa najlažji oziroma ga je moč najhitreje implementirati. Slabost je v tem, da je skorajda celoten podatkovni model izpostavljen odjemalcu, kar predstavlja velik varnostni problem. SmartClient podpira drugačen pristop z zbirko funkcionalnosti, ki omogočajo, da se podatkovni model odjemalcu ne odkrije v celoti, vendar podrobnosti tega presegajo obseg magistrskega dela in se v to ne bomo spuščali.

```
var Sprint_UserStory_DS = {
        ID:"Sprint_UserStory",
        dataFormat: "json",
        dataURL: "rest.php",
        fields:[
            {primaryKey:"true", name:"Sprint_ID", type:"sequence", hidden: "true"},
            {primaryKey:"true",   name:"UserStory_ID",   type:"integer",   hidden:   "true",   required:
"false"},
            {name:"Sprint_Description", length:65535, type:"text"},
            {name:"Sprint_Begin_Date", type:"date", required: "true"},
            {name:"Sprint_End_Date", type:"date", required: "true"},
            {name:"Sprint_Length", type:"integer", required: "true"},
            {name:"Accepted", type:"text", required: "false"},
            {name:"UserStory_Points", type:"float", required: "false"}
        ]
    };
    isc.RestDataSource.create(Sprint_UserStory_DS);

    var UserStory_DS = {
        ID:"UserStory",
        dataFormat: "json",
        dataURL: "rest.php",
        fields:[
            {primaryKey:"true", name:"UserStory_ID", type:"sequence", hidden: true},
            {name:"UserStory_Name", length: 65535, type:"text", required: "true"},
            {name:"UserStory_Description", length:65535, type:"text", required: false},
            {name:"Comments", length:65535, type:"text", required: false},
            {name:"UserStory_Constraint", type:"text", length:45, required: "true"},
            {name:"Acceptance_Tests", length:65535, type:"text", required: false},
            {name:"Initial_UserStory_Points", type:"float", required: false},
            {name:"Business_Value", type:"integer", required: false},
            {name:"UserStory_Priority_ID",                                          type:"integer",
foreignKey:"UserStory_Priority.UserStory_Priority_ID", required: "true"},
            {name:"Project_ID", type:"integer", foreignKey:"Project.Project_ID", required: "true"},
            {name:"Release_ID", type:"integer", required: false},
            {name:"Accepted", type:"text", required: false},//added for mysql view - Sprint_UserStory
            {name:"Date_Accepted",   type:"date",   required:   false},//added   for   mysql   view   -
Sprint_UserStory
            {name:"UserStory_Points",  type:"float",  required:  false},//added  for  mysql  view  -
Sprint_UserStory
            {primaryKey:       true,       name:"Sprint_ID",       type:"integer",       foreignKey:
Sprint_UserStory.Sprint_ID, required: false}//added for mysql view - Sprint_UserStory - primary only
because we want to send its value with a request
        ]
    };
    isc.RestDataSource.create(UserStory_DS);
```

Slika 25: Del kode datoteke `DataSources.js`

```
function GenerateSPIandCPIBacklogCSVFile()
{
    //here we have to send a request to generate a CSV file and to link it to it
    var SPIandCPIBacklogCSV_DS = {
        ID:"SPIandCPIBacklogCSV",
        dataFormat: "json",
        dataURL: "rest.php",
        fields:[
            {name:"link", length: 65535, type:"text"}
        ]
    };
    isc.RestDataSource.create(SPIandCPIBacklogCSV_DS);

    var recRelease= listReleasesGrid.getSelectedRecord();
    var recSprint = listSprintsGrid.getSelectedRecord();
    if (recSprint)
    {
        SPIandCPIBacklogCSV.fetchData({procedure:        "Create_SPIandCPI_CSV_File",        Sprint_ID:
recSprint.Sprint_ID},
                                      function (dsResponse, data)
                                      {
                                           var record = data.get(0);
                                          if(record && record.link)
                                          {
                                              window.open(record.link);
                                          }
                                          if(dsResponse.status == -4)
                                          {
                                              isc.warn("Operation   could   not   be   completed!<br>Please
verify that you have sufficient privileges!");
                                          }
                                      }
        );
    }
    else if(recRelease)
    {
        SPIandCPIBacklogCSV.fetchData({procedure:       "Create_SPIandCPI_CSV_File",       Release_ID:
recRelease.Release_ID},
                                      function (dsResponse, data)
                                      {
                                           var record = data.get(0);
                                          if(record && record.link)
                                          {
                                              window.open(record.link);
                                          }
                                          if(dsResponse.status == -4)
                                          {
                                              isc.warn("Operation   could   not   be   completed!<br>Please
verify that you have sufficient privileges!");
                                          }
                                      }
        );
    }
    else
    {
        isc.warn("Please select Release and/or Sprint record first !");
    }
}
```

Slika 26: Koda datoteke `SPIandCPIBacklog_gui.js`

V poddirektoriju `GUI` se nahajajo datoteke, ki vsebujejo kodo, napisano v skriptnem jeziku JavaScript. Vsebina teh datotek se ob dostopu odjemalca (datoteka `index.php`) prenese na njega in se tam tudi izvaja. Seznam vseh datotek se nahaja v datoteki `index.php`, kar je razvidno iz slike 20. Za boljše razumevanje omenimo prej izpostavljeno funkcijo prožanja zahteve za izračun indeksov SPI in CPI. Na sliki 26 je prikazana funkcija `GenerateSPIandCPIBacklogCSVFile()`, ki se proži ob tovrstni zahtevi uporabnika (preprosto rečeno ob »kliku« uporabnika na funkcijo *SPI and CPI Backlog* grafičnega uporabniškega vmesnika – glej poglavje Grafični uporabniški vmesnik, funkcije orodja in uporaba na primeru, Zbirka funkcij *View*). Funkcija ustvari podatkovni izvor `SPIandCPIBacklogCSV_DS`, ki bo vseboval podatke v formatu JSON, ki se bodo ob klicu funkcije `SPIandCPIBacklogCSV.fetchData` poslali kot vhodni podatki datoteki `rest.php`. Poslani podatki vsebujejo še ključno besedo `Create_SPIandCPI_CSV_File`, ki jo strežnik, kot že prej obrazloženo, zna pravilno izluščiti (datoteka `rest.php`).

V poddirektoriju `Icons` se nahajajo manj pomembni podatki oziroma gre le za slike/ikone, ki se uporabljajo za bolj nazoren grafični prikaz funkcij orodja.

5.5.4 Poddirektoriji s kodo, ki se izvršuje na strani strežnika

Na koncu se dotaknimo še poddirektorijev `DBConnection`, `WriteBacklogs` in `Backlogs`.

Poddirektorij `Backlogs` ne vsebuje nobene kode oziroma je prazen. Služi predvsem kot izbrana lokacija na strani strežniku, kjer se shranjujejo sledeče datoteke ob prožitvi uporabnikove zahteve:

- `UserStoryBacklog.csv` (seznam vseh uporabniških zgodb, ločeno glede na to, kateremu projektu, izdaji ali Sprintu pripadajo – prikazano na sliki 34),

- `SprintBacklog.csv` (seznam vseh nalog izbranega projekta, ločen glede na to, kateri izdaji, Sprintu in uporabniški zgodbi pripadajo vključno z vsemi pripadajočimi meritvami, opravljenimi med razvojem - sliki 34) in

- `SPIandCPI.csv` (izračuni indeksov SPI in CPI po dnevih za vse Sprinte vseh izdaj izbranega pojekta, vključno z delnimi izračuni posameznih nalog po dnevih – prikazano na sliki 35 in obrazloženo v poglavju *6.3 Uporaba orodja na primeru*).

V resnici se po uporabnikovi zahtevi naštete datoteke shranijo na strežniku in pošljejo uporabniku/odjemalcu, ki ima nato možnost, da jih lokalno shrani.

Poddirektorij `DBConnection` vsebuje dve datoteki: `DBConfig.php` in `DBConnection.php`. Obe vsebujeta funkcije za neposredno delo s podatkovno bazo. Implementacija datoteke `DBConfig.php` je precej preprosta, vsebuje pa funkcije, ki služijo nastavitvam za vpis v

podatkovni strežnik z ustreznim uporabniškim imenom in geslom, ki predstavlja vlogo uporabnika v orodju (glej poglavje 5.3 *Osnovne sistemske nastavitve*).

Glede na celotno orodje je datoteka `DBConnection.php` najobsežnejša in precej zahtevna za implementacijo. Vsebuje sledeče pomožne, a izjemno pomembne funkcije:

- `executeStoreProcedure` je funkcija (slika 27), s katero je možno prožiti izvajanje določene bazne procedure. V primeru povratnih podatkov izvedene bazne procedure jih funkcija tudi ustrezno vrne klicatelju.

- Funkcija `executeQuery` (slika 27) omogoča neposredno izvajanje stavka SQL nad izbrano podatkovno bazo. V primeru branja iz podatkovne baze, funkcija prav tako ustrezno vrne podatke klicatelju.

- Funkcija `composeQuery` je najobsežnejša. Edini vhodni parameter funkcije je prejeta zahteva odjemalca. Funkcija zna pravilno razčleniti zahtevo, ki je v formatu JSON. Z znanimi ključnimi besedami pobere dejanske podatke, skrite v zahtevi, na osnovi katerih sestavi stavek SQL. Tega je nato s pomočjo klica funkcije `executeQuery` možno izvesti nad podatkovno bazo.

Poddirektorij `WriteBacklogs` vsebuje le eno datoteko `WriteBacklogs.php` (slika 28). Ta je najzahtevnejša za implementacijo, saj vsebuje funkcije, ki dejansko ustvarijo že prej omenjene datoteke (`UserStoryBacklog.csv`, `SprintBacklog.csv` in `SPIandCPI.csv`), ki se ob proženi zahtevi uporabnika sprva shranijo v poddirektorij `Backlogs`. Poudariti je potrebno, da se v tej datoteki nahaja funkcija `WriteSPIandCPIBacklog()` z dejansko implementacijo algoritma za izračun indeksov SPI in CPI.

```
public function executeQuery($query, & $retLastID = null)
    {
        $retVal = 'false';
        //$ds = array();
        $ds = null;
        $result = $this->query($query);
        $retLastID = $this->insert_id;
        if($result)
        {
            if ($result->num_rows > 0)
            {
                $ds = $result->fetch_all(MYSQLI_BOTH);
                $result->free();
            }
            $retVal = 'true';
        }
        if($ds)
        {
            return $ds;
        }
        return $retVal;
    }

public function executeStoreProcedure($proc_name, $params = null, & $retLastID = null)
    {
        $retVal = 'false';
        //$ds = array();
        $ds = null;
        if($params)
        {
            $query = "CALL " . $proc_name . "('" . implode("', '", $params) . "');";
        }
        else
        {
            $query = "CALL " . $proc_name . ";";
        }
        if($this->multi_query($query))
        {
            $retlastID = $this->insert_id;
            do
            {
                $result = $this->store_result();
                if($result)
                {
                    if ($result->num_rows > 0)
                    {
                        $ds = $result->fetch_all(MYSQLI_BOTH);
                    }
                    $result->free();
                }
                $retVal = 'true';
            } while($this->next_result());
        }
        if($ds)
        {
            return $ds;
        }
        return $retVal;
    }
```

Slika 27: Del kode datoteke `DBConnection.php`

```
        private function WriteSprintBacklog()
        {
            $filePath = "Backlogs/";
            $fileName = "SprintBacklog_" . $this->wb_projectID . $this->wb_sprintID . ".csv";
            if(!opendir("Backlogs"))
                mkdir("Backlogs");

            $this->filePointer = fopen($filePath . $fileName, 'wb');
            if($this->filePointer)
            {
                $sprintData = $this->dbConnection->executeQuery("select * from Sprint where Sprint_ID
= " . $this->wb_sprintID);
                $userStoriesData      =      $this->dbConnection->executeQuery("select      *      from
Sprint_UserStory_View s_us_v where s_us_v.Sprint_ID = " . $this->wb_sprintID);
                $tasksData = null;

                $usLength = count($userStoriesData);
                if($sprintData && $sprintData != 'false' && $sprintData != 'true' &&
                   $userStoriesData && $userStoriesData != 'false' && $userStoriesData != 'true')
                {
                    $this->fileLink = "./Backlogs/" . $fileName;
                    $fileData  =  "User  Story  Name;Initial  Task  ID;Task  Description;Responsible;Task
Type;Task Status;" . ";" . "Initial Task Time Estimate;";
                    $emptyRow = ";;;;;;;;";

                    $measures = $this->dbConnection->executeQuery("select * from Measure");
                    if($measures && $measures != 'false' && $measures != 'true')
                    {
                        $numSprintDays      =      $this->DateDiff($sprintData[0]['Sprint_Begin_Date'],
$sprintData[0]['Sprint_End_Date']);
                        $daysRow = "";
                        for($j = 1; $j <= $numSprintDays; ++$j)
                        {
                            $daysRow = $daysRow . $j .";";
                            $daysRowTmp = $daysRowTmp . ";";
                        }
                        $firstRow = ";;;;;;;;";
                        for($j = 0; $j < count($measures); ++$j)
                        {
                            if(in_array($measures[$j]['Measure_ID'],                                $this-
>selectedMeasures['Measures']))
                            {
                                $firstRow = $firstRow . $measures[$j]['Measure_Name'] . $daysRowTmp;
                                $fileData = $fileData . $daysRow;
                            }
                        }
                        $fileData = $firstRow . "\n" . $fileData . "\n";

                        $measurements = null;
                        $totalSums = ";;;;;;Total Sums;";
                        $measurementsTotalSums = array();
                        $totalInitTaskTimeEstimateSum = 0;
                        for($i = 0; $i < $usLength; ++$i)
                        {
                            $tasksData[$i]  =  $this->dbConnection->executeQuery("select  *  from  Task  t
where t.Sprint_ID = " .
                                                                                $this->wb_sprintID  .  "
and t.UserStory_ID = " .

$userStoriesData[$i]['UserStory_ID']);
```

Slika 28: Del kode datoteke `WriteBacklogs.php`

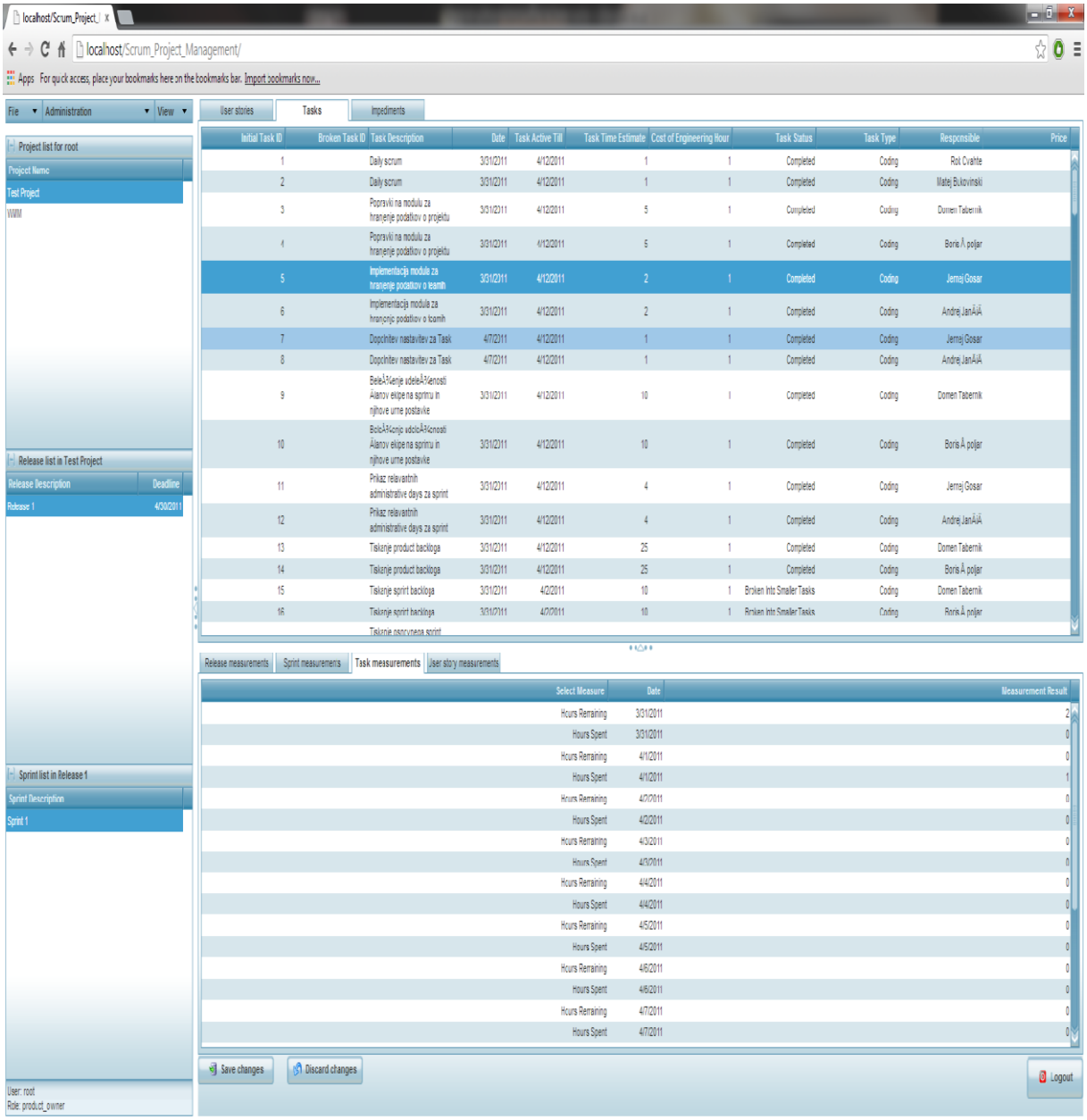
6 Grafični uporabniški vmesnik, funkcije orodja in uporaba na primeru

6.1 Splošno

Grafični uporabniški vmesnik (skrajšano GUV) predstavlja precej pomemben del orodja, saj omogoča osnovno interakcijo med uporabnikom in orodjem. Za izgradnjo niso bile zbrane posebne zahteve, ki bi krojile izgled GUV-a, temveč so bila bila upoštevana nekatera osnovna vodila. Ta so temeljila predvsem na hitri dostopnosti do razpoložljivih funkcij, intuitivnosti uporabe orodja ter visoki preglednosti prikazanih podatkov. Poleg omenjenega je bilo nekaj pozornosti namenjenih tudi estetskemu videzu orodja.

GUV je napisan v programskem jeziku JavaScript, uporablja pa že omenjeno tehnologijo SmartClient. V kodi so uporabljeni klici določenih funkcij, ki so del SmartClientovega kompleta funkcij za razvoj (ang. *Software Development Kit*, skrajšano SDK), katerih klic povzroči avtomatično generiranje dejanskega grafičnega vmesnika. Jasno je, da imajo te funkcije določene parametre, s katerimi se lahko vpliva na sam izgled, način prikaza, lokacijo, razmestitev in druge potrebne funkcionalnosti oziroma lastnosti posameznih objektov vmesnika. Ker se sama koda nahaja na strežniku, jo je potrebno najprej prenesti v odjemalca. Ob dostopu uporabnika s spletnim brskalnikom na glavno spletno stran (v tem primeru se izvrši koda datoteke `index.php`) se koda GUV-a v celoti naloži v odjemalčev brskalnik, nakar je pripravljena za uporabo. Vsak »klik« na neko lokacijo znotraj GUV-a lahko povzroči izvršitev določene JavaScript funkcije. Ta lahko povzroči prikaz lokalnih podatkov ali pa sproži komunikacijo s strežnikom (pošiljanje določenih zahtev po podatkih ali akcijah, ki naj bi se izvedle na strežniku).

Na sliki 29 je prikazan GUV izdelanega orodja za vodenje in merjenje učinkovitosti razvoja programske opreme po metodi Scrum. GUV je v osnovi razdeljen na več prikaznih področij. Na levi zgornji strani se nahaja osnovni meni s sklopom različnih funckij, ki bodo opisane v nadaljevanju: *File*, *Administration* in *View*. Pod menijem se nahajajo trije razdelki, ki prikazujejo seznam projektov, izdaj (ang. *Release*) in Sprintov. Seznam izdaj se prikaže le v primeru, če izberemo določen projekt, seznam Sprintov pa le, če izberemo določeno izdajo. V levem spodnjem kotu orodje prikaže, kateri uporabnik je vpisan (ang. *logged*) v orodje in kakšna je njegova vloga. Ta mu omogoča določen nivo dostopa in spreminjanja podatkov.



Slika 29: Grafični uporabniški vmesnik orodja

Na desni strani se nahajata dva dodatna, a zelo pomembna razdelka: prikaz seznama uporabniških zgodb (ang. *User stories*) in nalog (ang. *Tasks*); spodaj pa je prikaz zajetih meritev izdaj, Sprintov, nalog in uporabniških zgodb.

Prikaz seznama določenih podatkov deluje avtomatično. Ob izbiri oziroma dvokliku na določen projekt iz seznama projektov se avtomatično prikaže seznam vseh izdaj, uporabniških zgodb in nalog izbranega projekta. Z izbiro določene izdaje znotraj projekta se omenjeni seznami posodobijo tako, da se prikažejo le vnosi, ki so del izbrane izdaje. Dodatno

se še prikaže seznam Sprintov, ki sodijo v izbrano izdajo. Z izbiro Sprinta znotraj izdaje se ponovno posodobijo seznam uporabniških zgodb in nalog, saj želimo prikazati le tiste, ki sodijo v izbrani Sprint. Ker določene naloge pripadajo določenim uporabniškim zgodbam, je ob izbiri le-te možno dobiti pripadajoči seznam nalog.

Pomembno je še omeniti, da se ob izbiri katerekoli naloge posodobi seznam vseh zajetih metrik (na sliki 29 *Task measurements*), ki so izjemnega pomena za izračun indeksov CPI in SPI.

6.2 Osnovne funkcije orodja

6.2.1 Menijska vrstica

Osnovni meni sestavljajo tri zbirke različnih funkcij: *File, Administration* in *View*.

Zbirka funkcij *File*

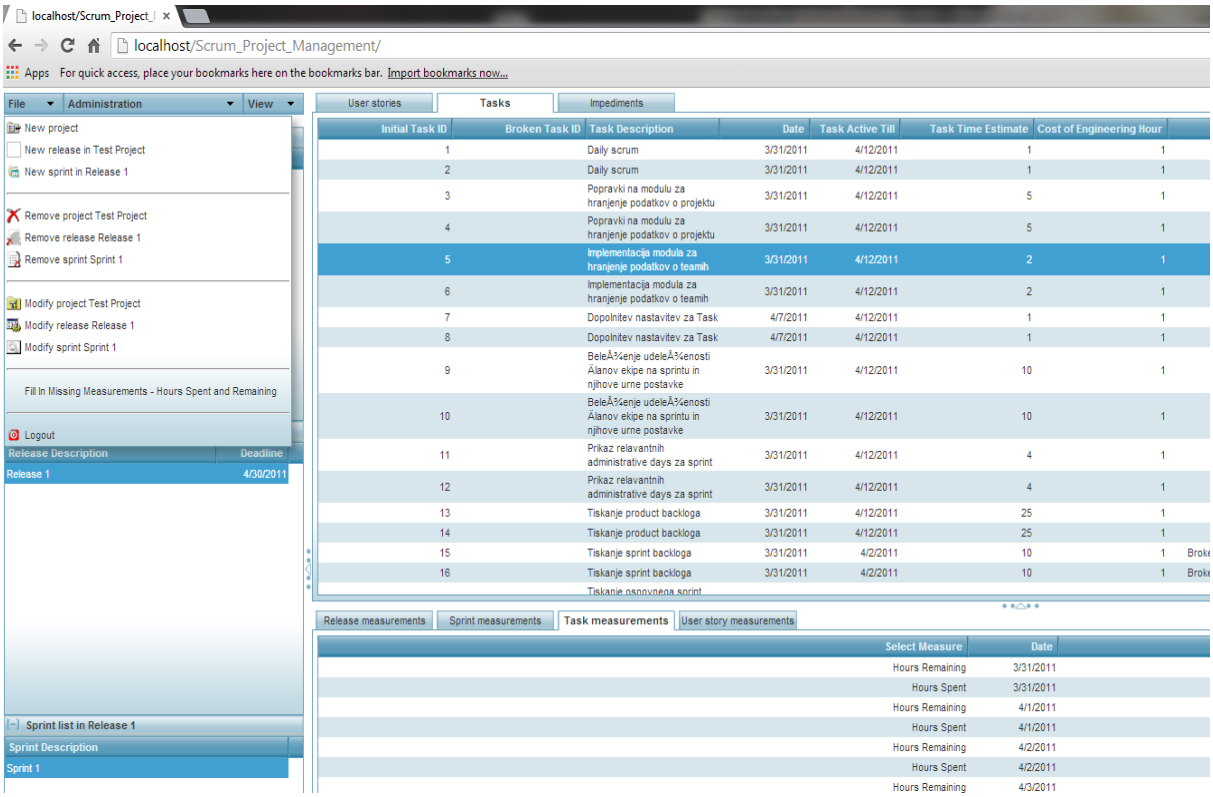
Na sliki 30 je prikazana zbirka funkcij *File*, ki vsebuje osnovne operacije nad projekti, izdajami in Sprinti. Z omenjenimi funkcijami jih je možno ustvariti, brisati ali pa jim spreminjati že vpisane podatke.

Pri ustvarjanju novega projekta je potrebno najprej vpisati osnovne podatke projekta, kot na primer ime projekta in krajši opis. Ob izbiri projekta se avtomatično aktivira (možno jo je sprožiti) opcija za sprožitev funkcije ustvarjanja izdaje z ustreznimi podatki: opis ter datum zaključka izdaje. Podobno velja tudi z ustvarjanjem Sprinta znotraj izdaje. Ta zahteva vpis datuma začetka in konca Sprinta ter poleg ostalih informacij še krajši opis.

Pri brisanju podatkov je potrebno biti izjemno previden, vendar pa so v orodje dodatno implementirani varnostni mehanizmi, ki preprečujejo vzvratno brisanje podatkov oziroma brisanje podatkov po celotni hierarhiji navzdol. Tako na primer ni mogoče izbrisati zapisa naloge, če že obstajajo zapisi meritev, ki so nanjo vezani. Najprej je potrebno izbrisati vse meritve in nato ustrezno nalogo. Prav tako ni mogoče izbrisati zapisa uporabniške zgodbe, če ji pripada vsaj en zapis naloge.

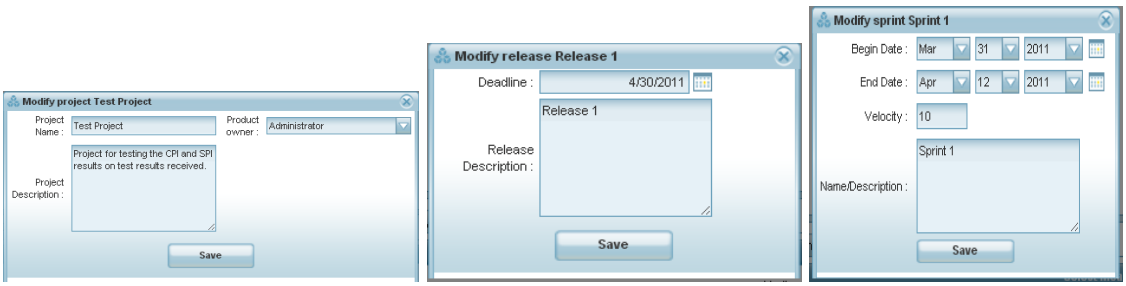
Po drugi strani pa je možno izbrisati izbrani Sprint, vendar le v primeru, da vsebuje samo zapise uporabniških zgodb, od katerih niti ena nima zapisa naloge (in posledično tudi ne zapisov meritev vezanih na pripadajoče naloge). V tem primeru se sicer izbriše zapis Sprinta, vendar pa ne tudi zapisi uporabniških zgodb, ki so mu pripadale. Te še vedno ostanejo del tiste izdaje, kateri je pripadal izbrisan Sprint.

Iz opisanega je razvidno, da je potrebno biti precej vešč oziroma natančno poznati, kaj določena funkcija izbrisa v resnici naredi. Zato je najbolje brisati podatke po hierarhiji navzgor – najprej brisati zapise meritev, nato nalog in ustreznih uporabniških zgodb in šele nato Sprintov, izdaj ter na koncu projektov.



Slika 30: Zbirka funkcij menijske vrstice *File*

Z zbirko funkcij menijske vrstice *File* je možno tudi spreminjati že vnešene podatke projektov, izdaj in Sprintov kot je ponazorjeno na sliki 31.

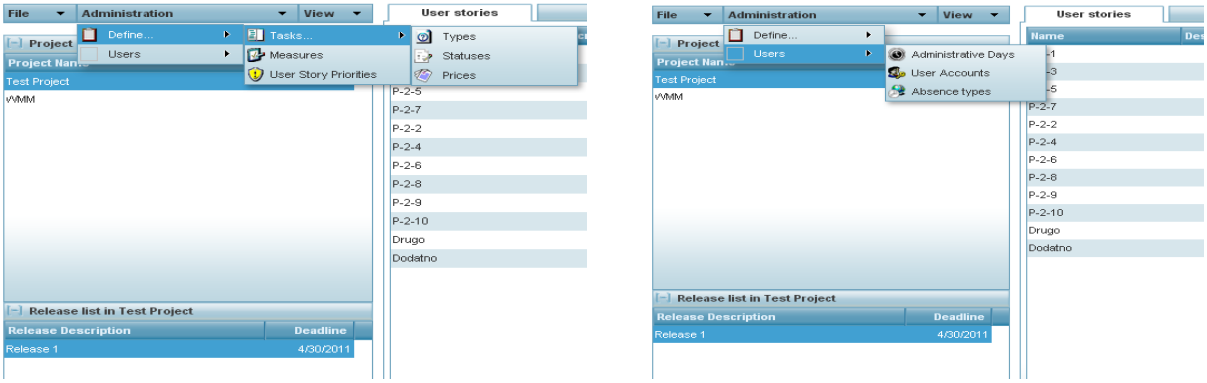


Slika 31: Spreminjanje podatkov projekta, izdaje in Sprinta

Velja omeniti še funkcijo *Fill in Missing Measurements – Hours Spent and Remaining*, ki ima izredno pomembno vlogo pri izračunih indeksov CPI in SPI. Funkcija, ki jo uporabnik izvrši po končanem Sprintu, generira zapise za *Hours Remaining* in *Hours Spent* za vse tiste dni v Sprintu, za katere tega podatka ni. Ta funkcija je relativno enostavna, saj sproži sprehod po datoteki vseh nalog v Sprintu, za vsako nalogo pa preveri vse dni Sprinta od začetka do konca veljavnosti naloge. Če za trenutni dan podatek *Hours Remaining* obstaja, si program zapomni to vrednost kot zadnjo veljavno vrednost, v nasprotnem primeru pa generira nov zapis, v katerem je vrednost *Hours Remaining* enaka zadnji veljavni vrednosti. V tem primeru se ustrezne vrednosti *Hours Spent* postavijo na vrednost 0, saj v omenjenem primeru ni porabljenega dela na nalogi. Ta funkcija omogoča, da za dneve, v katerih ni bilo vloženega nobenega dela na določeni nalogi, ni potrebno ročno vpisovati vrednosti *Hours Remaining in Spent*, temveč se le-te generirajo avtomatično po koncu Sprinta s sprožitvijo omenjene funkcije.

Zbirka funkcij *Administration*

Kot že samo ime pove, je zbirka funkcij menijske vrstice *Administration* (slika 32) namenjena predvsem administrativnim opravilom.



Slika 32: Zbirka osnovnih funkcij menijske vrstice *Administration*

Z njimi je možno urejati in definirati sledeče pomembnejše podatke:

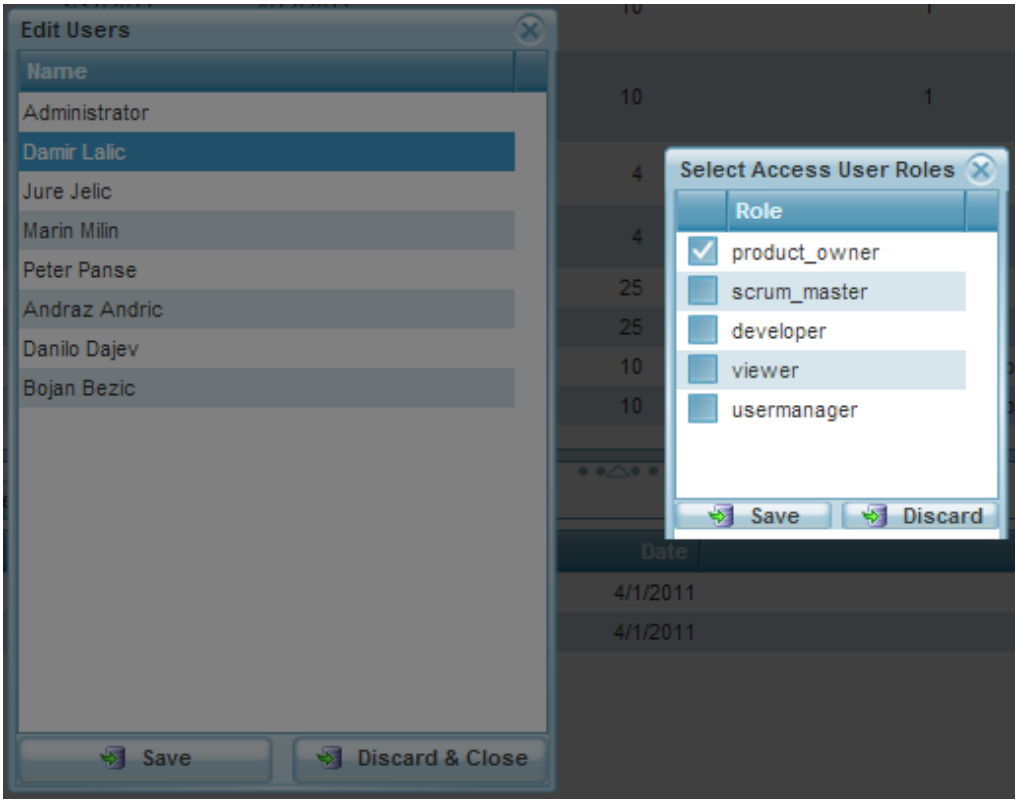
- Tipe oziroma vrste nalog kot na primer: *Coding*, *Testing*, *Documentation* itn. Tukaj definirani tipi nalog se pojavljajo kot izbira pri vnosu nalog v ustrezni seznam.

- Vrste statusov, ki jih naloga med razvojem programske oprema lahko ima, kot na primer: *Not Started*, *In Progress*, *Completed*, *Omitted*, *Moved Into Next Sprint*, *Broken Into Smaller Tasks* ipd. Prav tako se tukaj definirani statusi pojavljajo kot izbira pri vnosu nalog v seznam ali pri spreminjanju trenutnega statusa posamezne naloge.

- Vrste metrik, ki jih zajemamo, kot na primer tiste, ki jih uporabljamo za izračun indeksov CPI in SPI: *Hours Remaining* in *Hours Spent*. Definirane vrste metrik potem uporabnik lahko vnaša v razdelek, namenjen vpisom meritev za posamezne naloge.

- Prioritete uporabniških zgodb, ki se pojavljajo kot ustrezna izbira pri vnosu le-teh, kot na primer: *Must have, Would have, Should have, Won't have this time* ipd.

- Seznam uporabnikov skupaj z vlogami, ki jih lahko določen uporabnik poseduje (lahko ima več uporabniških vlog; slika 33), ter izbranim geslom.

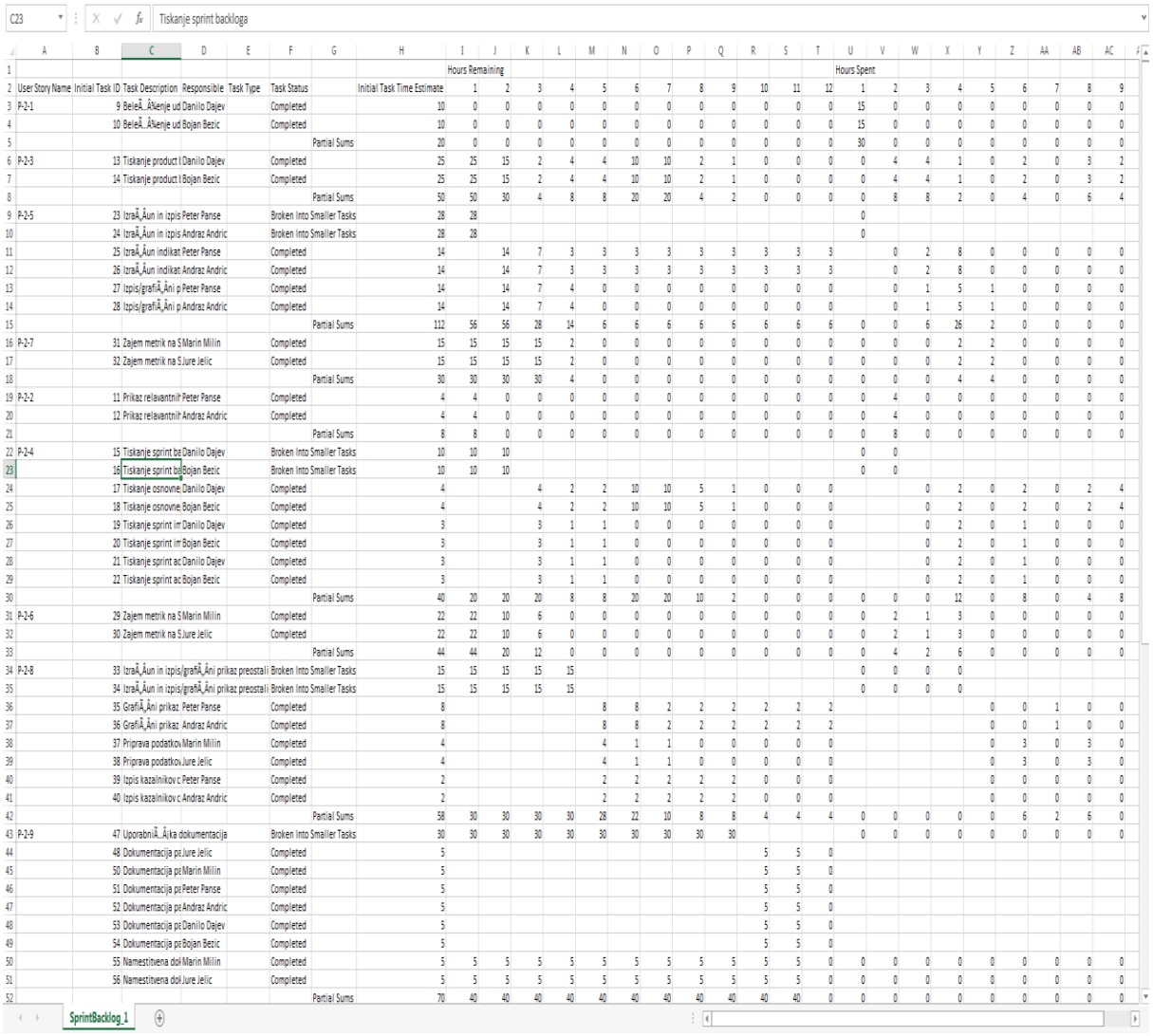


Slika 33: Uporabniški računi in izbira vlog

Zbirka funkcij *View*

S funkcijami menijske vrstice *View* je možno dobiti vpogled v sledeče pomembne informacije:

- *Product Backlog* – ob sprožitvi te funkcije program ustvari datoteko s končnico *csv* (skrajšano od ang. *Comma Separated Values*). Datoteko uporabnik lahko odpre s programom *Microsoft Excel* ali podobnim, ki podpira tovrstne vrste datotek. Ustvarjena datoteka vsebuje seznam vseh uporabniških zgodb, izdaj in Sprintov ter vključuje morebitne zajete meritve na nivoju celotnega projekta (te meritve predstavljajo dodatno razširitev orodja in niso uporabljene pri izračunavanju indeksov CPI in SPI).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1									Hours Remaining												Hours Spent								
2	User Story Name	Initial Task ID	Task Description	Responsible	Task Type	Task Status		Initial Task Time Estimate	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9
3	P-2-1	9	BeleÅ¾enje ud	Danilo Dajev		Completed		10	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0
4		10	BeleÅ¾enje ud	Bojan Bezic		Completed		10	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0
5							Partial Sums	20	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0
6	P-2-3	13	Tiskanje product	Danilo Dajev		Completed		25	25	15	2	4	4	10	10	2	1	0	0	0	0	4	4	1	0	2	0	3	2
7		14	Tiskanje product	Bojan Bezic		Completed		25	25	15	2	4	4	10	10	2	1	0	0	0	0	4	4	1	0	2	0	3	2
8							Partial Sums	50	50	30	4	8	8	20	20	4	2	0	0	0	0	8	8	2	0	4	0	6	4
9	P-2-5	23	IzraÄ…un in izpis	Peter Panse		Broken Into Smaller Tasks		28	28												0								
10		24	IzraÄ…un in izpis	Andraz Andric		Broken Into Smaller Tasks		28	28												0								
11		25	IzraÄ…un indikat	Peter Panse		Completed		14		14	7	3	3	3	3	3	3	3	3	3		0	2	8	0	0	0	0	0
12		26	IzraÄ…un indikat	Andraz Andric		Completed		14		14	7	3	3	3	3	3	3	3	3	3		0	2	8	0	0	0	0	0
13		27	Izpis/grafiÄ…ni p	Peter Panse		Completed		14		14	7	4	0	0	0	0	0	0	0	0		0	1	5	1	0	0	0	0
14		28	Izpis/grafiÄ…ni p	Andraz Andric		Completed		14		14	7	4	0	0	0	0	0	0	0	0		0	1	5	1	0	0	0	0
15							Partial Sums	112	56	56	28	14	6	6	6	6	6	6	6	6	0	0	6	26	2	0	0	0	0
16	P-2-7	31	Zajem metrik na S	Marin Milin		Completed		15	15	15	15	2	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0
17		32	Zajem metrik na S	Jure Jelic		Completed		15	15	15	15	2	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0
18							Partial Sums	30	30	30	30	4	0	0	0	0	0	0	0	0	0	0	0	4	4	0	0	0	0
19	P-2-2	11	Prikaz relavantnih	Peter Panse		Completed		4	4	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
20		12	Prikaz relavantnih	Andraz Andric		Completed		4	4	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
21							Partial Sums	8	8	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0
22	P-2-4	15	Tiskanje sprint ba	Danilo Dajev		Broken Into Smaller Tasks		10	10	10											0	0							
23		16	Tiskanje sprint ba	Bojan Bezic		Broken Into Smaller Tasks		10	10	10											0	0							
24		17	Tiskanje osnovne	Danilo Dajev		Completed		4			4	2	2	10	10	5	1	0	0	0			0	2	0	2	0	2	4
25		18	Tiskanje osnovne	Bojan Bezic		Completed		4			4	2	2	10	10	5	1	0	0	0			0	2	0	2	0	2	4
26		19	Tiskanje sprint im	Danilo Dajev		Completed		3			3	1	1	0	0	0	0	0	0	0			0	2	0	1	0	0	0
27		20	Tiskanje sprint im	Bojan Bezic		Completed		3			3	1	1	0	0	0	0	0	0	0			0	2	0	1	0	0	0
28		21	Tiskanje sprint ac	Danilo Dajev		Completed		3			3	1	1	0	0	0	0	0	0	0			0	2	0	1	0	0	0
29		22	Tiskanje sprint ac	Bojan Bezic		Completed		3			3	1	1	0	0	0	0	0	0	0			0	2	0	1	0	0	0
30							Partial Sums	40	20	20	20	8	8	20	20	10	2	0	0	0	0	0	0	12	0	8	0	4	8
31	P-2-6	29	Zajem metrik na S	Marin Milin		Completed		22	22	10	6	0	0	0	0	0	0	0	0	0	0	2	1	3	0	0	0	0	0
32		30	Zajem metrik na S	Jure Jelic		Completed		22	22	10	6	0	0	0	0	0	0	0	0	0	0	2	1	3	0	0	0	0	0
33							Partial Sums	44	44	20	12	0	0	0	0	0	0	0	0	0	0	4	2	6	0	0	0	0	0
34	P-2-8	33	IzraÄ…un in izpis/grafiÄ…ni prikaz preostali			Broken Into Smaller Tasks		15	15	15	15	15									0	0	0	0					
35		34	IzraÄ…un in izpis/grafiÄ…ni prikaz preostali			Broken Into Smaller Tasks		15	15	15	15	15									0	0	0	0					
36		35	GrafiÄ…ni prikaz	Peter Panse		Completed		8					8	8	2	2	2	2	2	2					0	0	1	0	0
37		36	GrafiÄ…ni prikaz	Andraz Andric		Completed		8					8	8	2	2	2	2	2	2					0	0	1	0	0
38		37	Priprava podatkov	Marin Milin		Completed		4					4	1	1	0	0	0	0	0					0	3	0	3	0
39		38	Priprava podatkov	Jure Jelic		Completed		4					4	1	1	0	0	0	0	0					0	3	0	3	0
40		39	Izpis kazalnikov c	Peter Panse		Completed		2					2	2	2	2	2	0	0	0					0	0	0	0	0
41		40	Izpis kazalnikov c	Andraz Andric		Completed		2					2	2	2	2	2	0	0	0					0	0	0	0	0
42							Partial Sums	58	30	30	30	30	28	22	10	8	8	4	4	4	0	0	0	0	0	6	2	6	0
43	P-2-9	47	UporabniÅ¡ka dokumentacija			Broken Into Smaller Tasks		30	30	30	30	30	30	30	30	30	30				0	0	0	0	0	0	0	0	0
44		48	Dokumentacija pr	Jure Jelic		Completed		5										5	5	0									
45		50	Dokumentacija pr	Marin Milin		Completed		5										5	5	0									
46		51	Dokumentacija pr	Peter Panse		Completed		5										5	5	0									
47		52	Dokumentacija pr	Andraz Andric		Completed		5										5	5	0									
48		53	Dokumentacija pr	Danilo Dajev		Completed		5										5	5	0									
49		54	Dokumentacija pr	Bojan Bezic		Completed		5										5	5	0									
50		55	Namestitvena dol	Marin Milin		Completed		5	5	5	5	5	5	5	5	5	5	5	5	0	0	0	0	0	0	0	0	0	0
51		56	Namestitvena dol	Jure Jelic		Completed		5	5	5	5	5	5	5	5	5	5	5	5	0	0	0	0	0	0	0	0	0	0
52							Partial Sums	70	40	40	40	40	40	40	40	40	40	40	40	0	0	0	0	0	0	0	0	0	0

Slika 34: Primer Sprint »backloga«

- *Sprint Backlog* (ponazorjeno na sliki 34) funkcija omogoča ustvarjanje datoteke s končnico *csv*, ki vsebuje celoten seznam uporabniških zgodb, ki pripadajo izbranemu Sprintu skupaj z ustreznimi nalogami in pripadajočimi pomembnejšimi polji, kot na primer *Task Description, Responsible, Task Type, Task Status ter Initial Task Time Estimate*. Poleg omenjenega vsaka naloga vsebuje še meritve, definirane in vnešene za vsak dan trajanja Sprinta. Na sliki 34 sta prikazani vnosi meritev *Hours Remaining* in *Hours Spent*, ki predstavljata osnovo za izračun indeksov CPI in SPI.

- S funkcijo *SPI and CPI Backlog* je mogoče ustvariti datoteko prav tako s končnico *csv*, ki pa vsebuje bistvene podatke v povezavi merjenjem razvoja programske opreme (ponazorjeno na sliki 35). Za vsak dan trajanja Sprinta sta tako preračunana in

zapisana oba indeksa SPI in CPI kot tudi nekatere druge delne skupne vrednosti vseh nalog znotraj izbranega Sprinta, kot je to na primer vrednost WRInit (začetna skupna vrednost preostanka dela v urah na vseh nalogah Sprinta po dnevih; skrajšano od ang. *Work Remaining Initial*). Ta funkcija predstavlja bistveni problem, na katerega se osredotoča magistrska naloga.

A1 | Sprint 1

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Sprint 1	1	2	3	4	5	6	7	8	9	10	11	12
2	SPI	1,118644068	0,961096243	1,122672107	1,460093438	1,338461538	1,066196183	0,93159376	0,9332541	0,850787338	0,801561381	0	0,952155852
3	CPI	0,818181818	0,966557018	1,354861111	1,445798817	1,574660633	1,389705533	1,406076829	1,396467996	1,339802934	1,278783308	0	1,244057646
4	WRInit	354	354	354	354	352	368	370	358	358	358	358	358
5	WSCEH	44	64	80	130	136	154	156	174	186	204	0	274
6													
7	ER (task, day)												
8													
9	Daily scrum	1	1	1	1	1	1	1	1	1	1	1	1
10	Daily scrum	1	1	1	1	1	1	1	1	1	1	1	1
11	Popravki na modulu za hranjenje podatkov o projektu	1	1	1	1	1	1	1	1	1	1	1	1
12	Popravki na modulu za hranjenje podatkov o projektu	1	1	1	1	1	1	1	1	1	1	1	1
13	Implementacija modula za hranjenje podatkov o teamih	1	1	1	1	1	1	1	1	1	1	1	1
14	Implementacija modula za hranjenje podatkov o teamih	1	1	1	1	1	1	1	1	1	1	1	1
15	Dopolnitev nastavitev za Task	0	0	0	0	0	0	0	1	1	1	1	1
16	Dopolnitev nastavitev za Task	0	0	0	0	0	0	0	1	1	1	1	1
17	BeleÃ…Â¾enje udeleÅ…Â¾enosti Ã„Âlanov ekipe na sprintu in njihove urne postavke	1	1	1	1	1	1	1	1	1	1	1	1
18	BeleÃ…Â¾enje udeleÅ…Â¾enosti Ã„Âlanov ekipe na sprintu in njihove urne postavke	1	1	1	1	1	1	1	1	1	1	1	1
19	Prikaz relavantnih administrative days za sprint	0	1	1	1	1	1	1	1	1	1	1	1
20	Prikaz relavantnih administrative days za sprint	0	1	1	1	1	1	1	1	1	1	1	1
21	Tiskanje product backloga	0	0,210526316	0,8	0,692307692	0,692307692	0,523809524	0,523809524	0,875	0,941176471	1	1	1
22	Tiskanje product backloga	0	0,210526316	0,8	0,692307692	0,692307692	0,523809524	0,523809524	0,875	0,941176471	1	1	1
23	Tiskanje sprint backloga	0	0	0	0	0	0	0	0	0	0	0	0
24	Tiskanje sprint backloga	0	0	0	0	0	0	0	0	0	0	0	0
25	Tiskanje osnovnega sprint backloga	0	0	0	0,5	0,5	0,285714286	0,285714286	0,545454545	0,909090909	1	1	1
26	Tiskanje osnovnega sprint backloga	0	0	0	0,5	0,5	0,285714286	0,285714286	0,545454545	0,909090909	1	1	1
27	Tiskanje sprint impediments	0	0	0	0,666666667	0,666666667	1	1	1	1	1	1	1
28	Tiskanje sprint impediments	0	0	0	0,666666667	0,666666667	1	1	1	1	1	1	1
29	Tiskanje sprint administrative days	0	0	0	0,666666667	0,666666667	1	1	1	1	1	1	1
30	Tiskanje sprint administrative days	0	0	0	0,666666667	0,666666667	1	1	1	1	1	1	1
31	IzraÄ…Ân in izpis/grafiÄ„Âni prikaz indikatorjev WE in EV	0	0	0	0	0	0	0	0	0	0	0	0
32	IzraÄ…Ân in izpis/grafiÄ„Âni prikaz indikatorjev WE in EV	0	0	0	0	0	0	0	0	0	0	0	0
33	IzraÄ…Ân indikatorjev WE in EV	0	0	0,222222222	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769
34	IzraÄ…Ân indikatorjev WE in EV	0	0	0,222222222	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769	0,769230769
35	Izpis/grafiÄ„Âni prikaz indikatorjev WE in EV	0	0	0,125	0,6	1	1	1	1	1	1	1	1
36	Izpis/grafiÄ„Âni prikaz indikatorjev WE in EV	0	0	0,125	0,6	1	1	1	1	1	1	1	1
37	Zajem metrik na Sprint review meeting	0	0,166666667	0,333333333	1	1	1	1	1	1	1	1	1
38	Zajem metrik na Sprint review meeting	0	0,166666667	0,333333333	1	1	1	1	1	1	1	1	1
39	Zajem metrik na Sprint retrospective meeting	0	0	0	0,5	1	1	1	1	1	1	1	1
40	Zajem metrik na Sprint retrospective meeting	0	0	0	0,5	1	1	1	1	1	1	1	1
41	IzraÄ…Ân in izpis/grafiÄ„Âni prikaz preostalih indikatorjev	0	0	0	0	0	0	0	0	0	0	0	0
42	IzraÄ…Ân in izpis/grafiÄ„Âni prikaz preostalih indikatorjev	0	0	0	0	0	0	0	0	0	0	0	0
43	GrafiÄ„Âni prikaz zadovoljstva naroÄ„Ânika in zadovoljstva razvijalcev	0	0	0	0	0	0	0,333333333	0,333333333	0,333333333	0,333333333	0,333333333	0,333333333
44	GrafiÄ„Âni prikaz zadovoljstva naroÄ„Ânika in zadovoljstva razvijalcev	0	0	0	0	0	0	0,333333333	0,333333333	0,333333333	0,333333333	0,333333333	0,333333333

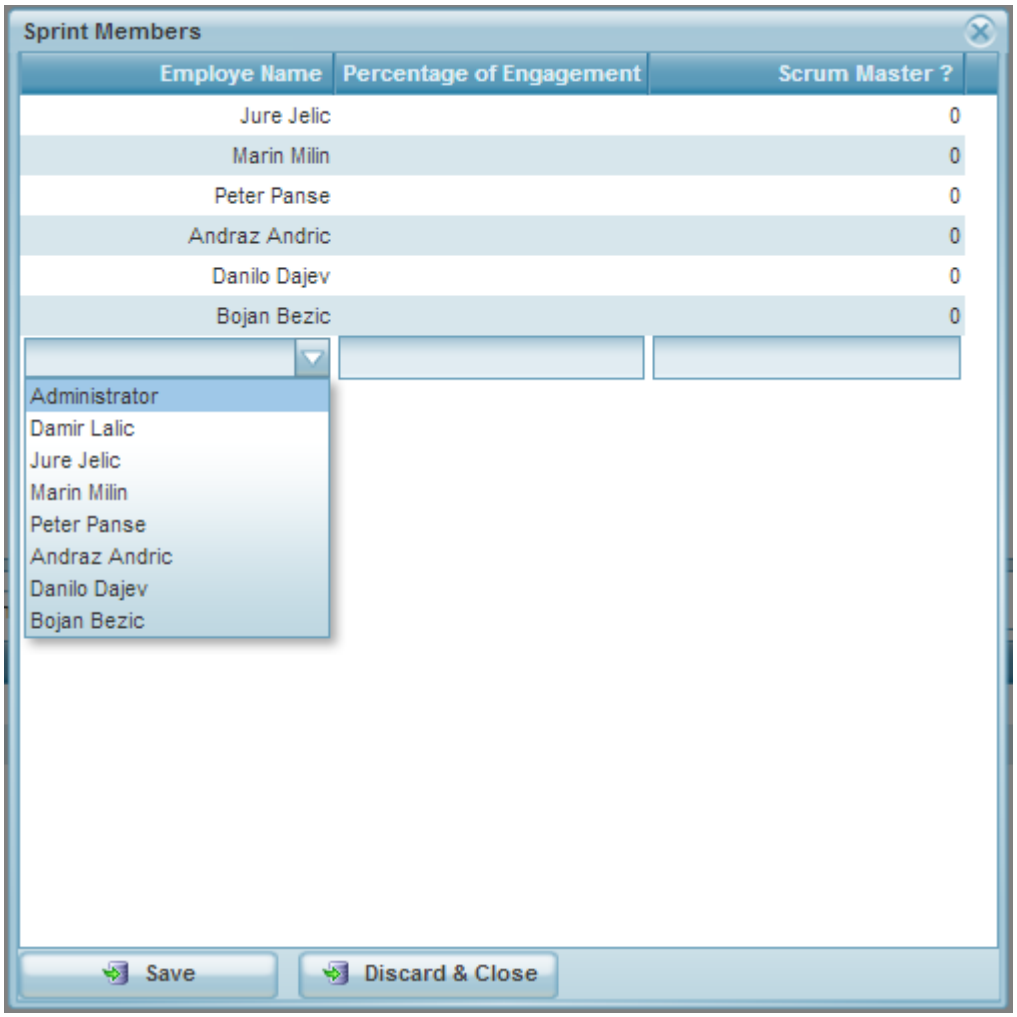
SPIandCPI_1

Slika 35: Primer CPI in SPI backloga

6.2.2 Funkcije za manipulacijo z bistvenimi zapisi orodja

Poleg funkcij menijske vrstice orodje podpira še ključne funkcije, ki se prikažejo in jih je možno sprožiti ob izbiri določenega zapisa.

Z desnim klikom na izbrani Sprint se odpre okno (slika 36) za dodajanje uporabnikov, ki lahko sodelujejo (opravljajo določene naloge) v tem Sprintu. Pri vnosu naloge v seznam izbranega Sprinta se tako dodani uporabniki pojavijo kot možna izbira za dodelitev odgovornosti za opravljanje naloge.

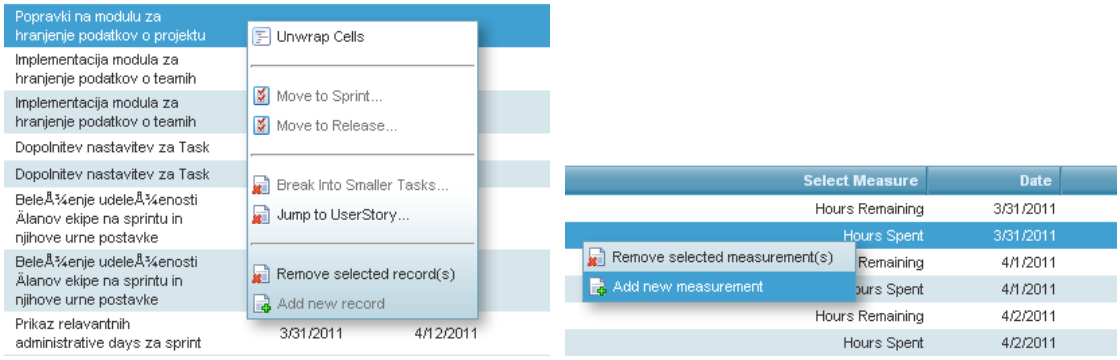


Slika 36: Dodajanje uporabnikov Sprinta

Bistvene funkcije (slika 37) za upravljanje z uporabniškimi zgodbami, pripadajočimi nalogami in meritvami, ki jih je je možno sprožiti z desnim klikom na ustrezen razdelek, so sledeče:

- Dodajanje novega zapisa (za naloge in uporabniške zgodbe ang. *Add new record*; za meritve ang. *Add new measurement(s)*). Tukaj velja omeniti, da orodje pri vpisu novih zapisov ne dovoli shranjevanja novih vpisov, ki ne vsebujejo vseh potrebnih podatkov (to je tudi grafično ponazorjeno).

- Brisanje izbranih zapisov (za naloge in uporabniške zgodbe ang. *Remove selected record(s)*; za meritve ang. *Remove selected measurement(s)*).



Slika 37: Funkcije za delo z uporabniškimi zgodbami, nalogami in meritvami

- Prenos zapisa v drugo ali povsem novo izdajo (ang. *Move to Release*) ter v drug ali nov Sprint (ang. *Move to Sprint*) – funkciji je moč uporabiti le za uporabniške zgodbe, avtomatično pa se prenesejo tudi pripadajoče naloge, vendar le tiste, ki še niso zaključene oziroma nimajo trenutnega statusa postavljenega na *Completed*. V tem primeru se naloge v resnici ne prenesejo, temveč se vrednosti kopirajo v nov zapis, ki hkrati ohranja referenco na prenešeno nalogo, in sicer v polju *Initial_Task_ID*. »Prenešena« naloga hkrati dobi nov status – *Moved Into Next Sprint*.

  Grobo gledano in le iz stališča uporabnika omenjeni funkciji obstajata iz več razlogov:

  - Pred samim začetkom razvoja programske opreme uporabniške zgodbe ni potrebno takoj razvrstiti v izdaje in Sprinte. Najprej jih lahko le vnesemo v orodje pod izbranim projektom in jih nato razvrstimo z uporabo omenjenih funkcij v izdaje in Sprinte.

    - Po drugi strani pa na koncu določenega Sprinta lahko ostanejo uporabniške zgodbe, ki jih ni bilo možno implementirati v celoti in jih zato lahko prenesemo bodisi v naslednje Sprinte trenutne izdaje bodisi v Sprinte kakšne druge izdaje.

  Ozadje omenjenih funkcij je precej bolj kompleksno in zahteva prilagoditev načrta podatkovne zbirke predvsem zaradi (opisano v poglavju *Podatkovni model orodja*) merjenja oziroma analiziranja aktivnosti znotraj neke časovno zaključene enote (na primer izdaje, Sprinta) ali aktivnosti, ki so vezane na neko funkcionalno zaključeno enoto (na primer naloga, uporabniška zgodba).

- Razbitje naloge na več manjših (ang. *Break Into Smaller Tasks*) velja le za obstoječe naloge. Uporabi se ponavadi v primerih, kadar se med razvojem ugotovi, da je naloga preobsežna in jo je smiselno razdeliti na več manjših, hkrati pa je bilo vloženega nekaj dela na njej. Naloga, ki jo »razbijemo«, tako dobi status *Broken Into Smaller Tasks*, manjše pripadajoče naloge pa dobijo dodatno informacijo *Broken Task ID*, ki predstavlja enolično referenco na »razbito« nalogo.

Pomembno je še omeniti, da je omenjene zapise možno spreminjati z dvoklikom na izbrani zapis, nakar orodje ponudi vnos novih ali popravek že obstoječih podatkov zapisa.

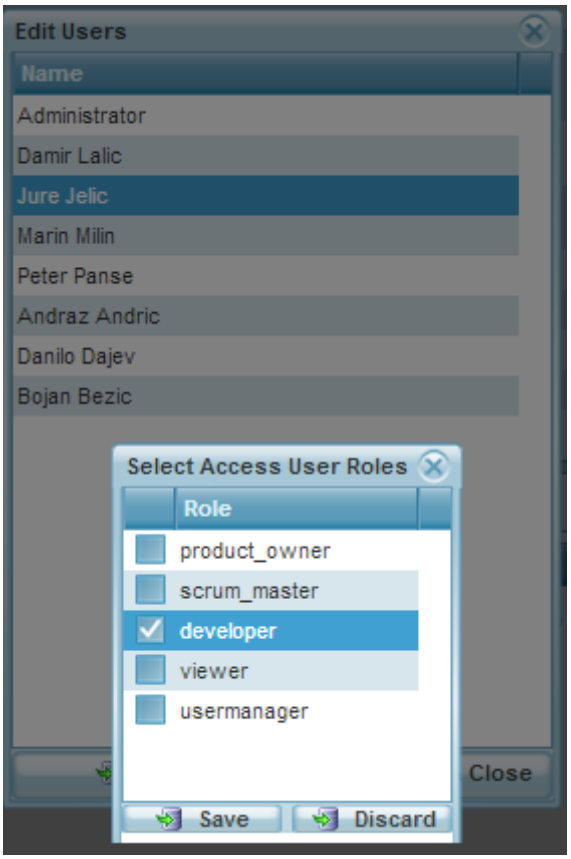
6.3 Uporaba orodja na primeru

V tem podpoglavju bo na kratko predstavljena uporaba orodja na preprostem primeru. Predpostavljeno je, da se je bralec že seznanil z metodo Scrum in tudi z vsemi funkcijami orodja, ki so razložene v poglavju 6. Zaradi obsežnosti se ne bomo spuščali v podrobnosti oziroma analizo projekta, temveč bo poudarek predvsem na praktični uporabi orodja. Namen je torej bralca seznaniti s samim postopkom in dejansko uporabo najpomembnejših funkcij.

Predpostavimo, da gre za projekt za katerega so že definirane zahteve. Produktni vodja mora najprej vzpostaviti podatke, ki so potrebni za delovanje orodja:

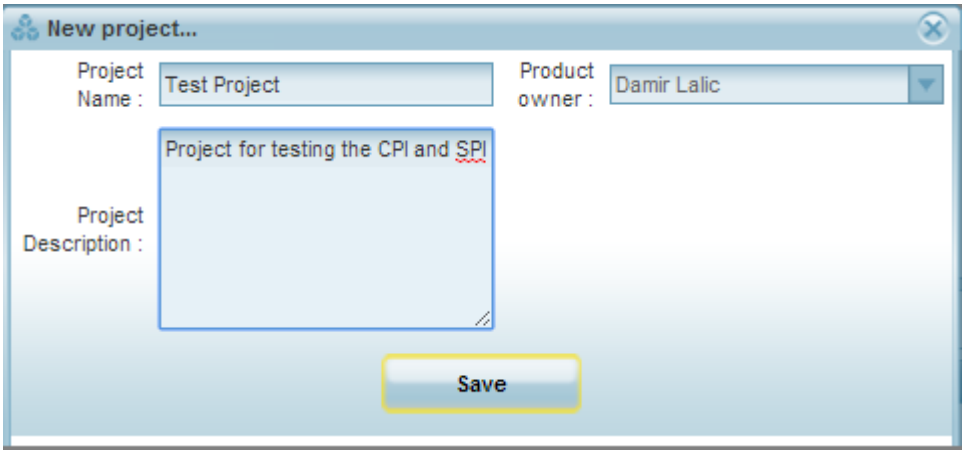
- Vrste nalog (*Coding, Testing, Documentation*);
- Vrste statusov nalog (*Not Started, In Progress, Completed, Omitted, Moved Into Next Sprint, Broken Into Smaller Tasks*);
- Vrste metrik (*Hours Remaining, Hours Spent*);
- Prioritete uporabniških zgodb (*Must have, Would have, Should have, Won't have this time*);

- Seznam uporabnikov skupaj z vlogami, ki jih lahko določen uporabnik poseduje (slika 38) kot je opisano v poglavju 6.2.1 - *Zbirka funkcij Administration*. Pri tem enemu uporabniku dodelimo vlogo *Product Owner*, enemu *Scrum Master* in vsem ostalim (razen *Administratorju*) dodelimo vlogo *Developer*. Poleg omenjenega je potrebno vsem uporabnikom nastaviti ustrezna uporabniška imena in gesla za dostop do orodja.



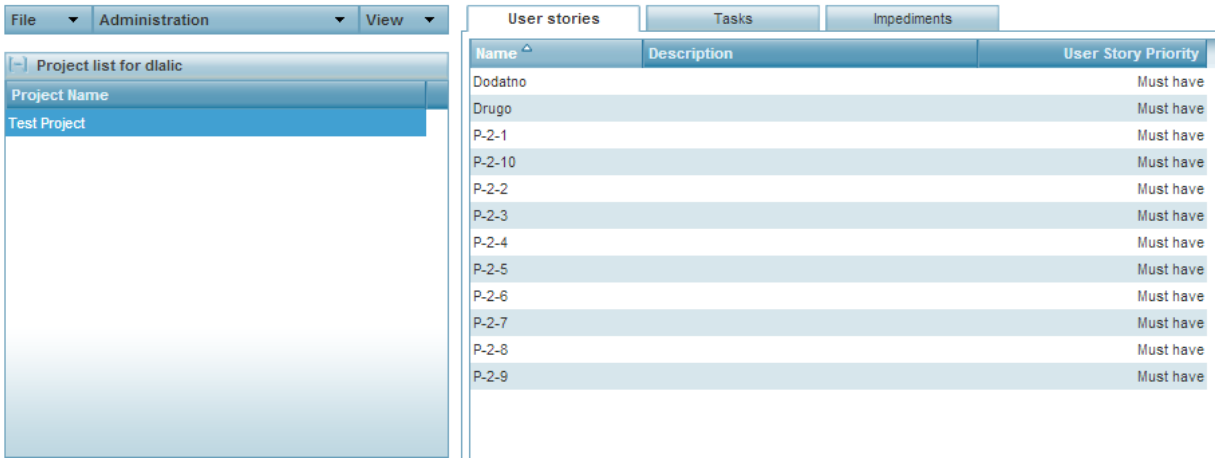
Slika 38: Seznam uporabnikov in vloge

Nato produktni vodja mora vnesti osnovne podatke o projektu (poglavje 6.2.1 – *Zbirka funkcij File*) kot je prikazano na sliki 39.



Slika 39: Vnos projekta

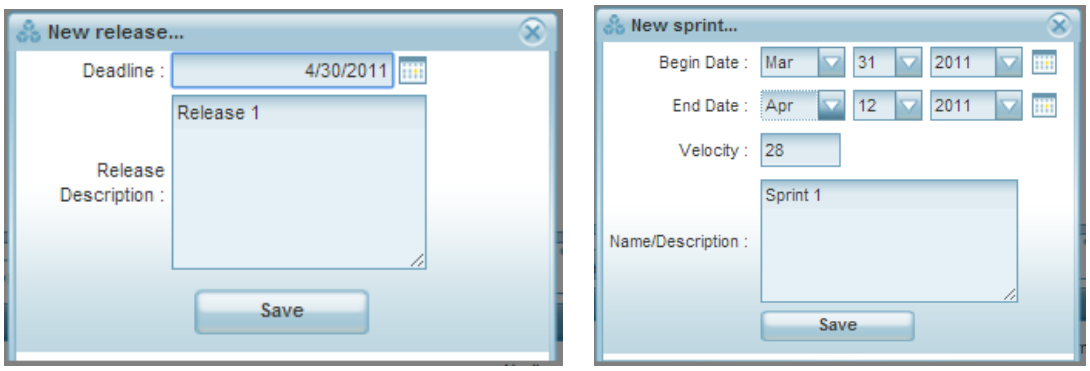
Po vnosu projekta je možno takoj vnašati zahteve v orodje v obliki uporabniških zgodb. Te zahteve (vsaj začetni seznam) produktni vodja vnese pred sestankom za načrtovanje Sprinta. Potrebno je le označiti projekt in v razdelku *User stories* se z desnim klikom na miško vnašajo zapisi zahtev. Pri tem opomnimo, da so te zahteve v tem trentku vezane le na projekt in niso še razvrščene po izdajah oziroma Sprintih.



Slika 40: Seznam zahtev

Na sliki 40 je prikazan seznam zahtev z dodeljenimi prioritetami. Ker gre za izmišljeni primer, so zahteve označene samo z identifikacijsko številko (npr. P-2-1 do P-2-10 itn.) in ni vnešenih nobenih drugih posebnih opisov (orodje podpira veliko več vhodnih podatkov za zahteve kot je prikazano na sliki – na primer *Initial Points, Description, Comments* itn.).

Pred sestankom za načrtovanje Sprinta produktni vodja vnese v orodje vsaj eno izdajo in vsaj en Sprint znotraj te izdaje (poglavje 6.2.1 – *Zbirka funkcij File*). Izdaji naj bo ime *Release 1* in Sprintu *Sprint 1* kot je prikazano na sliki 41. Pri tem je potrebno še določiti člane razvojne skupine, ki bodo sodelovali v *Sprintu 1* (slika 36).



Slika 41: Vnos izdaje in Sprinta

Med sestankom za načrtovanje Sprinta ima produktni vodja orodje aktivno in sproti predstavlja zahteve članom razvojne skupine. V prvem delu sestanka se produktni vodja in razvijalci dogovorijo, katere zahteve bodo realizirane v naslednjem Sprintu (v tem konkretnem primeru bo Sprint trajal 12 dni). Pri tem velja omeniti, da orodje omogoča njihov prenos v izbrano izdajo oziroma Sprint. Po doseženem dogovoru vsako izbrano zahtevo posebej prestavimo prvo v izdajo *Release 1* in nato še v *Sprint 1*. Da bi se izognili dvojnemu prestavljanju zahtev, je možno tudi neposredno vpisovati izbrane zahteve bodisi v izdajo ali pa v Sprint.

V drugem delu sestanka je potrebno izbrane zahteve razbiti na posamezne naloge (vsaka naj bi trajala približno 4 do 16 ur) in se na koncu dogovoriti o odgovornost za izpolnitev vsake. Ko so enkrat vse zahteve prestavljene v *Sprint 1*, je možno vpisovati dogovorjene naloge. Najprej je potrebno izbrati *Sprint 1*, nato posamezno zahtevo in šele potem v razdelku *Tasks* s klikom na desni gumb miške je možno vnesti nalogo z vsemi pripadajočimi podatki (poglavje 6.2.2). Delni seznam nalog se nahaja na sliki 42.

User stories | Tasks | Impediments

Initial Task ID	Broken Task ID	Task Description	Date	Task Active Till	Task Time Estimate	Cost of Engineering Hour	Task Status	Task Type	Responsible
1		Daily scrum	3/31/2011	4/12/2011	1	1	Not Started	Coding	Marin Milin
2		Daily scrum	3/31/2011	4/12/2011	1	1	Not Started	Coding	Jure Jelic
3		Popravki na modulu za hranjenje podatkov o projektu	3/31/2011	4/12/2011	5	1	Not Started	Coding	Danilo Dajev
4		Popravki na modulu za hranjenje podatkov o projektu	3/31/2011	4/12/2011	5	1	Not Started	Coding	Bojan Bezic
5		Implementacija modula za hranjenje podatkov o teamih	3/31/2011	4/12/2011	2	1	Not Started	Coding	Peter Panse
6		Implementacija modula za hranjenje podatkov o teamih	3/31/2011	4/12/2011	2	1	Not Started	Coding	Andraz Andric
7		Dopolnitev nastavitev za Task	4/7/2011	4/12/2011	1	1	Not Started	Coding	Peter Panse
8		Dopolnitev nastavitev za Task	4/7/2011	4/12/2011	1	1	Not Started	Coding	Andraz Andric
9		BeleÅ¾enje udeleÅ¾enosti Älanov ekipe na sprintu in njihove urne postavke	3/31/2011	4/12/2011	10	1	Not Started	Coding	Danilo Dajev
10		BeleÅ¾enje udeleÅ¾enosti Älanov ekipe na sprintu in njihove urne postavke	3/31/2011	4/12/2011	10	1	Not Started	Coding	Bojan Bezic
11		Prikaz relavantnih administrative days za sprint	3/31/2011	4/12/2011	4	1	Not Started	Coding	Peter Panse
12		Prikaz relavantnih administrative days za sprint	3/31/2011	4/12/2011	4	1	Not Started	Coding	Andraz Andric

Slika 42: Delni seznam nalog Sprinta 1

Za vsako posamezno nalogo so določeni sledeči podatki:

- *Task Description* – opis kaj je potrebno narediti v okviru naloge.
- *Date* – datum začetka veljavnosti naloge.
- *Task Active Till* – datum konca veljavnosti naloge.
- *TaskTime Estimate* – prvotna ocena časa potrebnega za dokončanje naloge v urah.
- *Cost of Engineering Hour* – strošek inženirske ure, ki je odvisna bodisi od naloge ali od tistega, ki je zanjo odgovoren. V tem konkretnem primeru so vse postavljene na 1.
- *Task Status* – trenutni status naloge. Na začetku so vse naloge postavljen na vrednost *Not Started*.
- *Task Type* – tip naloge, ki je v tem primeru lahko: *Coding*, *Testing* ali *Documentation*
- *Responsible* – član razvojne skupine, ki je hkrati del tima *Sprinta 1*, odgovoren za izvršitev naloge.

Po vnosu vseh nalog in ustreznih podatkov je sestanek za načrtovanje Sprinta zaključen. Naslednji dan sledi začetek dela na nalogah vseh članov Sprinta. V tem konkretnem primeru je predvideno, da se Daily Scrum sestanki izvajajo vsako jutro in se metriki *Hours Remaining* in *Hours Spent* zajemata takrat. Na začetku (na prvem Daily Scrum sestanku) privzamemo, da je vrednost *Hours Remaining* enaka časovni oceni potrebnega dela za dokončanje naloge, vrednost *Hours Spent* pa enaka 0. Nato pa vsak dan zabeležimo, koliko ur dela je bilo vloženega in koliko ur je še preostalo do dokončanja neke naloge. Podatke vnašamo samo za tiste naloge, na katerih je dejansko potekalo delo. Pri ostalih nalogah privzamemo, da je količina vloženega dela enaka 0, količina preostalega dela pa enaka kot prejšnji dan. Te vrednosti se avtomatično vzpostavijo s pomočjo funkcije *Fill in Missing Measurements – Hours Spent and Remaining* (poglavje 6.2.1), ki jo je potrebno sprožiti vedno pred izračunavanjem indeksov SPI in CPI.

S pomočjo podatkov o vloženem in preostalem delu lahko med Sprintom vsak dan izračunamo vrednosti indeksov SPI in CPI. To nam zagotavlja sproten vpogled v potek dela na projektu tako s časovnega kot s stroškovnega vidika. Primer izračuna prikazuje slika 35.

Praktični primer je sestavni del priloge, ki je v sklopu magistrske naloge shranjena na zgoščenki.

7 Sklep

Agilne metode so nastale kot odziv na nezadovoljstvo zaradi velikega odstotka neuspešnih projektov na področju razvoja programske opreme. Gre za drugačen pristop, podprt z bolj fleksibilnim načinom razmišljanja in bolj sproščujočim odnosom do projektov. Glavni cilj agilnih metod je povečanje odstotka uspešnosti projektov na področju razvoja programske opreme s sprotnim prilagajanjem spremembam v zahtevah naročnika. Odpravljanje nepotrebne dokumentacije, zmanjševanje nesporazumov med razvojem, sprotno odpravljanje težav in aktivno sodelovanje naročnika v razvoju so ene izmed mnogih lastnosti agilnega pristopa k razvoju, ki prispevajo k večji uspešnosti razvojnih projektov.

Ko govorimo o uspešnosti ali neuspešnosti projekta, moramo poudariti, da ne gre le za projekte, ki morda niso bili zaključeni oziroma so bili prekinjeni predčasno bodisi s strani naročnika ali izvajalca. Neuspešni projekti so tudi tisti projekti, pri katerih upoštevanje stališč različnih interesnih skupin (npr. vodstvo IT, člani razvojne skupine in naročnik oziroma stranka), ki sodelujejo v procesu razvoja, nakazuje na »neko« nezadovoljstvo. Pri tem je pomembno implementirati mehanizme, ki sproti merijo učinkovitost razvoja programske opreme. Sprotno merjenje omogoča hitro odpravljanje težav in ovir, ter takojšno izvedbo ukrepov za večanje učinkovitosti. Posledično to pomeni večanje verjetnosti, da bo projekt uspešen.

Rezultat magistrskega dela je izdelano programsko orodje, ki omogoča vodenje procesa razvoja programske opreme po metodi Scrum. Za boljše razumevanje so v magistrskem delu predstavljene tudi osnovne značilnosti, načela in priporočila agilnih metod (poglavje 2), podrobno pa je predstavljena metoda Scrum (poglavje 3), ki jo orodje podpira. Orodje dodatno vključuje podporo izbranim metrikam za spremljanje učinkovitosti procesa, ki temeljijo predvsem na upoštevanju stališč vodstva IT. To se v glavnem ukvarja s tradicionalnimi vidiki uspešnosti razvoja programske opreme glede na čas, stroške in kakovost. Pri tem se v magistrski nalogi osredotočam predvsem na en cilj vodstva IT: pravočasno obveščanje o uspešnosti projektov s poudarkom na projektih, ki se odmikajo od prvotno začrtanih postavk glede časa zaključka in proračuna projekta.

Tako pridemo do terminskega (SPI) in stroškovnega (CPI) indeksa, ki sta v magistrskem delu predstavljena v poglavju 4. Z indeksom SPI je možno meriti potek razvoja programske opreme glede na prvoten načrt, medtem ko pa z indeksom CPI spremljamo stroške dela glede na prvotno načrtovane. Orodje podpira izračuna obeh indeksov, ki temeljita na dveh osnovnih metrikah, ki ju je možno vnašati v orodje: količina preostalega dela (osnovna metrika prvotno

podprta s strani metode Scrum) in na novo uvedena metrika količina porabljenega dela. Orodje omogoča vnašanje omenjenih metrik na različnih nivojih, pri čemer je za namen izračunov indeksov SPI in CPI pomembno, da se obe metriki zajemata za vsako nalogo istočasno na vsakodnevnih Daily Scrum sestankih. Omenimo še, da je v okviru funkcij orodja implementirana tudi možnost dinamičnega vnosa dodatnih metrik brez posegov v programsko kodo.

V magistrskem delu je detajlno predstavljen pristop k izgradnji orodja (poglavje 5). Ta vključuje zajem zahtev, katerim mora programsko orodje zadoščati, ter izbiro tehnologij in drugih orodij, ki so bila uporabljena za uresničitev implementacije. Posebna pozornost je dodatno namenjena predstavitvi najpomembnejših delov implementacije orodja: osnovne sistemske nastavitve, predstavitev najpomembnejših elementov podatkovnega modela in izvršilne kode orodja (na strežniku in odjemalcu). V delu so še posebej predstavljeni grafični uporabniški vmesnik, najpomembnejše funkcije, ki jih orodje podpira, in uporaba orodja na preprostem primeru (poglavje 6).

Med izdelavo orodja sem se srečal tudi s težavami, ki jih je bilo potrebno sproti reševati. Kar se tiče tehničnih težav ni bilo kakšnih posebnosti. Tažave so se pojavljale sproti in jih je bilo potrebno reševati z detajlnim proučevanjem dokumentacije in uporabo različnih virov na spletu. Posebej pa bi omenil bolj konceptualno težavo v povezavi z merjenjem / analiziranjem aktivnosti. Podatkovni model namreč mora omogočati merjenje/analiziranje aktivnosti znotraj neke časovno zaključene enote (na primer izdaje ali Sprinta) in aktivnosti, ki je vezana na neko funkcionalno zaključeno enoto (na primer naloga, uporabniška zgodba). Podroben razmislek o omenjenem je zahteval spremememembo oziroma prilagoditev podatkovnega modela. V poglavju *5.4.4 Merjenje / analiziranje aktivnosti* je nazorno prikazana omenjena problematika in ustrezna rešitev, ki jo podatkovni model oziroma orodje ponuja.

V bodoče bi bilo zanimivo spremljati oziroma meriti tudi drugi cilj vodstva IT [6]: izboljšanje kakovosti. V orodje bi lahko uvedli dodatne metrike s katerimi bi lahko spremljali kazalnike kot sta na primer število napak na tisoč vrstic kode (ang. *number of errors per KLOC*) in stroški predelave (ang. *costs of rework*) določenih funkcionalnosti [6].

Omenimo še, da izjemno pomembno vlogo v razvoju predstavljajo tudi člani razvojne skupine. Glavni cilj te interesne skupine je zadovoljstvo pri delu. Člani skupine so najbolj produktivni, če imajo dobre delovne pogoje, ki jim omogočajo konstantno hitrost napredka brez prevelike delovne obremenjenosti in velikega števila nadur. Za merjenje cilja članov razvojne skupine so v [6] predlagani nekateri kvalitativni in kvantitativni kazalniki, ki bi jih prav tako bilo zanimivo preučiti in implementirati v orodje.

8 Viri in literatura

[1] F. T. Anbari, »Earned value project management method and extensions«, Project Management Journal, Vol. 34, No. 4, str. 12-23, 2003.

[2] M. Cohn, »User stories applied for agile software development«, Addison-Wesley, 2009.

[3] R. T. Fielding, »Architectural Styles and the Design of Network-based Software Architectures«, University of California, Irvine, 2000. Doktorska disertacija dostopna na: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[4] D. Hartmann, R. Dymond, »Appropriate Agile Measurement: Using Metrics and Diagnostics to Deliver Business Value«, Proceedings of AGILE 2006 Conference (AGILE'06), pp. 126-134.

[5] D. Ljubobratović, I. Padavić, M. Velić, »Agilni razvoj programskih proizvoda«, Infoteh-Jahorina, Vol. 10, Ref. E-I-16, str. 466-470, 2011.

[6] V. Mahnič, I. Vrana, »Using stakeholder driven process performance measurement for monitoring the performance of a Scrum based software development process«, Elektrotehniški vestnik, letnik 74, št. 5, str. 241-247, 2007.

[7] V. Mahnic, N. Zabkar, »Introducing CMMI Measurement and Analysis Practices into Scrum-based Software Development Process«, International Journal of Mathematics and Computers in Simulation, Vol. 1, Issue 1, str. 65-72, 2007.

[8] L. Rising, N. S. Janoff, »The Scrum software development process for small teams«, IEEE Software, Vol. 17, Issue 4, Jul/Aug 2000, str. 26-32, 2000.

[9] K. Schwaber, »Agile project management with Scrum«, Microsoft Press, Redmond, 2004.

[10] T. Sulaiman, B. Barton, T. Blackburn, »AgileEVM - Earned Value Management in Scrum Projects«, Proceedings of AGILE 2006 Conference (AGILE'06), pp. 7-16.

[11] L. Williams, »Agile Software Development Methodologies and Practices«, Advances in Computers, Vol. 80, str. 1-44, 2010.

Ostali viri

[12] Serena Software, Inc., »An Introduction to Agile Software Development«, 2007. Dostopno na: http://www.serena.com/docs/repository/solutions/intro-to-agile-devel.pdf.

[13] D. West, J. S. Hammond, »The Forrester Wave: Agile Development Management Tools«, Forrester Research, Inc., 2010.

[14] Simon Tutek, Janez Brest, Viljem Žumer, Inštitut za Informatiko, »FERI: Razvoj spletnih aplikacij z uporabo programskega ogrodja JCorporate Expresso«. Dostopno na: http://www.kiblix.org/2003/work_jsp.html.

[15] J. J. Garrett, Ajax: »A New Approach to Web Applications«, 2005. Dostopno na: http://adaptivepath.com/ideas/ajax-new-approach-web-applications.

[16] Standard ECMA-262, izdaja 5.1, junij 2011

[17] Json.org, »Introducing JSON«. Dostopno na: http://www.json.org/.

[18] D. K. Barry, »Web Services Explained«. Dostopno na: http://www.service-chitecture.com/web-services/articles/web_services_explained.html.

[19] G. Jackson, »Overview Active Server Pages (ASP)«, 2006. Dostopno na: http://www.newebia.co.uk/articles/asp-explained.html.

[20] Dostopno na: http://msdn.microsoft.com/en-us/library/aa286483.aspx.

[21] Microsoft, MSDN, 2011. Dostopno na: http://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx.

[22] Dostopno na: http://docs.oracle.com/javaee/1.4/tutorial/doc/.

[23] Dostopno na: http://www.php.net/.

[24] Dostopno na: http://smartclient.com.

[25] Dostopno na: http://agilemanifesto.org/iso/sl/.