UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andraž Cej

AGILNI RAZVOJ PROGRAMSKE OPREME PO METODOLOGIJI SCRUM

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: 01665/2010
Datum: 05.04.2010

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

Kandidat: **ANDRAŽ CEJ**

Naslov: **AGILNI RAZVOJ PROGRAMSKE OPREME PO METODOLOGIJI SCRUM**

AGILE SOFTWARE DEVELOPMENT WITH SCRUM

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite značilnosti agilnih metodologij za razvoj programske opreme in predstavite postopek razvoja po metodologiji Scrum. Posebno pozornost namenite specifikaciji zahtev v obliki uporabniških zgodb in njihovi uporabi pri izdelavi plana izdaje in planov posameznih iteracij. Izdelajte pregled orodij za podporo vodenju projektov, specificirajte kriterije za njihovo primerjavo in jih ovrednotite s pomočjo programa DEXi.

Mentor:
prof. dr. Viljan Mahnič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

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

Zahvala

Rad bi se zahvalil mentorju izr. prof. dr. Viljanu Mahniču za vso podporo in dobro voljo ter vsem ostalim, ki so kakorkoli prispevali k nastanku naloge.

KAZALO

Povzetek

Predstavljen je pomen in značilnosti agilnih metodologij za razvoj programske opreme, s poudarkom na trenutno najbolj razširjeni metodologiji – Scrum. Poleg osnovnih značilnosti ter predstavitve Scrum razvojnega procesa, se naloga dotakne tudi specifikacije zahtev v obliki uporabniških zgodb in njihove uporabe v omenjeni metodologiji. Predstavljene so tudi najbolj razširjene tehnike ocenjevanja uporabniških zgodb in podroben pregled tehnike z imenom poker planiranja. Na koncu je izdelan pregled orodij za podporo vodenju projektov in njihova primerjava s pomočjo odločitvenega modela in programa DEXi.

Ključne besede: agilne metodologije, Scrum, poker planiranja

Abstract

The thesis describes the importance and characteristics of agile methodologies for software development, focusing on the currently most widely used methodology - Scrum. In addition to basic features and the presentation of the Scrum development process, the paper describes the specification of user requirements through user stories and their use in the above-mentioned methodology. It also presents the most widely spread user stories estimation techniques and gives a detailed examination of the technique called Planning poker. The paper concludes with a review of tools that support project management tasks and their comparison with the help of a decision model and the program DEXi.

Key words: agile metgodologies, Scrum, Planning poker

1. Uvod

Razvoj programske opreme težko definiramo kot razvoj, ki je predvidljiv oziroma enostaven. Za razliko od klasične industrije tukaj ne gre za masovno produkcijo ali za predvidljive modele, temveč gre skoraj pri vsakem projektu za nov izdelek. To pomeni, da je redko mogoče vnaprej natančno določiti specifikacije izdelka ter natančno planirati potek projekta. Zelo težko je namreč postaviti pravilne ocene za projektne okvire, kot so čas za izvedbo, potrebni viri, cena, in definirati aktivnosti, povezane z razvojem izdelka.

Poleg vsega tega se razmere v inženirstvu programske opreme in z njim povezanim trgom spreminjajo. Močno se krajšajo časi izdelave programske opreme, podobno se krajša čas uporabe izdelkov in sistemov. Podjetja agresivno razvijajo nove izdelke in storitve. Razvojne ekipe v podjetjih, ki se ukvarjajo s produkcijo programske opreme, so prisiljene k hitrim odločitvam, kljub temu, da ne razpolagajo s popolnimi informacijami. Ko k temu prištejemo še pogosto spreminjanje uporabnikovih zahtev, se lahko hitro dokopljemo do razloga, ki je podkrepil razvoj agilnih metodologij.

Od sredine devetdesetih let pa do danes se je razvilo kar nekaj agilnih metodologij. Nekatere so zelo podrobne in predstavljajo celotno ogrodje razvoja, druge so bolj principi oziroma splošna navodila, ki se jih da integrirati tudi v druge, bolj klasične metodologije. Metodologija, ki je med agilnimi najbolj prepoznavna, je Ekstremno programiranje (XP, ang. Extreme Programming), v zadnjih letih pa vodilno vlogo glede na uporabo prevzema Scrum. Scrum ima precej skupnih lastnosti z Ekstremnim programiranjem, zaradi svoje popularnosti pa je tudi glavna tema tega diplomskega dela.

Scrum ni samo trenutno najbolj razširjena agilna metodologija, temveč tudi tista, ki ji uporabniki najbolj tesno sledijo. To pomeni, da se Scrum pravila tudi v praksi dobro obnesejo. Kljub temu pa vse agilne metodologije, ne samo Scrum, ne kontrolirajo vseh aspektov razvojnega procesa, temveč puščajo v marsičem proste roke. To je poleg pomanjkanja izkušenj lahko razlog za neuspelo integracijo agilnih metodologij v razvojni proces podjetja. V ta namen sem v nalogi pregled razvojnega procesa Scrum razširil na vlogo uporabniških zgodb v povezavi s Scrum-om ter tehnike planiranja in ocenjevanja, poleg tega pa sem dodal pregled nekaterih orodij, ki so na trgu na voljo za spremljanje Scrum procesa. Vse našteto občutno pripomore k uspehu implementacije agilnih metodologij v podjetjih.

Nalogo tako začnem s pregledom področja agilnih metodologij, najprej s splošnega vidika, potem pa z nekoliko podrobnejšim pregledom nekaterih najbolj razširjenih metodologij. Tretje poglavje je namenjeno podrobnemu opisu Scrum metodologije. Poleg Scrum vlog in izdelkov namenja naloga posebno pozornost tudi poteku sestankov, ki so ključni del razvojnega procesa Scrum.

Četrto poglavje opisuje uporabniške zgodbe, ki sicer niso nastale v sklopu Scrum-a, ampak se v praksi vse bolj uporabljajo tudi povezavi s to metodologijo. Poglavje se zaključi s

tehnikami planiranja in ocenjevanja s posebnim poudarkom na tehniki planiranja, imenovani poker planiranja. Zadnji dve poglavji sta namenjeni pregledu orodij za podporo Scrum-u in njihovi primerjavi s pomočjo odločitvenega modela.

Tudi sam sem se že srečal z agilnimi metodologijami v praksi. Bil sem priča prehodu s klasičnih metod k agilnim metodam. Kljub temu, da vsaka tovrstna sprememba prinaša določene nevšečnosti, pa sem tudi sam mnenja da so agilne metode prava smer. To je tudi spodbudilo nastanek te naloge.

2. Agilne metodologije za razvoj programske opreme

2.1. Uvod

Bistvo agilnega razvoja programske opreme je postavljanje prioritete na sam razvoj izdelka, na delujočo programsko kodo, na prenos odgovornosti na razvojno skupino in na zavedanje, da se uporabniške zahteve s časom spreminjajo [1]. Vse aktivnosti se morajo odvijati z namenom koristiti razvoju programske opreme, ki bo dostavljena. Tak način razvoja označimo za »agilen«, ker je sposoben hitro reagirati na uporabnikove spremenjene potrebe.

Agilne metodologije razvoja predvsem zagovarjajo hitre in stalne izdaje programske opreme. To se odraža v hitrejši dostavi novih verzij izdelka, kar pomeni, da lahko uporabniki izdelek takoj preizkušajo. Zaradi tega hitro pridemo do povratnih informacij, sprememb zahtev, splošnih komentarjev itd. Zaradi načina izdaj, ki ga zagovarjajo agilne metodologije, lahko uporabniki tudi hitro vidijo rezultate svojih pripomb, ki jih posredujejo po tem, ko izdelek sami preizkusijo.

Agilni razvoj programske opreme močno pripomore k izdelavi delujoče programske opreme v zastavljenem času in znotraj finančnih okvirjev. Predvsem pa agilnost zagotavlja razvoj izdelka, ki je dejansko tisto, kar si uporabnik želi oziroma potrebuje. Tradicionalne metode močno zaostajajo pri razmerah, ki so dandanes prisotne na trgu, saj se v tem hitro razvijajočem svetu uporabniške zahteve navadno res hitro spreminjajo. Poleg tega pa velikokrat nastane situacija, ko je izdelek zastarel, še preden se je sploh začel uporabljati. Posluževanje standardnih metod velikokrat rezultira v prepozni dostavi ali predragem izdelku, po navadi pa kar v obojem. Velikokrat je končna verzija brez določenih ključnih funkcionalnosti, z lastnostmi, ki ne koristijo nikomur ter mnogimi napakami. Agilnost sicer ne zagotavlja idealnega izida projekta, še manj lahkega oziroma enostavnega postopka. Zagotavlja pa hitro prilagajanje spremembam in uporabno programsko opremo znotraj projektnih meja.

Zamisel o agilnih tehnologijah se je rodila v devetdesetih letih prejšnjega stoletja in hitro dobila široko podporo predvsem pri razvijalcih. Že leta 2001 pa so se zbrali strokovnjaki s tega področja in sestavili tako imenovani Manifest agilnosti (ang. Agile Manifesto) [8]. Povezali so se v Agile Software Development Alliance, kasneje preimenovano v Agile Alliance. To je neprofitna organizacija, ki še danes promovira in raziskuje področje agilnih metodologij z namenom narediti programsko industrijo bolj produktivno, dostopno, humano itd. Kljub preteklim letom in nadaljnjemu razvoju področja pa organizacija še danes kot temelj delovanja uporablja Manifest agilnosti.

2.2. Manifest ter načela agilnosti

Manifest agilnosti je bil napisan februarja leta 2001 na konferenci v Snowbird-u (Utah, ZDA). Sedemnajst neodvisnih strokovnjakov, ki so delovali na različnih področjih in se ukvarjali z različnimi agilnimi metodologijami, je med drugim razpravljalo o skupnih značilnostih agilnih tehnik [1]. S skupnimi močmi so glavne vrednote agilnosti strnili v poseben dokument: Manifest agilnosti. V njem je zapisano:

Manifest agilnosti:

Z razvijanjem in nudenjem pomoči drugim pri razvijanju programske opreme odkrivamo boljše načine le-tega. Pri svojem delovanju dajemo prednost naslednjim vrednotam:

Posameznikom in sodelovanju pred *procesi in orodji*

Delujoči programski opremi pred *obsežno dokumentacijo*

Sodelovanju z naročnikom pred *pogajanjem o pogodbi*

Reagiranju na spremembe pred *sledenju načrtom*

Torej, čeprav imajo vrednost tudi zapisi na desni, bolj cenimo zapise na levi.

Sodelovanje ljudi je pri razvoju programske opreme ključnega pomena. Razvojni procesi, metodologije, orodja itd. so sicer pomembne sestavine, ampak izmenjava informacij med soudeleženimi mora biti na prvem mestu. Prav tako je dokumentacija lahko dobra podpora tako samemu razvoju kot kasnejši uporabi izdelka, ne sme pa postati sama sebi namen. Bistvo razvoja je programska oprema in ne dokumentacija.

Agilnost že sama po sebi pove, da je poudarek na prilagajanju. V kolikor se zahteve spremenijo, smo sposobni reagirati na spremembe in nismo toliko vezani na začetne zahteve oziroma na fiksen plan. To seveda ne pomeni, da planiranje ni potrebno. Plan je zelo pomemben, a se prilagaja uporabnikovim povratnim informacijam. Nasploh nam je v interesu, da je uporabnik glede tega čim bolj angažiran, da nam posreduje informacije ter opombe in da je vseskozi dosegljiv. Agilne metodologije zagovarjajo to, da bi moral biti čas porabljen s sodelovanjem in integriranjem uporabnikov v razvoj, in ne s podrobnimi pogajanji o pogodbi. Na žalost ni vedno tako, saj je v realnem svetu to težko doseči. Vodstva podjetji navadno vztrajajo pri striktnih pogodbah, da se zavarujejo njihovi interesi.

Vse to zajema Manifest agilnosti. Iz tega dokumenta so člani združenja Agile Alliance kasneje specificirali tudi nabor načel, ki naj bi pomagal pri agilnem razvoju programske opreme [9].

Načela agilnosti

- Najvišja prioriteta je zadovoljiti kupca z zgodnjim in kontinuiranim dostavljanjem programske opreme z neko težo, vrednostjo.
- Spremembe v specifikacijah so dobrodošle tudi v poznih fazah projekta.
- Potrebno je pogosto dostavljati delujoče verzije izdelka. Čim pogosteje, tem bolje.
- Interakcija vseh zainteresiranih pri projektu: vodstvo, uporabniki, razvojna ekipa itd.
- Bistvo projekta je zgraditi skupino okrog motiviranih posameznikov. Ti morajo biti deležni primerne podpore in zaupanja.
- Najboljši način izmenjave informacij je v živo, s formalnimi in neformalnimi srečanji.
- Najboljše merilo napredovanja je delujoča programska oprema.
- Potrebno je zagotavljanje trajnostnega razvoja.
- Mera napredka naj bi bila konstantna skozi celoten potek projekta.
- Poseben poudarek je potreben na dobrem načrtovanju.
- Enostavnost in preprostost imata prednost pred vsem ostalim.
- Razvojne skupine si delo organizirajo same.
- Razvojna skupina se mora sestajati v rednih intervalih in oceniti, kako se lahko proces dela izboljša.

2.3. Predstavitev agilnih metodologij

Scrum in metode planiranja, ki se jih da povezati s to metodologijo, bodo predstavljene v naslednjih poglavjih, na tem mestu pa bodo opisane ostale najbolj razširjene agilne metodologije:

- Ekstremno programiranje (ang. XP, Extreme Programming)
- Agilno modeliranje (ang. Agile Modeling)
- Metoda razvoja dinamičnih sistemov (ang. DSDM, Dynamic System Development Method)
- Razvoj, osredotočen na funkcionalnosti (ang. FDD, Feature-Driven Development)

Poleg načel agilnosti pa je agilnim metodologijam skupno tudi to, da so iterativne (ponavljajoče) in inkrementalne (obročne). Proces je iterativen takrat, ko se izdelek izboljšuje preko ponavljajočega dodajanja, rafiniranja. Najprej se naredi del, za katerega se ve, da je pomanjkljiv ali približen. Kasneje se ga skozi več iteracij dopolni do te mere, da zadošča zahtevam. Izdelek tako z vsako iteracijo napreduje z dodajanjem in izboljšanjem funkcionalnosti. Dobra primerjava za to je kiparstvo, kjer kipar najprej naredi okvirno obliko – recimo glave in trupa, in se šele kasneje loti detajlov – potez obraza.

Proces je inkrementalen, ko se ga lahko razdeli v več delov, vsak kos pa predstavlja določeno zaokroženo celoto. Inkrement je lahko velik ali majhen, odvisno od definicije. Vsak del je kodiran in testiran do konca in pričakuje se, da se podmnožic ne bo več dopolnjevalo.

2.3.1. Ekstremno programiranje

Ekstremno programiranje je bilo sprva razvito za manjše skupine, ki so delovale z negotovimi in spreminjajočimi se zahtevami [1]. Razvito je bilo na principih programskega inženirstva, s poudarkom na točni dostavi izdelka in na njegovem ustrezanju uporabniškim specifikacijam. Glavni akter tehnologije je razvijalec, zato je na njem tudi poudarek. S tem dobi večjo vlogo, a tudi večjo odgovornost. V ospredje stopita razvojna skupina in skupinsko delo.

Glavne značilnosti ekstremnega programiranja:

Programiranje v parih je ena izmed stvari, ki jih najprej povežemo z ekstremnim programiranjem. Pri tem načinu dva razvijalca dejansko sedita skupaj za istim monitorjem in sodelujeta pri kodiranju. S tem dosežemo, da je vsa koda pregledana že pri nastanku, kar pomeni, da je preglednejša ter kvalitetnejša. Tak način veliko prispeva tudi k stalni izmenjavi mnenj in informacij med razvijalcema.

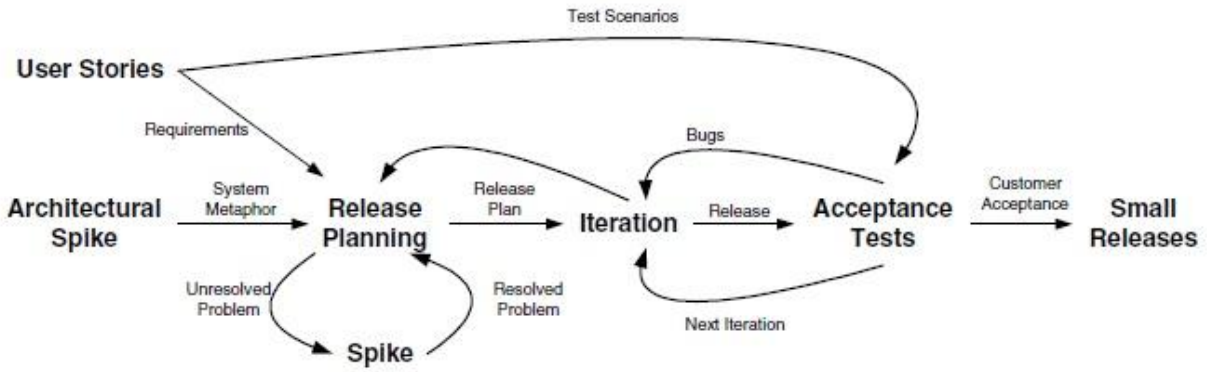
Vzdrževanje stalnega kontakta s stranko je ključnega pomena. Velikokrat se določi predstavnika, ki je v stalnem kontaktu z razvijalci oziroma postane del razvojne skupine. Potrebna so pogosta srečanja z naročnikom, da se stalno izmenjuje informacije in pripombe.

Teste za programsko opremo je potrebno napisati, **preden začnemo s kodiranjem**. Ekstremno programiranje sledi logiki, da v kolikor testov nismo sposobni napisati, to pomeni da ne poznamo vhodnih in izhodnih podatkov, torej o izdelku nimamo dovolj informacij in ni pravega smisla v tem, da bi z razvojem sploh začeli. Teste je potrebno tudi sproti poganjati, sploh ob večjih spremembah. S tem se zaščitimo pred nevarnostjo, da bi poškodovali že delujoče dele kode.

Potrebno je imeti **kratke iteracije**, tako da si zagotovimo hitre in stalne povratne informacije. Poleg tega je manjši sistem lažje spraviti v produkcijo in ga preizkušati. Naknadno se izdelek razširi v tisto smer, ki je potrebna.

Potreben je poudarek na **preprostem načrtovanju**. V kolikor imamo enostaven in preprost sistem, se izognemo prevelikim zapletom v začetnih iteracijah, ki so že tako ali tako izpostavljene velikim tveganjem. Dopolni in razširi se ga v kasnejših iteracijah. Potrebno je tudi ohraniti poudarek na lastnostih sistema, ki so potrebne zdaj, in ne razmišljati o funkcionalnostih, ki bodo potrebne v prihodnosti. Lahko se namreč izkaže, da so nepotrebne ali nepopolne. V vsakem primeru ni dobro delati prevelike znanosti iz enostavnih stvari in potrebno ohraniti pogled na danes in ne na jutri.

Vsi člani razvojne skupine so za delo **soodgovorni**. Zavedati se morajo, da so vsi v tem skupaj. V kolikor obstaja napaka, imajo vsi nalogo, da se jo čim prej odpravi. Vsi so odgovorni za celotno kodo oz. za celoten projekt.



Slika1.1: Razvojni proces ekstremnega programiranja

Pomemben del Ekstremnega programiranja so uporabniške zgodbe. To so zapisi uporabniških zahtev ali lastnosti sistema, ki imajo za uporabnika tudi neko uporabno vlogo. Navadno jih napiše uporabnik sam ali razvojna skupina v sodelovanju z uporabnikom. Sestavljene so iz kratkega opisa, opomb in sprejemnih testov (ang. Acceptance tests), po katerih se lahko preveri, ali je uporabniška zgodba dokončana.

Potek razvojnega projekta (Slika 1.1) pri ekstremnem programiranju se začne s sestankom za planiranje izdaje (ang. Release Planning). Tu se poskuša razdelati celoten projekt. Potrebno je ugotoviti, kako dolge bodo iteracije, koliko jih bo, kaj vse zajema projekt itd. Gre za usklajevanje med razvijalci in uporabniki izdelka glede celotnega projekta. Za podatke pri določanju in ocenjevanju se uporabljajo ocene iz uporabniških zgodb (npr. točke zahtevnosti). Sestanek oziroma pogovori trajajo, dokler niso vse strani zadovoljne. Ustreženo mora biti tako prioritetam naročnika, kot zmožnostim razvojne ekipe. Tudi ocene morajo biti v ugotovljenih okvirih.

Ekstremno programiranje daje poseben poudarek tako imenovanim eksperimentom v sklopu razvojnega procesa (ang. Spikes). Gre za iskanje rešitve za problematične dele kode. Obravnava jih posebej in jim daje posebno mesto, saj so to deli, ki jih je potrebno dodatno raziskati in preveriti.

Iteracije so kratki intervali, v katerih naj bi razvili delujoč izdelek. Ekstremno programiranje zagovarja kratke iteracije, ki naj bi trajale vsega skupaj nekaj tednov. Iteracije niso planirane vnaprej, ampak se jih planira tik pred začetkom, kar doda k agilnosti. Planiranje šele pred začetkom pomeni, da se njihova vsebina, kljub začetnemu planu, občutno spremeni.

Na koncu iteracije se vedno preveri še teste ustreznosti uporabniških zgodb. To je sicer integralni del uporabniških zgodb, a imajo v razvojnem ciklu ekstremnega programiranja tudi testi posebno mesto. Ko je iteracija zaključena, je potrebno preveriti ustreznost in se odzvati zaključkom primerno (v kolikor implementacija ni ustrezna, uporabniška zgodba ni dokončana – navadno se jo prenese v eno izmed naslednjih iteracij).

2.3.2. Agilno modeliranje

Agilno modeliranje poskuša integrirati agilno razmišljanje v modeliranje programske opreme. Cilj tega je najti primerno ravnotežje med preveč in premalo modeliranja v fazi načrtovanja projekta. To pomeni, da poskuša:

Modelirati do mere, da imamo dovolj informacij, da učinkovito raziščemo in dokumentiramo sistem, ampak ne toliko, da modeliranje postane obremenjujoče [1].

Strogo vzeto ne gre za metodologijo, temveč za nek prijem oziroma filozofijo, ki jo poskušamo vpeljati. Lahko je združena z drugimi tehnikami, tudi povsem standardnimi. Agilno modeliranje poskuša promovirati čim bolj enostavne procese ter nas zalaga z iztočnicami in nasveti, kako kreirati uspešne modele in biti pri tem čim bolj učinkoviti.

Agilno modeliranje ima tri glavne cilje:

- Definirati in promovirati vrednote, principe in prakse (ang. Practices), ki pomagajo zgraditi primerne modele.
- Pomoč pri apliciranju modeliranja v agilni produkciji programske opreme.
- Pomoč pri integraciji agilnega modeliranja v ostale procese izdelave programske opreme.

Agilno modeliranje predpisuje modele, ki so namenjeni zgolj določeni publiki. Modeli morajo biti razumljivi tistim, ki so jim namenjeni, kar pa še ne pomeni, da morajo biti razumljivi vsem. Vedno je potrebno imeti pred seboj tistega, ki mu je potrebno posredovati določeno informacijo. Narediti je potrebno model, ki je dovolj dober, da služi svojemu namenu, ne sme pa biti namenjen samemu sebi.

Agilno modeliranje zagovarja dejstvo, da mora imeti vsak model neko vrednost oziroma nositi določeno informacijo. Služiti mora svojemu namenu, ampak nič več kot to. Seveda pa je potrebno da so modeli natančni, pravilni, razumljivi itd. Poleg tega morajo biti dovolj konsistentni, vendar nič ne de, če so nekateri bolj podrobni kot drugi. Vedno se je treba ravnati po končni publiki. Znotraj omenjenih mej in poudarkov se stremi k temu, da je model kar čim bolj enostaven. Poleg tega, da nam tak model ne vzame preveč časa, pa so taki modeli tudi najbolj razumljivi.

Nekateri avtorji vidijo Agilno modeliranje bolj kot umetnost in ne toliko kot znanost [1]. Nasploh naj bi agino modeliranje bolj podajalo neka načela za agilno dokumentacijo kot neko ogrodje za razvoj. Tehnika tudi ne stoji na stališču, da je dokumentacija nepomembna, še več, prav modeli, ki jih predpisuje, so v bistvu neke vrste dokumentacija. Dokumentacija, ki sloni na agilnih modelih.

2.3.3. Metoda razvoja dinamičnih sistemov

Metoda razvoja dinamičnih sistemov (ang Dynamic System Development Method - DSDM) nudi ogrodje kontrol najboljših praktičnih prijemov za hiter razvoj aplikacij ali RAD (Rapid Application Development) [1]. Najbolj je ta metoda primerna za kompleksne sisteme, ki so še posebej omejeni s časom. Tehnika, ki je bila sprva v celoti poznana kot RAD, je bila zasnovana že v sredini devetdesetih let. Kasneje se oprime agilnih značilnosti in postane agilna metodologija, ki je občutljiva na čas, kvaliteto in ceno.

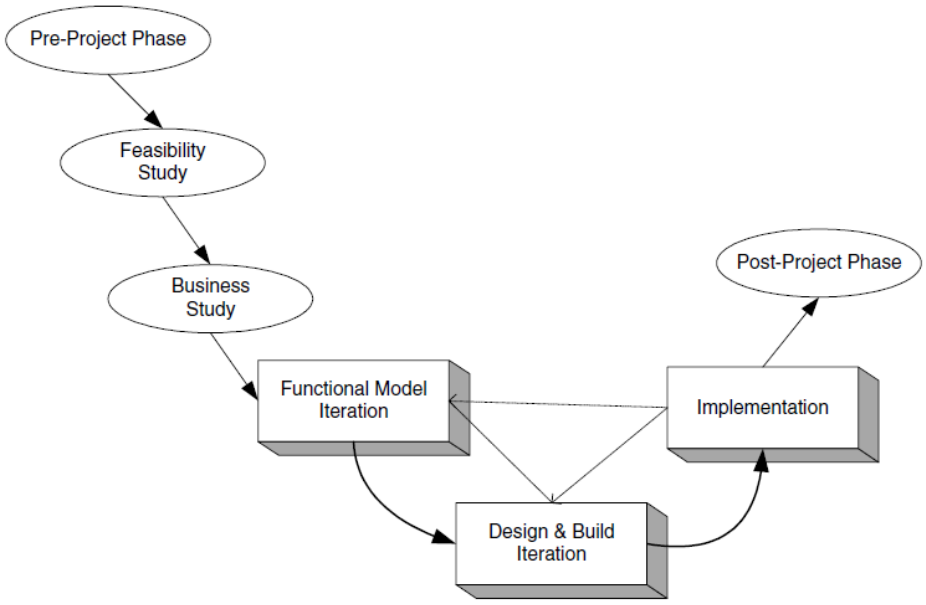
Metoda razvoja dinamičnih sistemov se drži devetih načel:

- Sodelovanje uporabnika je ključnega pomena.
- Razvojna skupina mora imeti dovolj podpore, da lahko odloča.
- Potrebna je hitra dostava izdelka.
- Izdelek mora ustrezati poslovnemu namenu.

- Razvoj mora biti iterativen in inkrementalen.
- Vse spremembe v razvoju so reverzibilne.
- Zahteve so podane na visokem nivoju.
- Testira se znotraj življenjskega cikla.
- Vsi, ki imajo tak ali drugačen interes v projektu, morajo med seboj sodelovati.

Ključni uporabniki, ki delujejo kot neke vrste posredniki med dejanskimi uporabniki in razvijalci, so znani kot ambasadorji. DSDM svoje projekte razširi v sedem faz (Slika 1.2). Prve tri predstavljajo neke vrste pripravo na projekt, kjer se sploh oceni, ali je primerno delati po tej metodi in ali se bo po njej tudi delalo. Zadnja predstavlja vse aktivnosti po koncu projekta, ko obstoj razvojne ekipe ni več nujen. Vmesne tri faze predstavljajo jedro. To so:

- Iteracija funkcijskega modela
- Načrtovanje in izgradnja projekta
- Implementacija



Slika 1.2: Razvojni proces DSDM

2.3.4. Razvoj, osredotočen na funkcionalnosti

Planiranje, spremljanje in vodenje projektov, ki so agilni in inkrementalni, je lahko zelo težko. Zaradi svoje narave lahko planiranje v agilnih projektih postane zelo težavno in tak projekt lahko hitro uide izpod nadzora, sploh pri razvijalcih, ki tovrstnih prijemov niso navajeni. Razvoj, osredotočen na funkcionalnosti (ang. FDD, Feature-Driven Development) zato

poskuša ta nadzor dobiti nazaj s tem, ko poskuša povezati enote zahtev z enotami planiranja in dela [1].

Razvoj, osredotočen na funkcionalnosti, je orientiran na to, da so v projektu v ospredju vsega funkcionalnosti. Funkcionalnosti so v tem primeru sorodne primerom uporabe (ang. Use case). S tem dosežemo, da uporabnik dobi, kar želi, obenem pa lahko aktivnosti ob tem planiramo in spremljamo. To je še posebej pomembno, ker smo velikokrat omejeni s strani vodstva z obveznostjo predstavitev rezultatov in napredka. Tak način omogoča lažje spremljanje in poročanje o poteku projekta. Razvoj, osredotočen na funkcionalnosti, je le eden izmed načinov, ki se osredotoča na ta pogled, poleg njega je zelo popularen tudi EVO (Evolutionary Development).

Definicija funkcionalnosti v tem primeru je 'Opredmetena zahteva, povezana z aktivnostjo, s katero jo realiziramo in zahteva, ki se jo da umestiti v časovnico'. Zahteva je lahko specificirana s strani uporabnika, vodstva, ali pa gre za interno zahtevo razvojne ekipe. Vsaki funkcionalnosti določimo poleg opisa oziroma definicije tudi ceno in prioriteto.

Funkcionalnosti naj bi imele naslednje atribute:

- Morajo biti majhne in uporabne.
- Lahko se jih združuje v množice, ki imajo za uporabnika skupno vrednost.
- Funkcionalnosti naj bi ohranjale fokus razvijalca na tistih zadevah, ki imajo določeno vrednost za zainteresirane pri projektu,
- Se jih da razvrstiti po prioriteti.
- Se jih da umestiti v določen urnik.
- Imajo določeno ceno.
- Lahko se jih združi v kratke iteracije.

Tudi to metodo se lahko povezuje z drugimi oblikami agilnih metodologij. Tako se na primer lahko poleg Razvoja, osredotočenega na funkcionalnosti uporabi Agilno modeliranje v fazi načrtovanja projekta ter principe Ekstremnega programiranja v fazi implementacije. Kljub vsem prednostim agilnega programiranja, ki smo jih omenili do sedaj, pa seveda agilnost odpira vrata tudi določenemu tveganju. Razvoj, osredotočen na funkcionalnosti, močno pripomore k brzdanju le-tega.

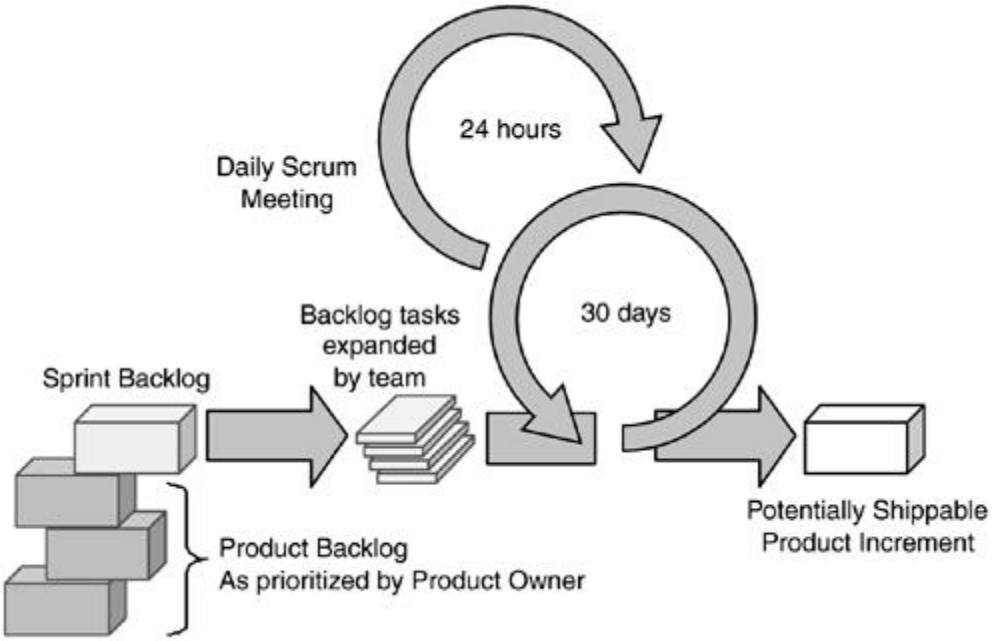
Zato FDD ponuja več procesov:

- Izdelava modela celotnega sistema in izdelava začetnega seznama funkcionalnosti.
- Izdelava podrobne liste funkcionalnosti, ki jim dodelimo prioritete.
- Planiranje glede na funkcionalnosti.
- Načrtovanje glede na funkcionalnosti.
- Grajenje sistema z poudarkom na funkcionalnostih.

3. Opis metodologije Scrum

3.1. Uvod

Prvi uradni zapis pravil Scrum je nastal leta 1995 izpod peresa J. Sutherland-a in K. Schwaber-ja, čeprav korenine nastanka metodologije segajo dlje. Že leta 1986 sta H. Takeuchi in I. Nonaka opisala postopek razvoja izdelka, pri katerem se faze procesa prekrivajo in razvojna skupina sodeluje pri celotnem procesu [6]. Kot »Scrum« je tehniko poimenoval J. Sutherland leta 1993, in sicer je metodologijo primerjal z istoimensko formacijo igralcev pri igranju rugby-ja [5]. Za prvo resno publikacijo na to temo se šteje knjiga, ki je izšla leta 2001 - Agile Software development with Scrum, katere avtorja sta K. Schwaber in M. Beedle.



Slika 3.1: Grafični prikaz Scrum razvojnega procesa [2]

Projekti SCRUM so navadno razdeljeni v serije 30-dnevnih iteracij, imenovanih »Sprint« ali cikel (Slika 3.1). Na začetku cikla razvojna ekipa določi količino dela, ki ga lahko opravi v danem intervalu. Naloge se določijo po prioritetnem spisku, imenovanem »Product backlog« ali seznam zahtev. Izbrane naloge se prenese v seznam nalog posameznega cikla ali »Sprint backlog«. Sprotno spremljanje dela pa se opravi s kratkimi jutranjimi sestanki ali dnevnimi cikli (ang. Daily Scrum) [4].

Pregled Scrum-a je razdeljen v pregled Scrum vlog, časovnih okvirjev ter Scrum izdelkov. Vsi, ki so vpleteni v delovanje Scrum, so Scrum skupina, kljub temu da znotraj nje obstajajo različne vloge.

Scrum skupina je oblikovana tako, da doseže kar največjo fleksibilnost in produktivnost. Za to je potrebno, da se skupina organizira sama organizira, da se člani ne razporejajo po funkcionalnih področjih in da delo poteka v iteracijah. Pri vsaki Scrum skupini ločimo tri vloge glede na odgovornost in naloge, ki jih posamezniki opravljajo. To so:

- Skrbnik metodologije (ang. Scrum Master)
- Skrbnik izdelka (ang. Product Owner)
- Razvojna skupina (ang. Team)

3.2. Scrum vloge

Kot že omenjeno, Scrum znotraj skupine loči tri vloge. Skrbnik metodologije mora poskrbeti, da Scrum proces teče gladko in da ga vsi razumejo, skrbnik izdelka je zadolžen, da ima opravljeno delo kar čim večjo vrednost za naročnika, razvojna skupina pa je zadolžena za implementacijo. Zelo pomembno je, da je vedno jasno, kdo opravlja katero vlogo in kdo je za kaj odgovoren [4].

Ljudje, ki izvajajo zgoraj opisane vloge, so tisti, ki so neposredno vključeni v projekt, niso pa edini, ki so zainteresirani za potek projekta. Veliko je s projektom tako ali drugače povezanih ljudi, ki pa zanj niso odgovorni. V Scrum postopkih prve, torej Scrum skupino, imenujemo »prašiči«, druge pa »piščanci«. Imeni izvirata iz neke šale, kjer piščanec in prašič skupaj odpirata restavracijo. Prašič, na koncu, na to ne pristane, ker ime restavracije (Šunka in jajca) ponazarja razliko med vlogama: prvi bi bil projektu predan, drugi pa v njem samo udeležen. To naj bi bila tudi glavna razlika med obema skupinama zainteresiranih za projekt. Scrum strogo ločuje obe skupini in poskrbi, da imajo odgovorni za projekt vsa pooblastila, da projekt uspešno izvedejo. Ostali se v to ne smejo vmešavati.

3.2.1. Skrbnik metodologije

Skrbnik metodologije skrbi, da celoten Scrum proces poteka nemoteno. Dolžan je pomagati razvojni skupini, da ta postane čim bolj produktivna in da dvigne kvaliteto končnega izdelka. Vse vpletene mora podučiti o Scrum-u in skrbeti, da se dosledno upoštevajo Scrum pravila. Metodologijo je potrebno tudi integrirati v obstoječe procese organizacije, tako da ta ne motiji delovanja družbe [4].

Skrbnik metodologije opravlja podobne naloge kot projektni vodja s to razliko, da ekipo predvsem usklajuje. Skrbnik metodologije je bolj v službi skupine. Njegova poglavitna naloga je odstranjevanje ovir in pomoč pri premagovanju problemov. Skrbnik metodologije je lahko tudi del razvojne ekipe, nikakor pa ne more biti tudi skrbnik izdelka. Nasploh je bolje, da se vloge v Scrum-u ne združujejo, temveč da jih opravljajo različni ljudje.

3.2.2. Razvojna skupina

Razvojna skupina skrbi za razvoj funkcionalnosti. V vsakem ciklu mora zapise s seznama zahtev spremeniti v implementacijo, tako da po vsakem ciklu dobimo novo verzijo izdelka, ki je tudi uporabna. Kako bo to doseženo, je popolnoma v domeni skupine, saj svoje delo organizira sama. Ne smejo obstajati zunanje direktive, ki bi bile povezane z delitvijo dela. V skupini so vsi člani med seboj enakopravni; uporaba nazivov, kot so programer, arhitekt, analitik,…, ni priporočljiva, kakor tudi ni priporočljiva organizacija v podskupine. Skupina naj bi delovala v duhu »Vsi za enega in eden za vse«, kar pomeni, da so vsi enako odgovorni ter zadolženi za vse. S tem se želi poudariti soodgovornost za projekt in rezultate [4].

Razvojna skupina mora biti sestavljena tako, da imajo vsi njeni člani skupaj dovolj znanja, da se projekt izpelje do konca. Ključnega pomena za kvaliteto izdelka je sinergija članov skupine in izmenjevanje znanja znotraj nje. Člani so lahko tudi ožje usmerjeni, npr. testerji, analitiki, administratorji podatkovnih baz, kljub temu pa si dela ne delijo po tem ključu. Nasploh Scrum zagovarja, da naj se dela ne bi delilo po funkcionalnih področjih in da mora vsak član prispevati del truda pri testiranju, načrtovanju, kodiranju itd., ne glede na svojo specializacijo. V vsakem primeru se ne sme poudarjati različnih specialnih sposobnosti posameznikov, temveč njihovo skupno lastnost, torej to, da so soodgovorni za potek in rezultate projekta. Ta lastnost je tudi glavna za uspeh projekta.

Razvojna skupina Scrum je navadno sestavljena iz štirih do sedmih ljudi [2]. Nekateri avtorji močno zagovarjajo, da je idealna izbira sedem plus ali minus dva člana [4]. Glavna utemeljitev je v tem, da se je v vsakdanji praksi enostavno pokazalo, da ima vsak večji odklon za posledico premajhno interakcijo članov ali pa je za normalno delo skupine potrebno preveč koordinacije. Obstajajo pa seveda tudi izjeme. Scrum prepoveduje kakršnokoli spreminjanje razvojne skupine med izvajanjem cikla, bodisi v sestavi bodisi v številu (npr. dodajanje članov, ker projekt zamuja). Tudi po koncu cikla se brez utemeljenega razloga odsvetujejo posegi v sestavo ekipe.

3.2.3. Skrbnik izdelka

Skrbnik izdelka je zadolžen, da zastopa vse, ki imajo take ali drugačne interese pri projektu. On je tisti, ki postavi temelje projekta s tem, ko razgrne osnovne zahteve, časovni načrt ter

pričakovano dodano vrednost (ang. ROI – Return on Investment) projekta. Skrbnik izdelka je glavni odgovorni za sestavo seznama zahtev in za to, da ima delo, ki ga opravlja razvojna skupina, za naročnika uporabno vrednost. Na splošno skrbi za seznam zahtev, nanj uvršča zahteve in pazi, da je dostopen vsem. Pomembna naloga skrbnika je tudi to, da poskrbi, da se glavne funkcionalnosti projekta implementirajo najprej. To doseže s postavljanjem prioritet zapisov na seznamu zahtev.

Skrbnik je lahko samo eden in ne skupina ali oddelek. Lahko obstaja skupina, ki svetuje ali ima kontakt s skrbnikom, vendar ima končno besedo vedno tisti, ki opravlja to vlogo. V kolikor želi kdo spremeniti ali dopolniti zapis v seznamu zahtev, mora najprej poskrbeti, da je podobnega mnenja tudi skrbnik izdelka.

Skrbnik izdelka mora uživati zadostno mero podpore v organizaciji in njegove odločitve morajo biti spoštovane. Nihče ne sme ekipi podajati navodil oziroma predlogov, ki niso v skladu s prioritetami ali navodili skrbnika. To seveda tudi pomeni, da se mora skupina tega držati. Skrbnikovo delo se najbolj odraža skozi sestavo in razlago seznama zahtev. Biti skrbnik izdelka je obenem zahtevna naloga in naloga, ki lahko tistemu, ki jo opravlja, veliko da.

3.3. Časovni okviri

3.3.1. Cikel

Cikel (ang. Sprint) predstavlja zaključeno in nedeljivo iteracijo, katere rezultat je uporabna verzija izdelka. Scrum predpisuje trideset koledarskih dni, v praksi pa mnogi zagovarjajo štirinajstdnevni cikel. Pri Scrum-u je cikel sestavljen iz [4]:

- Sestanka za izdelavo plana iteracije (ang. Sprint Planning Meeting)
- Razvoja izdelka
- Sestanka za pregled rezultatov dela (ang. Sprint Review Meeting)
- Sestanka za oceno kakovosti razvojnega procesa (ang. Sprint Retrospective Meeting)

Za vsak cikel je potrebno pred njegovim začetkom definirati cilje in vsebino. Skrbnik metodologije pa mora poskrbeti, da se med potekom iteracije njenih elementov ne spreminja (sestava skupine, vsebina iteracije, kvaliteta dela, cilji,…). Cikli si sledijo drug za drugim brez časovnega razmika. Pomemben del Scrum cikla so tudi dnevni sestanki, ki služijo za sprotni pregled dela [4].

Cikel se lahko tudi prekine oziroma ustavi, v kolikor se pokaže, da cilji, ki so bili zastavljeni na začetku projekta, niso več smotrni. Kljub vsemu je treba stvari zelo dobro pretehtati in premisliti. Skrbnik izdelka je edini, ki lahko prekine cikel. To se zgodi zelo redko.

3.3.2. Sestanek za izdelavo plana iteracije

Vsak cikel je potrebno pred njegovim začetkom definirati. Zato se ena iteracija vedno začne s sestankom za izdelavo plana naslednje iteracije, na katerem se podrobno razdela zahteve projekta in temu primerno definira seznam nalog za prihajajoči cikel (ang. Sprint Backlog). Slednji se tekom cikla vseskozi dopolnjuje, tako da lahko razvojna skupina po koncu iteracije doseže cilj cikla. Sestanek navadno traja cel dan oziroma 8 ur in je razdeljen na dva dela, vsak po 4 ure, če trajajo cikli po en mesec. V kolikor so iteracije krajše, se sestanek temu primerno skrajša [4].

V prvem delu se analizira zahteve, specificirane na seznamu zahtev izdelka (ang. Produkt Backlog), v drugem pa skupina sestavi seznam nalog. Zahtevana je prisotnost celotne razvojne skupine in obeh skrbnikov. Sestanku lahko občasno prisostvujejo tudi drugi zainteresirani, vendar samo, v kolikor nudijo vpogled v nove informacije oziroma odgovarjajo na morebitna vprašanja. Po tem sestanek zapustijo. Opazovalci niso dovoljeni.

Nekateri avtorji dela sestanka poimenujejo »Kaj?« (prvi del sestanka) ter »Kako?« (drugi del) [4]. V prvem delu, kjer se odgovarja na »Kaj?« - skrbnik izdelka razloži posamezne zapise na seznamu zahtev, ki mora biti pripravljen pred začetkom sestanka (pripravi ga skrbnik izdelka), in postavi prioritete. Razvojna skupina se mora v dovolj veliki meri seznaniti z zahtevami, tako da lahko v drugi polovici sestanka določi naloge, ki se jih bo lotila v prihajajočem ciklu. Skrbniku izdelka ni potrebno razlagati vseh zapisov s seznama, temveč se lahko omeji samo na tiste najpomembnejše in pusti ostalo za kasnejše cikle.

Razvijalci navadno že v tem delu postrežejo z okvirnimi ocenami, koliko in kaj je mogoče narediti v prihajajočem ciklu. Točno vsebino cikla se formira skupaj s seznamom nalog v drugem delu sestanka, kljub temu pa je v tem delu potrebno sprejeti odločitev, koliko zapisov s seznama zahtev z najvišjo prioriteto se še da uresničiti v danem obdobju. Morebitni preostali čas se zapolni z zapisi z nižjo prioriteto. Prednost imajo tisti, ki so v povezavi z že izbranimi. Pomembni podatki, s katerimi mora Scrum skupina tukaj razpolagati, so: kako je razvojna skupina delovala v preteklosti, kakšna je razpoložljivost razvojne skupine v naslednjem ciklu, kaj je bilo narejenega do sedaj itd.

Poleg zgoraj omenjenega udeleženi skupaj sestavijo krajši opis, ki opredeljuje cilj cikla (ang. Sprint Goal). To je neke vrste izjava, ki opredeljuje smer in namen cikla ter služi kot glavno vodilo skupine. Na koncu cikla se preveri predvsem, ali je bil dosežen glavni cilj iteracije. V primeru, da je bil, se šteje iteracija kot uspešna kljub temu, da morebiti niso bile v polni meri implementirane vse funkcionalnosti s seznama nalog.

V drugem delu (»Kako?«), se razvojna ekipa sestane ločeno. Člani se pogovorijo o tem, kar so slišali v prvem delu in se na podlagi tega odločijo, kako se bodo dela lotili v drugem delu. Glavni izdelek sestanka je izbrani seznam zapisov s seznama zahtev, ki se ga podrobno razčleni na naloge. Imenuje se kar seznam nalog (ang. Sprint Backlog). Seveda gre pri

formulaciji le-tega za dogovarjanje s skrbnikom izdelka, vendar je na koncu vedno ekipa tista, ki odloča o končni verziji seznama. Skrbnik izdelka je na voljo, vendar, kot rečeno, ne sodeluje pri sestavi seznama, temveč navadno le odgovarja na vprašanja.

Seznam nalog se tudi med potekom iteracije vseskozi spreminja, saj se stalno spreminjajo naloge. Scrum tudi ne zahteva, da se vse naloge izbere že na začetku cikla, temveč celo stimulira sprotno ugotavljanje nalog. Po drugi strani pa ni dovoljeno dodajati novih zapisov, izbranih s seznama zahtev, k seznamu nalog. Nihče, ki ni član razvojne skupine, ne more posegati v zahteve cikla. Edini način, da se to zgodi, je če razvojna skupina delo zaključi občutno pred urnikom. V tem primeru razvojna skupina prosi skrbnika izdelka, da jim pomaga izbrati še kakšen zapis s seznama zahtev.

3.3.3. Sestanek za pregled rezultatov dela

Na koncu vsakega cikla mora razvojna ekipa izdati delujočo programsko kodo, ki je sestavni del naslednje izdaje. To pomeni, da se v vsakem ciklu ustvari kos programske opreme, ki je tudi testiran in pripravljen za uporabo. Izdelek lahko stranka dostavi uporabnikom ali pa ga uporablja interno. Odločitev je odvisna od tega, ali ima nova verzija dovolj novih funkcionalnosti, ter od namena programske opreme.

Za pregled nove verzije se skliče sestanek za pregled rezultatov cikla (ang. Sprint Review Meeting). Tudi ta sestanek je časovno omejen – 4 ure za mesečni cikel. Med tem sestankom ekipa razvijalcev predstavi, kar je bilo v tem ciklu narejeno. Navadno je to kar demonstracija novih funkcionalnosti in pogovor o tem, v kakšno smer naj se razvoj obrne. Paziti je potrebno, da sestanek ne preide v golo napovedovanje nadaljnjega razvoja in bodočih funkcionalnosti s strani razvojne skupine.

Sestanek se navadno začne s tem, da skrbnik izdelka oznani, kaj vse je bilo narejeno v prejšnjem ciklu. Sledi pogovor skupine o tem, kaj je šlo po načrtih in kaj ne, na kakšne probleme so naleteli in kako so jih reševali. Nato pride na vrsto demonstracija novih funkcionalnosti. Po končani predstavitvi skupina navadno odgovarja na vprašanja. Skrbnik izdelka naredi še približen oris nadaljevanja celotnega projekta na podlagi podatkov, s katerimi razpolaga (storilnost, točke zahtevnosti uporabniških zgodb,…), za konec pa se celotna skupina pogovori o tem, kar je bilo predstavljeno in kako to vpliva na nadaljnji razvoj. Ta sestanek zagotovi precej koristnih informacij za sestanek za izdelavo plana naslednje iteracije, ki mu sledi kmalu potem.

Sestanek je namenoma čim manj formalen, uporaba raznih predstavitvenih programov (PowerPoint, Impress) je prepovedana in čas, ki je namenjen pripravi na sestanek, je omejen na dve uri. Postavljena pravila imajo namen, da sestanek za pregled rezultatov dela ne postane preveč moteč element, temveč neke vrste naravni zaključek cikla. Tudi na ta sestanek so povabljeni vsi – razvojna skupina, skrbnik izdelka ter skrbnik metodologije. Prav tako lahko

sestanku prisostvujejo vsi ostali zainteresirani, ki lahko tukaj tudi podajo svoj komentar in postavljajo vprašanja razvojni skupini.

Ključni del vsebine je pregled zastavljenih ciljev in ugotovitev, ali so bili doseženi. Bolj kot posamezni zapisi s seznama zahtev pa je pomembno, da je ekipa dosegla glavni cilj cikla, ki je bil zapisan na uvodnem sestanku.

3.3.4. Sestanek za oceno kakovosti razvojnega procesa

Po sestanku za pregled rezultatov dela in pred naslednjim sestankom za izdelavo plana iteracije ima Scrum skupina sestanek za oceno kakovosti razvojnega procesa (ang. Sprint Retrospective Meeting). Na tem sestanku, ki je omejen na tri ure za mesečne cikle, poskuša skrbnik metodologije dobiti povratne informacije s strani razvojne ekipe o razvojnem procesu.

Namen sestanka je analizirati, kako je potekal zadnji cikel z ozirom na ljudi, medsebojne odnose, kvaliteto komunikacije, proces dela in uporabo raznih orodij. Predvsem je treba poudariti, kaj vse se je izkazalo za koristno in kaj bi se dalo izboljšati. Sem spadajo predvsem: metode komuniciranja, orodja, potek sestankov, definicija tega, kdaj je naloga »dokončana« in podobno. Na koncu sestanka naj bi skupina prišla do predlogov, kako proces izboljšati in ga narediti bolj zanimivega.

3.3.5. Dnevni sestanek za sprotni pregled dela

Scrum je po vsej verjetnosti prva dokumentirana tovrstna metodologija, ki vključuje kratek dnevni sestanek [2]. Ta je pri Scrum-u poimenovan kar dnevni sestanek (ang. Daily Scrum) oziroma dnevni sestanek za sprotni pregled dela. Ideja se je hitro pojavila tudi v drugih agilnih procesih in postala neke vrste zaščitni znak agilnih metodologij. Dnevni sestanek je tako pomemben, ker ne zahteva veliko časa in ni pretirano moteč za delovni proces, hkrati pa služi kot sredstvo pridobivanja in izmenjave informacij vseh udeleženih v projektu. Predstavlja sprotni posnetek stanja in kontrolno točko projekta.

Najbolj primeren čas za dnevni sestanek je zgodaj zjutraj, vendar šele takrat, ko so vsi zainteresirani prišli na delovno mesto. To se navadno zgodi med 9:00 in 9:30 dopoldan. Sestanka se morajo udeležiti vsi v razvojni skupini, kakor tudi oba skrbnika. Sestanek je namerno kratek, navadno 15 minut ali manj, v vsakem primeru pa ne več kot 30 minut. Da se vse skupaj ne bi pretirano razvleklo, se v nekaterih primerih zahteva, da vsi, ki so neposredno udeleženi, stojijo celoten čas sestanka (ang. Stand-up meeting). V kolikor se kdo sestanka ne more udeležiti, je potrebno, da nekdo drug poroča namesto njega. Najboljše je, da so dnevni sestanki vsak dan ob isti uri in v istem prostoru skozi celoten cikel.

Med dnevnim sestankom vsi razvijalci odgovorijo na naslednja vprašanja:

- Kaj si naredil včeraj?
- Kaj boš naredil danes?
- Na kakšne ovire si naletel?

Zelo pomembno je, da sestanek ne preide v zasliševanje s strani skrbnika metodologije. Napovedi udeleženih so predvsem zaveza ostalim razvijalcem oz. eden drugemu in ne toliko vodilnim ali podjetju. Celoten potek sestanka se mora čim bolj oklepati zgornjih treh vprašanj, saj se le tako doseže njegov namen. V kolikor je potrebno določene stvari doreči ali se pogovoriti o njih, se za to skliče poseben sestanek. Dovoljeno je tudi, da se določi del skupine, ki se sestane takoj po dnevnem sestanku, da reši morebitne zaplete.

Skrbnik metodologije je tukaj predvsem povezovalec in tisti, ki skrbi, da se pogovor ne oddalji preveč od zastavljenih vprašanj. Na sestanku je prisoten tudi skrbnik izdelka, saj mora tudi on poročati o svojem delu – pisanju testov, specifikacij, itd. [2].

Prednost dnevnih sestankov je tudi v tem, da služi kot naključna kontrolna točka vsem tistim, ki imajo takšne ali drugačne interese v projektu, vendar niso del Scrum skupine (»piščanci«). Tako se navadno tudi nadomesti dodatne sestanke na koncu cikla, ki so namenjeni informiranju ostalih zainteresiranih (managerji, uprava, tehnični vodje,...). Četudi se »piščanci« lahko kadarkoli udeležijo sestankov, pa v njih ne smejo imeti aktivne vloge, prav tako pa ne smejo motiti vseh neposredno udeleženih.

Dnevni sestanek veliko pripomore k temu, da vsi v razvojni skupini vidijo posnetek celotnega trenutnega stanja. To je tudi idealen čas, da se premisli o trenutnih zadolžitvah. V kolikor pomembne zadeve zaostajajo, je smiselno kakšnemu razvijalcu dodeliti kakšno drugo nalogo.

Najtežjo nalogo pri dnevnih sestankih ima skrbnik metodologije, saj mora biti strikten glede pravil Scrum-a, ne sme pa prestopiti meje in sestanek spremeniti v pogovore, ki služijo samo njemu. Pomembno je tudi, da se ne postavlja preveč vprašanj glede časovnih zahtevnosti posameznih nalog, saj namen sestanka ni v planiranju ali operiranju z številkami. Ocenjevanje je seveda tudi potrebno, vendar ga je potrebno opraviti ločeno.

3.4. Scrum izdelki

3.4.1. Seznam zahtev

Nabor vseh uporabniških zahtev izdelka, ki ga razvija Scrum skupina, je zbran na seznamu zahtev (ang. Product Backlog). Sestavljen je iz različnih vnosov, ki predstavljajo lastnosti, funkcionalnosti, tehnologije, popravke, izboljšave itd., skratka vse, kar potrebujemo, da razvijemo in izdamo nov izdelek. Kljub temu, da je to eden glavnih dokumentov razvoja, pa ne gre za statičen dokument. Sicer se ga oblikujena začetku projekta, a se ga tekom razvoja vseskozi dopolnjuje in spreminja. Scrum namreč zahteva, da se na začetku identificira le tiste zahteve, ki so najbolj očitne in najbolj razumljive. Navadno je zapisov na seznam zahtev za prvi razvojni cikel več kot dovolj. Ostale vnose se doda ali razčleni kasneje, ko imamo več oprijemljivih podatkov in boljšo sliko izdelka [4].

Posamezen vnos na seznamu je sestavljen iz opisa ali vsebine, njegove ocene ter prioritete vnosa, po kateri je seznam tudi razvrščen. Prioriteto se določi na podlagi tveganja pri razvoju, vrednosti, ki jo ima vnos za uporabnika, ter ocene, kako nujna je realizacija vnosa. Za določanje atributov vnosa obstaja več tehnik, praksa v zadnjih letih pa je pokazala, da so za posamezne vnose v seznamu zahtev najbolj primerne uporabniške zgodbe. To je opis funkcionalnosti ali lastnosti, ki ima za naročnika neko uporabno vrednost. Navadno jo sestavi kar uporabnik sam. Podrobnejši opis uporabniških zgodb je podan v naslednjem poglavju.

Za seznam zahtev je v celoti zadolžen skrbnik izdelka. On postavi prioritete, skrbi za vsebino in dosegljivost seznama. Seznam zahtev je dinamičen in se razvija skupaj s izdelkom in okoljem, v katerem se ga uporablja. Višjo prioriteto kot ima vnos, večjo vrednost ima zapis za uporabnika, več informacij imamo na voljo, manj je neznank, ker se je o tej temi več razmišljalo, in tako dalje. Za zapise, ki so niže na seznamu, seveda velja obratno. Primer seznama zahtev je prikazan na Sliki 3.2.

Item ID	Bug ID	Summary	Rank	Category	Accomplished?
1		Search other hobby sites: bring back results in a list	1	Search	X
2		Search hobby content & bring back results in a list	3	Search	X
3		Show direct display of hobby content in results	14	Search	
4		Use variety of methods for relevancy	4	Search	X
5		Keep track of queries for buckets	37	Search	X
6		Include user-generated content (message boards) in index	19	Search	X
7		Include Ads on search	16	Search	X
8		Include sites to index	2	Search	X
9		Search infrastructure/ops/machines		Search	X
10		"Did you mean?" support for misspellings, etc.	18	Search	X
11		Media for Message boards	23	Boards	
12		View/post to boards	6	Boards	X
13		View list of forums		Boards	X
14		See a list of threads on this board	7	Boards	X
15		Show different views of threads	40	Boards	X

Slika 3.2: Primer seznama zahtev

3.4.2. Seznam nalog

Za vsak cikel se določi vnose s seznama zahtev, ki se jih bo implementiralo. Zatem se jih razčleni na naloge, ki so potrebne, da se doseže cilj cikla. Naloge so zbrane na seznamu nalog (ang. Sprint Backlog), bile pa naj tako obsežne, jih je mogoče opraviti v enem delovnem dnevu [4]. Razvojna skupina lahko spreminja seznam nalog, ne pa začetne vsebine. Ko razdeljujemo določene naloge, se lahko pojavijo novi pogledi na vsebino in razvojna skupina lahko razširja tako seznam zahtev kot seznam nalog. Spreminjanje seznama nalog je dovoljeno samo razvojni skupini. V primeru, da razvojna skupina dokonča vse uporabniške zgodbe, lahko z sodelovanjem skrbnika izdelka na seznam nalog doda nov vnos s seznama zahtev.

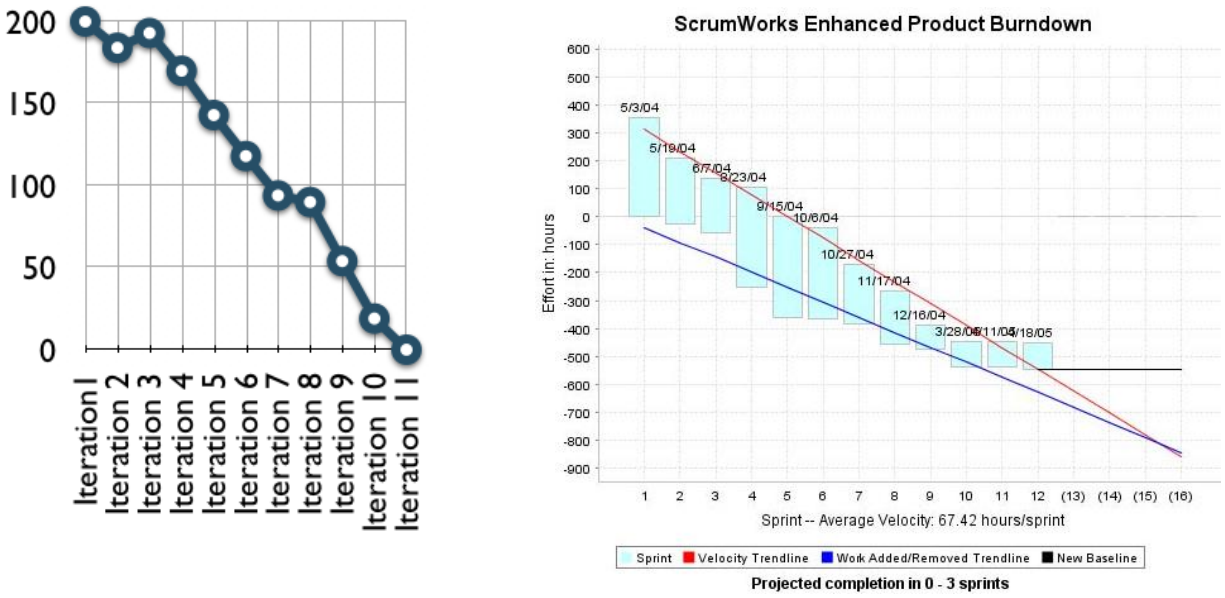
3.4.3. Graf preostalega dela

Prikaz oziroma graf preostalega dela (ang. Burndown chart) je pomemben del Scrum metodologije. Na njem prikazujemo, kako se s časom spreminja (praviloma zmanjšuje) obseg preostalega dela na projektu. Graf je dvodimenzionalen; navpična lestvica prikazuje obseg preostalega dela, vodoravna pa predstavlja čas, navadno v dnevih. Količina dela se sicer lahko izraža v različnih količinah, običajno enota pa so točke uporabniških zgodb. Časovni interval, ki ga spremljamo, je lahko vezan na posamezen cikel, na mejnik (če več ciklov združujemo v intervale, opredeljene z mejniki) ali na celoten projekt. V praksi so najpogosteje uporabljeni

grafi, ki spremljajo potek cikla. Na slikah 3.3 in 3.4 sta prikazana dva primera grafov preostalega dela.

Količina dela, ki nam je še ostalo do konca zastavljenega časa, je enostavno seštevek vseh točk uporabniških zgodb, ki še niso dokončane. Ko imamo na voljo že nekaj izmerjenih podatkov, lahko z grafa razberemo določen trend. Na graf se lahko doda tudi trend linijo, ki je navadno je v prvih dveh do treh ciklih netočna, razen če je ekipa že delala skupaj na podobnih projektih. Včasih se zgodi, da se zaradi dodatnega dela (projektu se dodaja več dela, kot se ga uspe sproti opraviti) linija popolnoma vzravna ali celo preide rahlo navzgor. Kljub vsemu pa se v večini primerov delo zmanjšuje.

Graf preostalega dela predstavlja dober pregled nad tem, ali smo na poti k zastavljenim ciljem, kako se nek trend odraža v našem delu, dobimo lahko ocene za naslednje cikle in podobno. Na podlagi spremljanja spreminjanja trenda in dela lahko tudi z drugačne perspektive gledamo na vzroke, ki so v projektu pomenili določeno zaustavitev tempa. Najbolj pogosta podatka na grafu sta seveda spreminjanje obsega dela s časom in trend določenega obdobja. Poleg prikaza navedenih podatkov nam različna orodja za podporo Scrum razvojnega procesa omogočajo prikaz nekaterih drugih grafov. Podrobnosti in primerjave bodo zajete v poglavju, namenjenem opisu Scrum orodij.



Sliki 3.3 in 3.4: Prikaz različnih variant grafa preostalega dela

4. Scrum in specifikacije zahtev ter planiranje

4.1. Uporabniške zgodbe

Uporabniška zgodba opisuje funkcionalnost, ki ima neko uporabno vrednost za uporabnika oziroma naročnika. V bistvu gre za relativno kratek opis določene uporabniške zahteve na visokem nivoju (ang. High level), ki je razumljiva tako naročniku kot razvijalcem (ne gre za tehnični jezik). Navadno so sestavljene iz naslednjih delov [2]:

- Opis zgodbe, ki se ga uporablja za načrtovanje in kot neke vrste opomnik.
- Opombe, ki služijo za to, da razjasnijo podrobnosti.
- Testi, ki jih uporabljamo za ugotavljanje, kdaj je razvoj uporabniške zgodbe zaključen. Služijo tudi za nadaljnje dokumentiranje podrobnosti.

Ne glede na tehniko agilnega načrtovanja, ki se jo uporabi (Scrum, XP), se pri razvoju z uporabniškimi zgodbami vselej zahteva, da je uporabnik prisoten pri celotnem razvoju od začetka do konca. To je v nasprotju z klasičnimi tehnikami, kjer je navadno uporabnik prisoten na začetku, pri zajemu zahtev, potem pa skoraj izgine iz projekta. Njegova prisotnost v drugih fazah ni tako nujna.

Uporabniki, ki so zainteresirani za uporabo nove programske opreme, morajo vedeti, da se od njih zahteva zelo aktivno vlogo pri pisanju uporabniških zgodb (to vlogo navadno prevzame skrbnik izdelka, ki je nekakšen predstavnik vseh uporabnikov ali strank). Navadno se uporabniške zgodbe razdeli glede na različne uporabniške vloge v sistemu, ki ga razvijamo.

Zgodbe so pogosto napisane na delavnicah ali sestankih, ki se jih lahko organizira kadarkoli med projektom. Ne glede na to, kdaj so bile napisane, pa se novo napisanih zgodb ne sme kar dodajati k zahtevam cikla, temveč se jih lahko samo definira. Za dodajanje in razvrščanje je potrebno imeti posebej določen sestanek (sestanek za izdelavo plana naslednje iteracije).

Na delavnicah, namenjenih zapisu uporabniških zgodb, ob pogovorih nastanejo določeni zapisi v obliki opomb, ki služijo kot dokumentacija nadaljnjih podrobnosti. Kljub temu pa se v času, ki je določen za pogovor o zgodbah, navadno ne da odgovoriti na vsa vprašanja in razkriti vseh podrobnosti. Zato se nadaljnje pogovore opravi v času razvoja na za to namenjenih sestankih ali v obliki neformalnih pogovorov s skrbnikom izdelka. Čas sestanka in pogovora mora biti omejen. Pogovor o detajlih in opombe, ki nastanejo iz tega, niso zavezujoče; služijo bolj kot opomnik. Pomemben je pogovor, ki razkrije zahtevane podrobnosti.

Velikost zgodbe se šteje za primerno, ko jo je mogoče razviti v intervalu velikosti od pol dneva do dveh tednov. Navadno je sicer bolje imeti več zgodb, vendar je včasih potrebno posegati tudi po velikih zgodbah. Takim zgodbam pravimo epi.

Epi so zanimivi predvsem v situaciji, ko določen del še ni definiran ali poznan in bo razvit šele enkrat v naslednjih ciklih, vendar bi vseeno radi vsaj približno zajeli tudi te zahteve. To naredimo tako, da vse te nejasnosti enostavno sestavimo v večjo zgodbo, ji damo maksimalno oceno in jo uvrstimo v začetni plan. Postopoma zadeve razbijemo in dobimo boljši vpogled v to, kaj je potrebno za rešitev problema in koliko časa bomo za to potrebovali.

Uporabniki navadno uporabniške zgodbe pišejo sami oziroma to za njih naredi skrbnik izdelka. To je pomembno zaradi tega, ker mora biti zgodba napisana v poslovnem jeziku in ne v tehničnem, pa tudi zato, ker uporabniki sami najbolje vedo, kaj je za njih najpomembnejše. Drži pa tudi, da uporabniki velikokrat potrebujejo pomoč pri pisanju zgodb.

4.2. Uporabniške zgodbe in Scrum

Najbolj eleganten način, da uporabniške zgodbe vključimo v Scrum je z vnosi na seznam zahtev. Vsaka zahteva je tako predstavljena kot ena uporabniška zgodba [2]. Namesto da dopustimo, da je seznam poln zapisov o popravkih, opisov novih lastnosti ter zadev, ki jih je treba raziskati, je seznam zahtev omejen samo na uporabniške zgodbe. Vsaka zgodba na seznamu zahtev mora predstavljati neko uporabno vrednost za uporabnika ali skrbnika izdelka. Za slednjega postane tako veliko lažje postavljati prioritete in ugotoviti, katere naloge so prednostne za izvedbo projekta, saj so vsi zapisi njemu razumljivi. Tudi pri tem načinu ni potrebno, da skrbnik izdelka definira vse zahteve vnaprej.

Med sestankom za izdelavo plana naslednje iteracije se skrbnik izdelka in razvojna skupina pogovorijo o najpomembnejših zapisih na seznamu zahtev (v tem primeru so zapisi uporabniške zgodbe). Skupina nato sestavi seznam nalog, ki je sestavljen iz vnosov, ki jih bo razvojna skupina implementirala v naslednji iteraciji. Vse zapise tudi razbije na manjše naloge. V kolikor uporabljamo uporabniške zgodbe, dosežemo to, da ima za uporabnika vsak zapis določeno težo. Tako so sestanki lažje vodljivi in razvojna skupina lažje razloži svoje namere ostalim, ne-tehničnim članom. Vsebina sestanka je razumljiva vsem in potek sestanka je tako hitrejši. V praksi se je pokazalo tudi, da je uporabniške zgodbe lažje in bolj naravno razdeliti na posamezne naloge, kar se izkaže kot velika prednost med samim razvojem, ko dopolnjujemo seznam nalog.

Tudi sestanek za pregled rezultatov dela ter dnevni sestanki so lažje izvedljivi, če uporabljamo uporabniške zgodbe. Razvojna skupina ima namreč tako boljšo predstavo o tem, kaj je končni cilj nalog, s katerimi se ukvarja. Lažje se tudi ugotovi, kaj je bilo že narejeno oziroma dokončano.

Kljub temu, da uporabniške zgodbe niso nastale v sklopu metodologije Scrum, pa se je njihova uporaba v sklopu te metodologije močno razširila. Že sama uporaba uporabniških zgodb predstavlja za naročnika dodano vrednost, ki pa je še večja, kadar jo uporabimo v kombinaciji s katero izmed agilnih metodologij. Tudi programska orodja, ki služijo za podporo Scrum-u, skoraj brez izjem podpirajo vnose v obliki uporabniških zgodb. Iz teh razlogov sem tudi sam v kasnejšem opisu in sestavi testnega primera vseskozi uporabljal Scrum v kombinaciji z uporabniškimi zgodbami.

4.3. Uporabniške zgodbe in planiranje

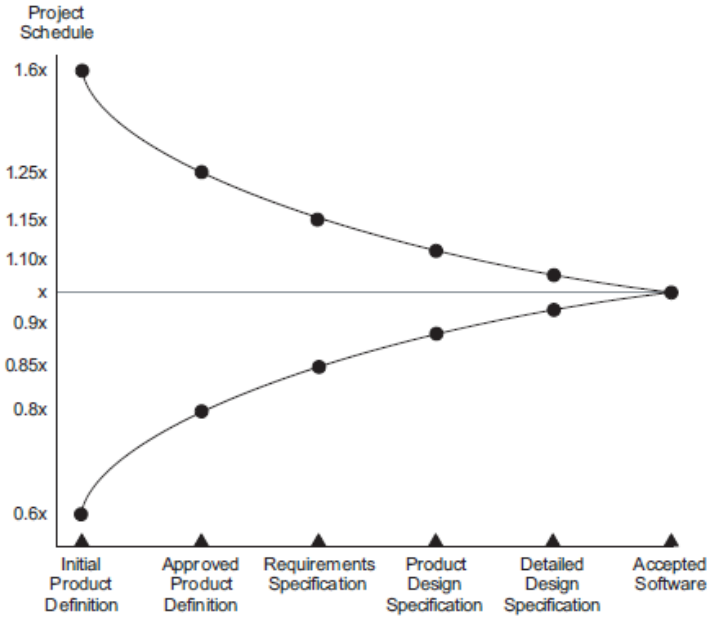
4.3.1. Uvod

Kljub temu, da programskega inženirstva ne moremo primerjati z ostalimi inženirskimi disciplinami, imata tudi pri njem planiranje, ocenjevanje ter zajem zahtev ne glede na velikost projekta kritičen pomen. Povsod, kjer se srečamo z investicijami, se namreč zahteva nekakšno podlago, ki temelji na načrtih in ocenah. Popolnoma možno je, da neko podjetje sprejme projekt, za katerega oceni, da bo gotov čez pol leta in bo družbo stal milijon denarnih enot ter zavrne projekt, ki bi trajal dve leti in stal podjetje štiri milijone.

Planiranje je zelo pomembno pri razporejanju kadrov, ugotavljanju, ali bodo vse zahtevane funkcionalnosti implementirane v predvidenem času, spremljanju, kako se naš napredek ujema z začetnimi ocenami, ugotavljanju zamude, pomoči pri reorganizaciji zaposlenih itd. Brez planiranja odpiramo vrata številnim težavam.

Planiranje pa kljub pomembnosti predstavlja eno težjih nalog. Največji problem je, da se pri tem velikokrat uštejemo. Zato se razvojne ekipe pogosto ujamejo v dveh ekstremih: planiranju se popolnoma izognejo ali pa v planiranje vložijo toliko truda, da začnejo zmotno razmišljati, da so njihovi zaključki definitivno točni. Ekipa, ki nima nikakršnega plana, ne zna odgovoriti niti na najbolj osnovna vprašanja, kot so: »Kdaj bo projekt zaključen?« ali »Ali se lahko verzijo izda junija?«. Ekipa, ki v začetno fazo vloži preveč truda, pa je prepričana, da bodo odstopanja od načrta zelo majhna. Res je, da je so v to vložili veliko več energije in časa, vendar to na žalost še ne pomeni, da so njihovi plani nujno boljši.

Problemi pri začetni fazi razvoja se niso rodili ob uveljavljanju novih programskih jezikov, temveč industrijo programske opreme spremljajo že od samih začetkov. Že leta 1981 je Barry Boehm narisal prvo verzijo grafa, ki je kasneje postal znan kot stožec nejasnosti (Slika 4.1) [3].



Slika 4.1: Stožec nejasnosti

Graf prikazuje različne mero nejasnosti v različnih fazah projekta. Kljub temu, da je bil razvit za zaporedni oziroma slapovni razvoj, so njegove ocene uporabne tudi v drugih metodologijah, čeprav namesto faz uporabljamo iteracije. Iz grafa lahko razberemo, da je mogoče, da projekt traja nekje v intervalu od 60% pa do 160% glede na začetno oceno. To recimo pomeni, da se projekt, za katerega smo planirali, da bo dokončan v 20 tednih, lahko zaključi v intervalu od 12 pa do 32 tednov.

Zakaj se potem sploh ubadamo s tem, če pa je ocenjevanje in planiranje tako zahtevna naloga, hkrati pa je v začetnih fazah projekta skoraj nemogoče dobiti točne vrednosti? Prvi razlog je definitivno ta, da organizacija, v kateri delamo, zahteva od nas določene odgovore. Časovne in druge ocene so za podjetje pomembne zaradi vrste aktivnosti, ki spremljajo osnovni razvoj projekta in so za končni uspeh projekta prav tako pomembne (oglaševanje, aktivnosti, povezane z izdajo verzije, izobraževanje uporabnikov, produkcija itd). Poleg tega pa obstaja za vsem tem še veliko globlji razlog za to, da se napornega dela vseeno lotimo.

Dobro planiranje ne predstavlja samo določitev končnega roka izvedbe ali urnika izvajanja nalog. Planiranje je predvsem določanje vrednosti, še posebej, če upoštevamo planiranje kot nedovršeno aktivnost. Najti poskušamo najboljšo možno rešitev: kaj moramo narediti, kaj je naš izdelek? Potrebno je upoštevati vse komponente in odgovor na navedena vprašanja poiskati v več iteracijah.

Dobro planiranje prispeva k:

- Zmanjševanju tveganja
- Zmanjševanju negotovosti
- Podpori boljšemu odločanju

- Vzpostavljanju zaupanja
- Oskrbi z informacijami

4.3.2. Zajem zahtev

Dobra definicija uporabniških zahtev je hkrati eden pomembnejših in eden najzahtevnejših členov pri razvoju programske opreme. Gre predvsem za komunikacijski problem. Na eni strani imamo skupino ljudi, ki na izdelek gledajo predvsem s poslovnega vidika, na drugi pa tehnično ekipo oziroma razvijalce. Prvo skupino lahko predstavljajo končni uporabniki, stranke, poznavalci poslovne domene, analitiki in drugi. Ne glede na to, kdo je v tej skupini, pa je uspešnost projekta odvisna od zmožnosti komuniciranja med stranema in kvalitete informacij, ki jih vsi vpleteni posredujejo drug drugemu.

Pomembna je tudi uravnoteženost komunikacije, saj se kvaliteta končnega izdelka lahko zmanjša, v kolikor ena stran prevzame vodilno vlogo. V kolikor je močnejša poslovna stran, so na prvem mestu predvsem roki ter izpolnjevanje specifikacije zahtev. Običajno je le malo posluha za to, ali razvijalci dejansko razumejo zahteve in ali so jih v določenem roku sposobni implementirati. Po drugi strani pa se kaj hitro lahko zgodi, da vsakdanji jezik zamenja tehnični žargon, če vodilno vlogo prevzame tehnična ekipa.

Zaradi omenjenih razlogov potrebujemo način, s katerim se razlike premostijo in problemi z administracijo virov postanejo skupni. Tako se lahko izognemo marsikateri nevšečnosti. V nasprotnem primeru se lahko znajdemo v situaciji, ko funkcionalnosti niso zahtevane kvalitete ali pa je okrnjen začetni seznam zahtev. Uporabnost programske opreme se tako zmanjša.

Navadno ni mogoče popolnoma napovedati poteka razvoja programske opreme. Ko uporabniki vidijo prve verzije izdelka, dobijo nove ideje in njihova pričakovanja se spremenijo. Drugi razlog pa je v neoprijemljivosti programske opreme. Razvijalci tako zelo težko ocenijo, koliko dela bo potrebnega, saj je končni izdelek zelo težko definirati. Zaradi teh in podobnih razlogov ne moremo kar narisati enostavnega PERT diagrama in upati, da smo zajeli vse zahteve.

Kaj torej narediti? Odločitve sprejemamo na podlagi informacij, s katerimi razpolagamo. Odločitve moramo čim bolj pogosto ponovno pretehtati oziroma odločitve spreminjati na podlagi novih informacij. Namesto da razgrnemo široko paleto odločitev takoj na začetku projekta, sprejemanje odločitev razporedimo preko celotnega poteka razvoja. Za to potrebujemo sistem, ki nas hitro in pogosto oskrbuje z informacijami.

4.3.3. Ocenjevanje zahtevnosti

Zahtevnost oziroma velikost uporabniških zgodb ocenimo s številom točk (ang. Story points). To naredimo tako, da vsaki uporabniški zgodbi dodelimo neko številčno oceno. Dodeljena ocena predstavlja njeno velikost zahtevnosti. Bolj pomembna kot njena absolutna vrednost pa je njena relativna vrednost. Na primer: v kolikor se določeni zgodbi določi vrednost 4, neki drugi pa 2, potem pomeni, da je prva dvakrat večja od druge. Obstaja sicer več tehnik dodeljevanja ocen, ne obstaja pa neko posebej predpisano pravilo ali točno določena formula, kako priti do ocene. V oceni točke naj bi bila upoštevana količina truda za razvoj, kompleksnost, tveganje ob razvoju itd. [3].

Za začetek sta priporočena dva osnovna načina. Tistim nalogam, za katere predvidevamo, da se bomo z njimi ukvarjali najmanj časa, dodelimo najmanjšo vrednost (npr. 1), ostalim povečujemo ocene glede na primerjavo z najmanjšimi. Druga možnost je, da srednje velikim zgodbam dodelimo srednje vrednosti (npr. če imamo lestvico od 1 do 10, dobi vrednost 5), vsem ostalim pa zmanjšujemo ali povečujemo vrednost glede na primerjavo velikosti.

4.3.4. Hitrost razvoja

Ocenjevanje uporabniških zgodb s točkami, ki nimajo enot, dobi smisel le, če poleg njih uvedemo še drugo mero ocenjevanja, ki ji pravimo hitrost razvoja (ang. Velocity). To je skupna vsota vseh točk uporabniških zgodb, ki jih skupina lahko dokonča v eni iteraciji. V kolikor ekipa dokonča tri zgodbe z oceno 5, to pomeni, da je hitrost razvoja v tej iteraciji 15. Pomemben pa je tudi podatek, da bo skupina, ki je imela hitrost 15 v prejšnji iteraciji, imela po vseh verjetnosti hitrost 15 tudi v naslednji [3].

Tako lahko, če povežemo točke, ki jih dodelimo uporabniških zgodbam, in podatke o hitrosti razvoja oz. meri napredovanja, hitro sestavimo urnik projekta. V kolikor poznamo skupno število točk za projekt in hitrost razvoja, lahko izračunamo število iteracij. V kolikor poznamo hitrost razvoja za neko iteracijo, lahko povemo, kaj vse bomo sposobni implementirati do konca cikla.

Obstaja več načinov za določitev hitrosti razvoja:

- Uporabimo zgodovinske podatke
- Izmerimo na iteraciji
- Naredimo predpostavko

Najbolje je seveda uporabiti že znane podatke. Problem nastane, ker je zelo težko najti dva projekta, ki bi si bila podobna, kaj šele enaka. Preden sklenemo, da bomo projekte

obravnavali na podoben način, je potrebno premisliti sledeče: ali so skupine enake (zasedba, število), ali je uporabljena enaka tehnologija, ali razvijamo z enakimi orodji, kakšna je primerjava strank, itd. Velikokrat bomo prišli do zaključka, da so se razmere občutno spremenile. Ali bomo uporabili podatke iz prejšnjega projekta ali ne in v kakšni meri, kar naenkrat postane težka odločitev.

V vsakem primeru pa je bolje imeti vsaj okvirno ali grobo informacijo kot nobene. Že po nekaj iteracijah pa postanejo znana odstopanja in tudi na podlagi teh lahko bolje opredelimo hitrost razvoja. Nasploh je najbolje, da v prvih nekaj iteracijah hitrosti razvoja ne napovedujemo, temveč ga določimo šele, ko je slika bolj jasna. Žal se tega ne da vedno storiti in velikokrat je potrebno dobesedno iz ničesar napovedati, koliko bo narejenega in do kdaj. Poleg tega pa so navadno odstopanja pri začetnih iteracijah večja kot sicer.

Ne glede na način ocenjevanja pa je vedno dobro, da hitrost razvoja podamo kot interval in ne kot fiksno vrednost. To pomeni, da morebitno izračunano ali pa ocenjeno fiksno vrednost razširimo v interval. Na primer: iz ocene 40 naredimo interval 30 – 50. Četudi ocena ni tako natančna, pa nam dobljena informacija vseeno pomeni dober približek. Bolje je imeti ohlapno, a pravilno oceno, kot pa natančno, a napačno.

Da so podatki res relativno točni, lahko pričakujemo le v primeru, ko ekipa zaključi z izdajo verzije in se pod podobnimi pogoji premakne na naslednjo. V vseh ostalih primerih je vsaj do določene mere potrebno poleg ocenjevanja vsaj malo ugibanja. Razširjanje na interval smo že omenili, poslužujemo pa se lahko tudi drugih tehnik. Vzamemo lahko najslabšo in najboljšo vrednost hitrosti uresničitve na prejšnjem projektu ali v zadnjem obdobju; možno je vzeti tudi neko srednjo vrednost in jo spremeniti v interval glede na stožec nejasnosti. Obstaja še nekaj tehnik, kljub vsemu pa je potrebno včasih enostavno priti do številke brez vsakršnega konkretnega podatka. V tem primeru se priporoča zgodbe razčleniti na naloge in te oceniti v urah. Seštevek vseh zgodb, ki jih uspemo uresničiti glede na načrtovane ure v določenem obdobju, predstavlja našo oceno.

V vsakem primeru je potrebno hitrost razvoja natančno spremljati. Le tako namreč lahko že takoj ob začetku projekta opazimo napake v planiranju. V kolikor smo svoje napovedi in urnik naredili na predpostavki, da bo hitrost razvoja 25 in se v prvih iteracijah izkaže, da je 20, je treba temu primerno prilagoditi plane.

Ne glede na to, kakšnih tehnik se poslužujemo za ocenjevanje, je znano dejstvo, da smo vedno učinkovitejši, v kolikor ocenjujemo stvari istega reda. Od tu izvira ideja, da se možne ocene poda z določenim naborom ali lestvico. Pogosti izbiri sta recimo Fibonaccijevo zaporedje: 1,2,3,5,8,13,... ali pa zaporedje, pri katerem je naslednik dvakratnik predhodnika: 1,2,4,8,16,... Te nelinearne lestvice so se izkazale za dobre predvsem zato, ker dobro ponazarjajo višji nivo nejasnosti ter tveganja v obsežnejših zgodbah. Dobra je tudi lestvica 20, 30, 40, 50... Primer slabe lestvice pa je recimo interval od 1 do 40, pri kateri eno zgodbo ocenimo s 34, drugo pa s 35. To nam namreč daje lažni občutek natančnosti, ki je v nobenem primeru nismo sposobni udejanjiti.

4.3.5. Ocenjevanje

Ena večjih dilem pri načrtovanju je seveda, kako sploh priti do ocene. V literaturi so navedeni trije osnovni načini [3]:

- Mnenje strokovnjaka
- Analogija
- Deljenje

Mnenje strokovnjaka

Mnenje strokovnjaka je dobrodošlo na vseh področjih. Tudi pri ocenjevanju zahtevnosti in velikosti programske opreme je tako. Ko potrebujemo oceno zahtevnosti, enostavno vprašamo za mnenje nekoga, ki se na to spozna, on pa odgovori na podlagi svojih izkušenj, znanja, občutkov in intuicije. Ta način je sicer bolj uporaben pri standardnih načinih planiranja kot pri agilnih, saj so agilne tehnologije usmerjene v oceno uporabniških zgodb, ki zahtevajo znanja več ljudi. Pri standardnih pristopih je ocena namenjena posameznim nalogam, ki jih navadno opravi ali pa vsaj nadzoruje en razvijalec. Pri agilnih metodah je to težje izvedljivo. Največja prednost takega pristopa je v tem, da ocenjevanje ne traja dolgo, saj strokovnjak navadno hitro pride do ocene.

Analogija

Ocenjevanje z analogijo ali primerjavo sloni na enostavni logiki primerjave dveh ali več zgodb med sabo. V kolikor je prva zgodba obsežnejša od druge, ji enostavno dodelimo razliki primerno višjo oceno. Več študij je namreč pokazalo, da je človek boljši pri relativnem ocenjevanju kot pa pri absolutnem. Pri tej tehniki si pomagamo tudi s tako imenovano triangulacijo. Recimo, da ocenjujemo zgodbo, ki ji želimo dati oceno 5. V tem primeru pred zaključkom ocenjevanja preverimo, ali je zgodba obsežnejša od tistih, ki imajo oceno 3 in manjša od tistih, ki imajo oceno 8.

Deljenje

Bistvo tehnike deljenja je v tem, da določeno zgodbo razdelimo na manjše dele, ki jih potem lažje ocenimo. Izkaže se namreč, da veliko bolje ocenimo zgodbe manjšega obsega, pa tudi primerjava med zgodbami je lažja, ko so zgodbe istega ali vsaj podobnega ranga. Recimo, da imamo zgodbo, ki se jo zaključi v 2 dneh in zgodbo, ki zahteva 100 dni. Primerjava med njima je skoraj nemogoča, poleg tega pa obstaja možnost, da smo se pri večji zgodbi močno ušteli z oceno. Rešitev je v tem, da večje zgodbe enostavno razbijemo na manjše dele.

4.4. Poker planiranja

Tehnika, ki jo mnogi avtorji priporočajo in je nekakšna zabavna kombinacija vsega zgoraj naštetega, je poker planiranja (Planning poker) [3]. Pri tej tehniki poskušajo razvijalci na podoben način, kot potekajo stave pri istoimenski igri s kartami, priti do neke končne ocene s, katero se vsi strinjajo. Seveda so pri tem pomembni argumenti in ne sreča.

Za izvedbo pokra planiranja potrebujemo prisotnost celotne razvojne ekipe in dogovor o načinu razdeljevanja ocen. Za optimalno izvedbo naj bi skupina ocenjevalcev štela nekje med 3 in 10, veljavne ocene pa so tiste, ki spadajo v nabor že prej dogovorjenih ocen. Poleg sestave lestvice ocen nas zanima tudi najvišja možna vrednost.

Igra poteka tako, da se vsakemu od razvijalcev da kup vnaprej pripravljenih kart z veljavnimi ocenami. Moderator nato prebere opis uporabniške zgodbe. Navadno sledi razprava oziroma postavljanje vprašanj skrbniku izdelka. Naloga ni natančno analizirati zahteve, temveč podati nekaj osnovnih podatkov in razložiti nejasnosti. Skrbnik izdelka navadno opravlja vlogo moderatorja ali povezovalca, v vsakem primeru pa je zraven in odgovarja na vprašanja. Po navadi ne daje ocen.

Ko so odgovori na vsa vprašanja znani, sledi ocenjevanje - vsak član ekipe izbere karto, ki predstavlja njegovo oceno. Pomembno je, da vsak poda oceno na podlagi lastnih občutkov in ne na podlagi ocen drugih članov. To pomeni, da je potrebno ocene razkriti hkrati, oziroma da se karte obrne istočasno. Zelo verjetno je, da se bodo ocene občutno razlikovale. V tem primeru sta tista dva, ki imata najnižjo in najvišjo oceno, svojo izbiro dolžna tudi utemeljiti. Pri tem je pomembno, da vse skupaj ne preide v borbo argumentov, temveč da se ostali člani predvsem seznanijo z načinom razmišljanja obeh. S tem se želi doseči, da ekipa razgrne okvirne poglede glede načina implementacije in se o tem vsaj približno zedini. Vedno se lahko zgodi, da kateri od razvijalcev ni upošteval vseh faktorjev ali pa da obstaja lažji način implementacije, na katerega ostali niso pomislili.

Skupina ima na voljo nekaj minut, da se o tem pogovori, potem pa se glasovanje ponovi. Da vse skupaj ostane v okvirih dovoljenega, skrbi moderator. Med pogovori si navadno moderator tudi napiše kakšno opombo, ki gre k opombam v uporabniški zgodbi. To je lahko v veliko pomoč pri kasnejšem kodiranju ali testiranju. Tudi drugo in morebitna nadaljnja glasovanja se izvedejo po istem že opisanem postopku. V večini primerov bodo rezultati pri drugi ponovitvi že močno konvergirali. V nasprotnem primeru se postopek ponavlja, dokler ne dosežemo želenega rezultata. Navadno glasovanje ne traja več kot tri kroge.

Cilj tehnike je v tem, da se ocenjevalci zedinijo glede določenega rezultata. Pri tem ni nujno, da morajo vsi oddati karte s točno enakimi odgovori. Dovolj je že, da moderator opazi, da so ocene dovolj blizu, tako da lahko kar sam predlaga končno oceno. Seveda se morajo s tem vsi strinjati. Na primer: oddane ocene so: 5, 5, 5, 3; moderator vpraša ocenjevalca, ki je dal najnižjo oceno, ali bi se strinjal tudi z oceno 5. Če privoli, ta ocena obvelja.

Pogovor med člani skupine ob ocenjevanju je ključni del pokra planiranja, kljub temu, pa je treba paziti, da ne preseže določenih časovnih okvirov. Tudi pri časovnem omejevanju različni avtorji predlagajo različne tehnike. Mogoče najbolj zanimiv je način časovnega omejevanja s peščeno uro. Na sredino mize se postavi dvominutno peščeno uro in vsak, ki bi rad začel argumentacijo, jo obrne. Zagovorniki tehnike zagotavljajo, da navadno ni potrebno ure obrniti več kot trikrat.

Poker planiranja naj bi trajal nekje med eno in tremi urami. V kolikor je zgodb več, je bolje, da se vse skupaj razdeli v več sestankov. Skupine navadno igrajo poker na začetku, torej preden se projekt v resnici začne, in pa seveda ob koncu iteracij, ko je treba oceniti vse zgodbe, ki so nastale med to iteracijo. Poker planiranja je mogoče igrati tudi v manjših skupinah oziroma podmnožicah. Skupino se razdeli v več ekip, od katerih mora vsaka imeti vsaj tri člane. Med skupine se razdeli uporabniške zgodbe tako, da vsaka dobi svoj nabor zgodb, ki jih mora oceniti. To je uporabno predvsem, kadar je potrebno oceniti res veliko število uporabniških zgodb.

Zakaj poker planiranja sploh deluje? Prvi razlog je nedvomno v tem, da igra doprinese k temu, da se pri ocenjevanju izrazijo mnenja večih strokovnjakov iz različnih segmentov razvoja. Zato so za ocenjevanje tudi najbolj kompetentni. Nasploh naj bi pri ocenjevanju veljalo načelo, da naj ocenjujejo zadeve tisti, ki so najbolj usposobljeni ali kompetentni za razvoj.

Diskusija, ki se vname med ocenjevanjem s to tehniko, precej pripomore k bolj točni oceni, še posebej pri zadevah, ki imajo veliko mero nedoločenosti. To, da je potrebno svojo odločitev tudi utemeljiti, pa pripomore k temu, da se ocena bolje prilagaja manjkajočim informacijam. Uporabniške zgodbe so namreč včasih namerno zelo splošne.

Poleg vsega so študije pokazale, da iskanje povprečja med posameznimi ocenami in skupinsko ocenjevanje enostavno prinesejo najboljše ocene. Poker planiranja v bistvu podaja neko mešanico vseh priznanih tehnik ocenjevanja, povrhu vsega pa je še zabaven [3].

5. Predstavitev orodij za vodenje projektov po metodologiji Scrum

5.1. Uvod

Na trgu je prisotnih kar nekaj orodij, ki omogočajo podporo in vodenje projektov po kopitu agilnih metodologij in lažje spremljanje celotnega razvojnega procesa. Na voljo imamo široko paleto možnosti, tako pri tistih bolj enostavnih orodjih kot pri tistih, ki predstavljajo že pravo ogrodje vodenja projektov. Kljub temu pa je zanimivo dejstvo, da sta še vedno najbolj uporabljani orodji za spremljanje razvojnih procesov Microsoft Excel in Microsoft Project [7]. To nakazuje, da so ljudje še vedno bolj domači pri orodjih, ki so bolj enostavna in so jih že navajeni uporabljati. To dejstvo je vplivalo tudi na našo izbiro nabora programov. V tej nalogi smo v ožji izbor vzeli štiri:

- ScrumNinja [12]
- Agilo [11]
- ScrumWorks [14]
- VersionOne [13]

VersionOne in ScrumWorks sta ta trenutek najširše uporabljani orodji za spremljanje in vodenje Scrum procesa [7] [10] [14]. VersionOne je na tržišču že od leta 2002, uporablja pa ga več kot 30.000 razvojnih ekip. ScrumWorks naj bi bil tako intuitiven kot Scrum sam. Obe orodji ponujata veliko število dodatnih možnosti, ki lahko močno pripomorejo k pozitivni izkušnji s Scrum-om. Kljub nekaterim skupnim lastnostim pa vse skupaj implementirata na različne načine in sta kljub obsežnosti še vedno enostavna in pregledna. Agilo je orodje, ki je v zadnjem času poželo veliko zanimanja zaradi svoje prosto dostopne verzije [10], je pa tudi edino, ki je izdelano na evropskih tleh. ScrumNinja spada pod med enostavna orodja, je pa zato bolj pregleden in lažje dostopen.

Vse, kar je bilo potrebno narediti za zagon programov, sem vnesel v virtualni računalnik, kjer je bil nameščen operacijski sistem Microsoft Windows XP. V kolikor je imel program profesionalno verzijo, sem poskušal dobiti licenco za slednjo. Poskusne licence so v vseh primerih veljale vsaj 30 dni. Poleg osnovnega spoznavanja s programi sem na vseh programih izvedel tudi testni primer. Najbolj pozoren sem bil na to, kako se programi obnašajo ob testnem primeru sem postal tudi najbolj pozoren, saj so te lastnosti ob enem tiste, ki podjetja najbolj zanimajo. Celoten testni primer je v prilogi, na naslednjih straneh pa je podan le njegov povzetek .

5.2. Povzetek testnega primera

Za boljšo predstavo o orodjih, ki jih opisujem, in lažjo primerjavo med njimi, sem za potrebe te naloge sestavil testni primer. Kljub temu, da gre za poenostavljen primer, sem se z njim poskušal čim bolj približati realnemu svetu, kjer so spremembe in napake pri poteku projekta na dnevnem redu. Testni primer je sestavljen iz opisa uporabniških zgodb, seznama zahtev, ki se spreminja po ciklih, ter scenarija poteka projekta. Povzetek testnega primera je v nadaljevanju, celoten pa je vključen kot priloga na koncu naloge.

Namišljeni naročnik projekta je turistična agencija, ki potrebuje enostavno namizno aplikacijo za vodenje svojega poslovanja. Agencija potrebuje možnosti pregleda prijav in namestitev, izpisa določenih internih seznamov, urejanja dostopnih pravic ipd. Uporabniške zgodbe so razdeljene med tri vloge:

- Prodajalec
- Računovodja
- Administrator

Prodajalec je tisti, ki ima neposredni stik s strankami. Opravlja lahko naloge pregleda ponudbe, prijavlja bodoče potnike na programe in podobno. Računovodja uporablja aplikacijo v interne namene in opravlja predvsem naloge, ki se tičejo finančnega pregleda ter izpisov internih seznamov. Administrator ureja in spreminja interne podatke ter skrbi za administracijo uporabniških računov.

Testni primer sem izvajal tako, da sem pred začetkom vsakega cikla posodobil seznam zahtev, sestavil seznam nalog za prihajajoči cikel ter nastavil primerno hitrost realizacije. Testni primeri vedno vpeljejo neko predvideno razporeditev na cikle ter začetne ocene uporabniških zgodb, predvidene realizacije in predvidenega konca projekta. Med potekom cikla se spreminja predvsem podatke v povezavi z opravljanjem nalog, vedno pa je potrebno slediti sestavljenemu scenariju, ki vpelje določene probleme in rešitve.

Na začetku projekta je seznam zahtev sledeč:

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
A1	Urejanje podatkov o terminih in namestitvah	13	1
P2	Prijava na izlet	13	2
P4	Pregled in urejanje prijav	21	2
P1	Pregled terminov, namestitev in destinacij	8	3
R1	Pregled in urejanje plačil	13	3
P3	Izpis računa	5	4
A2	Urejanje podatkov o uporabnikih	5	4

Tabela 5.1: Začetni seznam zahtev

Uporabniške zgodbe, ki se jim kratica začne s črko »A«, so namenjene vlogi administrator, zgodbe, ki so namenjene prodajalcu, imajo okrajšavo »P«, zgodbe z okrajšavo »R« pa spadajo pod vlogo računovodje.

Seznam zahtev se tekom projekta spreminja. Doda se nekaj popolnoma novih uporabniških zgodb, predvsem povezanih z raznimi izpisi, ki jih sprva naročnik ni zahteval. Zaradi velike obsežnosti pa se zgodbo P4 razdeli na dve zgodbi. Nasploh se predvidena realizacija in razpored po ciklih skozi celoten projekt vseskozi spreminjata. Pred začetkom projekta je predvidena sledeča:

	Hitrost realizacije	Uporabniške zgodbe
Cikel 1	26	A1, P2
Cikel 2	29	P4, P1
Cikel 3	23	R1, P3, A2

Tabela 5.2: Začetni plan projekta

Najprej optimistično nastavimo predvideno hitrost realizacije cikla na 26 točk in pričakujemo, da jo bomo v drugem ciklu celo malo dvignili. Že takoj v prvem ciklu pa se hitrost zaradi težav, ki nastopijo zaradi komunikacije z naročnikom, zniža na 18. Projekt na koncu traja 5 ciklov. Na koncu projekta lahko ugotovimo, da se realizacija po ciklih spreminja tako:

- Cikel 1: 18 točk
- Cikel 2: 29 točk
- Cikel 3: 21 točk
- Cikel 4: 18 točk
- Cikel 5: 13 točk.

5.3. ScrumNinja

ScrumNinja je program za vodenje projektov po metodologiji Scrum, ki ga je razvilo podjetje Inernaut Design iz ZDA. Zaradi svoje preglednosti in enostavnosti predstavlja dober prvi korak zlasti za razvojne skupine, ki se držijo glavnih načel agilnosti in nimajo potrebe po naprednih in velikokrat zapletenih dodatnih možnostih. Najdemo ga lahko na spletnem naslovu:

http://scrumninja.com/scrum_software.

Obstaja samo ena različica, lahko pa bodisi gostujemo na njihovih strežnikih bodisi si program v celoti prenesemo in ga poganjamo na svojem omrežju. Verzija za prenos deluje le na Linux platformah. Za ekipe do 3 ljudi je brezplačna, sicer pa se zaračunava mesečno glede na število uporabnikov. Cene so različne tudi glede na to, ali gostujemo pri njih ali smo si program prenesli.

Glede na to, da sem programe izvajal na Windows XP platformi, sem izkoristil gostovanje. Vse, kar je potrebno v tem primeru narediti, je pridobiti poskusno licenco in se prijaviti na njihovi spletni strani. Poskusna licenca traja trideset dni.

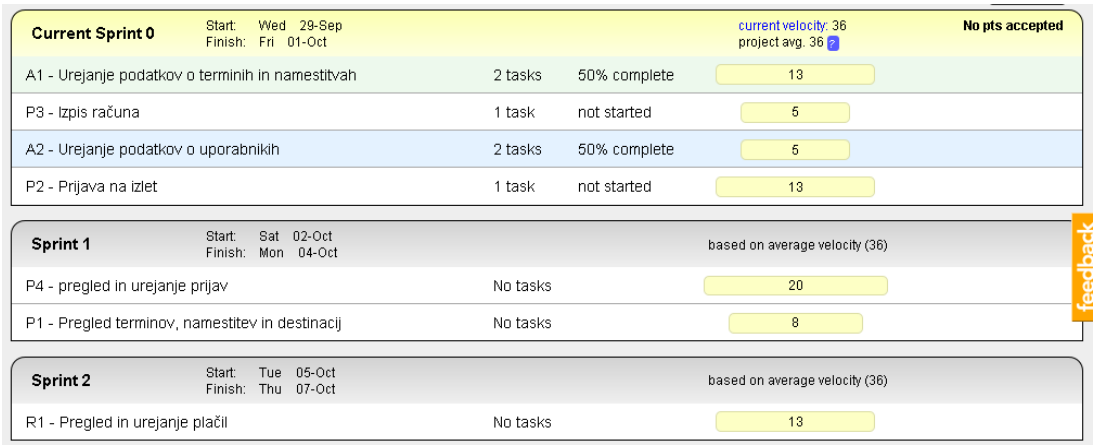
Ne glede na to, ali se odločimo za gostovanje ali za izvajanje svoje aplikacije, se do aplikacije dostopa preko spletnega brskalnika. Projekt analiziramo v treh osnovnih pregledih:

- Seznam zahtev (ang. Backlog)
- Pregled nalog (ang. Card Wall)
- Grafični pregled dela (ang. Burndown).

Seznam zahtev (Slika 5.1) služi kot osnovni pregled projekta. Na njem spremljamo stanja vseh uporabniških zgodb ter pregled njihove razdelitve po ciklih. V bistvu gre pri tem pogledu za kombinacijo seznama zahtev celotnega projekta (ang. Product Backlog) ter seznama nalog (ang. Sprint Backlog). Celoten nabor uporabniških zgodb predstavlja seznam zahtev, njihova razdelitev po ciklih pa seznam nalog oziroma uporabniške zgodbe, ki bodo z razbitjem na posamezne naloge realizirane v določenem ciklu. Za razporejanje uporabniških zgodb v iteracije skrbi kar program sam, upošteva pa navedene prioritete zgodb ter pričakovano hitrost realizacijo v ciklu.

Posebna lestvica vrednosti za določanje prioritet ne obstaja, temveč v ta namen uporabljamo mesto uporabniške zgodbe v seznamu zahtev. Višje kot je zgodba postavljena, višja prioriteta ji pripada. Uporabnik lahko sam poljubno prestavlja uporabniške zgodbe po seznamu, program pa na podlagi točk zahtevnosti uporabniških zgodb in predvidene hitrosti realizacije oblikuje cikle. Predvidena hitrost realizacije se izračuna iz povprečja realizacije že zaključenih ciklov oziroma jo uporabnik lahko poljubno spreminja. Dobra lastnost tega, da program kar sam skrbi za oblikovanje ciklov, je da se vseskozi avtomatsko popravlja plan

projekta. V kolikor se predvidena realizacija zmanjšuje, se temu primerno zmanjšuje tudi obseg cikla. V primeru dodajanja novih zgodb višje prioritete pa se ostale zgodbe premikajo po seznamu navzdol. Tako imamo vedno na voljo sveže informacije o predvidenem poteku in zaključku projekta.



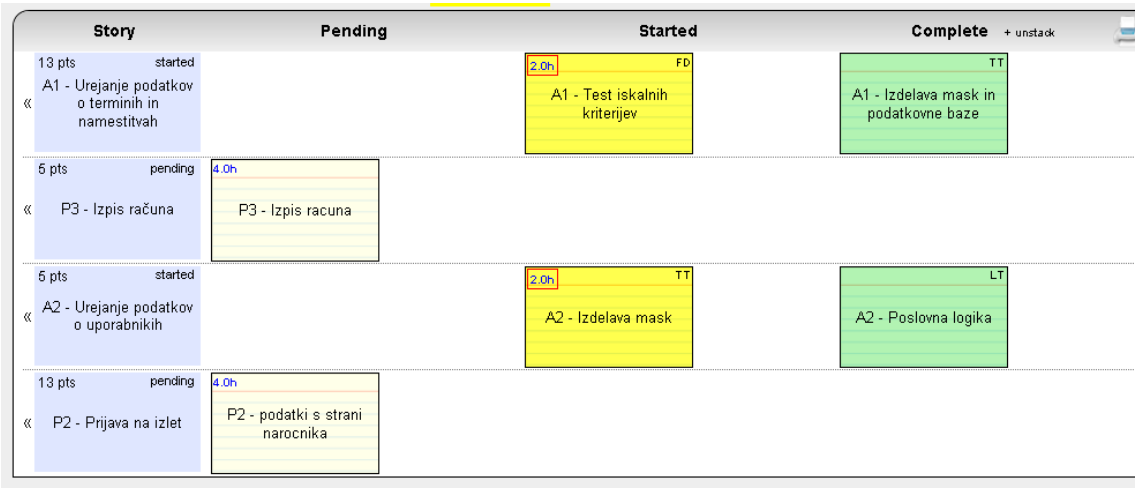
Slika 5.1: Seznam zahtev

Pri urejanju ciklov lahko sami določamo pričakovano realizacijo ter začetni in končni datum. Ti podatki so nam na seznamu zahtev ves čas na voljo. Uporabniške zgodbe lahko dodajamo posamično oziroma v skupini, v kolikor se poslužujemo funkcije uvozi/izvozi (ang. Import/Export).

Drugi pogled je pregled nalog (Slika 5.2) v obliki **simulacije table** (ang. Card Wall). Razdeljena je v štiri stolpce. Prvi predstavlja uporabniške zgodbe, ostali trije pa statuse, ki jih naloge lahko imajo. V stolpcu, namenjenem uporabniškim zgodbam, so izpisane zgodbe, ki naj bi se jih razvilo v trenutnem ciklu. Vsaki zgodbi pripada ena vrstica, v kateri je prikazana njena razčlenitev na naloge. Vsaki nalogi pripada svoja kratica, ki je umeščena v stolpec glede na status naloge. ScrumNinja pozna tri statuse nalog:

- Naloga v vrsti (ang. Pending)
- Začeta (ang. Started)
- Dokočana (ang. Complete)

Vsaki nalogi se poleg statusa določi še odgovorno osebo ter čas, ki je ostal do konca izvedbe naloge. Čas je zapisan v zgornjem levem kotu, kratice zadolžene osebe pa zgoraj desno. Ko so vse naloge dokončane, nam program ponudi izbiro, da zgodbo oddamo v pregled ali da jo zavrnemo. Še vedno obstaja tudi možnost dodajanja dodatnih nalog. S tem pogledom imamo pregled nad vsemi trenutnimi nalogami. Pregled velja samo za trenutni sprint. Tu lahko spremljamo podrobnejši razvoj in status naloge.



Slika 5.2: Pregled nalog

Tretji pregled je **grafični pregled dela**. Pri tem pregledu lahko izbiramo med dvema prikazoma. V kolikor nas zanima, koliko je še predvidenih ur do konca cikla, izberemo prvo možnost. V nasprotnem primeru graf prikazuje, koliko točk uporabniških zgodb nam je še preostalo. Prvi prikaz je vezan na ocene ur, ki smo jih dodelili nalogam, drugi pa na oceno točk uporabniških zgodb, ki še niso dokončane. Na grafu lahko spremljamo tri linije: dejansko izmerjeni napredek pri delu, trend opravljenega dela razvojne skupine in idealni potek projekta. Slednji v bistvu predstavlja cilj razvojne skupine oziroma kako bi morala skupina napredovati, da pravočasno dokonča predvideno delo.

Kot smo že omenili, je ScrumNinja precej osnoven, tako da ne ponuja veliko dodatnih možnosti za administracijo. Vseeno pa se med orodji skrivajo pogled za administracijo skupin, možnost pregledovanja zgodovine in urejanja nalog bodočih ciklov ter že prej omenjeni uvoz/izvoz.

Ob urejanju skupine nas predvsem zanimajo osnovni podatki, kot so ime, začetnice ter elektronski naslov, določiti pa moramo tudi njegovo vlogo. ScrumNinja pozna naslednje vloge:

- Administrator (mišljen tudi kot skrbnik metodologije)
- Član razvojne skupine
- Skrbnik izdelka
- Opazovalec

Pri pregledu skupin na žalost nimamo možnosti dodajanja in administracije večih Scrum skupin hkrati. ScrumNinja ponuja tudi pregled že zaključenih sprintov v obliki zgodovine, pod zavihkom 'bodoče naloge' pa lahko najdemo razporeditev nalog za bodoče cikle. Tu lahko planiramo naloge za uporabniške zgodbe, ki še sledijo.

5.4. Agilo

Orodje Agilo je izdelek podjetja Agile42, ki ima sedež v Nemčiji. Agilo, ki se ga dobi na spletnem naslovu: http://www.agile42.com/cms/pages/agilo/, ima prosto dostopno verzijo in plačljivo »Pro« verzijo. Tudi pri Agilu se lahko odločamo med različico za prenos in različico za gostovanje, ki ponuja tudi repozitorij kode. Prosto dostopna verzija se distribuira pod licenco Apache 2.0. Sam sem dobil poskusno različico za 30 dni uporabe plačljive verzije, ki je robustnejša in ponuja nekaj pomembnih dodatnih možnosti.

Uporabljam verzijo za prenos, tako da internetna povezava ni potrebna. Preden začnemo program uporabljati, je potrebno najprej pognati strežnik, do odjemalca pa nato dostopamo preko spletnega brskalnika. Od tu naprej lahko spremljamo potek celotnega projekta.

Kontrola nad pregledi je razdeljena v dve orodni vrstici. V stranski vrstici so vseskozi prisotni seznam glavnih akcij, pregled seznamov z zahtevami (tako glavnega seznama zahtev kot seznamov nalog za posamezne cikle), pregledi kartic (ang. Ticket) ter pregled strani, ki nudijo pomoč pri upravljanju orodja (ang. Wiki). Zgornja orodna vrstica pa služi za preklop med glavnimi pregledi projekta.

Seznam zahtev ter seznami nalog za posamezen cikel so tukaj ločeni. Uporabniške zgodbe moramo na seznam zahtev dodati preko posebne akcije, naloge pa se lahko uporabniškim zgodbam dodaja direktno na pregledu seznama. Uporabniške zgodbe razporejamo po ciklih tako, da v detajlnem pregledu zgodbe določimo cikel, v katerega spada. Cikle se dodaja preko posebnega pregleda ciklov, ki ga najdemo pod pregledom administracijskih orodij.

Vnosi so na seznamu razporejeni po vrstnem redu vnosa, prioriteto pa se jim določi z atributom »pomembnost« (ang. Importance). Na seznamu nalog imamo pregled uporabniških zgodb, določenih za posamezen cikel, ter njihovo razdelitev na naloge. Boljši pregled nad dogajanjem pri implementaciji nalog pa tudi tukaj nudi pregledna tabla.

Agilo omogoča pregled nalog na pregledni tabli (ang. Whiteboard) (Slika 5.3), kjer imamo dostop do vseh podatkov, ki se tičejo napredka nalog. Uporabniške zgodbe, ki so izbrane za trenutni cikel, so zbrane v prvem stolpcu, imenovanem seznam (ang. Backlog). Tukaj so zbrane tudi naloge, ki pripadajo izbrani zgodbi, vendar še niso začete. Ko so naloge v delu (ang. In Progress), se jih premakne v srednji stolpec, ko pa so zaključene (ang. Done), dobijo svoje mesto v zadnjem stolpcu.

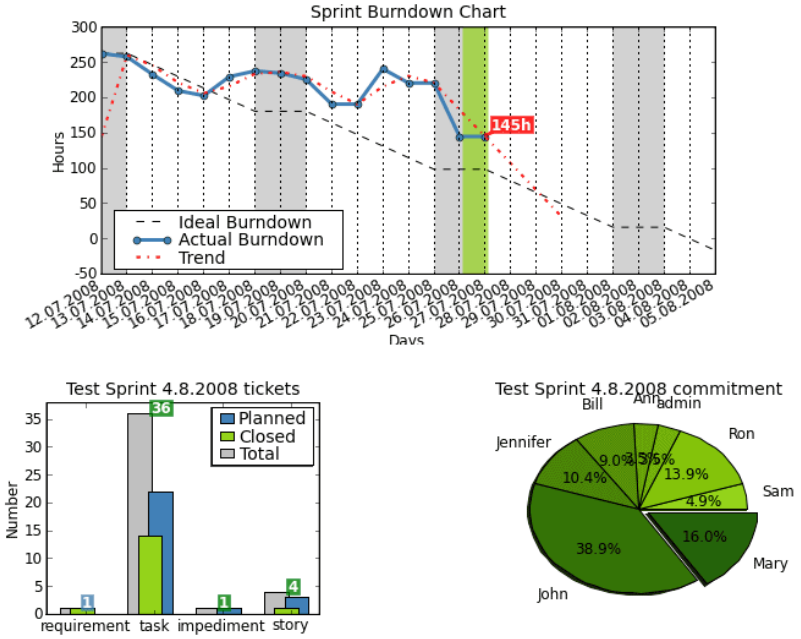
Vsaki nalogi pripada kartica, kjer je poleg šifre naloge zapisan tudi tisti, ki je za nalogo zadolžen in čas, ki je še potreben za dokončanje naloge. Opis naloge na karticah ni prikazan. Dobra stran prikaza pri Agilu je to, da je možno skrčiti vrstice, v katerih so zgodbe razčlenjene, tako da lahko pregledujemo samo izbrane uporabniške zgodbe. To naredi tablo preglednejšo, posebno če je seznam zgodb obsežen. Tabla omogoča tudi razporejanje nalog, ki niso vezane na nobeno izmed naštetih uporabniških zgodb. Ko se uporabniško zgodbo premakne v stolpec »začeto«, je potrebno nujno določiti tistega, ki je za nalogo odgovoren.

Slika 5.3 : pregled nalog

Grafični pregled

Grafični pregled je mogoče najboljša stran tega programa (Slika 5.4). Poleg grafa preostalega dela je prisoten prikaz, kako so člani razvojne ekipe trenutno razporejeni po nalogah ter razmerja med opravljenimi in planiranimi nalogami in uporabniškimi zgodbami. Deleži zasedenosti razvojne ekipe bi morali ob idealni razdelitvi dela biti med seboj čim bolj podobni. Na grafu preostalega dela imamo štiri prikaze:

- Trend
- Kapaciteta
- Idealni napredek
- Dejanski napredek



Slika 5.4 : Grafični pregled

Časovnica

Časovnica (ang. Timeline) služi kot neke vrste beležnica oziroma dnevnik vseh dogodkov. Tukaj lahko po dnevih pregledujemo spremembe, ki se tičejo nalog, uporabniških zgodb, ciklov itd. Pomemben podatek, ki je tudi povsod naveden, je avtor spremembe in točna ura ter datum. Nastavimo lahko, katere spremembe nas zanimajo in v kakšnem časovnem intervalu jih pregledujemo.

Napredovanje projekta

Pregled napredovanja projekta (ang. Roadmap). Delo za določen projekt se pri Agilu lahko razdeli na mejnike (ang. Milestone). Med posameznimi mejniki je lahko več ciklov. Za vsak cikel ter mejnik pa imamo grafični prikaz napredka razvoja in količine časa, ki je še na razpolago. Pregled sicer ni najbolj natančen, dobi pa smisel, kadar je struktura projekta kompliciran.

5.5. VersionOne

VersionOne je trenutno eno najširše uporabljanih orodij za agilno modeliranje na tržišču. Pozna tri različne verzije: Team, Enterprise ter Ultimate. Za vse tri verzije se lahko uporabnik odloči, ali bo uporabljal gostovanje na VersionOne strežnikih ali pa bo uporabljal verzijo za prenos. Dostopa se preko spletnega odjemalca. Sam sem uporabljal poskusno verzijo različice Enterprise za 30 dni in gostoval na njihovem strežniku.

Odjemalec VersionOne je organiziran v tri dele. Glavne sekcije so zbrane v orodni vrstici na vrhu zaslona, levo je orodna vrstica za pregled trenutnega projekta oziroma trenutnega konteksta, osrednji del pa je namenjen pregledu podrobnosti. V zgornji orodni vrstici lahko pregledujemo vse faze projekta – od planiranja izdelka, planiranja cikla, razdeljevanja na naloge, pa do pregleda raznih poročil. V levi orodni vrstici so zbrane glavne akcije glede na to, kje se trenutno nahajamo. Za prikaz lahko izbiramo različne kontekste, kot so trenutni projekt, podprojekt itd. V orodni vrstici na levi strani so tudi vseskozi prikazane zadnje spremembe, ki smo jih opravili. Glavni del je pregled podrobnosti, ki je razdeljen na razne zavihke, tako da se lažje sprehajamo oziroma znajdemo med različnimi deli. Vsi niso obvezni - uporabljamo lahko samo nekatere.

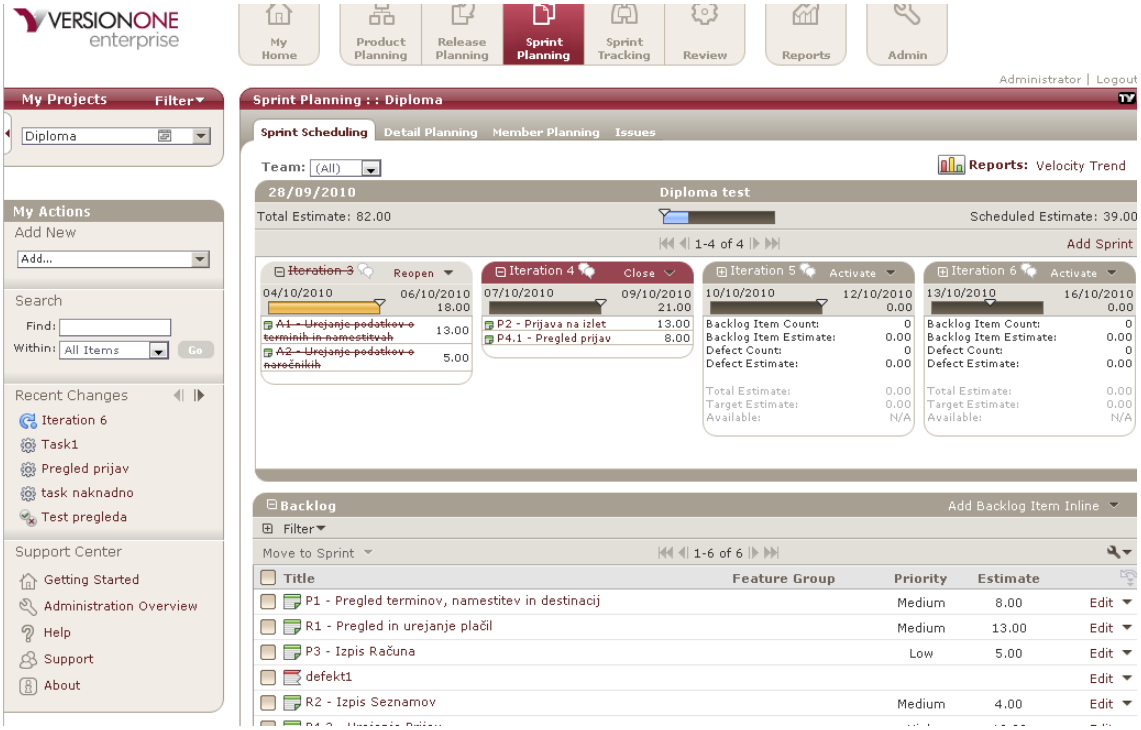
Kot pri drugih programih, tudi tukaj najprej urejamo seznam zahtev, ki je dostopen v prvem pregledu. To je pregled »Planiranje izdelka«, ki služi za identifikacijo in kategorizacijo zahtev ter administracijo izdelka z vidika celotnega projekta. Glavno orodje je definitivno seznam zahtev, poleg tega pa obstaja tudi zavihek za pregled in operiranje z epi, dodatnimi zahtevami (ang. Request), cilji izdelka in podobnim.

Na seznam zahtev lahko uvrščamo dve vrsti vnosov – napake (ang. Defects) in uporabniške zgodbe. Ločimo ju po barvi ikone. Za obe vrsti vnosov prioriteto določa vrstni red na seznamu, ki ga urejamo z enostavnim premikanjem vnosov. Za prioriteto imamo tudi poseben atribut, vendar je pravi pokazatelj nujnosti pozicija na seznamu. Za testni primer v tej nalogi pridejo v poštev samo uporabniške zgodbe. Vrstni red lahko uredimo tudi glede na vrednost atributa v posameznem stolpcu, na primer po številu točk. V kolikor želimo prvotno razporeditev, moramo sprožiti akcijo »ponastavi seznam«.

Uporabniške zgodbe dodajamo direktno na seznam zahtev. Za ta namen lahko uporabimo dva načina – hiter in standarden. Pri hitrem načinu se vnos prikaže direktno na seznamu, tako da izpolnjujemo samo glavne atribute, pri standardnem pa se nam za to odpre posebno okno, kjer imamo na razpolago vse atribute uporabniške zgodbe. Podoben sistem dodajanja nas spremlja na vseh seznamih, prisotnih v programu. Seznam zahtev lahko prikazuje vnose, namenjene celotnemu projektu, lahko pa nabor zgodb omejimo samo na določeno izdajo. Te predstavljajo podprojekt oziroma samostojno enoto v okviru glavnega projekta.

Poleg seznama zahtev so v tem pregledu v okviru zavihkov dosegljivi tudi epi, torej uporabniške zgodbe, ki se jih bo tekom projekta razčlenilo na več drugih uporabniških zgodb. Posebno mesto imajo tudi dodatne zahteve uporabnika, ki se pojavljajo tekom projekta. Tu je

mesto za njihov zapis, preden te dejansko postanejo uporabniške zgodbe. VersionOne omogoča, da vzpostavimo relacijo med temi zahtevami in uporabniškimi zgodbami, ki iz njih nastanejo. V sklopu planiranja projekta lahko določimo tudi cilje, ki bi si jih radi postavili znotraj razvoja, in njihovo povezavo z uporabniškimi zgodbami ter administracijo posebnih skupin ter morebitnih zapletov (ang. Issues). VersionOne omogoča tudi uvoz podatkov v obliki Microsoft Excel datotek.



Slika 5.5: Planiranje cikla

V kolikor imamo projekt razdeljen na izdaje, lahko to urejamo v okviru naslednjega pregleda. To je »Planiranje izdaj« (ang. Release planing**)**. Urejanje na tak način ni obvezno, v sklopu zavihkov pa lahko najdemo urnik izdaj, napovedovanje izdaj ter razporejanje skupin. Pregled je še posebej uporaben, v kolikor imamo več ciklov, urejenih v izdaje.

Preden lahko planiramo in pregledujemo cikle, moramo na projekt pripeti poseben urnik ciklov, ki ga formiramo v administrativnih orodjih. V urniku moramo določiti, kako dolgi bodo cikli, ter ali bo med njimi kakšen dan premora. načeloma narekuje, da naj bi si cikli sledili zaporedoma, torej brez premorov.

Pri planiranju cikla (ang. Sprint Planning) imamo štiri zavihke (Slika 5.5). Prvi je 'razporejanje ciklov', kjer urejamo razporeditev ciklov skozi cel projekt. S pomočjo seznama zahtev lahko razporejamo uporabniške zgodbe. Kljub temu, da smo cikle načeloma časovno omejili že z urnikom ciklov, lahko datume tukaj še vedno spremenimo. V pomoč pri razporejanju in določanju hitrosti razvoja nam je graf o trendu hitrosti razvoja, ki je dostopen desno zgoraj.

Zavihek 'podrobno planiranje' je v bistvu seznam nalog. Poleg planiranja nalog lahko tukaj dodamo tudi teste za posamezne uporabniške zgodbe. Seznam nalog je omejen na en cikel, s tem da lahko pregledujemo vse cikle, ki še niso zaprti. Ostala dva zavihka sta 'planiranje človeških virov' in 'urejanje zapletov'.

Spremljanje ciklov (ang. Sprint Tracking) je osredotočeno na podrobnejše preglede razvoja nalog in testov. Poleg seznama nalog, ki je zaradi boljše preglednosti prisoten tudi pri tem pogledu, ima VersionOne več tabel za pregled. Prva je tabla za razporejanje uporabniških zgodb (ang. Storyboard), kjer razporejamo uporabniške zgodbe glede nato, ali je zgodba začeta, opravljena, bo razvita v prihodnosti, ali pa sprejeta. Pri ostalih programih to opravljamo preko statusov njenih nalog, tukaj pa to lahko opravljamo ločeno. Tabla za razporejanje nalog (ang. Taskboard) je podobna kot v drugih programih, s to razliko, da imamo v stolpcu povzetek (ang. Summary), ki zajema poleg preostalega števila ur uporabniške zgodbe tudi grafični prikaz opravljenih testov. Te razporejamo na zadnji prikazni tabli, torej tabli za spremljanje testov (ang. Test Board). Zanima nas predvsem, ali je test opravljen ali ne.

VersionOne ima res veliko izbiro poročil. Pregledujemo lahko hitrost razvoja na več načinov, napredek v sprintih na več načinov, pozna tudi pregledno ploščo itd. Poročila so razdeljena na več skupin, nekatera pa se ponavljajo v več skupinah:

- Poročila, namenjena odločanju (ang. Executive Reports)
- Poročila o projektih in izdajah
- Poročila o ciklih
- Poročila o razvojnih skupinah
- Poročila o planiranju
- Poročila o testih in kvaliteti

Pri večini skupin so prisotne t. i. pregledne plošče (ang. Dashboard), kjer imamo zajete podatke, ki omogočajo širok pregled. Vsak uporabnik si lahko nastavi nabor grafov, ki se mu izpisujejo v zavihkih, kakor tudi začetno stran programa (ang. My home).

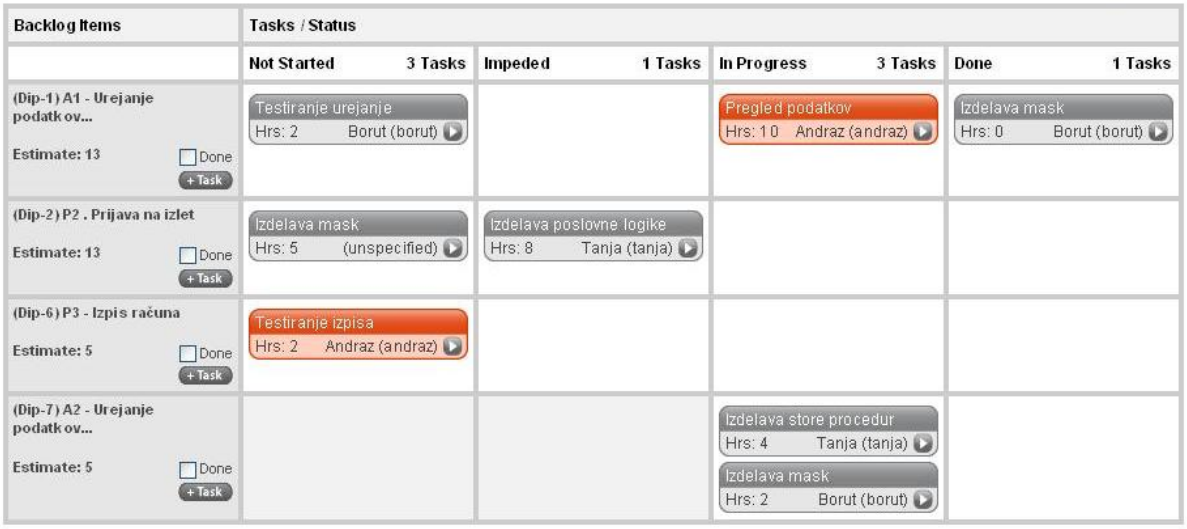
5.6. ScrumWorks

ScrumWorks je program podjetja Danube Technologies, ki se ukvarja predvsem z razvojem programske opreme in izobraževanji za Scrum. To ameriško podjetje z izdelkoma ScrumWorks Pro in ScrumWorks Basic predstavlja enega vodilnih podjetji na tem področju. Obe verziji sta namenjeni za prenos. V nalogi sem preizkušal plačljivo verzijo s 35 dnevno preizkusno licenco.

ScrumWorks ima dva odjemalca – spletnega, ki je namenjen sprotnemu spremljanju nalog v projektu, ter namiznega (ang. Desktop Client), preko katerega lahko dostopamo do vseh možnosti, ki jih ScrumWorks ponuja. Za naprednejše nastavitve se uporablja namizni odjemalec, za sprotno uporabo pa je boljši spletni.

Do spletnega odjemalca se dostopa preko spletnega brskalnika. Ker je namenjen predvsem sprotnemu pregledu nalog, je dokaj enostaven. Poudarek je na pregledu nalog. Spremljamo lahko, v kateri fazi so, kdo je za njih zadolžen, opis, opombe in druge podatke. Možne so različne nastavitve glede tega, katere in čigave naloge, projekte ali cikle bomo spremljali (Slika 5.6). V vsakem primeru so naše naloge obarvane drugače kot ostale. ScrumWorks pozna za naloge štiri statuse:

- Naloga še ni začeta (ang. Not started)
- Naloga ima ovire (ang. Impeded)
- Začeta (ang. In progess)
- Dokončana (ang. Done)



Slika 5.6: Prikaz nalog v spletnem odjemalcu

Tudi tukaj so naloge združene pod uporabniške zgodbe, ki jim pripadajo. Te so naštete na levi strani table. Poleg opisa in ocene zahtevnosti imamo še prostor, kjer lahko označimo, da je uporabniška zgodba dokončana. Pod pregledom nalog se nam izrisuje graf preostalega dela, ki prikazuje, koliko delovnih ur nam je še ostalo, ter tabela, namenjena možnosti dodajanja težav (ang. Impediments), ki se pojavljajo med razvojem in ovirajo potek projekta.

Namizni odjemalec (Slika 5.7) služi za celoten pregled nad aplikacijo oziroma projektom. Tukaj lahko nastavljamo projekte, urejamo razvojne skupine, pregledujemo napredek projekta in podobno. Zaženemo ga preko Java bližnjice. Prvo okno, ki se nam odpre pri projektu, je navadno seznam zahtev s seznami nalog za posamezne sprinte. Leva polovica zaslona predstavlja seznam nalog, desna pa seznam zahtev. Cikle moramo na seznam zahtev nanizati ročno, vseskozi pa imamo pregled nad vsemi cikli, ki so planirani v tem projektu.

Slika 5.7: Namizni odjemalec

Uporabniške zgodbe dodajamo v posebnem oknu, kjer poleg osnovnih podatkov lahko določimo tudi poslovno težo. Ta se izračuna iz vrednosti dobitka, v kolikor je zgodba vključena v izdelek, in vrednosti odbitka, v kolikor zgodba ni vključena v projekt. Iz tega se tudi izračuna dobljena vrednost glede na vložen trud (ang. Return on Investment) ter utežen delež zgodbe.

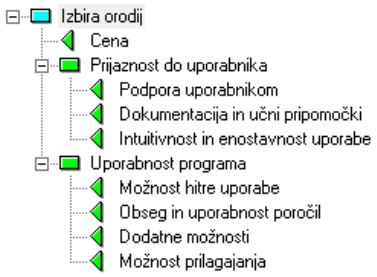
Ko nanizamo cikle na seznam nalog, lahko zgodbe s seznama zahtev enostavno premaknemo v želeni cikel. Za razčlenjevanje uporabniških zgodb na naloge je boljši spletni odjemalec. Tudi tukaj se za prioriteto določa pozicija na seznamu. Seznam zahtev je razdeljen na posamezne izdaje. Vse skupaj lahko zapakiramo v eno ali več izdaj.

6. Primerjava

Za namene primerjave programov sem razvil odločitveni model (Slika 6.1). Sestavljen je iz treh glavnih kriterijev in sedmih podrejenih kriterijev. Za pomoč pri sestavi modela sem uporabil program DEXi, ki je za neprofitne projekte brezplačen. Dobi se ga na internetnem naslovu: http://www-ai.ijs.si/MarkoBohanec/dexi.html. Glavni viri podatkov za primerjavo variant so bile izvedba testnega primera in spletne strani navedenih programov [11] - [14].

Kot že rečeno, ima odločitveni model tri glavne kriterije, ki imajo pri končni odločitvi enako težo. To so:

- Cena
- Prijaznost do uporabnika
- Uporabnost programa



Slika 6.1: Odločitveni model

Cena programov je izračunana za razvojno skupino petih ljudi in za obdobje treh let. Zaloga vrednosti kriterija »Cena« je razdeljena v štiri intervale, predstavlja pa vrednost plačila v valuti USD. V kolikor program dopušča več načinov licenciranja, je v modelu upoštevan tisti, ki se za to obdobje najbolj splača. Intervali, ki so enako obteženi, so sledeči:

- do 1500 $
- 1500 – 2000 $
- 2000 – 2500 $
- več kot 2500 $

Prijaznost do uporabnika je sestavljena iz naslednjih kriterijev:

- Podpora uporabnikom – utež 20
- Dokumentacija in učni pripomočki – utež 30
- Intuitivnost in enostavnost uporabe – utež 50

Kot podporo uporabnikom razumemo vsebino podpore, ki je vključena v licenco, obstoj forumov in drugih načinov izmenjevanja informacij. Podpora uporabnikom ima utež 20. Kriterij dokumentacije in učnih pripomočkov, ki ima utež 30, zajema količino in kvaliteto tekstovnega, video in ostalega gradiva, ki je na voljo, da se uporabniki lahko naučijo uporabljati program in nadgradijo svoje znanje. Intuitivnost in enostavnost uporabe pa ima kot glavni kriterij tudi največjo veljavo – utež 50.

Uporabnost programa je sestavljena iz naslednjih kriterijev:

- Možnost hitre uporabe – utež 10
- Obseg in uporabnost poročil – utež 20
- Dodatne možnosti – utež 35
- Možnost prilagajanja – utež 35

Možnost hitre uporabe se navezuje na to, ali se da program uporabiti z majhno količino znanja in samo za osnovne podatke, ki nam služijo pri projektu, brez zapletenega nastavljanja naprednih možnosti. Ta kriterij ima utež 10. Obseg in uporabnost poročil ima utež 20, dodatne možnosti pa utež 35. Slednji kriterij zajema količino dodatnih nastavitev, pregledov in funkcij, ki jih zajema program poleg osnovnega spremljanja Scrum procesa. Zadnji kriterij je možnost prilagajanja, ki ima prav tako utež 35.

Na sliki 6.2 so prikazane ocene za posamezne kriterije ter končna ocena. Ta je prikazana v vrstici »Izbira orodij«. Zaloga vrednosti vseh podrejenih kriterijev je tristopenjska, z vrednostmi: Slaba, Zadovoljiva in Dobra. Edina izjema je kriterij »Dodatne možnosti«, kjer so vrednosti: Malo, Srednje in Veliko. Glavni kriteriji imajo štiristopenjsko zalogo vrednosti:

- Slaba
- Zadostna
- Zadovoljiva
- Dobra

Na podlagi odločitvenega modela DEXi sem prvim trem programom dodelil končno oceno »Zadovoljiva«, programu ScrumWorks pa »Dobra«, kar je tudi najvišja možna ocena. ScrumNinja je imel največ vnesenih najnižjih vrednosti, kljub temu pa je dobil oceno »Zadovoljiva« zaradi nizke cene. Agilo ima največ srednjih vrednosti, VersionOne pa največ najboljših. Kljub temu je dobil samo oceno »Zadovoljiva«, in sicer zaradi najvišje cene.

Ne glede na rezultate odločitvenega modela menim, da je najboljša izbira VersionOne. Glavni razlog, da ScrumWorks dobi boljšo oceno, je v ceni, v resnici pa je razlika med cenama omenjenih orodij samo nekaj več kot 500$. To je ravno dovolj, da sta vsak v svojem intervalu. Tudi pri testnem primeru je najboljši vtis pustil ravno VersionOne. Poleg Dobra izbira se mi zdi tudi ScrumNinja. Kljub slabšim rezultatom cena odtehta okrnjene dodatne možnosti. Program namreč zaradi svoje preglednosti in preprostosti predstavlja dobro izbiro za prvi stik z metodologijo.

.

Varianta	Scrum Ninja	Agilo	VersionOne	ScrumWorks
. Izbira orodij	Zadovoljiva	Zadovoljiva	Zadovoljiva	Dobra
. . Cena	do 1500	1500 - 2000	2500+	2000 - 2500
. . Prijaznost do uporabnika	Zadovoljiva	Slaba	Dobra	Dobra
. . . Podpora uporabnikom	Zadovoljiva	Zadovoljiva	Dobra	Dobra
. . . Dokumentacija in učni pripomočki	Slaba	Zadovoljiva	Dobra	Zadovoljiva
. . . Intuitivnost in enostavnost uporabe	Dobra	Slaba	Dobra	Dobra
. . Uporabnost programa	Slaba	Zadovoljiva	Dobra	Dobra
. . . Možnost hitre uporabe	Dobra	Zadovoljiva	Slaba	Zadovoljiva
. . . Obseg in uporabnost poročil	Slaba	Zadovoljiva	Dobra	Zadovoljiva
. . . Dodatne možnosti	Malo	Srednje	Veliko	Veliko
. . . Možnost prilagajanja	Slaba	Zadovoljiva	Dobra	Dobra

Slika 6.2 : Rezultati odločitvenega modela

7. Zaključek

Agilne metodologije v inženirstvo programske opreme prinašajo val svežine. Z uvedbo agilnih metodologij v razvojni proces se veča kvaliteta kode, kakor tudi procesa samega. Znotraj področja trenutno prednjači metodologija Scrum in vse kaže na to, da bo se bo baza uporabnikov Scrum-a v naslednjih letih samo še razširila. Razloge lahko najdemo v tem, da je Scrum že v osnovi intuitiven in predpisuje enostavna pravila, ki se jih ni težko držati. Glavna prednost pa je, tako kot pri drugih agilnih metodologijah, poudarek na razvoju in dostavi uporabne programske kode.

Ne glede na povedano pa imajo tudi agilne metodologije svoje slabosti. Ena glavnih je ta, da je prehod s klasičnega na agilni razvoj še vedno težaven proces. Predvsem tisti, ki se z razvojem programske kode ne ukvarjajo neposredno, imajo težave s sprejemanjem procesa, ki se sproti razvija in dopolnjuje. Še posebej vodilni v podjetjih še vedno ne odstopajo od oprijemljivih in končnih ocen ter planov. Tudi uporabnikom programskih izdelkov ni vedno enostavno razložiti vsebine in prednosti agilnih metodologij. Ljudje smo preprosto po naravi taki, da imamo radi določena zagotovila.

Za pomoč pri težavah, ki nastanejo, obstajajo različni dodatni prijemi, kot so vpeljava uporabniških zgodb, ki imajo za uporabnika določeno težo in pomen. Uporabnik jih navadno napiše sam in se tako čuti bolj vključenega v razvoj, kar nedvomno pomaga pri reševanju problemov in poveča transparentnost procesa. Poleg tega nam različne tehnike ocenjevanj in planiranja, ki jih lahko uporabljamo pri agilnem razvoju, pomagajo pri tem, da kljub vsemu dobimo trdnejše in bolj zanesljive ocene, ki povečujejo kredibilnost in medsebojno zaupanje. Za podporo vsemu temu pa je na trgu prisotno precejšnje število različnih orodij, ki dodatno pripomorejo k boljši kvaliteti razvojnega procesa.

Glede na to, da področje ni več v začetni fazi in da je prisotnih kar nekaj tehnik in orodij, je agilnost vsekakor smer, ki se ji splača slediti. Predpogoj za uporabo pa sta zadostna količina znanja ter zavedanje prednosti in slabosti metodologije, ki jo uporabljamo.

Dodatek A - Opis testnega primera

Pregled uporabniških zgodb

Začetni seznam uporabniških zgodb

Kratica	Naziv
P1	Pregled terminov, namestitev in destinacij
P2	Prijava na izlet
P3	Izpis računa
P4	Pregled in urejanje prijav
R1	Pregled in urejanje plačil
A1	Urejanje podatkov o terminih in namestitvah
A2	Urejanje podatkov o uporabnikih

Dodatne uporabniške zgodbe

Kratica	Naziv
P4.1	Pregled prijav
P4.2	Urejanje prijav
R2	Izpis seznamov
R3	Izpis položnic

Opis Uporabniških zgodb

Vloga Prodajalec

*Pregled terminov, namestitev in destinacij – **P1***

- Opis: Prodajalec mora imeti na voljo pregled vseh razpisanih terminov na določenih destinacijah ter prostih namestitev, opisov in podrobnosti le teh ter cen.
- Testi:
  - Ko je namestitev zasedena se le ta ne sme izpisati.
  - Ob dodajanju novih namestitev in spreminjanju podatkov se morajo vse spremembe odražati tudi v prikazu.

*Prijava na izlet – **P2***

- Opis: Prodajalec glede na izbrane atribute (destinacija, namestitev,…) opravi prijavo. Pri tem se zahteva vnos podatkov o stranki ter podrobnosti o načinu prijave in plačevanja.
- Testi
  - Prodajalec ima možnost vnosa vseh zahtevanih podatkov.
  - Prodajalec ima na izbiro več, že v naprej pripravljenih, načinov plačevanja, lahko pa tudi sam doda posebnost.
  - Prodajalec lahko kadarkoli prekine prijavo.
  - Prodajalec mora imeti na voljo pregled vseh vnesenih podatkov preden prijavo potrdi.

*Izpis računa – **P3***

- Opis: Ob koncu prijave je potrebno natisniti račun.

- Testi:
    - Račun mora biti pravilnega formata.
    - Vneseni podatki se morajo pravilno prilagajati fiksnemu tekstu na izpisu.

*Pregled in urejanje prijav – **P4***

- Opis: Prodajalec ima voljo pregled vseh prijav in njihovo urejanje glede na izbrane atribute.
- Testi
    - Prodajalec ima na voljo pregled vseh podrobnosti izbranih prijav.
    - Prodajalec pridobi pregled prijav glede na poljubno kombinacijo izbranih atributov, tudi če se kriterije pusti prazne.
    - Prodajalec lahko spreminja posamezne podatke.

Vloga Računovodja

*Pregled in urejanje plačil – **R1***

- Opis: Računovodja mora imeti na voljo pregled in možnost urejanja vseh plačil ter obveznosti strank do agencije.
- Testi
    - Računovodja lahko spremeni status plačila.
    - Računovodja lahko vnese vknjižbo.
    - Računovodja lahko spreminja podatke o plačilih.
    - Mogoče je dobiti seznam potnikov za določen termin, ki še nimajo poravnanih vseh obveznosti.

Vloga Administrator

*Urejanje podatkov o terminih in namestitvah – **A1***

- Opis: Vnos ter spreminjanje podatkov o namestitvah, terminih, destinacijah, cenah in podobnem.
- Testi:
    - Cene se lahko nastavijo za množico izborov.
    - Kategorije izpisa podatkov se določi preko izbranih kriterijev.

*Urejanje podatkov o uporabnikih – **A2***

- Opis: Administrator mora imeti na voljo pregled podatkov o uporabnikih ter administracijo pravic ter dostopa.
- Testi:

- Administrator lahko ponastavi geslo uporabniku.
- Administrator lahko spreminja vnesene podatke o uporabnikih.
- Administrator določa pravice uporabniku, le ta na podlagi tega lahko dostopa do določenih pregledov do drugih pa ne.

Dodatne zgodbe

Vloga Prodajalec

*Pregled prijav – **P4.1***

- Opis: Prodajalec ima voljo pregled vseh prijav glede na izbrane atribute.
- Testi
  - Prodajalec ima na voljo pregled vseh podrobnosti izbranih prijav.
  - Prodajalec pridobi pregled prijav glede na poljubno kombinacijo izbranih atributov, tudi če se kriterije pusti prazne.

*Urejanje prijav – **P4.2***

- Opis: Prodajalec ima možnost spreminjanja podatkov o prijavah.
- Testi:
  - Prodajalec lahko spreminja posamezne podatke.
  - Prodajalec lahko prijavo popolnoma izbriše.

Vloga Računovodja

*Izpis seznamov – **R2***

- Opis: Glede na izbrane parametre je mogoče izpisati določene sezname za interno rabo.
- Testi:
  - Mogoče je natisniti seznam vseh potnikov v določenem terminu.
  - Mogoče je natisniti pregled zasedenosti določene namestitve skozi določeno obdobje.
  - Mogoče je natisniti seznam potnikov po namestitvah.

*Izpis položnic – **R3***

- Opis: Mogoč je izpis položnic za izbrane obroke.
- Testi:
  - Lahko se nastavi poljuben nabor strank.
  - Lahko se nastavi poljuben nabor obrokov.

- Datum se avtomatično nastavi, vendar ga je mogoče spremeniti.

Scenarij poteka

Pred vsakim ciklom se uporabniške zgodbe oceni in jih razdeli po prioriteti ter formira okviren plan zaključka. Oceni se tudi predvideno hitrost realizacije, pri tem, da metoda ocene ni fiksna za celoten projekt. Točke zahtevnosti so razdeljene po Fibonaccijevi lestvici: 1,2,3,5,8,13,21,34,55; s tem, da je 55 najvišja možna vrednost ter rezervirana za epe. Prioritete so Postavljene od 1 do 5. Pri tem 1 pomeni najvišjo vrednost 5 pa najmanjšo. Skrbnik produkta lahko prioritete pred začetkom cikla poljubno spreminja. Začetna ocena hitrosti realizacije je postavljena na 26 točk zahtevnosti, ki v drugem ciklu rahlo naraste.

Cikel 1

Seznam zahtev

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
A1	Urejanje podatkov o terminih in namestitvah	13	1
P2	Prijava na izlet	13	2
P4	Pregled in urejanje prijav	21	2
P1	Pregled terminov, namestitev in destinacij	8	3
R1	Pregled in urejanje plačil	13	3
P3	Izpis računa	5	4
A2	Urejanje podatkov o uporabnikih	5	4

Predviden potek projekta pred začetkov prvega cikla

V prvi cikel se uvrstita prvi dve zgodbi s seznama prioritet: A1 in P2. Ker je začetna hitrost realizacije postavljena na 26, to tudi pomeni, da ni prostora za dodatne uporabniške zgodbe.

	Hitrost realizacije	Uporabniške zgodbe
Cikel 1	26	A1, P2
Cikel 2	29	P4, P1
Cikel 3	23	R1, P3, A2

Potek cikla

Pravila o načinih plačevanja niso dostopna: čaka se na specifikacije vodstva podjetja naročnika. Realizacija zgodbe P2 torej ni mogoča. Ker ja razvoja skupina dokončala zgodbo A1, se v sodelovanju s skrbnikom produkta ciklu doda zgodba A2.

Realizirano v prvem ciklu

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
A1	Urejanje podatkov o terminih in namestitvah	13	1
A2	Urejanje podatkov o uporabnikih	5	4

Cikel 2

Doda se zgodba R2, Spremni se oceno zgodbi P1, zgodbo P4 se razdeli na zgodbi P4.1 ter P4.2

Seznam zahtev

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
P2	Prijava na izlet	13	1
P4.1	Pregled prijav	8	2
P4.2	Urejanje prijav	13	2
P1	Pregled terminov, namestitev in destinacij	13	3
R1	Pregled in urejanje plačil	13	3
R2	Izpis seznamov	8	3
P3	Izpis računa	5	4

Predviden potek projekta pred začetkom drugega cikla

Kljub realizaciji 18 v prvem ciklu se predvidena realizacija postavi na 21, saj je predvideva, da bo tokrat naročnik tokrat hitreje posredoval informacije. Kljub temu pa se hitrost realizacije zmanjša glede na prvotno oceno.

	Predvidena realizacija	Uporabniške zgodbe
Cikel 2	21	P2, P4.1
Cikel 3	21	P4.2, R2
Cikel 4	18	P1,P3
Cikel 5	13	R1

Potek cikla

Razvojna skupina predvideni zgodbi implementira predčasno, tako da se tekočemu ciklu doda zgodba R2, tudi zaradi tega, ker naročnik pristane na tehnologijo izpisa, ki jo obstoječi člani razvojne skupina že poznajo. Tudi ta uporabniška zgodba je uspešno zaključena do konca cikla.

Realizirano v drugem ciklu

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
P2	Prijava na izlet	13	1
P4.1	Pregled prijav	8	2
R2	Izpis seznamov	8	3

Cikel 3

Doda se zgodba R3 z visoko prioriteto. Predvideno realizacijo se dvigne, ker je bil prejšnji cikel uspešen. Postavi se jo na 26.

Seznam zahtev

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
P4.2	Urejanje prijav	13	1
R3	Izpis položnic	8	1
P1	Pregled terminov, namestitev in destinacij	13	3
R1	Pregled in urejanje plačil	13	3
P3	Izpis računa	5	4

Predviden potek projekta pred začetkom tretjega cikla

	Predvidena realizacija	Uporabniške zgodbe
Cikel 3	26	P4.2,R3,P3
Cikel 4	26	P1, R1

Potek cikla

Izpis položnic se izkaže za veliko trši oreh kot sprva predvideno. Zgodbe P3 ekipa ne uspe implementirati.

Realizacija v tretjem ciklu

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
P4.2	Urejanje prijav	13	1

R3	Izpis položnic	8	1

Cikel 4

Realizacijo se ponovno zniža na 21.

Seznam zahtev

Okrajšava	Uporabniške zgodbe	Točke zahtevnosti	Prioriteta
P1	Pregled terminov, namestitev in destinacij	13	2
R1	Pregled in urejanje plačil	13	3
P3	Izpis računa	5	4

Predviden potek projekta pred začetkom četrtega cikla.

	Predvidena realizacija	Uporabniške zgodbe
Cikel 4	18	P1, P3
Cikel 5	13	R1

Potek cikla in nadaljevanje projekta

Projekt se v nadaljevanju odvija kot planirano.

Literatura

[1] J. Hunt, Agile Software construction, London: Springer-Verlag, 2006, pogl. 2

[2] M. Cohn, User Stories Applied For Agile Software Development, Boston: Addison-Wesley, 2004, pogl. 1,2 15.

[3] M. Cohn, Agile Estimating and Planning, Prentice Hall, 2005, pogl. 1,4,6, 16.

[4] (2010) K. Schwaber, J. Sutherland, »Scrum«. Dostopno na: http://www.scrum.org/scrumguideenglish/

[5] (2010) Scrum Is an Innovative Approach to Getting Work Done. Dostopno na: http://www.scrumalliance.org/pages/what_is_scrum

[6] (2010) Introduction to SCRUM Methodology. Dostopno na: http://www.scrummethodology.org/

[7] (2010) 2009 State of Agile Development Survey Results. Dostopno na: http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf

[8] (2010) http://agilemanifesto.org/

[9] (2010) http://agilemanifesto.org/principles.html

[10] (2010) http://www.userstories.com

[11] (2010) http://www.agile42.com/cms/pages/agilo/

[12] (2010) http://scrumninja.com/scrum_software

[13] (2010) http://www.versionone.net/

[14] (2010) http://danube.com/scrumworks