UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

ALEŠ KOPRIVNIKAR

SKUPINSKI RAZVOJ PROGRAMSKE OPREME Z IBM RATIONAL TEAM CONCERT

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Viljan Mahnič

LJUBLJANA, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 01599/2009
Datum: 15.10.2009

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

Kandidat: **ALEŠ KOPRIVNIKAR**

Naslov: **SKUPINSKI RAZVOJ PROGRAMSKE OPREME Z IBM RATIONAL TEAM CONCERT**

COLLABORATIVE SOFTWARE DEVELOPMENT USING IBM RATIONAL TEAM CONCERT

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Predstavite izzive globalnega razvoja programske opreme in vlogo sodobnih orodij za pomoč pri skupinskem delu. Opišite značilnosti orodja IBM Rational Team Concert in možnosti za njegovo uporabo pri agilnem razvoju programske opreme po metodi Scrum. Za ilustracijo prikažite postopek priprave uporabniških zgodb in začetnega plana izdaje za projekt izgradnje študijskega informacijskega sistema.

Mentor:
prof. dr. Viljan Mahnič

Fakulteta za računalništvo in informatiko
UNIVERZA V LJUBLJANI
2

Dekan:
prof. dr. Franc Solina

Zahvala

V prvi vrsti bi se najlepše zahvalil svojemu mentorju, izr. prof. dr. Viljanu Mahniču. Njegove ideje, usmerjanje, odzivnost in predlogi so mi bili v zares neizmerno pomoč pri pravočasnem dokončanju diplomskega dela.

Prisrčno se zahvaljujem tudi družini in prijateljem, ki so mi v času študija stali ob strani. Posebna zahvala gre mami, očetu, Katarini, Poloni, Jelki in Bojanu. Včasih se sprašujem, če bi mi brez njih sploh uspelo.

Kazalo

Kratice in simboli

ALM – Application Lifecycle Management

CALM – Collaborative Application Lifecycle Management

RTC – Rational Team Concert

SCM – Source Code Management

RUP – Rational Unified Process

OpenUP – Open Unified Process

JTS – Jazz Team Server

JFS – Jazz Foundation Services

Povzetek

V nalogi opisujem orodje za podporo skupinskemu razvoju programske opreme Rational Team Concert. Raziskal in opisal sem platformo Jazz, na kateri bazira in ga umestil v upravljanje življenjskega cikla aplikacije (Application Lifecycle Management - ALM). Posebno pozornost sem namenil podpori agilnim metodologijam pri razvoju, ki jih lahko izvajamo in prilagajamo v omenjenem orodju.

Za boljši prikaz zmožnosti orodja pri razvoju sem predstavil izvajanje vzorčnega projekta po metodologiji Scrum. Izvedel sem začetni del razvojnega projekta. Zbral sem uporabniške zahteve in jih razdeli na več iteracij. Na podlagi tega so prikazane funkcionalnosti orodja.

Moj cilj je bil detajlno spoznavanje z orodjem predvsem iz metodološkega stališča. Najbolj sem se foukusiral na metodologijo Scrum in kako je ta implementirana v orodju, na kak način se orodje uporablja z njo in kakšne so koristi razvojne ekipe pri uporabi RTC in Scrum.

Ključne besede:

Razvoj programske opreme

Upravljanje izvorne kode

Skupinski razvoj

Agilni razvoj

Projektno vodenje

Metodologije razvoja programske opreme

Abstract

In this paper I am describing a tool supporting the collaborative development of software, called IBM Rational Team Concert (RTC). I analyzed and described the Jazz platform on which it is based and described its role in the Application Lifecycle Management (ALM). I paid special attention to the support of agile methodologies in software development, which the tool helps to execute and adapt to specific needs.

For a better understanding of the capabilities of RTC I envisioned a sample project. In it I implemented the planning part of the development lifecycle. I defined the project's requirements and split the work in several iterations. Based on that I have shown the capabilities of the tool.

My goal was to get acquainted with RTC, mainly from a methodological standpoint. I focused primarily on Scrum and how it is supported in the tool, on how the tool is used when executing a Scrum project and on the benefits for the development team, while using both RTC and Scrum.

Key words:

Software development

Source code management

Collaborative development

Agile development

Project management

Software development methodology

1. Uvod

Pred leti je bil razvoj programske opreme preprostejši. Vloge so večinoma obsegale programerje, vodje razvoja in analitike. Celotna razvojna skupina pa se je nahajala samo na eni lokaciji, kar je olajševalo komunikacijo in sodelovanje. Sama zahtevnost projektov ni bila na tako visokem nivoju kot v zadnjih letih.

Dandanes se je število vlog in kompleksnost projektov močno povečala. V razvoj so vključeni analitiki, programerji, testerji, revizorji, vodje projektov, strokovnjaki za varnost, pravniki in drugi netradicionalni udeleženci. Razvojne skupine so lahko razpršene po celem svetu – in ni nujno, da so vsi zaposleni v organizaciji, ki razvija rešitev. Partnerji, distributerji in pogodbeniki so lahko tudi del razvojnega procesa. Vsaka vloga ima svojo perspektivo, informacije, procese in nadzorne postopke. Vsi ti udeleženci morajo slediti spremembam in zahtevam, saj te lahko v veliki meri vplivajo tudi na njihove zadolžitve pri izvajanju razvojnega projekta.

Ali bo projekt uspešen, je odvisno od sodelovanja in komunikacije vseh naštetih udeležencev v procesu nastajanja programske opreme. V veliki meri pa uspešnost projekta merimo tudi s tem, ali je bil izveden v vnaprej določenem roku in še bolj pomembno, v okviru proračuna.

Doseganje teh ciljev ni preprosto, ne glede na metodologijo, uporabljeno tehnologijo in kompleksnost. Sredi 90-ih let se je pričela razvijati nova vrsta metod, kot reakcija na bolj "težke" metode, kamor spadajo tiste, ki zahtevajo ogromno planiranja, dokumentiranja, načrtovanja, ... Ti novi pristopi so z letom 2001 in obljavljenim manifestom agilnosti [10] tudi uradno dobili ime agilne metodologije.

Dandanes so mnenja o agilnem razvoju deljena. Na vsak način je sprememba načina dela v razvojni skupini težaven proces in so primeri, ko posvojitev agilnega razvoja ne uspe. Dejstvo pa je, da je s klasičnimi razvojnimi tehnikami izredno težko razvijati projekte, kjer se uporabniške zahteve hitro in pogosto spreminjajo. Boehm [11] predlaga uvedbo agilnih metod v projektih, kjer se uporabniške zahteve spreminjajo za več kot 1% na mesec. Hkrati Cohen [2] ugotavlja, da pri skupinah večjih od 20-40 komunikacija in koordinacija pri uporabi agilnih metodologij postane problematična. Zaradi teh težav predlagajo rešitve za prilagajanje praks na večje skupine. To je seveda izziv, saj komunikacija pri večjem številu ljudi postane težavna.

Vedno bolj se izzivov na tem področju zavedajo tudi podjetja, ki razvijajo in prodajajo razvojna orodja. Perforce, CollabNET, Accurev, IBM in Microsoft so zelo aktivni na tem področju, če omenim samo večje komercialne akterje. Obstaja pa še množiča brezplačnih in manjših igralcev na področju. Poudarek v tej diplomski nalogi je na IBM rešitvi na tem področju, ki kot podjetje s 40.000 razvijalci [12], tudi deluje v tem prostoru in ponuja rešitve za vodenje razvojnega procesa z velikim poudarkom na sodelovanju, za uporabo v manjših in večjih skupinah. To je platforma Jazz in orodja, ki temeljijo na njej. Najbolj razširjeno pa je Rational Team Concert (RTC), ki je v IBM interno odlično sprejeto. Po podatkih iz konca 2009 [12] je interno v uporabi približno ene tretjine razvijalcev in se še vedno širi. V tem diplomskem delu sem se lotil raziskovanja platforme, njene arhitekture, kako dobro podpira integracijo z agilnimi metodologijami, predvsem Scrum in ali premika mejnike v načinu razvoja programske opreme.

2. Razvoj progamske opreme danes

Namen tega poglavja je orisati področje razvoja programske opreme glede na uporabljene jezike, orodja, metodologije in izzive, s katerimi se srečujejo razvojne skupine dandanes. Na podlagi tega lahko bolje umestim Rational Team Concert v področje razvoja programske opreme.

2.1. Programski jeziki

Dandanes je nabor programskih jezikov, ki se uporabljajo, velik. Najbolj na grobo jih lahko razdelimo na prevajane in interpretirane jezike [13]. Razlika med njimi je, da pri prvih s pomočjo prevajalnika iz izvorne kode naredimo strojno kodo, ki se potem izvaja. Primer takih jezikov so FORTRAN, COBOL, C, C++ in drugi. Pri interpretiranih jezikih interpreter procesira izvorno kodo vrstico po vrstico. To so večinoma skriptni jeziki kot naprimer PHP, Perl, Python, ... Obstaja pa še posebna kategorija interpretiranih jezikov, ki uporabljajo prevod izvorne kode, ki potem teče na navideznem stroju, kar predstavlja vmesno rešitev. V tem primeru prevajalnik naredi vmesno kodo (angl. byte code), ki jo potem navidezni stroj izvaja v realnem času. Primer takega jezika sta Java in C#. Sicer so jeziki osnovani na takšnem principu počasnejši, vendar hkrati veliko bolj fleksibilni, saj jih lahko načeloma poganjamo na katerikoli platformi, za katero je razvit navidezni stroj. K njihovi široki uporabi veliko pripomore uporaba prevajanja ob ravno pravem času (angl. Just-In-Time compiler – JIT) [14] in HotSpot prevajalnika [15]. S pomočjo teh je hitrost izvajanja veliko večja.

Kateri jezik je najbolj popularen, je težko določiti. V različnih sferah so popularni različni jeziki. Zanimiv primer je uporaba COBOL-a. Ta je bil razvit leta 1959 [17] in se še vedno ogromno uporablja v velikih organizacijah, predvsem bankah. Podobna situacija je tudi z RPG, PL/I,.. Preprosto je prehod na novejše tehnologije preveč tvegan, saj so programi v uporabi že več desetletij in so do sedaj že temeljito preverjeni. Na področju razvoja spletnih aplikacij je zelo popularen PHP [16], ki je zelo zmogljiv skriptni jezik za razvoj dinamičnih spletnih strani. Dandanes se večino sodobnih poslovnih aplikacij razvija v Javi. Njena glavna prednost je verjetno ravno to, da lahko teče na velikem številu operacijskih sistemov. Veliko pa je k njeni široki uporabi pripomoglo tudi to, da je to strateška platforma kar nekaj večjih ponudnikov komercialne programske opreme (IBM, Sun, Oracle, ...) Zelo popularen je tudi Microsoftov .NET za razvoj spletnih in namiznih aplikacij.

2.2. Pristopi pri razvoja programske opreme

Pri razvoju programske opreme obstajata, kar se tiče pristopov, dva tabora [1, 2]. Prvi je bolj zgodovinsko naravnan, govorim seveda o discipliniranem pristopu, ki temelji na »waterfall« modelu. Drugi tabor pa so bolj sodobne metodologije. Tukaj srečamo iterativni in inkrementalni, spiralni in agilni razvoj.

2.2.1. Tradicionalni pristop – »waterfall« model

Po nekaterih trditvah je bil to prvi formaliziran način za razvoj funkcionalnosti za končnega uporabnika. Model deli cikel razvoja neke rešitve na pet faz [18], prikaz na sliki 1.

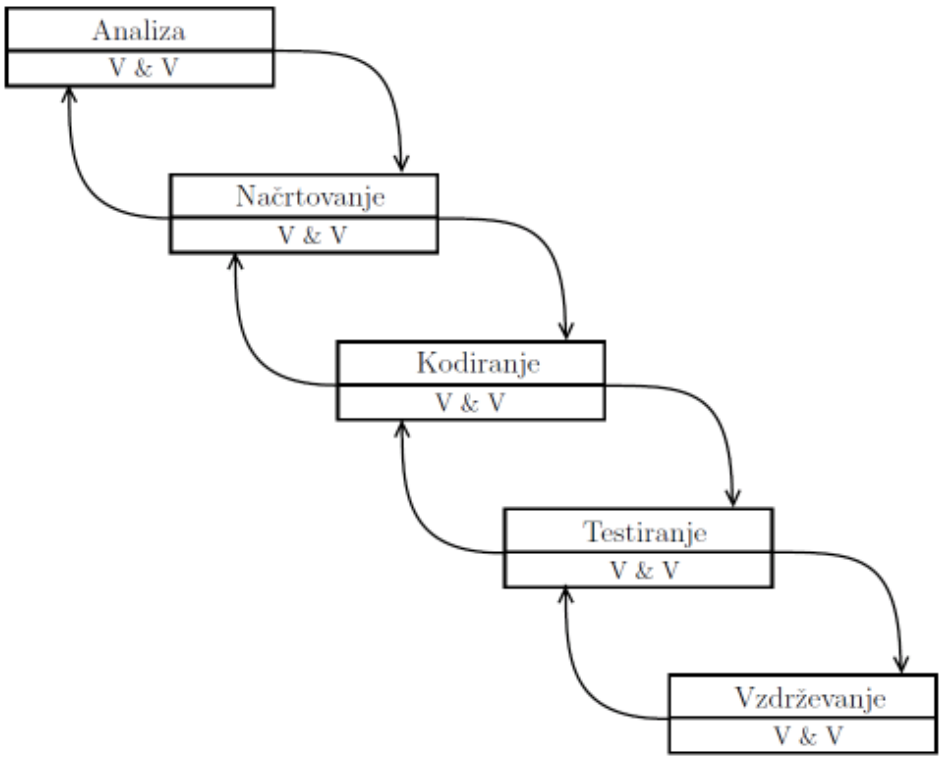
1. Analiza zahtev

2. Načrtovanje
3. Kodiranje
4. Testiranje
5. Vzdrževanje

Skozi dolgotrajno analizo analitiki in razvojni inžinirji pridejo do celotnega nabora zahtev (slika 1), ki ga temeljito dokumentirajo za naslednjo fazo načrtovanja. Tukaj arhitekti z upoštevanjem znanega zastavijo optimalno arhitekturo rešitve, ki jo razvijajo. Nato programerji implementirajo sistem. Faza testiranja pomeni temeljito preverjanje pravilnosti delovanja razvite programske opreme in pa tudi sprejemno testiranje s strani naročnika. Sledi prehod v produkcijo, kjer se vrši vzdrževanje.

V teoriji na ta način vedno pridemo do delujoče končne rešitve. Vendar pogosto temu ni tako [2]. Sama faza zbiranja zahtev je lahko pri nekaterih projektih tako obsežna, da lahko traja tudi mesece včasih celo leta. V tem času se same zahteve s strani uporabnikov spremenijo v tolikšni meri, da dokumentirana rešitev sploh ni več tisto, kar uporabniki želijo. Problemi nastanejo tudi ob vsaki spremembi s strani uporabnikov, kjer je potrebno spremeniti velike količine dokumentacije, arhitekturo, že implementirano funkcionalnost, ... Ker uporabniki v samem razvoju ne sodelujejo, je za njih končni izdelek lahko presenečenje, saj ni nujno, da so si analitiki in razvijalci pravilno razlagali njihove zahteve.

Iz težav, s katerimi so srečevali med razvojem na ta način, so postopek razvoja spremenili in nastale so nove metodologije.

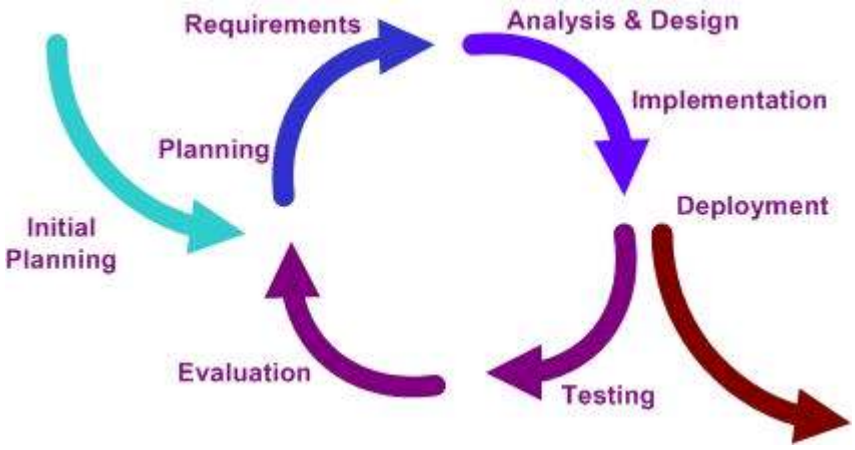


Slika 1: Shema waterfall principa razvoja programske opreme

2.2.2. Iterativni in inkrementalni razvoj

Inkrementalni razvoj je bil evolucija waterfall metode [2]. Še vedno se zahteve zberejo na začetku, vendar se potem implementacija izvaja v več inkrementnih interacijah, ki se lahko delno prekrivajo. Na ta način v vsaki iteraciji realiziramo neko delujočo podmnožico funkcionalnosti. Primarni namen je bil prihranek časa.

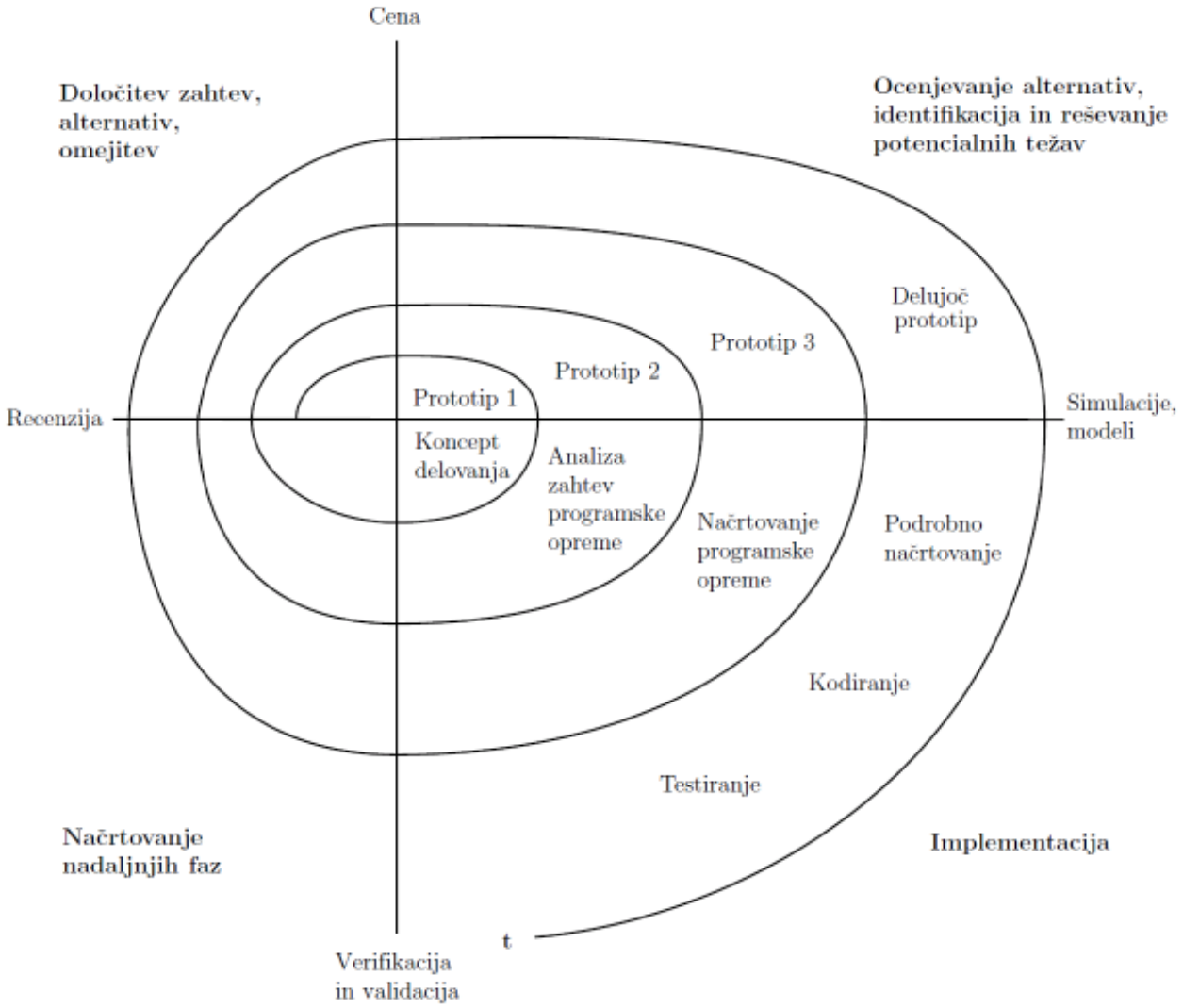
Iterativni razvoj (slika 2) pa se v nasprotju s inkrementalnim ukvarja predvsem z boljšim obvladovanjem tveganj in sprememb med razvojem. Vsaka iteracija se razbije na več faz, ki imajo lahko tudi različne dolžine. Končni rezultat vsake iteracije je delujoč nabor funkcionalnsti, ki gradi na kodi in dokumentaciji prejšnje iteracije. Produkt prve iteracije je zelo osnovna funkcionalnost, ki je osnova za naslednjo. Vsaka iteracija sledi waterfall procesu z analizo, načrtovanjem, implementacijo in testiranjem. Na ta način se razvojna skupina lahko dobro odziva na spremembe, ker so samo zahteve trenutne iteracije končne. Kljub temu je treba imeti v mislih okvirne zahteve za naslednjo iteracijo, ker te niso popolnoma definirane do naslednje faze analize. To omogoča spremembe s strani naročnika brez večjih vplivov na sam potek projekta.



Slika 2: Iterativni razvoj

2.2.3. Spiralni razvoj

Spiralni razvoj, v nasprotju z iterativnim (razvoj prioritiziran po funkcionalnosti) prioritizira razvoj po rizikih [2].



Slika 3: Shema spiralnega razvoja

Princip spiralnega razvoja (slika 3) vsebuje več ciklov, kjer v vsakem naredimo analizo, načrtovanje, implementacijo in testiranje [18]. Izdelki zgodnjih ciklov so prototipi, medtem ko se v kasnejših ciklih razvije že delujoč prototip rešitve, na katerega mora odzive podati tudi naročnik. Sledeči cikli gradijo na že obstoječem (npr. vsak cikel en del sistema). Na koncu lahko vsak cikel pomeni le eno vzdrževalno iteracijo rešitve, ki že deluje v produkcijskem okolju.

Tipičen primer uporabe sprilnega razvoja je v industriji razvoja iger, kjer so projekti veliki in imajo precej visoko frekvenco sprememb skozi potek razvoja.

2.2.4. Agilni pristop k razvoju

Manifest agilnosti [10] je bil produkt konvencije 17 vodilnih strokovnjakov na področju Ekstremnega programiranja (XP), SCRUM, DSDM; Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming in drugih. Skupni cilj je bila potreba po alternativi dokumentacijsko orientiranim, težkim procesom pri razvoju programske opreme. Povzetek njihovega pogleda na agilne metode je bil Manifest Agilnosti. V njem navajajo naslednje ključne vrednote [2]:

- **Posamezniki in interakcija** nad procesi in orodji
- **Delujoča programska oprema** nad obsežno dokumentacijo
- **Sodelovanje s končnimi uporabniki** nad pogajanji o pogodbah
- **Odzivanje na spremembe** nad sledenjem načrta

Kljub temu, da vidijo vrednost v desnih pojmih, bolj cenijo leve. Pomemben sklep konvencije pa je bil tudi, da agilno gibanje ni anti-metodologija, ampak samo odmik od zelo formalističnih klasičnih metod.

V nadaljevanju bom zelo na kratko predstavil nekaj bolj znanih agilnih metodologij.

2.2.4.1 Scrum

Scrum je ena izmed bolj znanih metod in široko uporabljenih agilnih metod. Dokončno pa sta ga utemeljila Jeff Sutherland in Ken Schwaber leta 1995, njegovi začetki pa segajo še približno desetletje nazaj. Metoda je iterativno, inkrementalno ogrodje za agilen razvoj in vzdrževanje pogramske opreme.

Kot omenjeno je Scrum ogrodje, ki vsebuje predefinirane procese, vloge in izdelke. Najbolj pomembni izdelki so [2, 21]:

- **Seznam zahtev** (angl. Product Backlog) ki vsebuje vse zahteve (angl. Product Backlog items), ki jih je potrenbno končati, da bo projekt uspešno zaključen. Vsaka zahteva je običajno predstavljena v obliki uporabniške zgodbe (angl. User Story).
- **Seznam nalog** (angl. Sprint Backlog) vsebuje vse naloge, ki so predvidene za trenutno iteracijo.
- **Delujoča programska koda**, je seveda tisto, kar razvoj želi ustvariti kot končni izdelek. Vse ostalo so, v določeni meri, le pomožni izdelki na poti. Ti sicer koristijo kot dokumentacija razvijalcem, ki kasneje vzdržujejo izdelek.

Vloge v scrum procesu pa so naslednje:

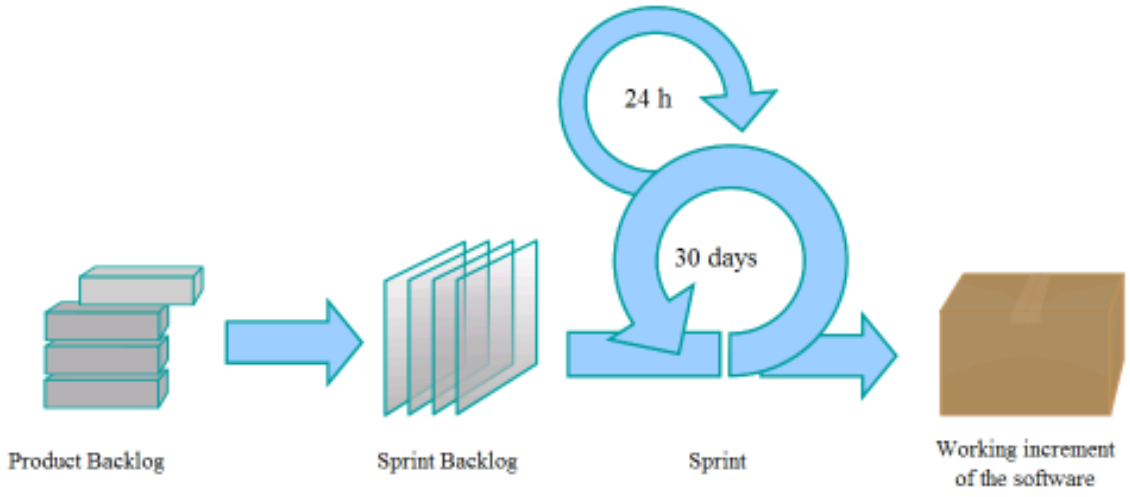
- **Skrbnik metodologije** (angl. Scum Master) je oseba, ki skrbi, da se Scrum proces izvaja, kot je bilo načrtovano. Deluje kot vmesni člen med razvojno skupino in mogočimi motečimi vplivi.
- **Razvojna skupina** (angl. Team) je zbirka ljudi (manjša od 10, zaslediti je priporočila od 5-9 [2]), ki opravi dejansko delo. Sestavljajo jo analitiki, načrtovalci, razvijalci, inžinirji za kvaliteto, ...
- **Predstavnik naročnika** (angl. Product Owner) v razvoju predstavlja glas stranke. Skrbi da razvoj teče v pravi smeri iz poslovne perspektive. Zbira, zapisuje uporabniške zgodbe in jih glede na prioriteto zbira v seznamu zahtev.

Pomemben del Scruma pa so tudi različni tipi sestankov skupine. To so:

- **Dnevni scrum** (angl. Daily Scrum) je 15 minutni dnevni sestanek razvojne skupine, ki poteka vsak delovni dan ob istem času, na istem mestu. Vsak član pove, kaj je naredil od prejšnjega sestanka, kaj bo naredil danes in s kakšnimi težavami se srečuje.

- **Sestanek za planiranje iteracije** (angl. Sprint Planing Meeting) se zgodi pred začetkom vsake iteracije (angl. Sprint). Tukaj se odloči, kaj se bo naredilo v prihajajočem sprintu, razvojna skupina razdeli oceni potreben čas, za posamezno nalogo. Takrat se naloge tudi razdelijo na člane skupine. Sestanek je časovno omejen na 8 ur. Zgodi se pred začetkom vsake iteracije.
- **Sestanek za pregled opravljenega dela** (angl. Sprint Review Meeting) se zgodi po koncu vsake iteracije. Namenjen je pregledu opravljenega in nedokončanega dela ter demonstraciji dokončanega naročniku. Omejen je na 4 ure.
- **Sestanek za oceno izvajanja procesa** (angl. Sprint Retrospective Meeting) je 3 urni sestanek vseh članov skupine namenjen izboljšanju procesa. Vsak član pove, kaj je po njegovem mnenju delovalo dobro, in kaj se ni izkazalo za dobro prakso. Na podlagi tega se lahko naredijo spremembe, ki skupini omogočijo bolj učinkovito izvajanje procesa.

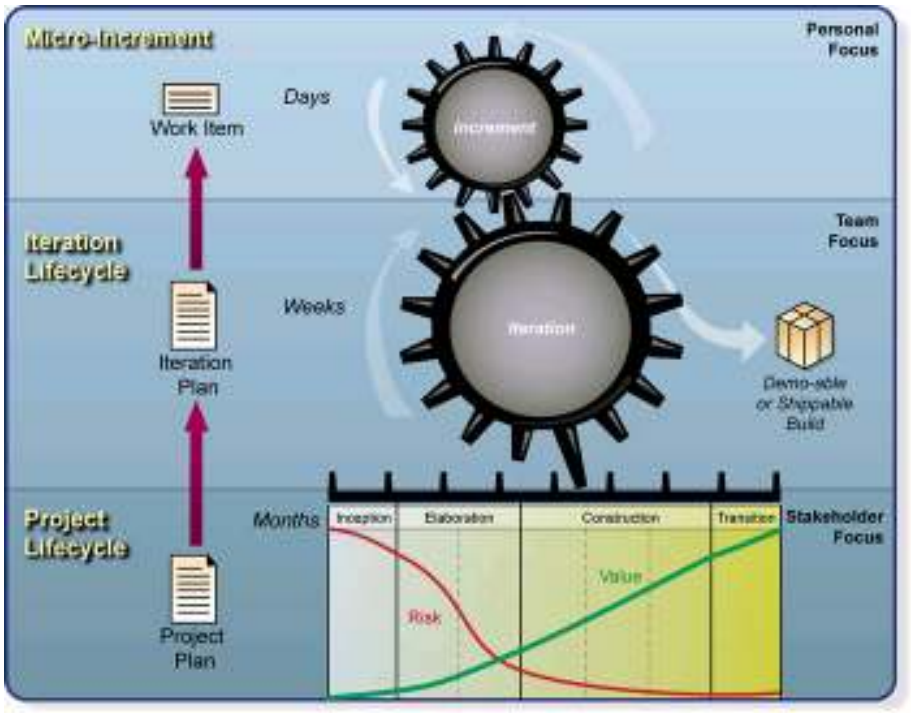
Iz predstavljenih vlog, izdelkov in sestankov pa sledi še potek dela, ki ga prikazuje slika 4. Iz celotnega seznama zahtev se na sestanku za planiranje iteracije izberejo uporabniške zgodbe, ki jih skupina želi implementirati v iteraciji. Vsi pomagajo pri časovni oceni nalog in si jih razdelijo. Iteracija je po procesu določena na 30 dni[21], vendar je v praksi navadno dolga od dveh do štirih tednov[2]. Med iteracijo se dogajajo vsakodnevni scrum sestanki. Po koncu iteracije se naredi oceno le te in predstavi dokončane zahteve naročniku izdelka. Naredi se tudi ocena izvajanja procesa, kjer je fokus na izboljšanju procesa za doseganje še večje učinkovitosti in kvalitete.



Slika 4: Potek Scrum procesa

2.2.4.2 OpenUP

Open Unified Process je del procesnega ogrodja Eclipse (Eclipse Process Framework - EPF), ki se razvija v okviru fundacije Eclipse [8]. V veliki meri bazira na Rational Unified Processu, ki je v svojem bistvu ogrodje, ki ga mora vsaka orgranizacija prilagoditi svojim potrebam.



Slika 5: OpenUP

Slika 5 predstavlja sestavo organizacije projekta v OpenUP metodologiji.

Projektni življenjski cikel je razdeljen v 4 faze, skozi katere se razvija rešitev. S potekom razvoja pridobivamo na vrednosti že narejenga, hkrati pa se tveganja znižujejo. Projektni plan omogoča razvojni skupini in drugim udeležencem v razvoju dober vpogled v trenutno stanje in olajša odločanje skozi celoten potek razvoja[8].

- **Začetek** (angl. Inception**)** vključuje vizijo in zajem zahtev.
- **Izpopolnitev** (angl. Elaboration) vključuje analizo zahtev in načrtovanje.
- **Konstrukcija** (angl. Construction) predstavlja razvoj in preverjanje pravilnosti delovanja.
- **Predaja v uporabo** (angl. Transition), ki predstavlja prehod v produkcijo.

Projekt je razdeljen v iteracije, njih dolžina se navadno meri v tednih. Namen iteracij je, da skupina inkrementalno ustvarja vrednost za naročnika/končnega uporabnika v predvidljivih časovnih okvirjih. Načrt iteracije nalaga, kaj se bo v trenutni iteraciji naredilo. Rezultat vsake iteracije je produkt, ki ga je mogoče demonstrirati ali celo poslati v distribucijo.

Delo osebe v OpenUP je organizirano v kratke enote dela, ki se navadno merijo v urah ali nekaj dneh. Te sestavljajo napredek trenutne iteracije in vsaka od njih daje informacije, ki so podlaga za odločitve o prilagajanju znotraj iteracije. Majhne enote dela pomenije kratko obdobje za pridobivanje povratnih informacij, ki omogoča prilagodljivo odločanje med iteracijo. Proces daje velik poudarek sodelovanju skupine med razvojem.

2.2.4.3 Ekstremno Programiranje (XP)

Ekstremeno programiranje, ponavadi označeno z XP je agilna metodologija za razvoj programske opreme, njeni začetki segajo v 1996 in sicer na velik projektu v korporaciji Chrysler, ki se je imenoval C3. Zaradi težav na projektu je Kent Beck razvil metodologijo, ki jo je 1999 opisal v knjigi Extreme Programming Explained[22].

Glavni cilj metode je zmanjševanje cene sprememb v razvojnem ciklu na ta način, da tega razdeli na več krajših podciklov. Na ta način postanejo spremembe nekaj vsakdanjega. Za doseganje teh ciljev so predlagane vrednote, dobre prakse in aktivnosti.

Definirane aktivnosti[2]:

- **Kodiranje** je najpomembnejši del v razvoju programske opreme – brez kode ni izdelka.
- **Testiranje** nam pove, ali funkcionalnosti, ki smo jih implementirali, delujejo tako kot smo jih načrtovali. XP izhaja iz izreka, da s testiranjem odkrivamo napake, s še več testiranja odkrivamo še več napak. Testi se delijo na 2 dela: test enot (angl. Unit Tests), ki so avtomatizirani in uporabniške sprejemne teste (angl. Acceptance Tests).
- **Poslušanje** s strani razvijalca je ključno za razumevanje, kakšna je poslovna logika funkcionalnosti, ki jo želi končni uporabnik.
- **Načrtovanje** postane vedno večji dejavnik s tem, ko se povečuje kompleksnost sistema.

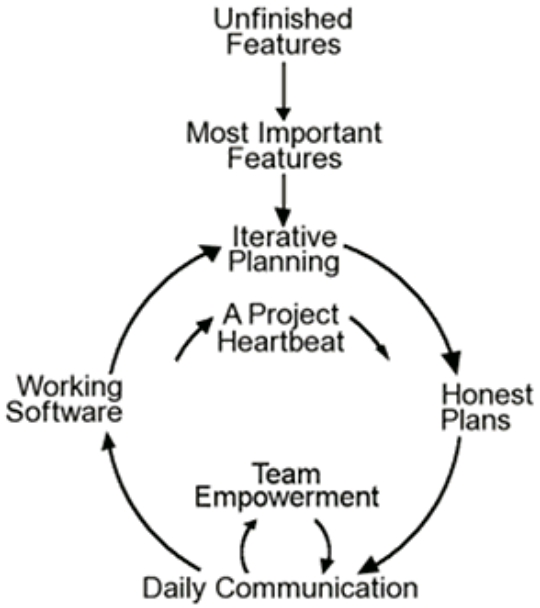
Definirane vrednote:

- **Komunikacija** je ključna pri izdelavi sistemov. Pri bolj formalnih metodologijah se to doseže s pomočjo dokumentacije.
- **Enostavnost** je zaželjena, saj tudi tukaj velja načelo, da se dodatna funkcionalnost lahko doda tudi kasneje. Po načelih XP je bolje razvijati za današnje potrebe.
- **Povratne informacije** se pri XP delijo na 3 dele. Rezultati testov enot ali rezultati integracijskih testov so povratna informacija od sistemov. Drugi del so odzivi naročnika na teste, ki so jih sestavili sami. Ti se izvajajo na nekaj mesecev. Tretja povratna informacija pa so ocene potrebnega časa za implementacijo sprememb, ki jih zahteva naročnik.
- **Pogum** je potreben pri načrtovanju in implementaciji funkcionalnosti za danes in ne za jutri. Razvijalci morajo pogumno spremeniti že obstoječo kodo, da implementirajo potrebne spremembe, ali pa jo zavreči, če le ta ni več uporabna.

- **Spoštovanje** med člani skupine pomeni, da vsak upošteva to, da nikoli zavestno v sistem ne vnaša sprememb, ki ne prestanejo vseh testov enot, se ne prevedejo ali na kakršen koli drug način ovirajo ostale člane razvojne skupine.

Poleg naštetih vrednot in aktivnosti XP definira še 12 praks, od katerih bom omenil samo nekaj najbolj zanimivih.

- **Programiranje v parih** (angl. Pair Programming). Vsa koda je napisana v paru za enim računalnikom, kjer eden kodira, drugi pa pregleduje. Ob tem se pogosto menjujeta. Tudi člani parov se menjujejo, tako da ne delajo vedno v istih parih. Ker je lastništvo kode skupno (vsak član skupine si lasti vso kodo) takšno programiranje pomaga pri poznavanju celotnega sistema. Hkrati manj izkušeni člani skupine hitreje napredujejo ob stalnem sodelovanju z bolj izkušenim sodelavcem.
- **Testno voden razvoj** (angl. Test Driven Development) razvoj pred samim kodiranjem funkcionalnosti predvideva razvoj avtomatiziranih testov, ki le to testirajo.
- **Predstavnik naročnika v razvojni skupini** (angl. On Site Customer) je praksa vključevanja končnega uporabnika v vseh fazah razvoja. Vedno mora biti na voljo za vprašanja razvijalcev.
- **Skupno lastništvo kode** (angl. Collective Code Ownership) pomeni, da so vsi odgovorni za vso kodo. Izmenjevanje parov pomaga pri spoznavanju z drugimi deli sistema. To pohitri razvoj, saj ob primeru napake katerikoli razvijalec lahko popravi napako v kateremkoli delu sistema.
- **Enakomeren tempo** izhaja iz tega, da razvijalci nikoli ne delajo več kot 40 ur na teden. Ljudje se najbolje izkažejo, če so spočiti in motivirani.



Slika 6: Shema poteka XP

Slika 6 prikazuje potek XP vodenega projekta. Iz seznama nedokončanih zahtev izberemo najbolj pomembne, začrtamo plan iteracije, ki je izvedljiv in usklajen med naročnikom ter izvajalcem. S pomočjo razvojne skupine razvijemo delujočo programsko opremo. Izdelki iteracije so podlaga za naslednjo iteracijo.

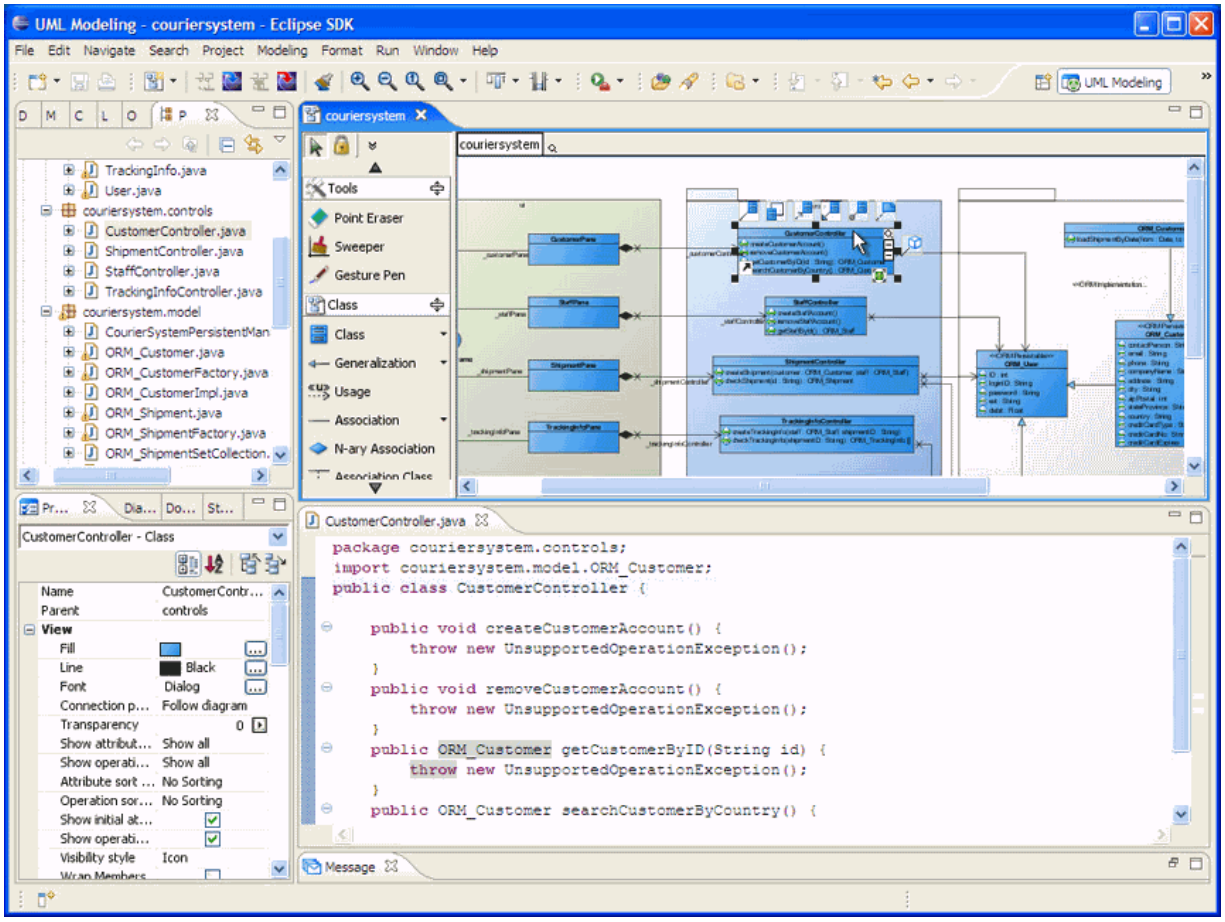
2.3. Orodja

Če primerjamo orodja za podporo razvoju programske opreme pred na primer že 15 leti je napredek očiten. K temu je veliko pripomogel hiter razvoj zmogljivosti osebnih računalnikov z zadnjem času. Orodja so postala vedno bolj zmogljiva in pokrivajo vedno večji spekter v procesu razvoja programske opreme. Če je pred 30 - 40 leti razvoj potekal v preprostem tekstovnem urejevalniku (slika 7), koda se je shranjevala na glavnem računalniku, za vse ostalo je bil v uporabi papir in svinčnik. Pred 15 leti so bili sicer jeziki bolj napredni, vendar orodja niso veliko napredovala (npr. Borland Turbo Pascal vizualno ni veliko drugačen od prej omenjenega vmesnika).

```
 File  Edit  Confirm  Menu  Utilities  Compilers  Test  Help
--------------------------------------------------------------------------------
EDIT       SCOMSTO.M214.LIBRARY(MSGTSTCO) - 01.07          Columns 00001 00072
Command ===>                                                Scroll ===> CSR
000553        do-the-work.
000554            move in-part-number        to reprt-part-number
000555            move in-description        to reprt-description
000556            move in-quantity-on-hand   to reprt-quantity-on-hand
000557            move in-quantity-on-ord    to reprt-quantity-on-ord
000558            move in-unit-price         to reprt-unit-price
000559            move in-reorder-level      to reprt-reorder-level
000560            write reprt-rec from reprt-record
000561            perform read-a-record.
000562
000563
000564        print-table.
000565            move parts-no(part-index)
000566                      to reprt-part-number
000567            move parts-desc(part-index)
000568                      to reprt-description
000569            move parts-on-hand(part-index)
000570                      to reprt-quantity-on-hand
000571            move parts-on-ord(part-index)
000572                      to reprt-quantity-on-ord
```

Slika 7: Primer razvoja COBOL-skega programa v ISPF urejevalniku na glavnem računalniku

Dandanes pa so orodja že na zelo visokem nivoju in pokrivajo celoten spekter razvoja od zajema zahtev in nadaljnjega razvoja le teh s prototipi, modeliranja arhitekture aplikacije, razvoja v integriranem okolju (slika 8, prikazuje razvoj Jave v Eclipse okolju), do avtomatizirane graditve (angl. build) in testiranja. Na koncu seveda tudi do avtomatiziranih namestitev. Čez cel cikel lahko sledimo od zahteve do sistema za upravljanje z izvorno kodo, testa in končne zgrajene verzije programa.



Slika 8: UML in Java razvoj v orodju Eclipse

Ves ta napredek je v veliki meri osredotočen na produktivnost enega razvijalca. Sodelovanje skupin v orodjih tega tipa ni zelo dobro podprto, čeprav predstavlja ključen del pri uspešnem razvoju, še posebno danes, ko zunanje izvajanje del in razdeljene skupine niso nekaj nenavadnega. Pomanjkanje pregleda nad nalogami, problemi in delitvijo dela med člani skupine, ki se ne nahajajo na isti lokaciji lahko vodi v težave [19]. V zadnjem času se izdelovalci orodij vedno bolj zavedajo tudi tega vidika razvojnega procesa in tudi orodja se razvijajo tudi v to smer.

2.4. Izzivi na področju razvoja

Orgranizacije, ki razvijajo programsko opremo, so v današnjih časih v nezavidljivem položaju. Globalizacija je na splošno trg naredila izredno konkurenčen in podpora kakršnemu koli poslovanju se mora zelo hitro odzivati na spremembe. To za razvojne organizacije pomeni, da je kritično, da izdelke dokončajo v roku, v okviru proračuna in na visoki ravni kvalitete. Če k vsemu temu dodamo še visoko frekvenco sprememb, dobimo relativno težko nalogo. Dodatno trend zunanjega izvajanja (angl. outsourcing) še bolj poveča zahteve po sposobnosti skupine, da sodeluje na razdaljo.

Za uspešno spopadanje z nekaterimi od teh izzivov je potrebno prilagajanje načina dela, pri drugih nam lahko v veliki meri pomagajo orodja. Vse narekuje razvoj orodij za podporo razvojnemu procesu, ki ne bodo osredotočena le na posameznika, ampak bodo usmerjena v usklajevanje dela celotne razvojne skupine. Na tem področju se orodja pojavljajo zadnjih nekaj let.

3. Platforma Jazz

3.1. Uvod

Platforma Jazz [23, 24] je bila osnovana na podlagi IBM-ove pobude za preoblikovanje razvoja in razvojnih skupin v bolj učinkovito celoto. Jazz želi to doseči z inovativnimi prijemi za dvig nivoja sodelovanja, produktivnosti in transparentnosti. Platforma je sestavljena iz treh ključnih delov:

- **Arhitektura za integracijo življenjskega cikla aplikacij** stremi k enotnemu sistemu za podporo upravljanja za aplikacijo od same ideje, zbiranja in razvoja zahtev, načrtovanja, kodiranja, testiranja do na koncu namestitve. Na tem bazira platforma Jazz.
- **Portfelj orodij, ki skupino postavlja na prvo mesto** je sestavljen iz vseh orodij, ki se uporabljajo za razvoj aplikacije. Pomembno pri tem pa je, da ta v veliki meri olajšajo sodelovanje med člani širše razvojne skupine. Ta vključuje vse vloge od naročnika do razvijalca.
- **Skupnost ljudi, ki lahko aktivno vpliva na razvoj arhitekture** pa se nahaja na spletnem portalu Jazz. Vsi, ki želijo, lahko s svojimi idejami in izkušnjami pri uporabi vplivajo na smer, v katero se platforma razvija. To ni samo omgočeno ampak tudi zelo zaželjeno s strani snovalcev.

Vse to cilja na pogosto pozabljeno dimenzijo razvoja: Razvojno skupino. Sodelovanje je v tej panogi izrednega pomena in samo usklajena skupina lahko uspešno pripelje zapletene projekte do konca. Drugi pomemben cilj platforme Jazz pa je poenostavitev integracije celotnega življenjskega cikla razvoja v dobro povezano celoto s poenotenjem načina povezovanja nabora orodij iz različnih faz razvojnega cikla.

3.2. Cilji

3.2.1. Skupinsko delo

V tradicionalnem okolju za razvoj programske opreme je vedno dilema, kaj je bolj pomembno: strog proces ali produktivnost posameznika. To se v veliki meri izključuje. Tudi naročniki, oz. končni uporabniki, imajo redko možnost vplivati na zahteve, razen pri večjih revizijah projekta. Orodja, ki temeljijo na platformi Jazz skušajo premostiti te ovire in delujejo na ta način, da center razvoja ni orodje, proces, posameznik ampak komunikacija. V razvojno skupino vključuje vse, ki imajo delež v projektu. Cilj je transparentnost skupin in projektov za boljše sodelovanje v pravem kontekstu.

3.2.2. Avtomatizacija

Vse organizacije lahko pridobijo z avtomatizacijo zapletenih, dolgotrajnih nalog, kjer zaradi komplesnosti prihaja do napak. Hkrati je potrebno ohraniti ali izboljšati ponovljivost in pregled nad napredkom projekta. Cilj platforme Jazz je avtomatizacija procesov, delovnih tokov in nalog, da lahko organizacije prihranjen čas bolje izkoristijo. Hkrati pa z avtomatizacijo

zapletenih procesov tudi zmanjšujemo število človeških napak, ki se dogajajo ob njihovem pogostem izvajanju.

3.2.3. Poročanje

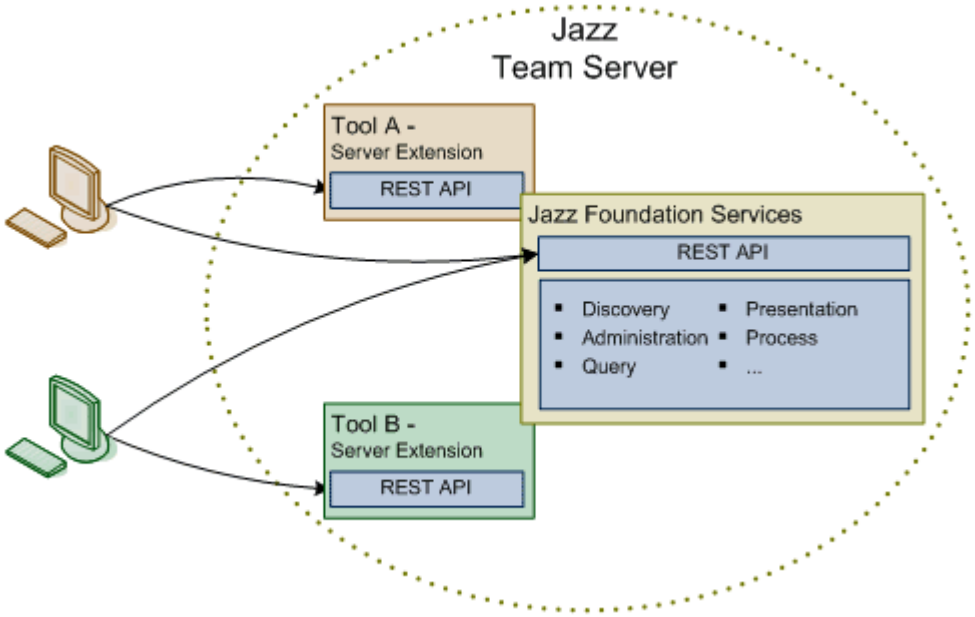
Pogosto se informacije o stanju projektov na področju razvoja zbirajo ročno in od takrat, ko so podatki zbrani, do interpretacije in predstavitve ključnim osebam, so le ti že zastareli. Zaskrbljujoče je, koliko časa lahko ključni ljudje v razvoju, razvijalci, porabijo za izpolnjevanje poročil o opravljenem delu. Platforma Jazz stremi k zagotavljanju vpogleda v programe, projekte in vire v realnem času. To nam omogoča hitrejše zaznavanje težav, kar nam pomaga pri lažjem odločanju. Hkrati so podatki točni in z njimi lahko lažje izboljšujemo učinkovitost razvojne skupine in posameznika.

3.3. Jazz Integration Architecture (JIA)

3.3.1. Pregled

Glavni cilj integracijske arhitekture platforme Jazz [5] je omogočanje sočasne uporabe nabora raznolikih orodij z zagotavljanjem povezane izkušnje svojim uporabnikom. Sicer to ni enovita rešitev, ampak množica medsebojno povezanih tehnologij in specifikacij. JIA vsebuje referenčno arhitekturo, specifikacijo API in množico osnovnih storitev in osnovnih gradnikov. To naslavlja potrebe pri gradnji novih orodij in omogočanju hitre integracije obstoječih. Vsa orodja, ki temeljijo na platformi Jazz, uporabljajo našteto za doseganje dobre povezanosti med posameznimi orodji.

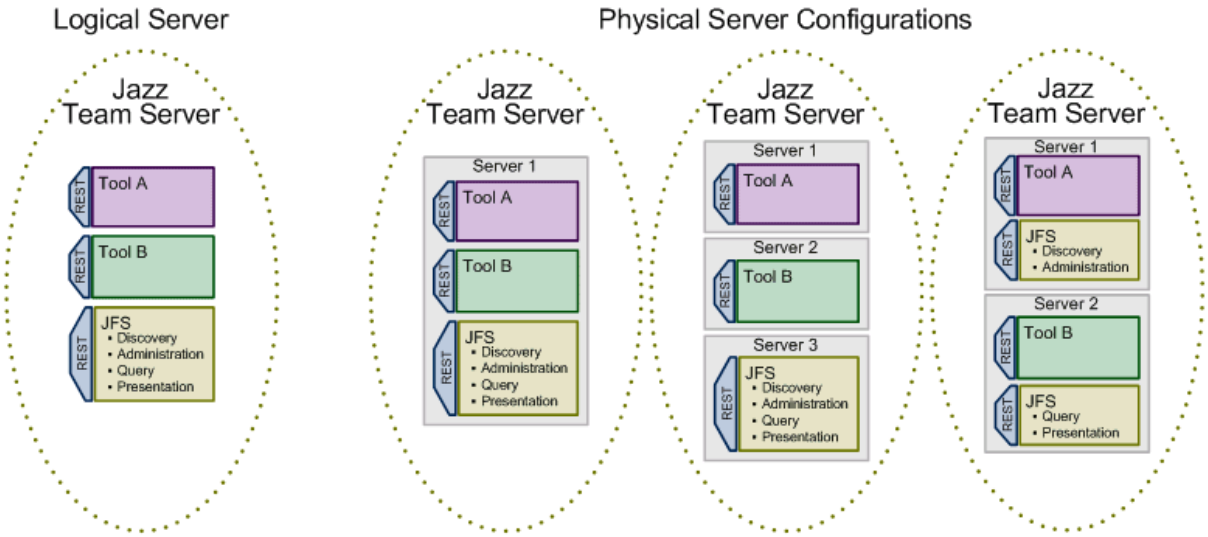
Srce JIA je Jazz Team Server (JTS), ki zagotavlja Jazz Foundation Services (JFS). To so temeljne storitve, ki množici orodij omogočajo skupno delovanje. To so upravljanje z uporabniki in projekti, varnost, sodelovanje, iskanje in druge. Za dostopanje do storitev je uporabljen standardiziran vmesnik REST API, ki sodelujočim orodjem omogoča preprosto skupno delovanje. Orodja so šibko vpeta (angl. loosely coupled), kar pomeni, da fizično ni pomembno ali se določene storitve fizično nahajajo na enem ali na več JTS. Slika 9 prikazuje en JTS, na katerem tečejo JFS, ki nudijo svoje storitve uporabnikom. Dodatno sta na tem JTS na voljo še 2 razširitvi, ki izkoriščata JFS za ponudbo dodatnih storitev.



Slika 9: Jazz Team Server okolje

3.3.2. Razširitev JTS

Podpora razširitvam osnovnih storitev (JFS) je ključnega pomena za gradnjo novih ali vključevanje obstoječih orodij v Jazz. Vsaka razširitev je specifična funkcionalnosti nekega orodja, ki seveda izkorišča že obostoječe funkcije JFS, hkrati pa kot razširitev izpostavi tudi svoje storitve na enak način (s pomočjo REST). Hkrati je arhitektura Jazz zasnovana na tak način, da fizično ni pomembno, ali se nabor orodij nahaja na istih JTS, ker se integracija vrši prek vmesnika REST. Slika 10 prikazuje več fizičnih načinov implementacije enega logičnega JTS.



Slika 10: Primer več načinov fizične implementacije logičnega Jazz Team Serverja z dvemi orodji.

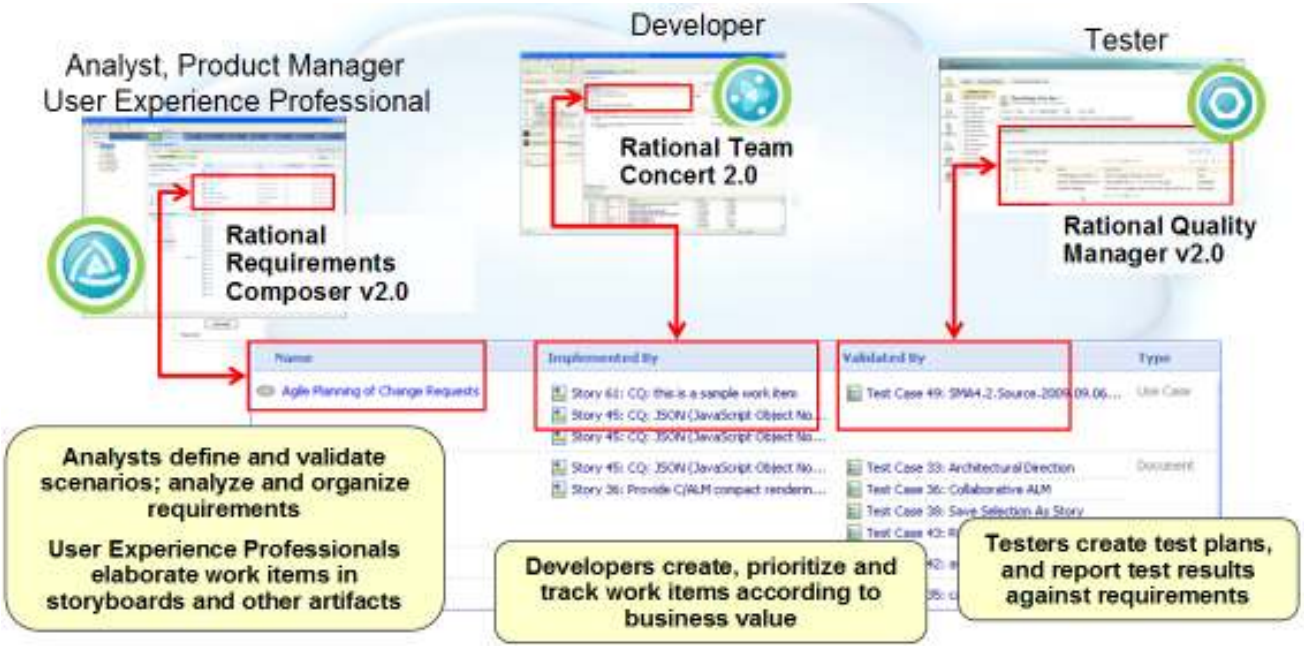
Iz vidika uporabnika je dostop do več orodij baziranih na Jazz arhitekturi povezana izkušnja. Naj to bolje pojasnim na primeru. Na JTS imamo nameščene 2 razširitivi. Prva je Rational Team Concert, kjer upravljamo z razvojem. Druga je Rational Quality Manager (RQM), ki omogoča upravljanje s testnimi primeri, izvajanjem testiranja in poročanjem nad stanjem aplikacije iz vidika kvalitete. Arhitektura Jazz predvideva povezovanje izdelkov iz enega in drugega orodja (uporabniška zgodba v RTC s testnim primerom v RQM) za vzpostavljanje sledljivosti in boljšega pregleda med procesom razvoja. To povezovanje danes poteka prek povezave projektov v obeh orodjih (anlg. project area link). To se naredi prek spletnega naslova drugega strežnika, če pa projekt obstaja na istem strežniku pa komunikacija prav tako poteka preko REST klicev. Zato je uporabnika popolnoma vseeno ali se 2 razširitvi, ki jih povezujemo med sabo nahajata na istem strežniku. Ob kliku na povezavo je preusmerjen na izdelek na drugem strežniku. Ob uporabi enotne prijave (angl. single sign on) je izkušnja popolnoma povezana in uporabnik se ne zaveda, da gleda izdelek iz drugega orodja.

3.4. Skupinsko upravljanje z življenjskim ciklom aplikacije (CALM)

Upravljanje z aplikacijskim življenjskim ciklom (ALM) združuje vse discipline, ki jih srečamo pri razvoju programske opreme. Od zbiranja zahtev, upravljanja z viri, razvoja, graditve (angl. build), testiranja do izdaje in podpore izdelka v produkciji [9]. Dandanes je povezovanje med njimi bolj nujno kot kdajkoli prej zaradi vedno večje kompleksnosti, krajših življenjskih dob aplikacij in vedno bolj omejenih časovnih intervalov za razvoj.

Izolirano delovanje razvojnih skupin v takšnem okolju preprosto ni več mogoče. Posamezni deli razvojnega cikla morajo biti med sabo dobro povezani. Člani skupine se ne morejo več popolnoma specializirati v npr. razvoj zahtev, kodiranje ali testiranje. Analitiki so vedno bolj povezani s procesom razvoja, razvijalci morajo detajlno razumeti potrebe naročnika, testerji pa morajo bit tesno povezani z analitiki, da lahko zagotovijo visoko kvaliteto končnega izdelka.

Od tu skovanka skupinsko upravljanje z življenjskim ciklom aplikacije (CALM). Daje pomen sodelovanju med poslovnim, razvojnim in upravljalskim segmentom aplikacije. Ta scenarij pa morajo podpirati tudi orodja. Slika 11 prikazuje, na kakšen način orodja, zasnovana na plaformi, Jazz podpirajo CALM. V Rational Requirements Composer-ju analitiki ustvarijo zahteve, modele poslovnih procesov, ki jih pokrivajo zahteve in uporabniške zgodbe. Načrtovalci uporabniških vmesnikov napravijo načrte ekranov. Te zahteve in drugi izdelki se povežejo z nalogami v razvoju in so razvijalcem z lahkoto na voljo prek spletnega vmesnika in obratno. Povežemo pa tudi testne primere v Rational Quality Manager-ju (RQM)in delovne naloge iz razvoja. Na ta način sledimo scenariju od zahteve do testa.



Slika 11: Prikaz delovanja orodij, ki so zasnovana na platformi Jazz v CALM

4. IBM Rational Team Concert

4.1. Uvod

Rational Team Concert (RTC) je bilo ob izdaji prvo orodje, ki bazira na arhitekturi Jazz in za svoje delovanje uporablja JTS in JFS. Je okolje za razvoj programske opreme, ki pomaga posameznikom in skupinam dosegati visoko raven učinkovitosti. Ustvarjeno je, da povezuje razpršene razvojne skupine, za izboljšanje posameznikove in skupinske produktivnosti, za skrajšanje razvojnih ciklov in dviganje kvalitete programske opreme. Nudi nadzor nad verzijami programske opreme, procesno vodenje, upravljanje z delovnimi nalogami, upravljanje z graditvami izvorne kode (angl. build) in še veliko več.

Rational Team Concert skupinam omogoča kreiranje delovnih predmetov na projektu in sledenje njihovemu napredku v skladu z razvojnim procesom in projektnimi pravili. To omogoča učinkovit tok informacij o napakah, izboljšavah in olajšano komunikacijo med udeleženci v razvoju. Z napredkom dela beleži tudi kdo, kaj, kdaj in zakaj v povezavi z vsakim predmetom, kar nam predstavlja kontekst, ko se delo deli med člani skupine. Izboljšuje produktivnost s sledenjem sprememb na končnih produktih razvojnega projekta in omogoča preprost povratek na prejšnje verzije. Podpora potrjevanju sprememb in diskusijam skrbi, da se pregledi kode in potrditve zgodijo pravočasno in člani razvojne skupine ostanejo usklajeni.

4.2. Kako deluje

RTC je zasnovan kot odjemalec strežnik aplikacija. Strežniški del je nabor nekaj Java Enterprise Edition spletnih aplikacij, ki tečejo na aplikacijskem strežniku, ki je lahko Tomcat ali pa Websphere Application Server. Za shranjevanje vseh podatkov pa uporablja relacijsko podatkovno bazo. Za namestitev do 10 uporabnikov je to lahko Apache Derby, za več pa IBM DB2, Microsoft SQL Server ali Oracle Database.

Za komunikacijo z odjemalci uporablja spletni protokol HTTP oz. kriptirano verzijo HTTPS. Bolj natančno je to mešanica klicev spletnih storitev in HTTP zahtev/odgovorov, ki se imenuje REST. Za zagotavljanje integracije s storitvijo klepetanja in zavedanja prisotnosti pa uporablja protokol jabber ali Lotus Sametime.

RTC ima 3 vmesnike. Primarni je integrirano razvojno okolje Eclipse z naborom vtičnikov, ki omogočajo komunikacijo z strežniško komponento. Drugi je polno funkcionalen spletni vmesnik, ki omogoča vse od pregledovanja poročil, urejanja delovnih nalog do ogleda izvorne kode. Tretji pa je orodje za poganjanje iz ukazne vrstice, ki omogoča dostop do funkcionalnosti za upravljanje z izvorno kodo.

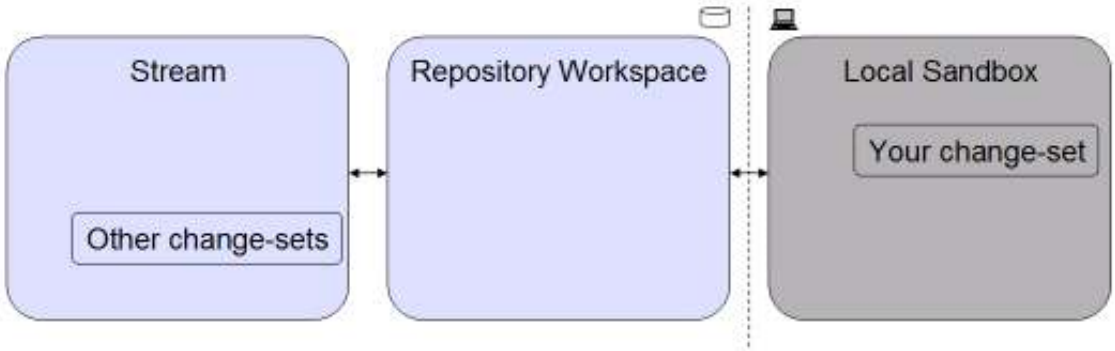
4.3. Pregled funkcionalnosti

4.3.1. Upravljanje izvorne kode

Sistem za upravljanje z izvorno kodo v RTC je hkrati zmogljiv in v primerjavi z nekaterimi drugimi orodji, ki omogočajo delovanje v tako velikih okoljih (RTC lahko podpira tudi več tisoč uporabnikov), relativno preprosto za administracijo [19]. Najprej je potrebno povdariti, da sistem ne temelji na datotekah, ampak kodo shranjuje v relacijsko bazo. Za končnega uporabnika sicer to ne predstavlja razlike.

Uporabnik, ki se prvič sreča z RTC Source Code Management (SCM) mogoče potrebuje nekaj več časa za spoznavanje s sistemom, zaradi dvofaznih potrditev. Če uporabnik iz repozitorija prenese kodo, se ta najprej shrani v kopiji na strežniku, ki je vidna samo uporabniku (slika 12). Sinhronizacija njegovega lokalnega delovnega prostora in strežniškega uporabniškega delovnega prostora je avtomatska. S to sinhronizacijo svojih sprememb še ni delil s celotno skupino. To se zgodi v drugi fazi, ki se imenuje dostava (angl. deliver) in pomeni, da se vse spremembe iz lokalnega in privatnega strežniškega delovnega prostora (ki sta vsebinsko enaka) zapakirajo v množico sprememb (angl. change set), ki se v celoti deli s skupino. Ta dostava je atomarne narave in se zgodi v celoti ali pa sploh ne. To onemogoča, da bi uporabnik delil samo del spremenjenih datotek in s tem ustvaril zmedo v repozitorju. Vsako tako množico sprememb lahko povežemo tudi z delovno nalogo in s tem vsaki spremembi damo tudi kontekst. Hkrati ima uporabnik lahko več privatnih strežniških prostorov in v vsakem dela na drugi veji kode, med njimi preklaplja brez težav.

Še nekaj besed o organizaciji v repozitoriju. Projekte, na katerih delamo (npr. nabor java projektov v Eclipse), združimo v komponente. Ko se odločimo za prenos projekta v sistem, določimo, v katero komponento sodi. Komponente nato združujemo v tokove, ki predstavljajo večje celote (celotna aplikacija, velik podsistem). Organizacija je seveda odvisna od potreb razvoja. Celotne komponente lahko preprosto zamrznemo v določeni točki in jih obdržimo kot referenco (mejniki, izdane verzije, ...).



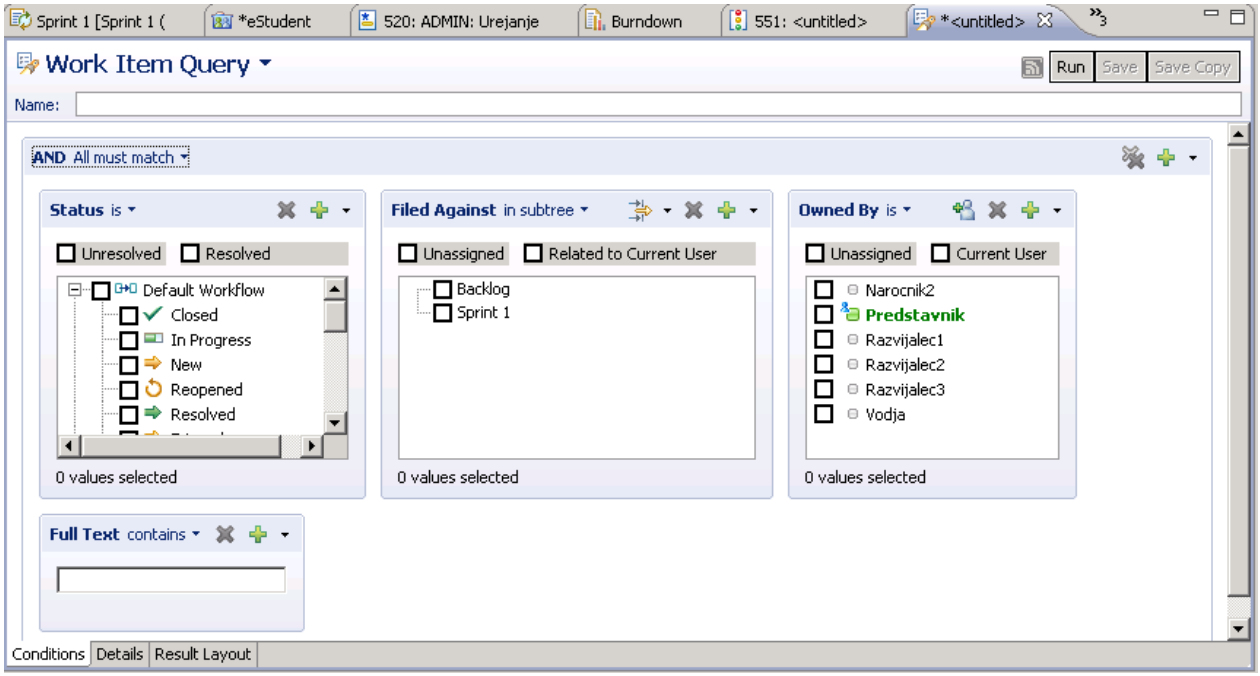
Slika 12: Osnovna anatomija RTC SCM

4.3.2. Upravljanje z delovnimi nalogami

RTC omogoča popolnoma prilagodljivo strukturo delovnih nalog različnih tipov (naloga, zgodba, napaka, ...), ki omogočajo zbiranje informacij o tem, kaj je treba opraviti, kje, do kdaj, s kakšno prioriteto, s kakšno oceno oceno in sledenjem časa. Vsaka delovna naloga vsebuje tudi možnost diskusije, priponk, obveščanja in potrjevanja. Prilagajamo pa lahko tudi, na kakšen način naloga prehaja skozi življenjski cikel (preprost primer: ustvarjeno, popravljeno, rešeno, testirano). Delovne naloge se lahko povezujejo med seboj, tako da lahko ustvarjamo hierahije in prehajamo od splošnih do vedno bolj specifičnih. Povezujemo lahko tudi razne druge artefakte od dokumentov do zgrajenih verzij. Delovne naloge so predstavljene v poglavju 4.3.1.

Vse te naloge lahko damo v plan, ki mu lahko določimo začetek in konec. Na podlagi prioritete, že opravljenega dela in zasedenosti razvijalcev, ki so jim naloge določene, orodje tudi oceni s kakšno verjetnostjo bo določena naloga opravljena pred koncu plana.

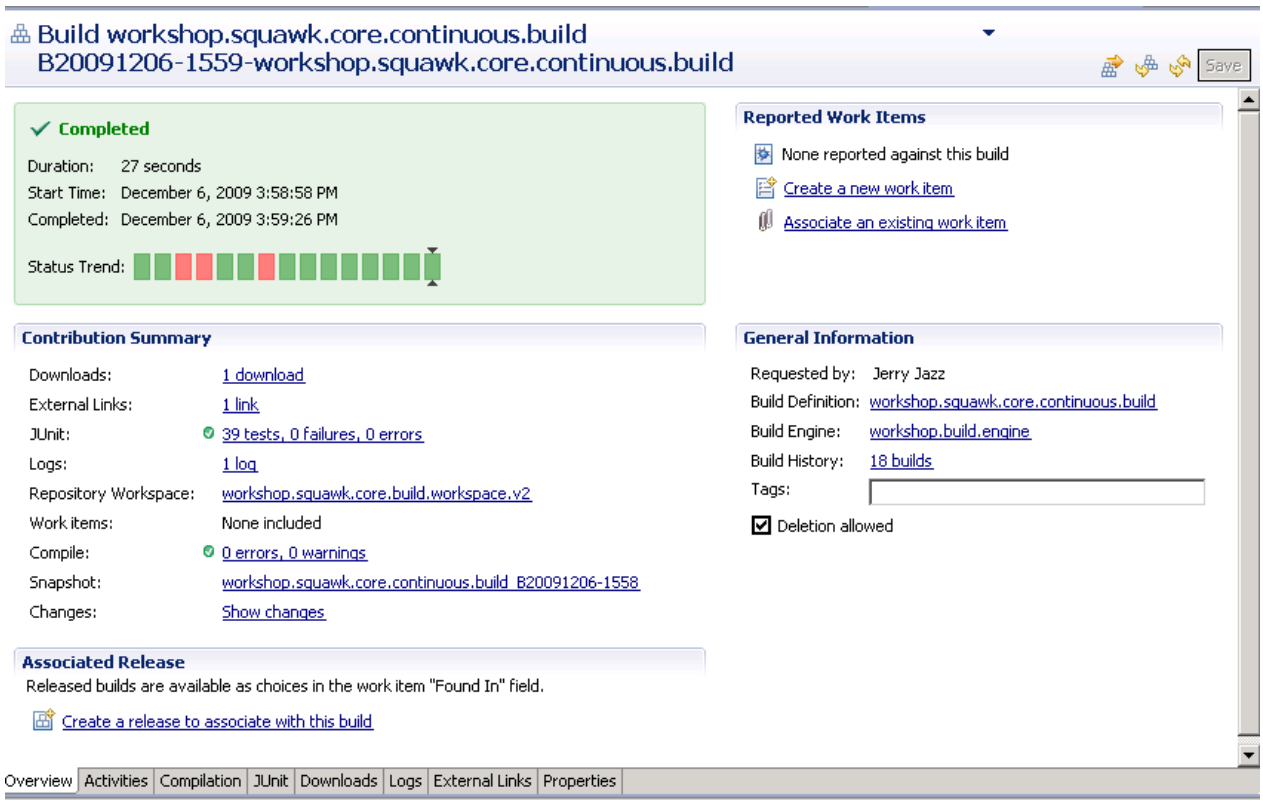
Glede na to koliko informacij je zbranih v delovnih nalogah je pomembno tudi iskanje po njih. RTC ima zmožnosti iskanja s pomočjo poizvedb (angl. Queries). Te poizvedbo imajo možnost vizualnega urejanja, lahko jih razvijemo in shranimo za kasnejšo uporabo, lahko pa jih tudi delimo z drugimi skupinami, ali pa samo specifičnimi člani skupin. Slika 13 prikazuje oblikovanje takšne poizvedbe. Prilagajamo jo lahko v detajle, glede na polja, glede na plan v katerem se nahaja, glede na to v katerem stanju je. Hkrati lahko uredimo tudi na kakšen način so predstavljeni rezultati, ki jih poizvedba vrne.



Slika 13: Prikaz oblikovanja poizvedbe za iskanje med delovnimi nalogami

4.3.3. Upravljanje z graditvijo izvorne kode (angl. build)

Graditev izvorne kode med razvojem je najboljši pokazatelj trenutnega zdravja projekta v vsakem trenutku. Poganjanje testov enot (angl. unit tests) pri vsaki graditvi pa nam da še dodatno informacijo o stanju. RTC v povezavi z Jazz Build Engine (JBE) nam omogoča prav to. Definiranje več različnih urnikov za posamezne dela projekta, je lahko vezano na podskupino ali pa integracijsko za celoten projekt. Ob izvršeni graditvi razvojna skupina takoj dobi povratno informacijo o napakah ali opozorilih prevajalnika, o rezultatih poganjanja testov, o trajanju in pa tudi opcijo prenosa končnega artefakta (slika 14). Vsaka graditev ima tudi povezavo do vseh delovnih nalog, ki so povezane na spremembe, ki so se zgodile od prejšnje graditve (Show changes povezava spodaj desno na sliki 14). Tako lahko takoj vemo, kaj je bilo spremenjenega, če slučajno pride do napake v testih ali prevajanju. Pred vsako graditvijo pa RTC tudi avtomatsko naredi posnetek stanja. Tako lahko ob napakah razvijalec uporabi točno tisto kodo, ki je bila uporabljena. Na preprost način si ustvari nov strežniški delovni prostor iz posnetka in dela na kodi. Če pa nekdo ustvari delovno nalogo, ki se sklicuje na to graditev pa je tudi to prikazano (zgoraj desno na sliki 14).



Slika 14: Primer graditve v RTC

4.3.4. Procesno vodenje projekta

RTC ima tudi zmožnosti definicije pravil za razvojni proces. Že vključene v produkt so predloge za Scrum, OpenUP in EclipseWay. Uporabniki lahko proces prilagodijo svoji organizaciji in željam. Zraven sodi še planiranje izdaj in iteracij, hkrati pa tudi ocena napredka prek avtomatiziranih nadzornih plošč. Možno je tudi definirati pravila glede na fazo v razvoju. Npr. ob stabilizaciji končne verzije je treba vsako spremembo, ki se doda v repozitorij, povezati z delovno nalogo, ki ima potrditev od vodje skupine.

Največja prednost funkcionalnosti za procesno vodenje je njegova prilagodljivost. Vsaka organizacija si lahko popolnoma po svoje definira delovne naloge, atribute v njih in na kakšen način prehajajo med sabo. Za veliko večino operacij v RTC je možno tudi zelo detajlno nastaviti varnostne pravice glede na uporabnikovo vlogo.

4.3.5. Poročanje

Izrednega pomena pa je tudi vpogled v samo stanje projekta. Za ta namen RTC omogoča izdelavo obširnih poročil o vseh aspektih projekta. V produkt so že v startu vključene predloge poročil, ki pokrijejo vse od frekvence buildov do števila odprtih nalog poljubnega tipa. Podprto pa je tudi izdelovanje svojih poročil s pomočjo BIRT ogrodja. Nekatera poročila lahko poganjamo na živi podatkovni bazi, druga pa iz podatkovnega skladišča, ki se sinhronizira z živo podatkovno bazo enkrat dnevno.

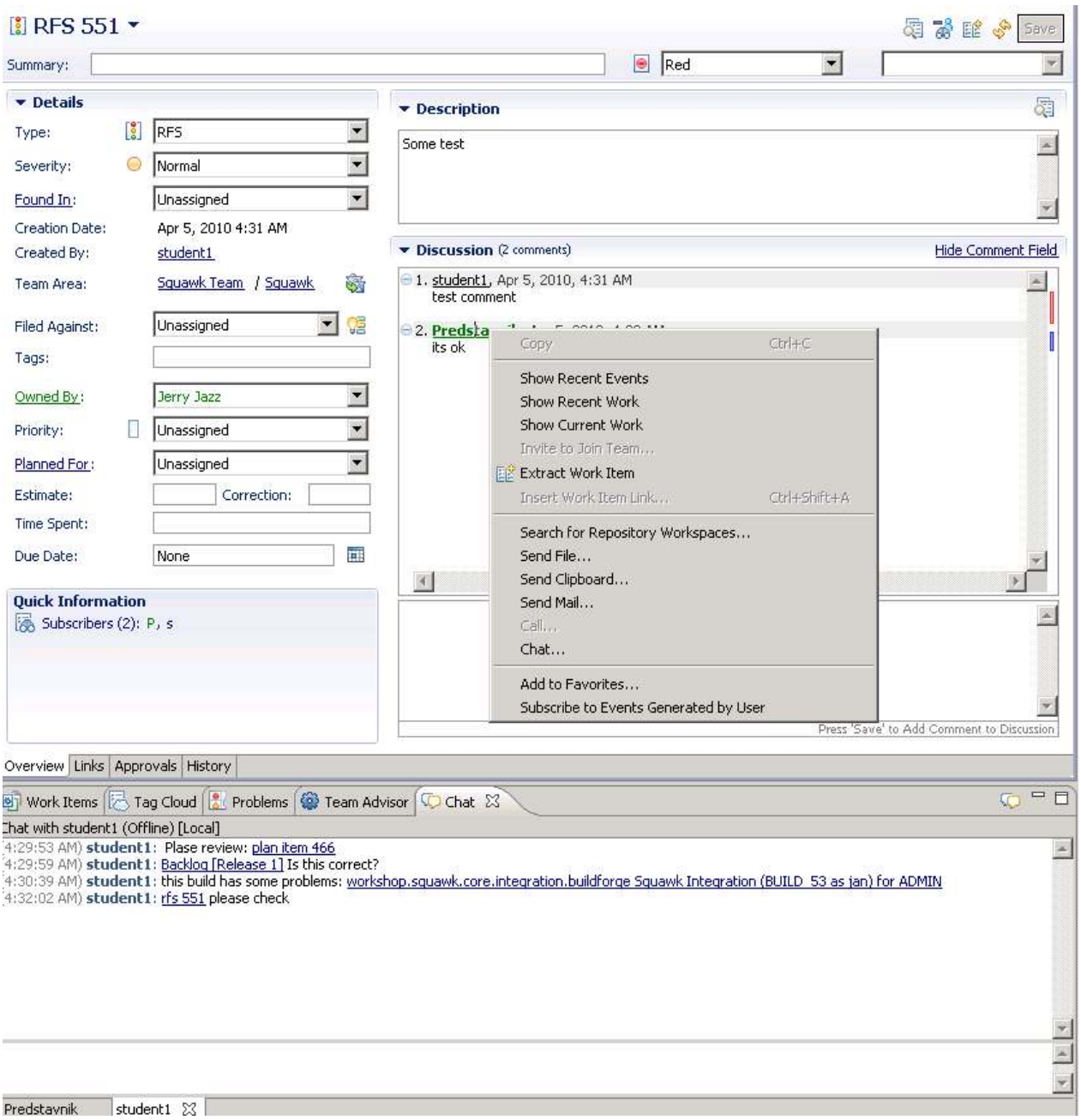
Primeri poročil: Zdravje graditve po iteracijah, prihajanje delovnih nalog z napakami glede na čas, zmanjševanje števila odprtih delovnih nalog glede na čas v iteraciji.

S pomočjo boljšega vpogleda v stanje je celotni razvojni skupini olajšano odločanje in prilagajanje na morebitne spremembe v poteku razvoja.

4.3.6. Sodelovanje

Zelo pomemben del RTC pa je vidik sodelovanja. Orodje je v veliki meri namenjeno ravno temu, v kontekstu razvoja. RTC je dobro integriran z storitvijo klepetanja, kar pomeni, da je odjemalec na voljo že kar v samem razvojnem orodju. Hkrati kjerkoli je omenjen nek uporabnik, zraven njegovega imena vidimo tudi, ali je na voljo ali ne. To pomeni, da je vsak član skupine oddaljne le en klik. Slika 15 prikazuje kako lahko 2 uporabnika preprosto ustvarita povezave v pogovornem oknu. Tudi v delovni nalogi, kjer uporabniki pustijo komentarje, so tisti, ki so trenutno na voljo obarvani zeleno. Z desnik klikom na njih pa lahko tudi hitro odpremo okno za pogovor.

Podpora hitri izmenjavi mnenj na posameznih delovnih nalogah in obveščanju o odgovorih je tudi ključnega pomena za hitro izmenjavo informacij. Obveščanje lahko poteka skozi vključene spletne vire (angl. feed) ali pa tudi z obveščanjem po elektronski pošti.



Slika 15: Primer uporabe storitve klepetanja v RTC

5. Vzorčni projekt v Rational Team Concert

5.1. Opis rešitve

Za vzorčni projekt sem izbral poenostavljeno obliko spletne aplikacije za vodenje študentske in izpitne evidence, tj. poenostavljen sistem e-študent, ki je v uporabi na Fakulteti za računalništvo in informatiko. Projekt je izveden samo do plana izdaje po metodologiji Scrum za primer sledeče skupine: Vodja projekta (scrum vloga: Skrbnik metodologije) en vodja razvoja (scrum vloga: Product Owner), 3 razvijalci in 1 naročnik. V nadaljevanju je definirana izbrana funkcionalnost, orisana v uporabniških zgodbah. Te uporabniške zgodbe so nato ocenjene in razvrščene na iteracije. Te so razbite na naloge za implementacijo. Za lažje prikazovanje nekaterih funkcionalnosti sem določene naloge tudi navidezno označil za končane.

5.2. Uporabniške zahteve

5.2.1. Uporabniške zahteve - študenti

5.2.1.1 Podatki o uporabniku

Uporabniki so shranjeni v sistemu in imajo definirane njihove osebne podatke (ime, priimek, naslov, vpisna številka, program in letnik v katerega so vpisani ter zabeležene ocene opravljenih izpitov). Vsi uporabniki so ustvarjeni ob vpisu s strani administracije. Takrat dobijo tudi podatke za prijavo v sistem.

5.2.1.2 Prijava v sistem

Študent se lahko v sistem prijavi z uporabniškim imenom in geslom. Nove uporabnike se ustvarja ob vnosu v sistem, njihovo uporabniško ime je vpisna številka, geslo pa je naključno določeno. Ob prvi prijavi je potrebna zamenjava gesla. Ko se uporabnik prijavi, je preusmerjen na prvo stran. Ob napačnem vnosu uporabniškega imena ali gesla se pojavi sporočilo, ki uporabnika na to opozori. Sporočilo ne vsebuje informacije kateri izmed vnešenih podatkov je napačen. Če uporabnik 5 krat vpiše napačno uporabniško ime in/ali geslo, se njegov račun zaklene, odklene ga lahko samo administrator.

5.2.1.3 Sprejemni testi za prijavo v sistem

- Preveri ali se uporabniški račun zaklene ob 5 kratnem napačnem vpisu kombinacije uproabniškega imena in gesla.
- Preveri ali se lahko prijaviš z veljavnim uporabniškim imenom in ali se ob uspešni prijavi prikaže prva stran z obvestili.
- Preveri ali se ob vpisu napačnega uporabniškega imena in gesla pojavi pravilno sporočilo o napaki.
- Preveri ali se ob prvi prijavi pojavi ekran za spremembo gesla.

5.2.1.4 Prva stran in obvestila

Ob prijavi je študent preusmerjen na stran, ki vsebuje obvestila. Prikazani so samo naslovi obvestil, ob kliku na naslov pa se prikaže njegova vsebina. Obvestilo je lahko študentom poslano tudi po el. pošti. Prikazanih je zadnjih 10 obvestil.

5.2.1.5 Sprejemni testi za prvo stran in obvestila

- Klikni na naslov obvestila. Preveri ali se prikaže njegova vsebina.
- Dodaj 11 obvestil nekemu študentu. Preveri ali se ob dodajanju več kot 10 obvestil starejša ne prikažejo.
- Pošlji študentom 2 obvestili, eno tudi po el. pošti. Preveri ali tista poslana tudi po elektronski pošti pridejo v el. predal in ali tista, ki niso poslana po elektronski pošti, ne pridejo.

5.2.1.6 Pregled in urejanje profila

Študent ima možnost izbire pregledovanja in urejanja profila. Ob izbiri se mu odpre stran, ki je v načinu samo za branje in vsebuje gumb za aktiviranje urejanja. Ob kliku nanj stran omogoča tudi spreminjane podatkov. Prikazani podatki so naslednji: ime in priimek študenta (urejanje onemogočeno), stalno prebivališče, kontaktni podatki (el. pošta, telefonska številka), naslov za pošiljanje pošte in elektronska pošta. Ob spremembi elektronskega naslova, se pošlje email s povezavo, s klikom na katero potrdijo delovanje njihovega novega email naslova.

5.2.1.7 Sprejemni testi za pregled in urejanje profila

- Spremeni naslov elektronske pošte. Preveri ali uporabnik dobi elektronsko sporočilo s potrditveno povezavo ob spremembi naslova elektronske pošte. Preveri ali povezava deluje in se naslov uspešno spremeni.
- Omogoči stran s podatki za urejanje. Preveri ali je mogoče urejati podatke v vseh poljih.
- Uredi podatke in pusti nekaj obveznih polj praznih ter sproži shranjevanje. Preveri ali aplikacija dovoli shranjevanje?
- Uredi podatke na veljaven način in sproži shranjevanje. Preveri ali se shranijo.
- Vnesi neveljavno poštno številko. Preveri ali aplikacija opozori na to?

5.2.1.8 Pregled izpitnih rokov in prijava

Študent ima možnost izbire strani izpiti, kjer se mu glede na neopravljenje izpite iz njegovega programa prikažejo tisti, na katere se lahko prijavi. V tabeli izpitov ob kliku na izpit pride na stran, kjer lahko izbere datum izpita in nato z gumbom prijava potrdi svojo prijavo na izpit. Možnost prijave je omogočena samo za izpite, za katere še ni potekel rok za prijavo.

5.2.1.9 Sprejemni testi za pregled izpitnih rokov in prijave

- Odpri stran za pregled prijav na izpite. Ob odpiranju strani prijava se prikažejo vsi neopravljeni izpiti študenta.
- Prijavi se na izpit. Preveri ali je izpit v prikazu označen kot prijavljen. Preveri ali je študent na seznamu prijavljenih.
- Prijavi se na izpit, za katerega je potekel rok za prijavo. Prever ali aplikacija to omogočeno?
- Razvrsti izpite po različnih parametrih, ki so prikazani (letnik, smer, profesor, ...). Preveri ali se podatki uredijo kot razvrščeno..

5.2.1.10 Pregled prijav

Študent ima možnost izbire strani pregled prijav, kjer se mu prikažejo njegove prijave na izpite. V primeru da rok za odjavo ni potekel, ima poleg prijave gumb odjava, s klikom na katerega se odjavi od tega izpita. Če je rok za odjavo potekel, je študent na to opozorjen s sporočilom pri temu izpitu

5.2.1.11 Sprejemni testi za pregled prijav

- Prijavi se na izpitni rok. Preveri ali je na seznamu izpitov ob omenjenem opcija odjave.
- Odjavi se od izpita. Preveri ali gumb odjave izgine.
- Prijavi se na izpit in počakaj, da poteče rok za odjavo. Preveri ali izgine opcija za odjavo in se prikaže primerno sporočilo.

5.2.1.12 Pregled ocen

Študent ima možnost izbire pregled ocen, kjer se mu prikaže stran z izbiro letnika, za katerega želi izpisane ocene. Ko izbere letnik, se mu za le tega prikaže preglednica z nazivom predmeta, dnevom opravljanja, številom poizkusov opravljanja, imenom profersorja in oceno. Na dnu preglednice ima tudi izpisano, koliko izpitov mu v izbranem letniku še ostaja in kakšna je njegova povprečna ocena za izpite, vaje in skupno. Za izbrani letnik lahko tudi izbere možnost kreiranja poročila (pdf), ki vsebuje vse omenjene podatke.

Poleg izbire po letniku ima še možnost generiranja poročila o vseh opravljenih izpitih čez vse letnike.

5.2.1.13 Sprejemni testi za pregled ocen

- Izberi letnik v katerem študent ni opravil nobenega izpita. Preveri ali na mestu, kjer bi morala biti njegova povprečna ocena piše 0.
- Izberi letnik, kjer je študent opravil nekaj izpitov. Preveri ali je izračun povprečne ocene in število opravljenih izpitov pravilen.
- Generiraj poročilo za naključno izbran letnik. Preveri ali aplikacija pokaže poročilo in ponudi prenos PDF datoteke.

5.2.1.14 Naročanje potrdil o vpisu in opravljenih izpitih

Študent v možnost naročanja potrdil. Tam izbere koliko potrdil želi naročiti in kakšnega tipa (izpiti, vpis). Ta so po pošti poslana študentu.

5.2.1.15 Sprejemni testi za naročanje potrdil o vpisu in opravljenih izpitih

- Naroči potrdilo o vpisu. Preveri ali administrator dobi naročilo za potrdila z vsemi potrebnimi podatki.

5.2.2. Uporabniške zahteve – adminstrativni del

Administrativni del podpira 2 uporabniške vloge: Prva je administratorji, ki lahko upravlja z vsemi vidiki sistema, druga pa pedagogi, ki so omejeni samo na ta način, da lahko urejajo ocene, izpitne roke samo predmetom, na katerih poučujejo in nič drugega.

5.2.2.1 Prijava

Uporabnik se prijavi v sistem z uporabniškim imenom in geslom. Uporabniško ime je šifra uporabnika, namesto vpisna kot pri študentih. Geslo se ob ustvarjanju uporabnika naključno določi in ga je potrebno ob prvi prijavi spremeniti. Ko se uporabnik prijavi, je preusmerjen na prvo stran.

5.2.2.2 Sprejemni testi za prijavo

- Prijavi se z napačnim uporabniškim imenom in geslom. Preveri ali aplikacija res ne sprejme prijave in ali uporabnika na to opozori pravilno.
- Prijavi se z veljavnim uporabniškim imenom in geslom. Preveri ali aplikacija prikaže prvo stran z obvestili.

5.2.2.3 Urejanje predmetov

Uporabnik ima možnost urejanja predmetov. Ob izbiri urejanja predmeta dobi seznam vseh predmetov. Ima možnost filtriranja po letniku in smeri, hkrati pa lahko dodaja nove predmete in jih tudi briše. Ob izbiri obstoječega ali ustvarjanju novega se mu odpre nova stran, kjer se nastavi ime predmeta, nosilec predmeta, drugi pedagogi, ki lahko urejajo ocene pri predmetu, smer, letnik, kreditne točke in semester. Smer in letnik se izbereta iz seznama letnikov in smeri. En predmet je lahko v več letnikih in smereh.

5.2.2.4 Sprejemni testi za urejanje predmetov

- Odpri stran za urejanje predmetov in izberi filter za letnik. Preveri ali so prikazani samo izpiti za izbran letnik. Ponovi za filter smeri.
- Odpri stran za urejanje predmetov in odpri predmet za urejanje. Dodaj pedagoga na predmet in ga shrani. Zapri in odpri aplikacijo. Preveri ali je bil pedagog shranjen.
- Ustvari nov predmet in ga poskušaj shraniti brez vpisanih obveznih podatkov. Preveri ali aplikacija opozori na to.

5.2.2.5 Urejanje letnikov in smeri

Uporabnik ima možnost urejanja letnikov in smeri. Ob izbiri urejanja smeri se mu prikaže seznam vseh smeri. Lahko ureja in dodaja nove smeri. Enako tudi za vsako smer določi koliko letnikov ima in tudi v katerem letniku se smer prične (nekatere smeri se pričnejo v višjih letnikih).

5.2.2.6 Sprejemni testi za urejanje letnikov in smeri

- Odpri seznam smeri in izbriši neko smer. Preveri ali se smer res izbriše.
- Odpri seznam letnikov in smeri in neki smeri uredi ime in jo shrani. Preveri ali se vrednost res spremeni.
- Odpri sezname letnikov in smeri. Dodaj novo smer in letnik in poizkusi shranit brez vnešenih podatkov. Preveri ali aplikacija opozori, da to ni mogoče.

5.2.2.7 Dodajanje uporabnikov in urejanje uporabnikov

Administrativni uporabniki imajo možnost dodajanja uporabnikov. Aplikacija omogoča vnos vseh potrebnih podatkov za uporabnika (ime, priimek, el. naslov, bivališče, kontaktni podatki, uporabniška skupina, povezani predmeti). Na podoben način je omogočeno tudi urejanje le, da je uporabnika, ki ga želimo urejati, prej potrebno poiskati.

5.2.2.8 Sprejemni testi za dodajanje uporabnikov

- Dodaj uporabnika z vsemi potrebnini podatki in ga shrani. Preveri ali ga potem lahko najdeš na seznamu uporabnikov.
- Poskušaj dodati uporabnika brez vseh potrebnih podatkov. Prveri ali aplikacija ob shranjevanju na to opozori.
- Dodaj uporabnika z vsemi potrebni podatki in ga shrani. Poišči ga in ga odpri za urejanje. Izbriši podatek, ki je obvezen in poizkusi uporabnika shranit. Preveri ali aplikacija opozori na to.
- Poskušaj poiskati uporabnika po parametru, kjer uporabnik sigurno obstaja. Preveri ali se pojavi na seznamu.
- Poskušaj z iskanje uporabnika po parametru, kjer uporabnik ne obstaja. Preveri ali se uporabnik ne pojavi na seznamu.

5.2.2.9 Ustvarjanje obvestil

Uporabnik ima možnost ustvarjanja novih obvestil. Člani pedagogov lahko pošiljajo obvestila samo tistim smerem in letnikom, kjer trenutno poteka njihov predmet.

Pri ustvarjanju obvestila lahko uporabnik izbere, komu je namenjeno (letnik, smer, vsi) in vnese poljubno sporočilo. Obstaja tudi opcija, da se sporočilo pošlje uporabnikom tudi po elektronski pošti.

5.2.2.10 Sprejemni testi za ustvarjanje obvestil

- Ustvari novo obvestilo in ga pošlji. Preveri ali ga imajo študenti katerim je naslovljeno med obvestili.
- Kot pedagog poizkusi poslati obvestilo letniku, v katerem nimaš predmeta. Preveri ali aplikacija ta letnik pokaže med možnimi naslovniki.
- Ustvari obvestilo, ki se mora poslati tudi po elektronski pošti. Preveri ali je bilo dostavljeno.

5.2.2.11 Urejanje izpitnih rokov

Uporabniki lahko ustvarjajo in urejajo izpitne roke za predmete. Pedagogi lahko ustvarjajo izpitne roke samo za predmete, kjer poučujejo ali asistirajo. Pri ustvarjanju novega izpitnega roka je potrebno določiti datum in prostor in predmet.

5.2.2.12 Sprejemni testi za urejanje izpitnih rokov

- Kot pedagog poizkusi ustvariti izpitni rok za predmet, kjer ne poučuješ. Preveri ali aplikacija na to opozori.

- Ustvari izpitni rok brez vseh potrebnih podatkov. Preveri ali aplikacija opozori na manjkajoče podatke.

5.2.2.13 Pregled prijav in vpis ocen

Uporabnik lahko pregleda seznam vseh razpisanih izpitnih rokov. Obstaja možnost filtriranja po letniku, smeri, nosilcih predmetov, semestru in izpiti, ki so se že odpisali. Ob kliku na izpit se prikaže seznam vseh prijavljenih. Na seznamu obstaja tudi možnost vpis ocen posameznemu študentu, ampak samo v primeru, da je uporabnik član skupine administratorji ali pa eden izmed pedagogov na predmetu. Obstaja možnost vračanja prijavnice študentu, vendar je za to potrebno navesti razlog.

5.2.2.14 Sprejemni testi za pregled prijav in vpis ocen

- Odpri seznam izpitov in jih filtriraj po letniku in smeri (izberi poljubno). Označi tudi da pokaže samo že odpisane izpite. Izberi poljubnega. Preveri ali omogoča vpis ocen študentom.
- Odpri seznam izpitov in izberi prikaz izpitov, ki še niso bili odpisani. Izberi poljubnega. Preveri ali aplikacija onemogoča vpis ocen.
- Odpri seznam izpitov in jih filtriraj po letniku in smeri (izberi poljubno). Označi tudi, da pokaže samo že odpisane izpite. Poljubnemu prijavljenemu študentu vpiši neveljavno oceno (npr. 11) in shrani. Preveri ali aplikacija dopušča ta vpis.

5.2.3. Nefunkcionalne zahteve

5.2.3.1 Revizijska sled

Vse spremembe ocen, izpitov, izpitnih rokov, uporabnikov, urejanje skupin morajo biti zabeležene v dnevniku sprememb z uporabnikom, ki je podatke spreminjal, časom spremembe in prejšnjo vrednostjo. Na ta način je omogočena ročna povrnitev originalnega stanja.

5.2.3.2 Skladnost z zakonom o varovanju osebnih podatkov (ZVOP)

Vsak dostop do osebnih podatkov študentov, pedagogov in administratorjev mora biti zabeležen. Bolj natančno, kdo in kdaj je dostopal do podatkov.

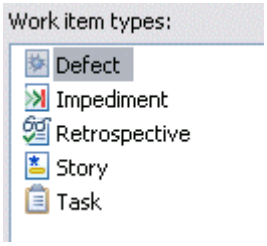
5.3. Planiranje

5.3.1. Podpora Scrum v Rational Team Concert

V Rational Team Concert obstaja predloga za Scrum metodologijo. To pomeni, da ima definirane sledče tipe delovnih nalog:

- Seznam zahtev (angl. Product Backlog) je zbirka nalog tipa zgodba (angl. story). Te kasneje ovrednotimo s točkami in jih porazdelimin porazdelimo med scrum iteracije.
- Seznam nalog (angl. Sprint Backlog) je zbirka uporabniških zgodb. Zgodba je razdeljena v naloge, ki jih je potrebno izpolniti za implementacijo zgodbe.
- Ovira (angl. impediment) je tip delovne naloge, ki predstavlja nekaj, kar ovira napredovanje projekta.
- Retrospektiva (angl. retrospective) predstavlja ugotovitve sestanke po koncu iteracije.

Tipi delovnih nalog iz RTC so prikazani na sliki 16



Slika 16: Tipi delovnih nalog v predlogi scrum

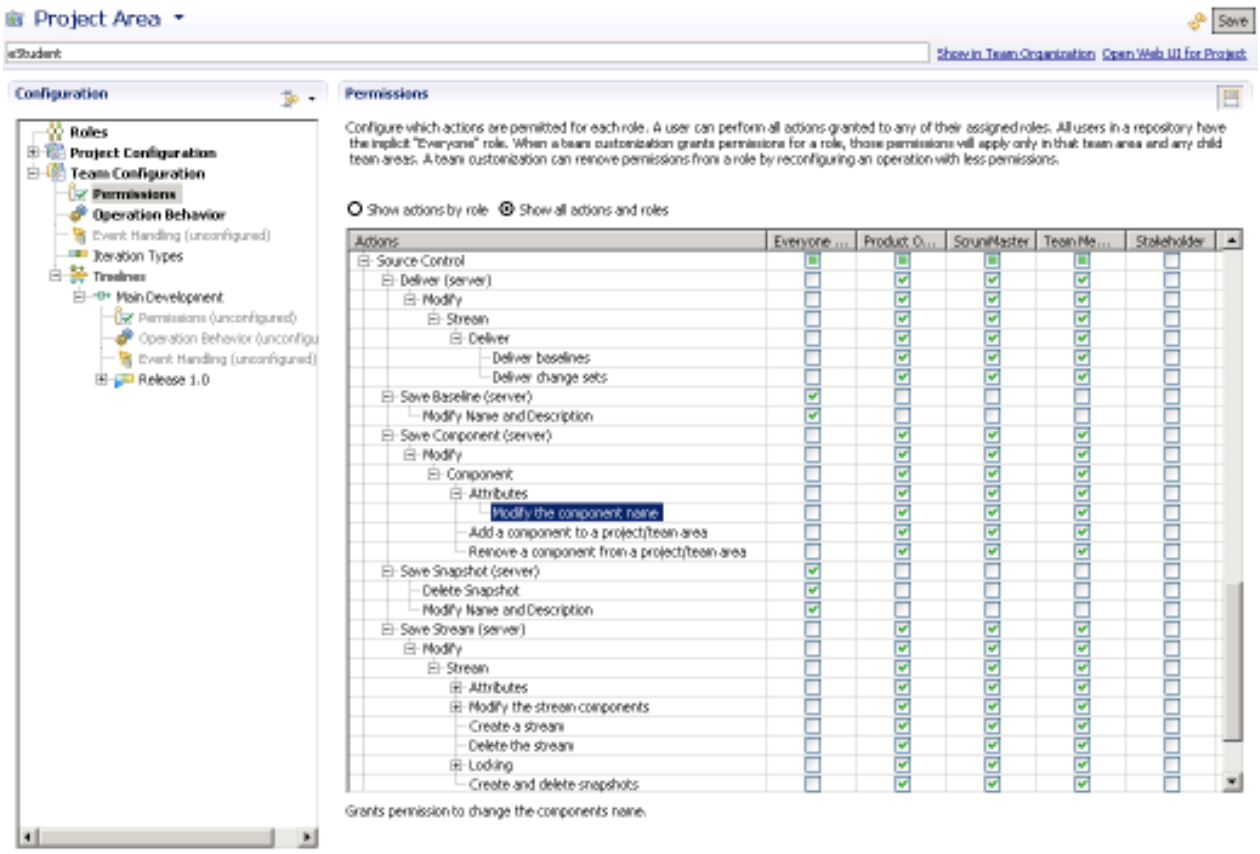
Definirane pa ima tudi vse Scrum uporabniške vloge: skrbnik metodologije, predstavnik naročnika, razvijalec in udeleženec. Prikazane so na sliki 14.

Predstavnik naročnika (angl. Product Owner) napravi začetni seznam zahtev, ki jih potrebno opraviti. Te so predstavljene v delovnih nalogah tipa zgodba (angl. Story). Na sestanku za planiranje iteracije se odloči katere zgodbe bodo implementerirane v trenutni iteraciji. Nato se zgodbe razdelijo v naloge. Ta se razdelijo med razvijalce in skupinsko se oceni čas potreben za dokončanje posamezne naloge. Nato se izvede iteracija. Razvijalci dokončajo naloge ali jih ob slučajnem pomanjkanju časa premaknejo v naslednjo iteracijo. Ob koncu itercije se na. pregledu iteracije pregledajo dosežki. Naslednji sestanek pa je retrospektiva iteracije, kjer se identificira kaj je potekalo dobro in kaj se lahko izboljša iz vidika procesa. Potem se ponovi celotna iteracija od sestanka za planiranje naprej.

Glede na uporabniško vlogo pa lahko določamo tudi pravice. Predloga Scrum sicer ne omejuje uporabnikov v veliki meri. Kot primer, skrbnik metodologije, predstavnik naročnika in razvijalci imajo več ali manj vse pravice. Razlika je za udeleženca, ki lahko ureja in dodaja delovne naloge in si ogleduje poročila. Ne more pa delati z izvorno kodo. Glede na to da so Scrum skupine samo organizirane enote, je logično, da je razdelitev pravic takšna. Ni pa takšen

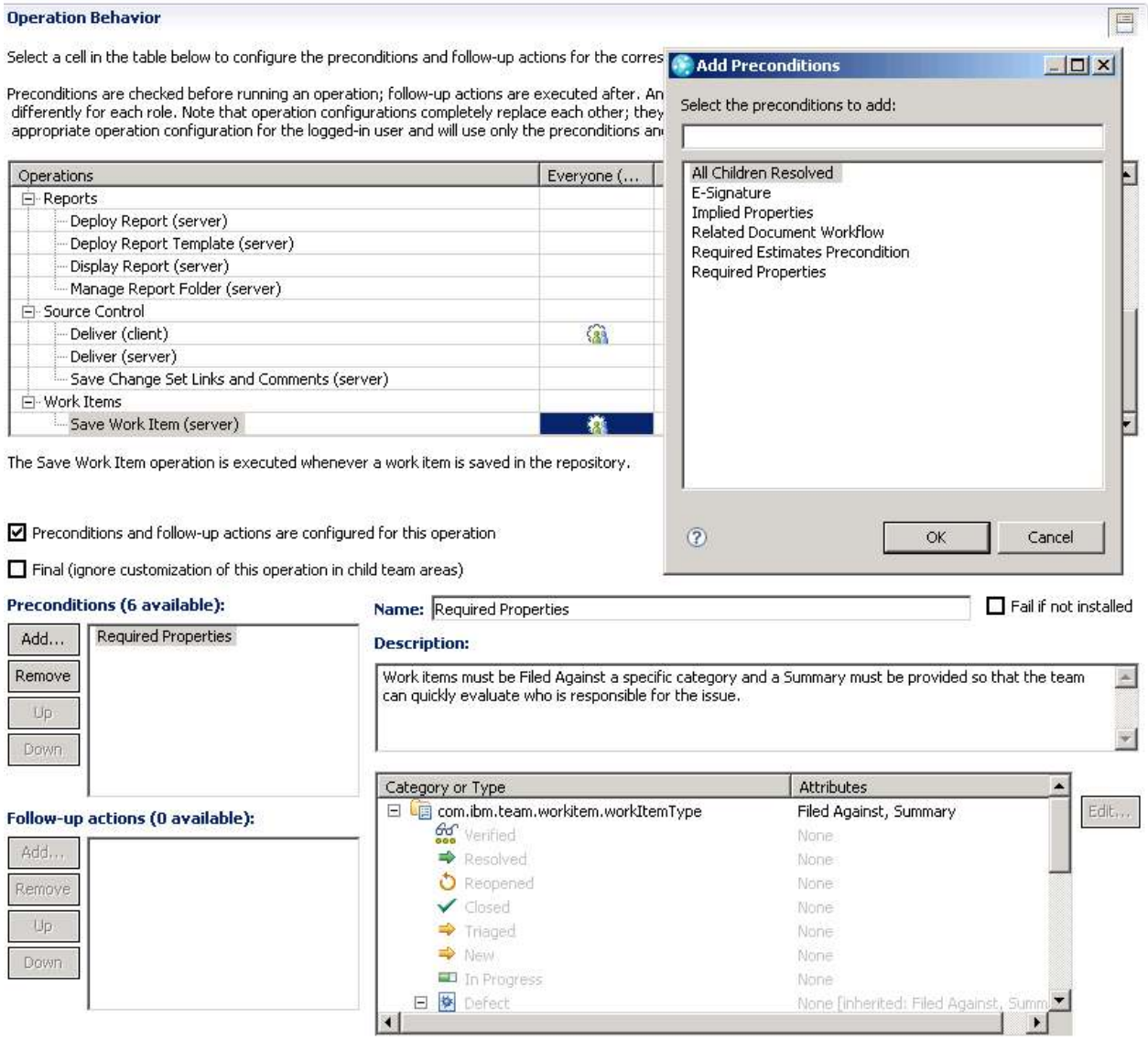
pristop vedno praktičen. Zaradi tega razloga ima funkcionalnost urejanja procesa možnosti prilagoditve pravic in obnašanja posameznih operacij glede na uporabniško vlogo.

Slika 17 prikazuje kako granularno lahko nastavimo pravice za vsako vlogo. Te nastavitve so na voljo za vse vidike s katerimi upravlja RTC od graditve, upravljana z izvorno kodo, poročanjem, planiranjem, delovnimi nalogami in procesom razvoja. Pravice se lahko nastavljajo v splošnem ali pa tudi samo za posamezno skupino. To omogoča preprosto delovanje v primeru majhnih skupin, kjer imajo vsi pregled nad vsem, ali pa uporabo v večjih skupinah, kjer morajo biti varnostne nastavitve veliko bolj rigorozne.



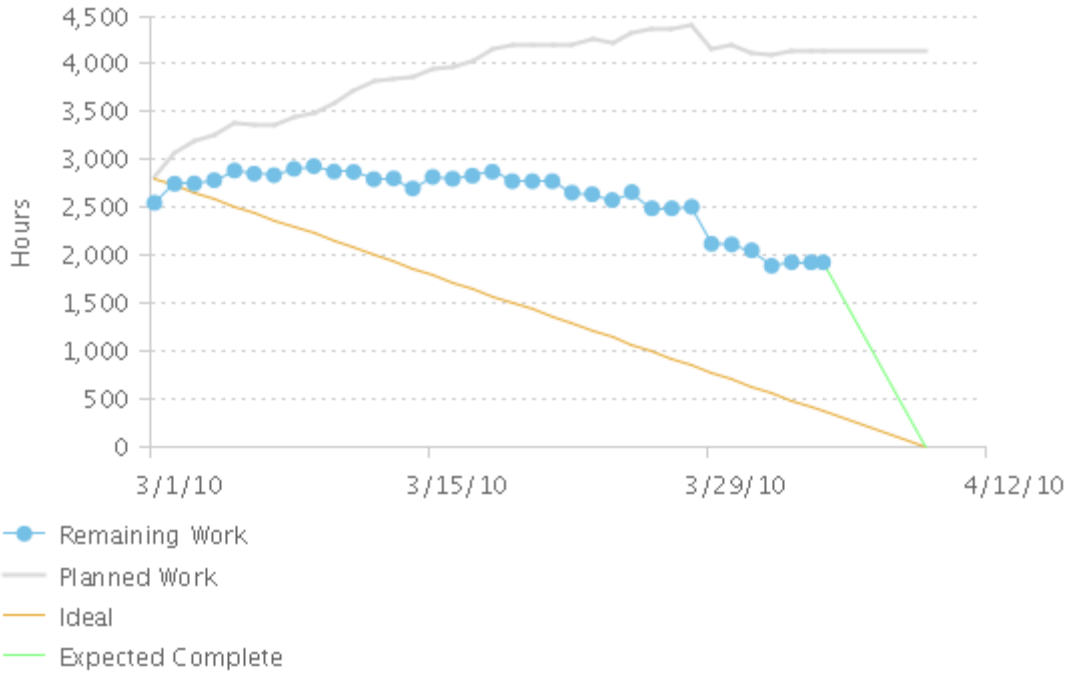
Slika 17: Možnosti prilagajanja uporabniških pravic glede na vlogo

Poleg urejanja pravic do proženja posameznih operacij pa lahko urejamo tudi njihovo obnašanje. Za vsako operacijo lahko določimo kakšni so predpogoji za njeno proženje, določimo pa lahko tudi različne akcije, ki se zgodijo po izvedbi operacije. Slika 18 prikazuje, kakšni predefinirani predpogoji so na voljo za operacijo shranjevanja delovne naloge. V tem primeru imamo nastavljen predpogoj za preverjanje prisotnosti določenih polj v delovni nalogi. V odprtem oknu Add preconditions pa imamo prikazane še druge možnosti, ki so na voljo. Spodaj desno za vsak tip delovne naloge določimo, katera polja so obvezna. V prikazanem primeru je obvezen vnos skupine, ki ji je določena delovna naloga in povzetek naloge, da skupina lahko hitro določi, komu je naloga namenjena. Hkrati pa lahko določimo tudi katera polja so obvezna glede na stanje v življenjskem ciklu delovne naloge. To pomeni, da delovna naloga tipa naloga, ne more biti zaključena, če nima vpisanega polja za porabljen čas.



Slika 18: Prilagajanje posameznih operacij v RTC

Pomemben del Scrum je tudi ocena opravljenega med samo iteracijo. Za ta namen RTC omogoča generiranje poročil za pregled ravno tega. Dober primer je t.i. »Burn down chart«, ki pove, koliko delovnih nalog je še odprtih. Slika 19 prikazuje omenjen graf iz razvoja najnovejše verzije orodja, kjer za razvoj RTC uporabljajo orodje samo. Ta primer sem izbral iz preprostega razloga, ker je tam zbranih ogromno podatkov in ker lepo prikazuje funkcionalnost. Iz tega lahko hitro sklepamo, da se njihovo planirano delo povečuje in glede na to, da opravljeno ne pada linearno s potekom iteracije, bo potrebno nekaj prilagajanja, če želijo doseči cilje za ta mejnik.

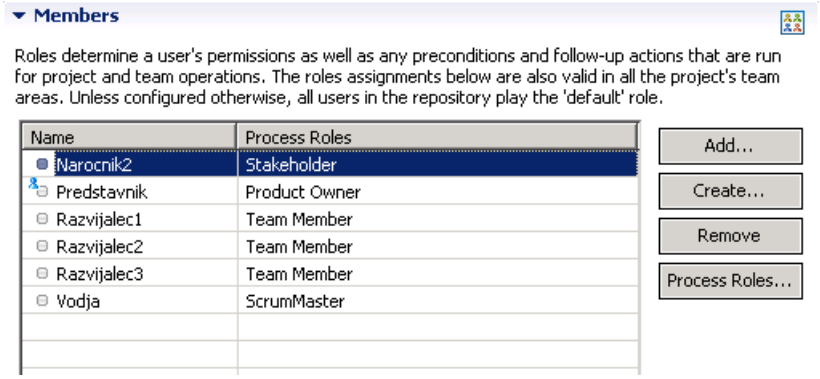


Slika 19: »Burn down chart« iz razvoja verzije 3.0 RTC

5.3.2. Plan vzorčnega projekta eŠtudent

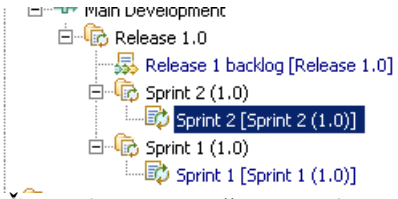
Najprej sem za vzorčni projekt definiral projektno področje in mu kot predlogo določil Scrum. To pomeni, da so avtomatsko ustvarjeni tipi delovnih nalog, predefinirah poizvedb za delovne naloge, uporabniške vloge, ...

Najprej sem ustvaril skupino, sestavljeno iz razvijalcev, skrbnika metodologije in predstavnika naročnika. Slika 20 predstavlja člane skupine in njihove vlog. Narocnik2 predstavlja mogočega končnega uporabnika, ki bi prek spletnega vmesnika lahko imel vpogled v napredek projekta, sporočal odkrite napake v testiranju, komenitral ali popravljal delovne naloge, itd.



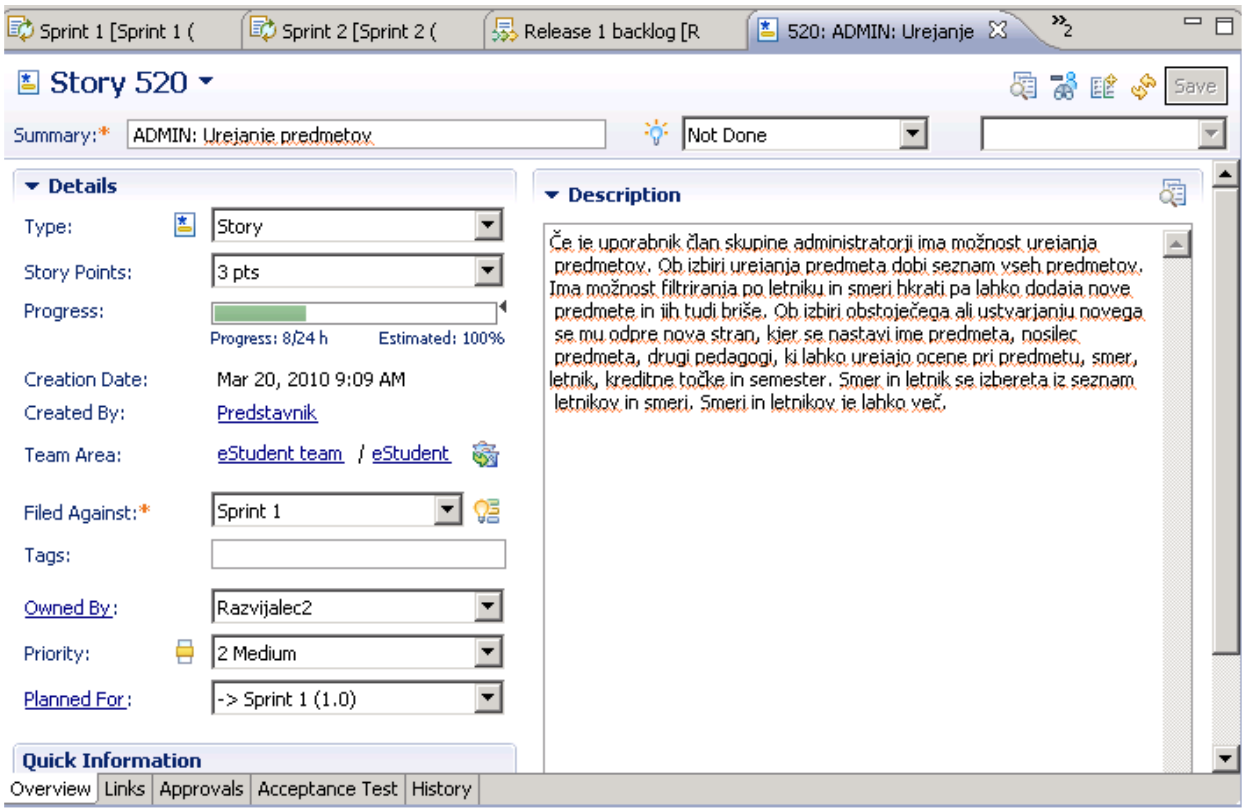
Slika 20: Definicija razvojne skupine

Nato sem za projekt definiral časovni potek projekta (slika 21). Odločil sem za 2 iteraciji. Definiral sem tudi seznam zahtev tukaj poimenovan Release 1 backlog.



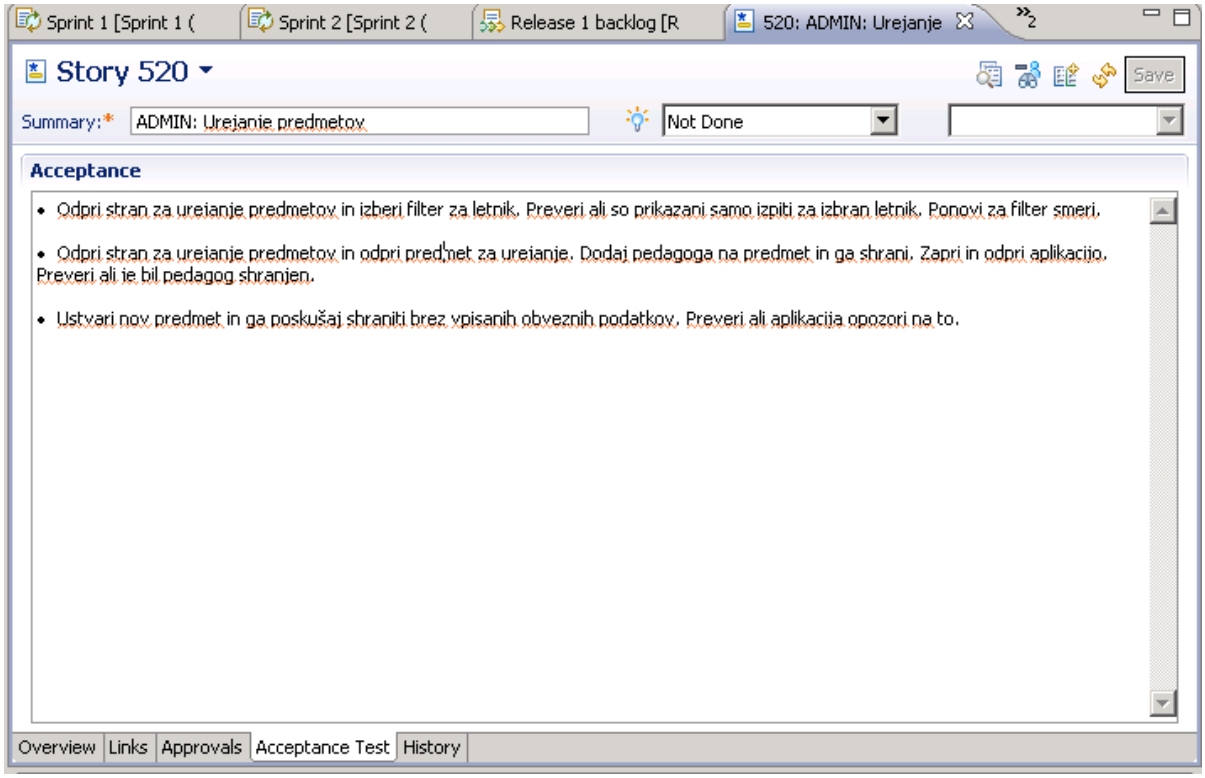
Slika 21: Časovni potek vzorčnega projekta eŠtudent

Seznam zahtev je sestavljen iz delovnih nalog tipa zgodba. Slika 22 in slika 23 prikazujeta kako je v RTC zabeležena zgodba. Na sliki 22 je prikazana pregledna stran zgodbe. Vsebuje opis, oceno zahtevnosti (točke), možnost sodelovanja in druge parametre, lastnika, v katero iteracijo sodi, kateri skupini je namenjena, ...



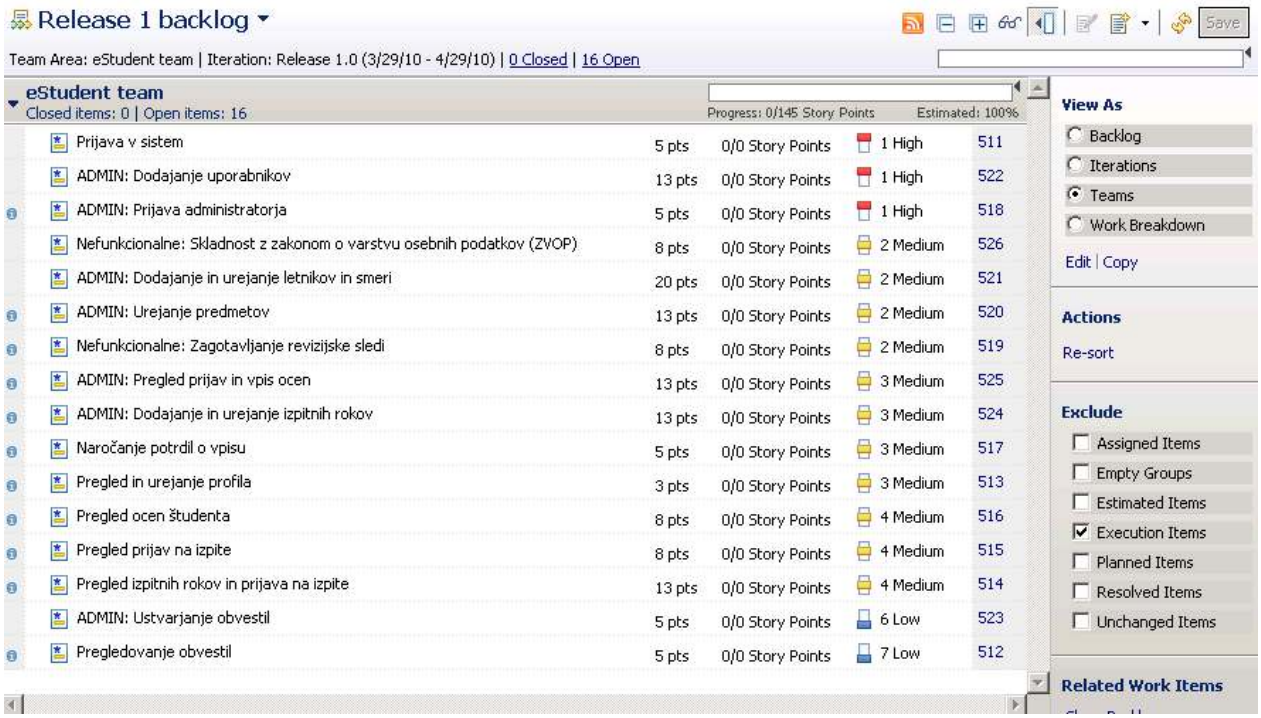
Slika 22: Uporabniška zgodba v RTC

Slika 23 pa prikazuje del zgodbe, ki vsebuje sprejemne teste, ki sem jih zapisal v seznamu uporabniških zgodb v poglavju 4.2. Sprejemni testi so seveda primarno namenjeni za testiranje implementirane rešitve. Toda zelo pa so uporabni tudi zato, da razvijalec bolje razume delovanje aplikacije, ki jo opisuje uporabniška zgodba.



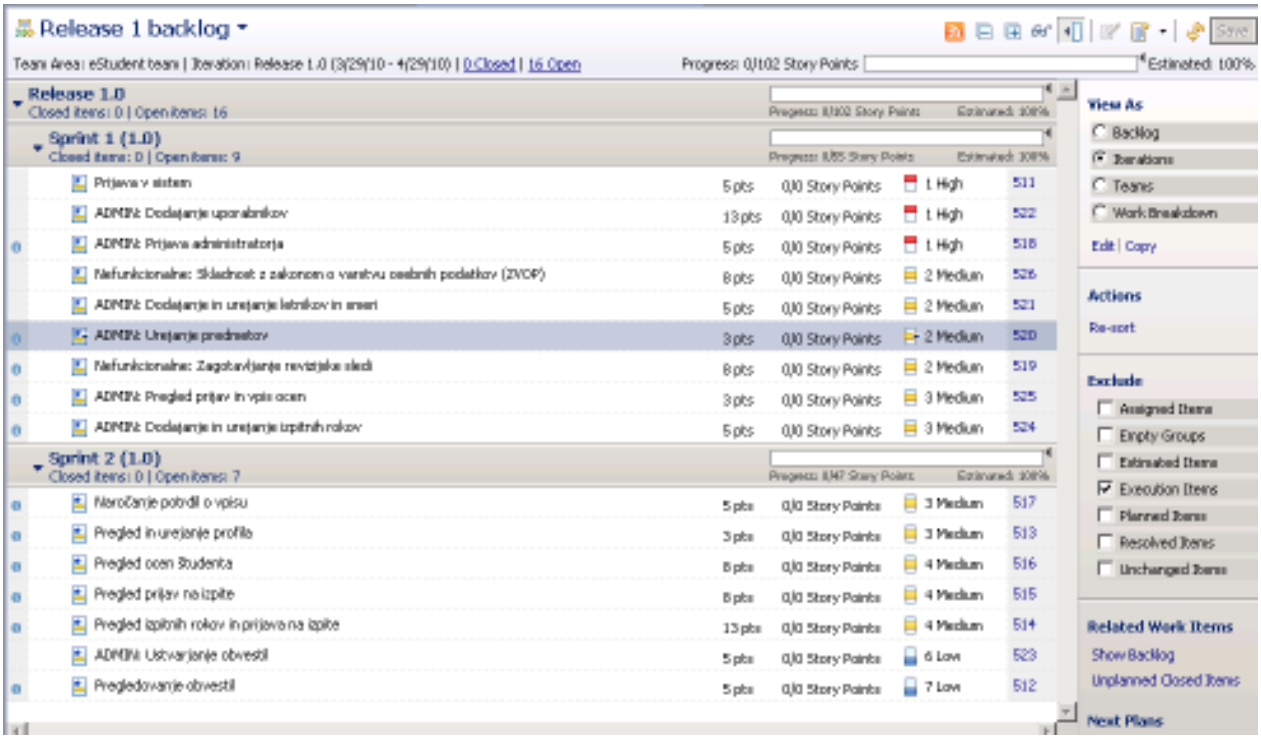
Slika 23: Sprejemni testi v uporabniški zgodbi

V seznam zahtev sem vnesel vse zgodbe iz podpoglavja 4.2 (slika 24). Glede na ocenjeno zahtevnost sem jih ovrednotil s točkami in jim določil prioriteto glede na to, kaj želimo imeti implementirano najprej.



Slika 24: Zbrane zgodbe v seznamu zahtev

Zgodbe sem nato razdelal razdelil v iteracije, glede na prioriteto (slika 25). Ker je projekt vzorec in skupina izmišljena, je težko oceniti hitrost (angl. velocity), kar je pomemben podatek za razdeljevanje zgodb glede na število točk. Iz tega razloga sem kasneje, ko sem zgodbe razdelal v naloge in jih časovno ocenil, nekaj zgodb premaknil. Glede na to, da vemo, kako dolga je iteracija (izbral sem si 20 dni) in koliko ur na dan so na voljo razvijalci, orodje prikaže koliko ur je na voljo v iteraciji. Na podlagi tega sem ugotovil, da sem za prvo iteracijo planiral preveč dela. Iz tega razloga sem potem nekaj zgodb premaknil v drugo iteracijo. Hkrati bi opozoril tudi na indikacijo napredka zgoraj desno, ki nam pove, koliko točk je bilo že implementiranih. V trenutku, ko je bil narejen ta posnetek zaslona, napredka še ni bilo. Vendar, ko neko zgodbo zaključimo, se napredek avtomatsko premakne za število točk, ki so določene zgodbi.

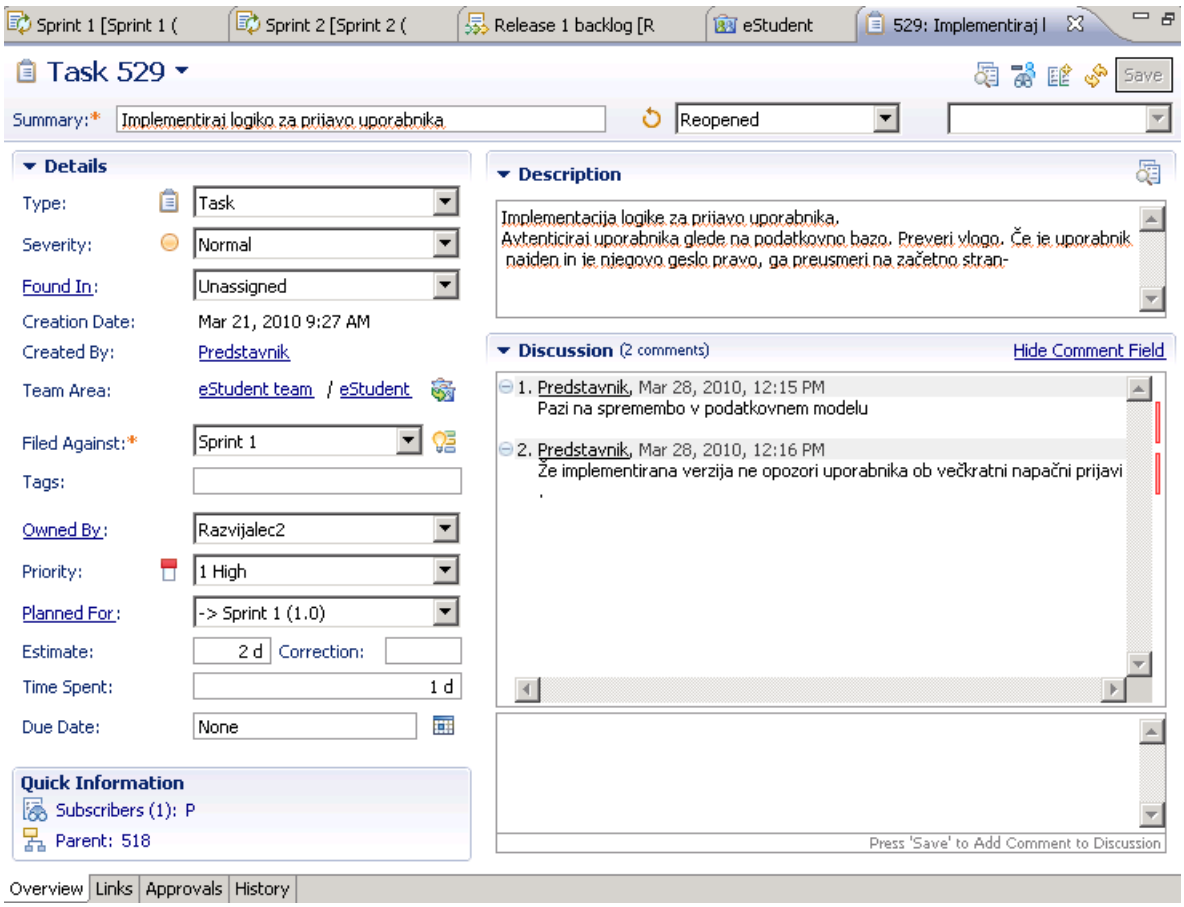


Slika 25: Razdelitev zgodb čez 2 iteraciji

V naslednjem koraku sem zgodbe razdelil med razvijalce in vsako razčlenil na več delovnih nalog tipa naloga (angl. Task). Vsaki nalogi sem določil tudi časovno oceno (slika 26). Vsak član skupine ima v svojem profilu določeno, koliko dni tedensko dela in kakšen procent svojega časa namenja določeni skupini. Na podlagi tega in števila delovnih dni v iteraciji orodje avtomatsko ocenjuje, koliko dela lahko ta razvijalec opravi. Ker je iteracija določena datumsko, za vsakega razvijalca dela oceno, kako napreduje in ali je premalo oz. preveč obremenjen. Na sliki 26 na primer, razvijalec 1 ni polno obremenjen (obremenitev 40/64, 24 ur +). Zgoraj desno pa je tudi celotna ocena iteracije glede na seštevek vseh ocenjenih ur, opravljenih ur in število že preteklih dni iteracije. V tem primeru sem nekaj nalogam že dodal nekaj opravljenega dela, zato napredek kaže, da smo 19.25 ure v zaostanku. Na ta način lahko v tem primeru skupina prilagaja delo, da bo ob koncu iteracije opravljena optimalna količina. Uporabniški vmesnik na sliki 18 deluje tudi tako, da lahko naloge z lahkoto samo primemo in spustimo na drugega razvijalca in delovna naloga je določena njemu. Hkrati lahko na tem ekranu urejamo tudi prioriteto in oceno potrebnega časa. To seveda v primeru, da imamo temu primerne pravice.

Slika 26: Delo razdeljeno med razvijalce v planu iteracije

Delovna naloga, ki je prisotna v planu iteracije ob odprtju pa je prikazana na sliki 27. Ocena časa se vrši na podlagi polj ocena (angl. Estimate) in porabljen čas (angl. Time spent). Omenil bi še okvir v spodnjem levem kotu, ki nam pove katera delovna naloga je starševska trenutni in kdo je naročen na obveščanje o spremembah, ki se zgodijo na tej nalogi. Možne so tudi druge relacije, npr. povezana delovna naloga, povezana zahteva, povezan testni primer, duplikat, odvisen od, itd. Starševska naloga v tem primeru je uporabniška zgodba, ki govori o prijavi uporabnika.



Slika 27: Delovna naloga tipa naloga

Tako izgleda faza planiranja v Rational Team Concert z uporabo Scrum metodologije. Če bi hotel projekt še izvesti, bi potreboval skupino ljudi, ki bi aplikacijo dejansko implementirali s sledenjem tega plana. Tukaj tudi ni bilo planiranega nobenega načrtovanja aplikacije, kar v realnem primeru sigurno ni dobra praksa.

6. Sklepne ugotovitve

V nalogi sem opisal Rational Team Concert in na podlagi vzorčnega projekta opisal njegovo uporabo z metodologijo Scrum. Ugotovil sem, da je RTC orodje z dobro podporo agilnim metodam. Že privzeto omogoča start produktivnega projekta v le nekaj dneh, saj je po večini že vse nastavljeno tako, da dobro podpira metodologijo. Težava, ki bi se lahko pojavila, je skaliranje te Scrum predloge na večje ekipe. V tem primeru so varnostne nastavitve privzeto preveč ohlapne za ohranjanje discipline pri večji ekipi. Sicer je ta težava z lahkoto rešljiva z uporabo precej naprednih varnostnih nastavitev, ki pa hkrati niso preveč težavne za uporabo.

Sam vmesnik za planiranje in vodenje razvoja je pregleden in učinkovit. Najbolj me je presentila preprostost uporabe agilnega planerja, kjer lahko kar direktno vpisujemo naloge v plan s pomočjo nekaj bližnjic in enega samega uporabniškega ekrana.

Ocenjevanje napredka je intuitivno in z lahkoto dostopno. Vsi imajo v vsakem trenutku dober pregled nad svojim napredkom, napredkom sodelavcev v skupini in stanjem celotnega projekta. Po agilnih načelih je to zelo pomemben vidik delovanja skupine in na tem področju se RTC izkaže zelo dobro.

Če se navežem na temo orodij za podporo sodelovanju iz poglavja 1.4 je RTC sigurno eno izmed orodij, ki je dobro pozicionirano na tem prostoru. Podpora enovitosti ekipe, četudi se ta nahaja na različnih lokacijah je dobra. Veliko k temu pripomore tudi enoten uporabniški vmesnik, ki pokriva več vidikov razvoja od upravljanja delovnih nalog, graditev, planiranja, komunikacijo, poročanje in tudi upravljanje z izvorno kodo. Veliko dodano vrednost, tudi v primerjavi s konkurenco, pa ima RTC na področju upravljanja s celotnim življenjskim ciklom aplikacije. To, poleg razvoje, vključuje še zajem, analizo in upravljanje z zahtevami ter seveda podporo zagotavljanju kvalitete. Platforma Jazz to funkcijo že danes odlično opravlja. Zamera, ki jo tukaj najdemo je, da trenutno to povezovanje še vedno poteka preko integracije. Ta je sicer dobro definirana in standardizirana, toda platforma v svoji viziji obljublja enoten podatkovni repozitorij za vse te izdelke. To je sicer obljubljeno v prihodnosti, vendar danes tega še ni.

Glede na to, da je Jazz odprta platforma, je trenutno vmesnik API, ki je na voljo še vedno precej slabo dokumentiran. Pogrešam bolj formalno gradivo na to temo, pri IBM navadno kakšen RedBook, ki bi bolj v detajle obravnaval to tematiko.

Vendar prihodnost obeta dosti. Glede na to, da je orodje na trgu samo 2 leti, ponuja zelo obsežen nabor funkcionalnosti, ki hkrati tudi zelo dobro deluje in je stabilen. Ob izvajanju planiranja za vzorčni projekt in konfiguracija okolja sploh nisem naletel na kakšne težave z napakami. Sklepam, da k temu dosti pripomore tudi uporaba RTC pri samem razvoju RTC s strani produkte skupine.

Kar pri orodju žal nisem mogel preizkusiti in dokumentirati, pa je sama uporaba v dejanski razvojni skupini. Zanimivo bi bilo videti vzorce uporabe pri razvjalcih in njihove odzive na način dela z opisano metodologijo Scrum v RTC. To žal ni bilo mogoče, ker nisem imel dostopa do nobene razvojne skupine, ki bi orodje uporabljala. Še bolj interesantno pa bi bilo spremljanje projekta pred in kasneje z uporabo orodja in primerjava načina dela in produktivnosti razvijalcev in skupine v obeh primerih.

7. Viri

[1] S. Mitchell, C. Seaman: A Comparison of Software Cost, Duration, and Quality for Waterfall vs. Iterative and Incremental Development: A Systematic Review; v Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, 511-515

[2] D. Cohen et al: An Introduction to Agile methods, ADVANCES IN COMPUTERS, VOL. 62, (2004) Elsevier Inc

[3] (2010) Scrum development. Dostopno na: http://en.wikipedia.org/wiki/Scrum_(development)

[4] (2010) Agile software development. Dostopno na: http://en.wikipedia.org/wiki/Agile_software_development

[5] (2010) Jazz Integration Architecture Overview. Dostopno na: https://jazz.net/projects/DevelopmentItem.jsp?href=content/project/plans/jia-overview/index.html

[6] (2010) A. Cockburn: Using Both Incremental and Iterative Development. Dostopno na: http://www.stsc.hill.af.mil/crosstalk/2008/05/0805Cockburn.html

[7] (2010) Sprial Model. Dostopno na: http://en.wikipedia.org/wiki/Spiral_model

[8] (2010) OpenUP documentation. Dostopno na, http://epf.eclipse.org/wikis/openup/index.htm

[9] M. Göthe, C. Pampino, P. Monson,K Nizami, K Patel, B. M. Smith, N Yuce: Collaborative Application Lifecycle Management with IBM Rational Products, 2008, IBM Redbooks

[10] (2010) Agile Manifesto. Dostopno na: http://agilemanifesto.org/history.html

[11] B. Boehm: Get ready for agile methods, with care, IEEE Computer (Jan. 2002) 64–69

[12] (2010) Rational Team Concert PoT presentation, Dostopno na:, http://www-05.ibm.com/at/events/software_experience/pdf/PoT.Rational.07.2.038.03-Presentations.pdf , stran 12

[13] S. Vinoski: The Language Divide, Internet Computing, IEEE, March-April 2006, 82 – 84

[14] (2010), JRE Installation notes, Just-In-Time Compilers, Dostopno na: http://java.sun.com/developer/onlineTraining/Programming/JDCBook/perf2.html#jit

[15] (2010) Hotspot compiler, Dostopno na: http://java.sun.com/javase/technologies/hotspot/

[16] (2010) Programing languages popularity, Dostopno na: http://langpop.com/

[17] (2010) COBOL, Dostopno na: http://en.wikipedia.org/wiki/COBOL

[18] F. Solina: Projektno vodenje razvoja programske opreme, Založba FE in FRI, (1997)

[19] (2010) Getting Started with Jazz Source Control, Dostopno na: http://jazz.net/library/article/41

[20] P. Wongthongtham, E. Chang, T.S. Dillon, I. Sommerville: Ontology-based multi-site software development methodology and tools, Journal of Systems Architecture, vol. 52, (2006) 640–653

[21] K. Schwaber: Agile Project Management with Scrum, (2004), Microsoft Press.

[22] K. Beck, C. Andres: Extreme Programming Explained: Embrace Change, 2nd Edition, (2005) Addison-Wesley

[23] (2010), Jazz architecture vision, Dostopno na: http://jazz.net/about/about-jazz-vision.jsp

[24] (2010) About Jazz Architecture, Dostopno na: http://jazz.net/about/about-jazz-architecture.jsp

[25] (2010), RTC devlopment Burn Down chart, Dostopno na: http://jazz.net/jazz/web/projects/Rational%20Team%20Concert#action=com.ibm.team.reports.viewQuery&queryUUID=__rHa8GsuEd6cw4CjcBsafw&name=Burndown%20%28Live%29