UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Uroš Smolnik

ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

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

Št. naloge: 01725/2011
Datum: 15.02.2011

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

Kandidat: **UROŠ SMOLNIK**

Naslov: **ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM**
SCRUM PROJECT MANAGEMENT TOOL

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite postopek razvoja programske opreme po metodi Scrum in možnosti za vpeljavo meritev, s katerimi bi lahko spremljali učinkovitost razvojnega procesa v skladu z zahtevami modela CMMI. Na podlagi tega realizirajte orodje za vodenje projektov po metodi Scrum, ki bo za vsak projekt omogočalo (poleg vzdrževanja seznama zahtev in seznamov nalog po posameznih iteracijah) tudi zajem podatkov, potrebnih za izračun indikatorjev prislužene vrednosti in učinkovitosti dela. Izračunane indikatorje prikažite v obliki ustreznih grafov.

Mentor:
prof. dr. Viljan Mahnič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Uroš Smolnik

ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Viljan Mahnič

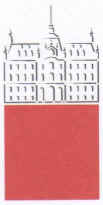
Ljubljana, 2011

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

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: 01725/2011
Datum: 15.02.2011

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

Kandidat: **UROŠ SMOLNIK**

Naslov: **ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM**
SCRUM PROJECT MANAGEMENT TOOL

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Proučite postopek razvoja programske opreme po metodi Scrum in možnosti za vpeljavo meritev, s katerimi bi lahko spremljali učinkovitost razvojnega procesa v skladu z zahtevami modela CMMI. Na podlagi tega realizirajte orodje za vodenje projektov po metodi Scrum, ki bo za vsak projekt omogočalo (poleg vzdrževanja seznama zahtev in seznamov nalog po posameznih iteracijah) tudi zajem podatkov, potrebnih za izračun indikatorjev prislužene vrednosti in učinkovitosti dela. Izračunane indikatorje prikažite v obliki ustreznih grafov.

Mentor:
prof. dr. Viljan Mahnič

Dekan:
prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Uroš Smolnik,

z vpisno številko 63040152,

sem avtor diplomskega dela z naslovom:

ORODJE ZA VODENJE PROJEKTOV PO METODI SCRUM

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 29.04.2011 Podpis avtorja:

Zahvala

Zahvalil bi se svojemu mentorju, izr. prof. dr. Viljanu Mahniču, ki mi je bil s svojo potrpežljivostjo in odzivnostjo na moja vprašanja v neizmerno pomoč pri dokončanju diplomskega dela.

Kazalo

Povzetek

Za diplomsko nalogo sem naredil aplikacijo za vodenje projektov po metodologiji Scrum. Namen je bil narediti aplikacijo, ki jo bodo lahko študentje uporabljali pri delu na projektu ob uporabi metodologije Scrum. Aplikacija naj bi poleg vnosov in urejanja podatkov projekta omogočala tudi prikaz rezultatov meritev v obliki tabel, pa tudi v obliki raznih diagramov.

V teoretičnem delu diplomske naloge sem se usmeril v predstavitev metodologije Scrum, opis vpeljave meritev v to metodologijo in nato bolj detajlno predstavitev uporabniških zgodb, ki so ena od osnovnih struktur metodologije. V zadnjem delu sem predstavil še funkcionalnosti implementirane aplikacije in nekaj primerov njene uporabe.

Ključne besede:

Scrum, uporabniške zgodbe, meritve v programski opremi, agilne metodologije, razvoj aplikacij

Abstract

For this thesis I made an application that will help manage projects using Scrum methodology. The objective was to make an application that students will use to work on their projects when using Scrum methodology. The application should enable managing project data by displaying measurement results in tabular form and also in the form of various diagrams.

The theoretical part of the thesis is focused on presenting Scrum methodology by describing the introduction of measurements in methodology, and then giving a more detailed presentation of user stories that are one of the basic structures of the methodology. In the last part I have introduced the functionality of the implemented application and some examples of its use.

Key words:

Scrum, user stories, software metrics, agile methodologies, application development

Poglavje 1

Uvod

Agilne metodologije se vse bolj uporabljajo pri razvoju programske opreme. Ena izmed tistih, ki se je v zadnjih letih najbolj razširila, je metoda *Scrum* [1]. V raziskavi, ki jo je naredilo podjetje *VersionOne* leta 2010 [6], vidimo, da že več kot 58 odstotkov uporabnikov agilnih metodologij razvoja uporablja ravno metodo Scrum. Ker v času pisanja diplome ni obstajala nobena prosto dostopna programska oprema za vodenje projekta po metodi Scrum, ki bi omogočala prikaz indikatorjev, opisanih v nadaljevanju, smo se odločili, da bomo to naredili mi.

Poleg vstavljanja in urejanja osnovnih podatkov, ki so definirani po metodi Scrum, naj bi aplikacija omogočala tudi prikaz rezultatov našega dela na projektu. Odločili smo se prikazati nekaj rezultatov meritev, ki temeljijo na praksah modela CMMI (*Capability Maturity Model Integration*) [4]. V obliki diagramov bomo tako prikazali *terminski indeks* (angl. *Schedule performance index - SPI*), *stroškovni indeks* (angl. *Cost performance index - CPI*), *indeks učinkovitosti* (angl. *Work effectiveness - WE*), *diagram za analizo trenda* (angl. *Burndown chart*) in *razmerje med planiranim in dejanskim delom* v obliki tortnega diagrama. Več o tem, kaj posamezna meritev je, kasneje.

V drugem poglavju, ki sledi, bomo najprej opisali metodo Scrum. Ta je dokaj enostavna, tako da nima veliko pravil. Verjetno je ravno to razlog za njen uspeh, saj njeni uporabniki, razvijalci, ne porabijo veliko časa za uvajanje. Naslednje poglavje bomo začeli z opisom, kako smo se lotili vpeljave meritev v metodo Scrum, nadaljevali pa s tem, kako so meritve realizirane. V četrtem poglavju bomo napisali več o *uporabniških zgodbah*. Te so osrednja in najpomembnejša struktura metode Scrum. V naslednjem poglavju bomo na kratko opisali podatkovni model, v zadnjem, šestem poglavju, pa opisali funkcionalnosti implementirane aplikacije.

Poglavje 2

Metoda Scrum

Zaradi poplave različnih orodij, arhitektur, komponent, podatkovnih baz ter razmaha interneta v današnji informacijski tehnologiji se kompleksnost razvoja programske opreme eksponentno veča. Posledica tega je vse večja nepredvidljivost v tem procesu. Metode, kot je Scrum, poskušajo kljub kaosu razvoj programske opreme obdržati pod kontrolo.

Ravno kontrola oziroma uporaba meritev, s katerimi izvajamo kontrolo, je ključni element do uspeha.

Scrum skrajša čas povratne informacije, ki jo dobimo od stranke, med seznamom želj in implementacijo ter med investicijo in donosom investicije. Če je sistem enostaven, ni težko vedeti, kaj hočemo, če pa je sistem kompleksen in se zahteve s časom spreminjajo ter tehnologija napreduje, se agilen sistem, kot je Scrum, dosti bolje obnese kot sistem z izrazitim načrtovanjem vnaprej.

Scrum je sestavljen iz 30-dnevnih ciklov, na koncu katerih je produkt pripravljen za produkcijo. Če že vse vemo in nimamo več ničesar odkriti, ne rabimo uporabljati Scruma. Če pa se moramo o problemu še kaj naučiti, nas Scrum z vztrajnostjo o izdajah inkrementov produkta nauči o pravilnosti in kompletnosti opravljenega. Ob delu lahko mislimo, da naš pristop deluje, vendar se na koncu izkaže, da je bil popolnoma napačen. To izvemo šele, ko je programska oprema testirana in integrirana v produkcijo. Scrum nas sili, da to opravimo za vsak cikel, oziroma čim pogosteje.

Dodaten razlog, zakaj Scrum deluje, je zato, ker skupaj deluje več ljudi in je tako večja verjetnost, da bo nekdo opazil, če gre kaj narobe.

Scrum spremeni majhne skupine v kovalce svoje usode. Skupina sprejme izziv in je nato sama odgovorna ugotoviti način, kako izziv rešiti. Tako je lahko veliko bolj kreativna in načrtuje stvari, ki vnaprej ne bi mogle biti načrtovane.

„Majhnost“ skupine spodbuja člane k sodelovanju in k občutku pripadnosti

ter k temu, da sami odločajo o svoji usodi. Počutijo se, kot da imajo nadzor v svojih rokah, kar je tudi res. Ko si člani delijo skupen cilj, v katerega vsak verjame, bodo tega tudi dosegli, če jim bo le na voljo dovolj virov.

Prednost Scruma nasproti tradicionalnim metodam, kjer sistem najprej popolnoma analizirajo in začrtajo, lahko pokažemo tudi na naslednjem primeru. Recimo, da imamo sistem, ki bo nekemu podjetju prinesel 100.000 EUR dohodka v prvih dveh letih po implementaciji. Z uporabo tradicionalnih metod bi razvoj trajal eno leto, stal pa bi 40.000 EUR. Iterativni pristop pa nam omogoča selektivni in inkrementalni razvoj in implementacijo funkcionalnosti sistema. V primeru, da za upravljanje razvoja uporabljamo eno izmed agilnih metod, bi na primer proces potekal na naslednji način:

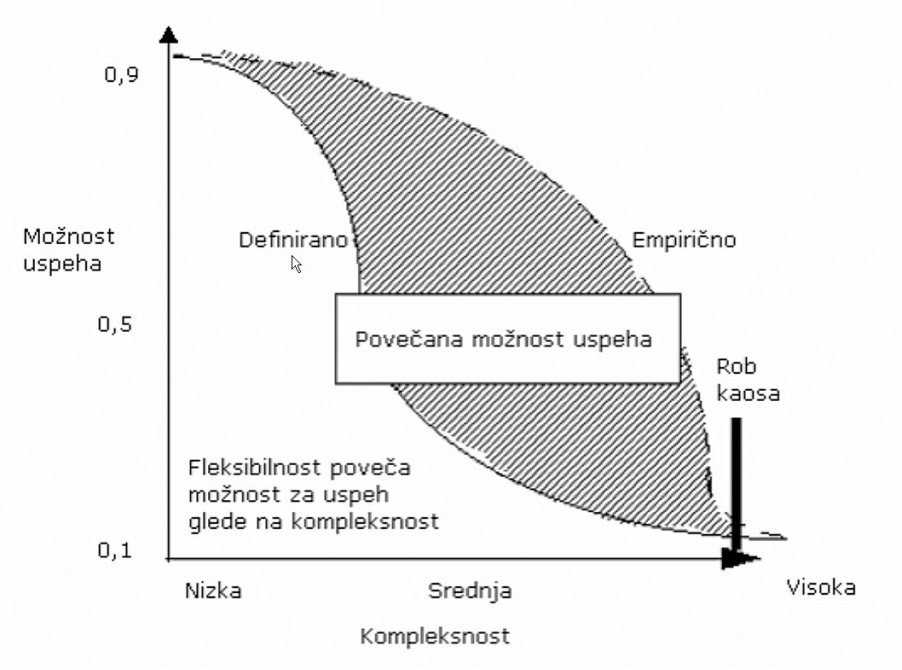
- Izdela se seznam vseh funkcionalnosti, ki jim priredimo neko prioriteto.
- Razdelimo seznam funkcionalnosti na dva dela: prvega ocenjujemo, da bo pripravljen pol leta po začetku razvoja.
- Z uporabo iterativnega in inkrementalnega razvoja dokončamo prvo izdajo po pol leta za 20.000 EUR. Tako lahko dobiček stranke prične rasti že čez pol leta.

V tem primeru vidimo, da je stranka lahko začela dvigati dobiček, ki ji bo pokril stroške investicije pol leta prej, kot bi drugače, prav tako pa ima čez pol leta možnost, da se odloči, ali je pripravljena potrošiti še 20.000 EUR za ostale funkcionalnosti.

2.1 Pregled

Razvoj programske opreme je zelo kompleksen proces. Scrum je oblikovan tako, da strmi h kreiranju uporabnih produktov iz kompleksnih problemov. V preteklosti, v zadnjih petnajstih letih, je bil uporabljen že na tisočih projektih. Temelji na teoriji industrijskega procesnega krmiljenja, ki temelji na mehanizmih, kot so samoorganizacija.

Po eni strani je Scrum zelo enostaven. Zelo malo je namreč praks in pravil, ki so povrhu še zelo enostavna. Po drugi strani pa je lahko enostavnost Scrum procesa zavajajoča. Scrum ni natančno določen proces. Ne opiše točno, kaj početi ob vsaki okoliščini. Uporablja se pri projektih, kjer zaradi kompleksnosti ne moremo natančno napovedati njihovega poteka. Scrum ponuja samo ogrodje in prakse, ki ponujajo pregled nad projektom in strmijo k načrtovanemu cilju. Večina ljudi, odgovornih za upravljanje s projekti, je



Slika 2.1: Graf odvisnosti uspeha od kompleksnosti

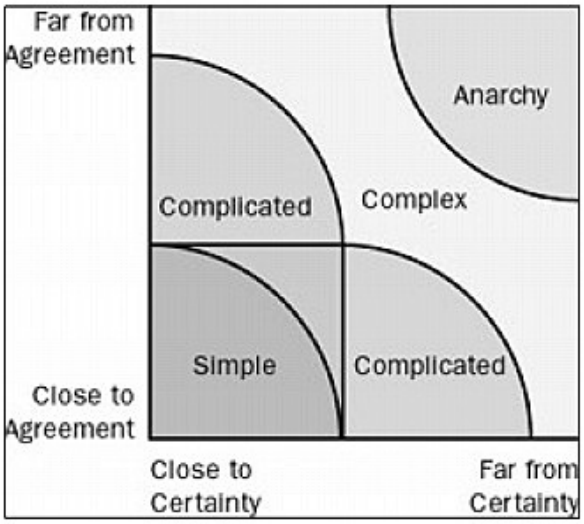
naučenih deterministčnih pristopov vodenja projekta, ki uporabljajo podrobne načrte in urnike dela. Scrum je pravo nasprotje. Za razliko od deterministčnih pristopov, ki projekt blokirajo pred začetnim zagonom, Scrum pokaže, kako optimalno voditi proces, ki se razvija s tem, ko se projekt nadaljuje.

2.2 Razvoj kompleksne programske opreme

Za oceno kompleksnosti razvoja programske opreme se omejimo samo na tri dimenzije: zahteve, tehnologijo in ljudi.

Skoraj nikoli se ne srečujemo z enostavnimi zahtevami. Primer enostavne zahteve je, ko imamo enega samega naročnika, ki je hkrati edini uporabnik produkta in si lahko vzame dovolj časa z razvijalcem, da točno določita, kaj razviti, ob tem pa se tudi njegove zahteve s časom ne bodo spreminjale. Pogosteje se srečujemo z zahtevami, kjer imamo veliko uporabnikov, ki nimajo dovolj časa za določitev točne specifikacije in velikokrat tudi še točno ne vedo, kaj bi radi. Njihove zahteve se tudi pogosto spreminjajo skozi čas.

Tudi pojem enostavna tehnologija le redko zasledimo pri razvoju programske opreme. Pogosto gre za skupek več nezanesljivih tehnologij, ki komunici-



Slika 2.2: Kompleksnost razvoja

rajo med seboj in tako povečujejo verjetnost napake in komplekstnost razvoja.

Na sliki 2.2 je prikazano, kako kompleksnost narašča in postaja neobvladljiva z naraščanjem kompleksnosti obeh mer, ki smo jih do sedaj predstavili. Če pademo v neobvladljiv del grafa, pomeni, da je potrebno nekatere kompleksnosti rešiti, preden nadaljujemo z delom.

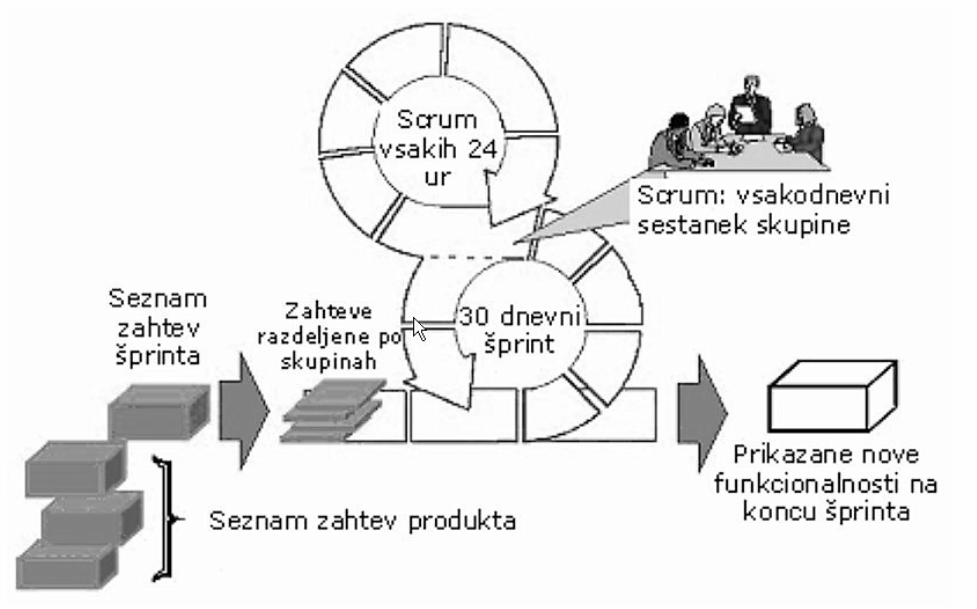
Tretja dimenzija kompleksnosti so ljudje, ki pri razvoju sodelujejo. Vsi imajo drugačne spretnosti, izkušnje, poglede, odnose in predsodke. Poleg tega je tudi vsakodnevno razpoloženje odvisno od drugih zunanjih dejavnikov.

2.3 Ogrodje procesa Scrum

Scrum je iterativni, inkrementalni proces, kot je prikazano na sliki 2.3. Spodnji krog predstavlja iteracijo razvoja, katere izhod je inkrementiran produkt. Zgornji krog pa predstavlja dnevni pregled stanja, ki se izvaja med iteracijo in kjer se člani skupine dobijo na 15-minutnem sestanku. Tu pregledajo dejavnosti med seboj in po potrebi naredijo potrebne prilagoditve. Iteracija se ponavlja, dokler je projekt financiran.

Proces poteka na način:

- Na začetku vsake iteracija skupina pregleda, kaj mora narediti. Nato izbere zahteve, za katere verjame, da jih lahko v času iteracije naredi in



Slika 2.3: Ogrodje procesa Scrum

vgradi v potencialni produkt, ki lahko gre v produkcijo.

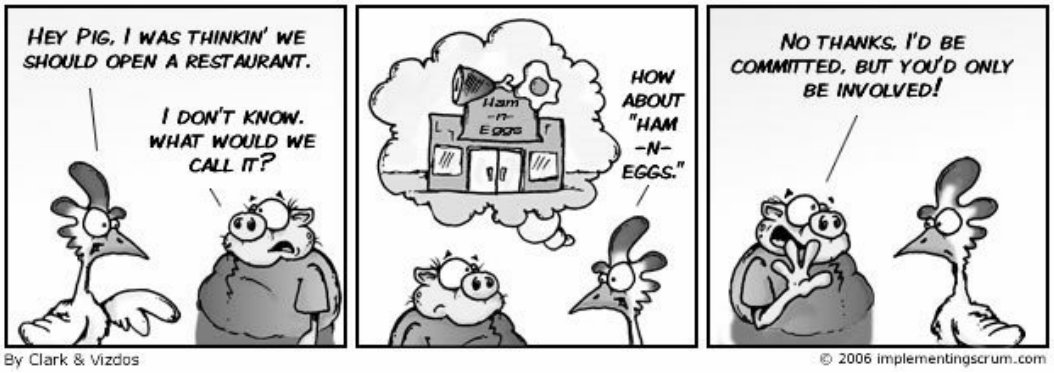
- Vsaka skupina ima proste roke med samo iteracijo.
- Na koncu vsake iteracije vsaka skupina predstavi na novo vključene funkcionalnosti tako, da lahko nadzorniki preverijo celotno funkcionalnost in po potrebi časovno prilagodijo projekt.

Skupina na začetku iteracije pregleda zahteve in razpoložljive tehnologije ter oceni svoje sposobnosti in zmogljivosti. Na podlagi tega nato določi, kako bodo dano zahtevo izvršili. Ob naletu na nove težave in kompleksnosti lahko dnevno pristop spreminjajo. Tako je skupini prepuščena izbira, kaj in kako mora biti opravljeno. Ta kreativni proces je jedro Scrum produktivnosti.

2.3.1 Vloge pri procesu Scrum

Scrum implementira to iterativno, inkrementalno ogrodje skozi tri vloge: produktni vodja (angl. *Product Owner*), razvojna skupina (angl. *Team*) in skrbnik metodologije (angl. *Scrum master*).

- **Produktni vodja**: Odgovoren je za predstavitev interesov vseh, ki imajo delež v projektu. Produktni vodja inicializira projekt s tem, ko



Slika 2.4: Pigs and chickens

poda splošne zahteve in načrt izdaje, odgovoren pa je tudi, da se najpomembnejše funkcionalnosti izvršijo najprej.

- **Razvojna skupina**: Odgovorna je za razvoj funkcionalnosti. Skupina ja samoupravna, samoorganizirana in navzkrižno funkcionalna. Ugotoviti mora način, kako spremeniti seznam zahtev v nove funkcionalnosti v okviru iteracije. Člani skupine so skupaj odgovorni za uspeh projekta kot celote.

- **Skrbnik metodologije**: Odgovoren je za Scrum proces, za učenje Scrum procesa vseh, ki so vključeni v projekt, za implementiranje Scrum procesa in za zagotavljanje, da vsi sledijo pravilom in praksam Scrum procesa.

Scrum jasno razlikuje ljudi, ki nastopajo v eni od teh treh vlog in ostalimi ljudmi, ki so nekako povezani s projektom. Uporabljata se izraza prašiči in kokoši (angl. *Pigs and chickens*), slika 2.4. To razlikovanje je pomembno, saj lahko tako povečamo produktivnost. Medtem ko so eni predani projektu in za njega tudi odgovorni (prašiči), so drugi v projekt samo vključeni (kokoši).

2.3.2 Tok pri procesu Scrum

Scrum projekt se začne z vizijo sistema, ki naj bi bil razvit. Vizija je lahko na začetku nejasna in navedena z marketinškimi pogoji, ampak bo postajala bolj jasna s tem, ko se bo projekt razvijal. Produktni vodja je odgovoren za tvorbo plana, ki bo kar najbolje vrnil investicijo, in del tega plana je tudi *seznam zahtev* (angl. *Product backlog*). Seznam zahtev je seznam funkcionalnih in nefunkcionalnih zahtev, ki predstavljajo začetno vizijo projekta. Seznam zahtev je prioritiziran, tako da so funkcionalnosti, ki bodo najverjetneje prinesle

vrednost, na vrhu seznama in razdeljene v predlagane izdaje (angl. *release*). Prioritiziran seznam zahtev je štartna točka, njegova vsebina pa se začne spreminjati s trenutkom, ko začnemo delati na projektu (kot je bilo pričakovano).

Vse delo je opravljeno v iteracijah. Vsaka iteracija traja 30 zaporednih koledarskih dni. Število dni (dolžina iteracije) je lahko tudi drugačno, tako se pogosto uporabljajo tudi krajše dolžine.

Vsaka iteracija se začne s **sestankom načrtovanja iteracije** (angl. *Sprint planning meeting*), kjer *produktni vodja* in *razvojna skupina* skupaj določijo, kaj bo opravljeno v naslednji iteraciji. Produktni vodja izbere zahteve z najvišjo prioriteto in nato skupina pove, kaj je v okviru iteracije možno narediti. Sestanek načrtovanja iteracije ne sme biti daljši od osmih ur. Čas je omejen, da se izognemu predolgemu premlevanju o tem, kaj je mogoče in kaj ne. Cilj je začeti delati, ne razmišljati, kako delati.

Sestanek načrtovanja iteracije je sestavljen iz dveh delov:

- V prvih štirih urah produktni vodja predstavi razvojni skupini zahteve z najvišjo prioriteto in od skupin sprejema vprašanja o vsebini, namenu in njihovem pomenu. Pred potekom teh štirih ur skupina določi, koliko v okviru predstavljenega lahko naredi.

- Druge štiri ure razvojna skupina porabi za planiranje iteracije. Zahteve, ki se izberejo v okviru tega sestanka, se zapišejo v seznam zahtev iteracije (angl. *Sprint Backlog*). Nove naloge v seznamu zahtev iteracije se pojavljajo s tem, ko se iteracija izvaja. Na začetku drugih štirih ur se je iteracija že začela.

Vsak dan se razvojna skupina dobi na **15-minutnem dnevnem Scrum sestanku** (angl. *Daily Scrum meeting*). Tukaj vsak član skupine odgovori na tri vprašanja [1]:

- Kaj si naredil na tem projektu od zadnjega dnevnega sestanka?

- Kaj planiraš narediti na tem projektu do naslednjega dnevnega sestanka?

- Kaj te ovira pri doseganju tvojih pričakovanj na tem projektu, na tej iteraciji?

Namen tega sestanka je dnevno usklajevanje del vseh članov razvojne skupine.

Na koncu iteracije je še **sestanek pregleda iteracije** (angl. *Sprint review meeting*). To je štiriurni, časovno omejen sestanek, kjer razvojna skupina

zainteresiranim uporabnikom predstavi, kaj je razvila med to iteracijo. Na sestanku je lahko navzoča še kakšna druga oseba, ki je kako drugače povezana s projektom. Namen sestanka je določiti, kaj naj skupina naredi v naslednji iteraciji.

Za sestankom pregleda iteracije in pred sestankom načrtovanja naslednje iteracije, se odvija še **Retrospektivni sestanek Sprinta** (angl. *Sprint retrospective meeting*) s skupino. Na tem triurnem, časovno omejenem sestanku, *skrbnik metodologije* spodbudi razvojno skupino k izboljšavam. Tema sestanka je [1]:

- Kaj je bilo uspešno?
- Kaj ni bilo uspešno?
- Kaj bi lahko izboljšali v naslednji iteraciji?

To je še posebej pomembna priložnost za skupino, da se osredotoči na svoje splošne učinkovitosti in opredeli strategije za izboljšanje svojih postopkov.

2.4 Struktura procesa Scrum

2.4.1 Seznam zahtev (angl. *Product Backlog*)

Zahteve sistema, razvrščene po prioriteti, so podane v seznamu zahtev. Za vsebino tega seznama je odgovoren *produktni vodja*. Seznam zahtev ni nikoli končen in je samo začetna ocena o zahtevah projekta. Razvija se s tem, ko se projekt razvija. Je dinamičen.

2.4.2 Seznam zahtev iteracije (angl. *Sprint Backlog*)

Definira tiste zahteve iz seznama zahtev, ki jih skupina izbere v drugem delu sestanka načrtovanja iteracije za izvedbo v trenutni iteraciji. Vsak element v seznamu zahtev je razdeljen še na naloge (angl. *Task*). Vsaka naloga naj ne bi trajala več kot 16 ur.

2.4.3 Izdaja (angl. *Release*)

Scrum zahteva, da je na koncu vsake iteracije pripravljena verzija produkta, ki gre lahko v produkcijo. To zahteva, da je projekt temeljito testiran, dobro

strukturiran in dobro napisan. Zahteva tudi, da je funkcionalnost dokumentirana. Ker pa se po koncu vsake iteracije ne odločimo za novo verzijo produkta, govorimo o izdajah. Tako lahko izdaja obsega več iteracij.

Poglavje 3

Vpeljava meritev v metodo Scrum

Za boljše spremljanje poteka projekta so nam v veliko pomoč razne meritve. Tako smo se poleg osnovnih meritev, ki so definirane že s samo metodo Scrum, odločili vpeljati še nekaj izpeljanih meritev. Te bodo v večini prikazane v obliki stolpčnih in tortnih diagramov. Nekatere od teh meritev so *terminski indeks*, *stroškovni indeks*, *indeks učinkovitosti* itd. Več o njih kasneje.

3.1 Vzpostavitev merskih ciljev

CMMI [4] predstavlja različne primere ciljev merjenja, kot so: zmanjšaje časa dostave, zmanjšanje skupnih stroškov življenjskega cikla, zagotavljanje določene funkcionalnosti v celoti itd: Vsi ti cilji so lahko merjeni skozi stalno spremljanje procesa. Tako implementacija CMMI meritev, predstavljena v nadaljevanju, predpostavlja, da je glavni cilj meritev spremljanje in izboljšanje procesa razvoja programske opreme.

Uspešnost procesa razvoja programske opreme lahko enačimo z zadovoljstvom interesnih skupin. Tako moramo upoštevati vidike vseh skupin, ki so navzoče v procesu. Najbolj smo uspešni, ko so vse skupine zadovoljne. Meritve pa morajo biti tako kvantitativne kot kvalitativne. Ob zaključku smo prišli do štirih interesnih skupin [3]:

- **Vodstvo**: Zasleduje tradicionalne vidike razvoja programske opreme, kot so porabljen čas, stroški in kvaliteta.
- **Člani razvojne skupine**: Glavni cilj članov skupine je zadovoljstvo pri delu. Člani so najbolj produktivni, če imajo dobre delovne pogoje.

- **Skrbnik metodologije**: Glavni cilj skrbnika metodologije je olajšanje uporabe Scruma in kreiranje pogojev za učinkovit razvoj.
- **Naročnik**: Glavni cilj naročnika je njegovo zadovoljstvo.

3.2 Določitev meritev

V skladu s CMMI obstajata dve vrsti meritev. „Osnovne“ in „izpeljane“. Medtem ko dobimo podatke za osnovne meritve direktno z merjenjem, so podatki za izpeljane meritve dobljeni tipično iz več osnovnih meritev.

Prvotno je imel Scrum samo eno osnovno meritev: oceno količine preostalega dela, ki ga je potrebno opraviti, da končamo z zahtevo oziroma z nalogo. Na nivoju naloge se ta meritev izvaja dnevno, medtem ko se na nivoju zahteve izvaja na začetku vsake iteracije. Na podlagi teh meritev lahko dobimo *diagram za analizo trenda iteracije* (angl. *Burndown chart*), ki prikazuje preostanek dela skozi čas.

Da bi lahko merili zadovoljitev vseh interesnih skupin, ki smo jih uvedli v prejšnjem poglavju, smo uvedli nekatere dodatne meritve [3]:

- Cilj: Pravočasne informacije o uspešnosti projekta.
  - Preostalo delo na dan d za vsako nalogo v seznamu zahtev iteracije.
  - Opravljeno delo na dan d za vsako nalogo v seznamu zahtev iteracije.
- Cilj: Izboljšanje kakovosti
  - Število odkritih napak na sestanku pregleda iteracije. Za vsako zahtevo posebej.
  - Število napak, ki nam jih sporoči uporabnik v določenem času po oddaji kode v produkcijo. Za vsako zahtevo posebej.
  - Velikost kode. Za vsako zahtevo posebej.
  - Število vseh zahtev, na katerih smo delali med iteracijo.
  - Število vseh zahtev, ki smo jih dokončali med iteracijo.
  - Število vseh nalog v iteraciji.
  - Število vseh nalog, ki smo jih dokončali v iteraciji.
- Cilj: Zadovoljstvo pri delu

  - Rezultati raziskave zadovoljstva članov skupine.

- Cilj: Učinkovito odpravljanje ovir
  - Število ovir, na katere smo naleteli na določeni nalogi/zahtevi/iteraciji.
  - Datum, kdaj smo naleteli na oviro.
  - Datum, kdaj smo oviro rešili.

- Cilj: Zadovoljstvo strank
  - Rezultati raziskave zadovoljstva kupcev na koncu iteracija/izdaje.

3.2.1 Zbiranje podatkov

Vse te meritve, ki smo jih pravkar določili, lahko zbiramo med predpisanimi sestanki Scruma. Edina izjema je število napak, ki nam jih sporoči uporabnik v določenem času po oddaji kode v produkcijo.

Meritve po sestankih [3]:

- Sestanek načrtovanja iteracije.
  - Dolžina iteracije (število delovnih dni v iteraciji).
  - Število članov v skupini.
  - Odstotek udeležbe vsakega člana v projektu.
  - Cena ure dela vsakega člana.

- Dnevni Scrum sestanek.
  - Preostalo delo na dan d za vsako nalogo v seznamu zahtev iteracije.
  - Opravljeno delo na dan d za vsako nalogo v seznamu zahtev iteracije.
  - Ovire, na katere smo naleteli.

- Sestanek pregleda iteracije
  - Število odkritih napak na sestanku (za vsako zahtevo posebej).
  - Rezultati raziskave zadovoljstva kupcev na koncu iteracije/izdaje.

- Retrospektivni sestanek Sprinta

- Velikost kode (za vsako zahtevo posebej).
- Število vseh zahtev, na katerih smo delali med iteracijo.
- Število vseh zahtev, ki smo jih dokončali med iteracijo.
- Število vseh nalog v iteraciji.
- Število vseh nalog, ki smo jih dokončali v iteraciji.
- Rezultati raziskave zadovoljstva članov skupine.

3.2.2 Izpeljane meritve

Z grupiranjem več osnovnih meritev dobimo izpeljane meritve ali indikatorje, ki so nam v pomoč pri analizi procesa razvoja programske opreme.

Za dosego cilja *Pravočasne informacije o uspešnosti projekta* uporabljamo nasledje indikatorje:

- **Indeks učinkovitosti** - se nanaša na razmerje med zmanjšanjem preostalega dela in opravljenim delom. Idealno bi bilo, da bi zmanjšanje preostalega dela med dnevi *d1* in *d2* bilo večje oziroma enako, kot opravljeno delo med tema dvema dnevoma. Tako je cilj obdržati to vrednost nad 1, čeprav vrednosti dosti višje kot 1 pomenijo slabo planiranje.
- **Terminski indeks** - se nanaša na razmerje med prisluženo vrednostjo (angl. *Earned value* - Vrednost vseh zaključenih nalog) in planirano vrednostjo (prvotna ocena vrednosti vseh nalog v določeni časovni točki). Tudi cilj tega indikatorja je vrednost nad 1. Vrednost nad 1 pomeni, da projekt prehiteva predvideni rok.
- **Stroškovni indeks** - se nanaša na razmerje med pridobljeno vrednostjo (merjeno v valuti) in dejanskimi stroški. Tudi cilj tega indikatorja je vrednost nad 1, kar pomeni, da je cena del skladna s planom oziroma nižja.

Za dosego cilja *Izboljšanje kakovosti* uporabljamo naslednje indikatorje:

- **Gostota napak** - število napak na tisoč vrstic kode.
- **Stroški predelave** - zmnožek števila ur, porabljenih na predelavi s ceno ure dela razvijalca.

- **Izpolnjevanje ciljev** - razmerje med številom zaključenih nalog v iteraciji in številom vseh nalog v iteraciji ali razmerje med številom zaključenih zahtev v verziji in številom vseh zahtev v verziji.

Za dosego cilja *Zadovoljstvo pri delu* uporabljamo naslednje kvalitativne in kvantitativne indikatorje:

- Povprečno število projektov, na katerih razvijalci delajo vzporedno.
- Kakovostna ocena o delovnih pogojih (komunikacija, timsko delo, fizično neugodje, psihično dobro počutje, delovne obremenitve, nadzor, priložnosti za rast ...).

Za dosego cilja *Učinkovito izogibanje ovir* uporabljamo naslednje indikatorje:

- Povprečno število ovir na nalogo/zahtevo/iteracijo/skupino.
- Povprečni čas za rešitev ovir.

3.2.3 Izračuni

V prejšnjem odseku smo podali meritve, kot so *terminski indeks* in *stroškovni indeks*. Ker Scrum ne predpostavlja nekega časovnega načrta, predpostavljamo, da je količina dela, ki mora biti opravljena do določene časovne točke, proporcionalna s časom, ki je pretekel od začetka iteracije. Meritve *preostalo delo* in *opravljeno delo* za vsak delovni dan nam omogočajo natančen izračun *deleža prislužene vrednosti* (angl. *Earning rule*), ki je definiran za vsako nalogo j, za vsak dan d *iteracije*. Izračunan je kot razmerje med količino že opravljenega dela na nalogi in celotno količino potrebnega dela [3]:

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

$WS_{i,j}$ označuje količino že opravljenega dela (angl. *Work spent*) za nalogo j na dan i, medtem ko $WR_{d,j}$ označuje količino preostalega dela za nalogo j na dan d.

Z uporabo *deleža prislužene vrednosti* izračunamo ***terminski indeks*** na dan d po naslednji formuli [3]:

$$SPI_d = \frac{\sum_{j=1}^{n} ER_{d,j} WR_{init,j}}{\sum_{j=1}^{n} WR_{init,j}} \frac{SL}{d}$$

kjer $WR_{init,j}$ označuje začetno oceno potrebnega dela na nalogi j, SL dolžino iteracije (angl. *Sprint length*) in d dan, za katerega računamo indikator.

Medtem ko računanje *terminskega indeksa* dovoli, da je pridobljena vrednost izražena v poljubni enoti (mi uporabljamo prvotno oceno v urah), računanje ***stroškovnega indeksa*** zahteva, da je pridobljena vrednost izražena v valuti [3].

$$CPI_d = \frac{\sum_{j=1}^{n} ER_{d,j} WR_{init,j} CEH_j}{\sum_{i=1}^{d} \sum_{j=1}^{n} WS_{i,j} CEH_j}$$

kjer CEH_j predstavlja ceno ure razvijalca, ki dela na nalogi j, d pa število preteklih dni (delovnih) od začetka iteracije (angl. *Days elapsed*).

Poglavje 4

Uporabniške zgodbe

Za specifikacijo zahtev naročnika se v metodi Scrum, vedno bolj uveljavljajo *uporabniške zgodbe* (angl. *User stories*) [2]. Začeli bomo s predstavitvijo, kaj uporabniške zgodbe so ter kako so uporabljene, nadaljevali pa z več podrobnostmi, kako uporabniške zgodbe pisati in kako napisati teste, s katerimi testiramo uspešnost implementacije uporabniške zgodbe.

4.1 Pregled

4.1.1 Kaj je uporabniška zgodba

Uporabniška zgodba predstavlja funkcionalnost, ki je pomembna za končnega uporabnika oziroma kupca programske opreme. Zgodbe so sestavljene iz treh vidikov:

- Pisni opis zgodbe, ki se uporablja za načrtovanje in kot opomnik.
- Pogovor o zgodbi, ki nam služi za natančnejšo opredelitev zgodbe.
- Testi, ki posredujejo podatke, ki jih uporabljamo za ugotavljanje, kdaj je zgodba končana.

Ker so ponavadi te zgodbe napisane na karticah, so ti trije vidiki poimenovani *Kartica, Pogovor in Potrditev*. Kartice so najvidnejša oblika uporabniške zgodbe, vendar niso najpomembnejše. Medtem ko kartica vsebuje besedilo zgodbe, so podrobnosti razjasnjene skozi pogovore in zabeležene v potrditvi.

Primeri uporabniških zgodb, ki jih bomo podajali, se nanašajo na spletno posredovalnico dela, ki naj bi delodajalcem omogočala prikaz oglasa za delovno mesto, zainteresiranim kandidatom pa oddajo prošnje za to delovno mesto.

Primer kartice uporabniške zgodbe je lahko: *Uporabnik lahko vstavi svoj življenjepis.* Ker pa uporabniške zgodbe predstavljajo funkcionalnosti, ki so vrednotene s strani končnega uporabnika, nasledji primeri zgodb niso primerni:

- *Programska oprema bo napisana v programskem jeziku C++.*
- *Program se bo povezal z bazo podatkov preko bazena povezav.*

Ta dva primera ne predstavljata dobre uporabniške zgodbe, ker končnega uporabnika ne zanima, v kakšnem programskem jeziku je aplikacija napisana in ga na splošno ne zanimajo tehnične podrobnosti aplikacije.

Ob branju teh zgodb nam lahko pride na misel: „Počakajte, uporaba bazena povezav je zahteva v našem sistemu“. Ob tem moramo pomisliti, da morajo uporabniške zgodbe biti napisane tako, da jih lahko uporabniki vrednotijo. Obstajajo načini, kako napisati zgodbe tako, da jih lahko uporabnik vrednoti. Kako, bomo videli kasneje.

Zakaj uporabniške zgodbe piše stranka

Stranka piše zgodbe raje kot razvijalec iz dveh glavnih razlogov. Prvi razlog je, da mora biti vsaka zgodba napisana v poslovnem jeziku, ne pa v tehničnem žargonu, tako da lahko stranka prioritizira zgodbe, kar potrebuje pri vključevanju zgodb v iteracijo. Drugi razlog pa je, da je kot glavni vizionar proizvoda v najboljšem položaju, da opiše obnašanje produkta.

4.1.2 Kje so podrobnosti

Da bi lahko začeli s kodiranjem in testiranjem, potrebujemo več podrobnosti o uporabniški zgodbi. Enostavno je reči *Uporabnik lahko išče prosta delovna mesta.* Kje so pa odgovori na vsa neodgovorjena vprašanja, kot so:

- Katere atribute lahko uporabniki uporabljajo pri iskanju? Mesto? Naziv delovnega mesta?
- Mora biti uporabnik član spletnega mesta?
- Lahko iskalne parametre shranimo?
- Katere informacije prikažemo o najdenih delovnih mestih?

Veliko podrobnosti se lahko izrazi kot dodatne zgodbe.

Kakšne naj bodo dolžine zgodb? Dolžina zgodbe naj bi bila taka, da bi jo lahko izvedli in testirali v obdobju od polovice dneva do dveh tednov. Tukaj upoštevamo, da na zgodbi dela en oziroma največ dva razvijalca. Predolge zgodbe lahko razdelimo na več krajših zgodb. Na primer, zgodbo *Uporabnik lahko išče delovna mesta* lahko razdelimo na zgodbe:

- Uporabnik lahko išče delovna mesta po atributih, kot so: naziv delovnega mesta, ime podjetja.
- Uporabnik lahko vidi informacije o vsakem delovnem mestu, ki ga vrne iskanje.
- Uporabnik lahko vidi podrobne informacije o podjetju, za katero je razpisano delovno mesto.

Z delitvijo zgodbe pa ne nadaljujemo, ko imamo zgodbo, ki pokriva vse podrobnosti. Na primer, zgodbe *Uporabnik lahko vidi informacije o vsakem delovnem mestu, ki ga vrne iskanje*, ni potrebno deliti na zgodbe:

- Uporabnik lahko vidi opis delovnega mesta.
- Uporabnik lahko vidi lokacijo delovnega mesta.

Raje kot pisati vse te podrobnosti v uporabniške zgodbe, lahko do teh podrobnosti pridemo s pogovorom, na točki v času, ko postanejo te podrobnosti pomembne. Nič ni narobe, če na kartico uporabniške zgodbe dodamo nekaj pripomb, ki temeljijo na razpravi. Na primer, na kartico *Uporabnik lahko vidi informacije o vsakem delovnem mestu, ki ga vrne iskanje*, lahko dodamo pripombo, *Marko pravi, da prikažemo opis, lokacijo in razpon možne plače.* Vendar je ključen pogovor, ne pa pripombe na kartici uporabniške zgodbe. Niti razvijalec, niti stranka, ne smeta čez tri mesece pokazati na kartico in reči „Glej, saj sem jaz tako rekel tam“. Zgodbe niso pogodbene obveznosti. Kot bomo videli kasneje, so sporazumi dokumentirani kot testi, ki dokazujejo, da je bila zgodba razvita pravilno.

Opisi testov naj bi bili kratki in nepopolni. Dodaja ali odstranjuje se jih lahko v vsaki časovni točki med projektom. Cilj je posredovati dodatne informacije o uporabniški zgodbi, tako da razvijalci vedo, kdaj so končali z delom na njej.

4.1.3 Skupina stranke

Na idealnem projektu bi imeli samo eno osebo, ki opredeljuje prednostne zgodbe za razvijalce, odgovarja na njihova vprašanja, uporablja programsko opremo, ko je končana in piše vse zgodbe. To je skoraj nemogoče, zato vzpostavimo skupino stranke. Ta vsebuje tiste ljudi, ki zagotavljajo, da bo programska oprema odgovarjala ciljnim uporabnikom. To pomeni, da ta skupina lahko vsebuje testerje, vodjo produkta, prave uporabnike in oblikovalce.

4.1.4 Kaj so sprejemni testi

Sprejemno testiranje (angl. *acceptance testing*) je proces preverjanja uporabniških zgodb. Testi preverijo, če implementacija zgodb deluje na tak način, kot skupina stranke pričakuje. Ko se iteracija prične, razvijalci pričnejo s kodiranjem, skupina stranke pa začne pripravljati teste. Odvisno od tehničnega znanja skupine stranke, so lahko ti testi v pisni obliki na hrbtni strani kartice uporabniške zgodbe, oziroma avtomatski testi.

Testi naj bi bili napisani karseda zgodaj v iteraciji. Lahko se jih prične pisati tudi pred začetkom iteracije, s tem da moramo v tem primeru ugibati, katere zgodbe bodo vključene v prihajajočo iteracijo. Zgodnje napisani testi so v veliko pomoč razvijalcem, saj tako pridejo do natančnih pričakovanj skupine stranke.

Vzemimo za primer zgodbo *Uporabnik lahko plača izdelke v svojem nakupovalnem vozičku s kreditno kartico.* Za njo so kasneje napisani naslednji testi:

- Preveri z Visa, MasterCard in American Express. Test mora uspeti.
- Preveri z Diner's Club. Test mora pasti.
- Preveri z Visa debit card. Test mora pasti.
- Preveri s poteklo kartico. Test mora pasti.
- Test z različnimi zneski nakupa (vključno z enim čez mejo kartice).

S tem, ko dobi programer te teste zgodaj, skupina stranke ne izrazi samo pričakovanja, ampak lahko programerja spomnijo na situacije, ki bi jih drugače spregledal.

4.2 Pisanje uporabniških zgodb

Za kreiranje dobrih uporabniških zgodb se moramo osredotočiti na šest stvari [2]:

- neodvisnost
- možnost dogovora
- koristnost za uporabnike ali stranko
- ocenljivost
- majhnost
- primernost za testiranje

4.2.1 Neodvisnost

Kolikor je mogoče, zmanjšamo odvisnosti med zgodbami. Odvisnosti med zgodbami namreč vodijo k težavam pri planiranju in prioritiziranju. Predvidimo, da je stranka določila visoko prioriteto zgodbi, ki je odvisna od zgodbe z nizko prioriteto. Na primer zgodbe:

- *Podjetje lahko plača za objavo oglasa za delo s kartico Visa.*
- *Podjetje lahko plača za objavo dela s kartico MasterCard.*
- *Podjetje lahko plača za objavo dela s kartico American Express.*

Recimo, da razvijalci ocenjujejo, da bo trajalo tri dni, da bodo podprli prvo vrsto kreditne kartice (ne glede na to, katera je), nato pa en dan za vsako od preostalih. S tako odvisnostjo zgodb, kot pri tem primeru, se ne ve, kakšno oceno dati kateri zgodbi. Katera zgodba naj dobi oceno treh dni?

Ko pride do takih odvisnosti, lahko to rešimo na dva načina:

- Z združitvijo odvisnih zgodb v eno večjo, vendar neodvisno uporabniško zgodbo.
- Z drugačnim razbitjem problema na zgodbe.

Združevanje zgodb se v tem primeru obnese dobro, saj je skupna zgodba dolga samo pet dni. Če pa bi bila skupna zgodba dosti daljša, je ponavadi boljši pristop ta, da najdemo drugačno razbitje problema. Na primer:

- Stranka lahko plača s kreditno kartico enega izdajatelja.
- Stranka lahko plača s kreditnimi karticami ostalih izdajateljev.

Če pa zgodb ne želimo združiti in ne najdemo drugačnega primernega razbitja, lahko zgodbi priredimo dve oceni. Ena, če je zgodba izvedena pred drugo odvisno zgodbo, in ena, če je izvedena za njo.

4.2.2 Možnost dogovora

Uporabniške zgodbe niso zapisane pogodbene obveznosti. So samo opisi funkcionalnosti, ki naj bi jih aplikacija podprla, katerih podrobnosti bodo določene skozi vrsto pogovorov med razvijalci in stranko. Tako zgodbe ne vsebujejo vseh podrobnosti. Vendar, če so v času pisanja zgodbe te podrobnosti že poznane, se jih lahko napiše na kartico kot opombe. Izziv predstavlja napisati zgodbe, ki vsebujejo ravno prav podrobnosti.

Primer dobre zgodbe je *Podjetje lahko plača za objavo oglasa za delo s kreditno kartico.* Zgodbi pa je še dodana opomba *Sprejmemo Viso, MasterCard in American Express. Mogoče Discover.* Ko bo razvijalec bral to zgodbo, bo opomnjen, da mora podpreti tri že izbrane kreditne kartice, za četrto pa mora vprašati stranko, če je bila že sprejeta odločitev. Opombe pomagajo razvijalcu in stranki nadaljevati pogovor od tam, kjer se je prejšnjič končal. To naj bi bilo možno tudi, če na zgodbi sodeluje drug razvijalec (ali stranka) kot prej.

Če bi pa bil prejšnji zgodbi, kot opomba, dodan naslednji tekst: *Sprejmemo Viso, MasterCard in American Express. Mogoče Discover. Ob znesku nad 100 vprašaj za ID kartice. Sistem lahko prepozna kartico iz prvih dveh številk številke kartice. Sistem lahko shrani kartico za naslednje uporabe. Uporabnik mora vnesti tudi leto in mesec izteka kartice.* Taka zgodba vsebuje preveč podrobnosti in vsebuje tudi informacije, ki bi bile lahko zapisane kot samostojna zgodba: *Sistem lahko shrani kartico za naslednje uporabe.* Delo na takih zgodbah je zelo težko. Bralec zgodbe lahko misli, da je zgodbi pripisanih dovolj podrobnosti, da pogovor s stranko sploh ni potreben.

Tako je kot opombe zgodbi dobro napisati težave, na katere smo naleteli pri pisanju zgodbe. Te nam služijo tudi kot opozorilo, da moramo o tej zgodbi imeti še razpravo.

Podrobnosti, ki so bile že določene med pisanjem zgodbe, pa napišemo kot teste uporabniške zgodbe. Te ponavadi pišemo na zadnjo stran kartice uporabniške zgodbe, če gre za tako obliko testa.

4.2.3 Koristnost za uporabnike ali stranko

Narobe bi bilo reči, da mora biti vsaka uporabniška zgodba koristna za uporabnika. Veliko projektov vsebuje zgodbe, ki za uporabnika niso pomembne. Moramo razlikovati med uporabnikom (nekdo, ki bo aplikacijo uporabljal) in naročnikom. Vzemimo za primer aplikacijo, ki bo uporabljena v velikem podjetju. Naročnika na primer zanima, da bo aplikacija podpirala možnost centralne konfiguracije, medtem ko končnega uporabnika ta podatek ne zanima.

Podobno se spodnje zgodbe vrednotijo s strani naročnika, končnega uporabnika pa ne zanimajo:

- *Med razvojem programske opreme bodo razvijalci pripravili ustrezno dokumentacijo za ISO 9001 revizijo.*
- *Razvijalci bodo razvili aplikacijo v skladu s CMM stopnjo 3.*

Čemur pa se želimo izogniti, so zgodbe, ki bodo vrednotene samo s strani razvijalca:

- *Vse povezave do podatkovnih baz uporabljajo bazen povezav.*
- *Vsa obravnavanja napak se izvajajo preko skupnih razredov.*

Kot že napisano, so te zgodbe usmerjene k razvijalcu. Velikokrat je možno, da ideja, ki stoji za temi zgodbami, ni slaba. Tako lahko te zgodbe napišemo na način, usmerjen naročniku. Tako dobimo naslednje variante zgornjih zgodb:

- *Več kot petdesetim uporabnikom bi moralo biti omogočeno uporabljanje aplikacije, z licenco za pet uporabnikov za dostop do baze.*
- *Vse napake so predstavljene uporabniku in redno beležene.*

Najboljši način za zagotovitev, da je vsaka zgodba dragocena za uporabnika oziroma kupca je, da kupec (stranka) piše uporabniške zgodbe. Stranki je ponavadi na začetku to početje neprijetno. Večina strank pa začne s pisanjem zgodb, ko verjamejo v koncept, da so zgodbe na kartici samo opomniki za pogovor, ki bo potekal kasneje, ne pa formalne obveze.

4.2.4 Ocenljivost

Za razvijalca je zelo pomembno, da je sposoben oceniti dolžino zgodbe. Torej, koliko časa bo potreboval za njeno implementacijo. Obstajajo pa trije pogosti razlogi, ko zgodba ni dobro ocenljiva:

- Razvijalčevo pomanjkanje znanja o domeni.
- Razvijalčevo pomanjkanje tehničnega znanja.
- Predolga uporabniška zgodba.

Če razvijalec napisane zgodbe ne razume dobro, naj se o njej pomeni s stranko, ki je to zgodbo napisala. Ni potrebno razumeti vseh podrobnosti o zgodbi, vendar mora razvijalec imeti splošno razumevanje o njej.

V primeru predolgih zgodb lahko zgodbo razdelimo na manjše zgodbe, ki jih nato lažje ocenimo.

4.2.5 Majhnost

Zgodbe so lahko prekratke, predolge ali pa so ravno prave dolžine. Dolžina zgodbe je pomembna, saj s prekratkimi oziroma predolgimi zgodbami težko planiramo.

Deljenje zgodbe

Dolge zgodbe ponavadi sodijo v enega od naslednjih tipov:

- sestavljene zgodbe ali
- kompleksne zgodbe.

Sestavljena zgodba je zgodba, ki je sestavljena iz več manjših zgodb. Med začetnim planiranjem je normalno, da so zgodbe dolge in sestavljene, vendar pa nato te zgodbe pred začetkom dela na njih ponavadi razdelimo na več manjših zgodb. Vendar moramo paziti, da ne gremo v drugo skrajnost in sestavljeno zgodbo razdelimo na veliko premajhnih zgodb.

Za razliko od sestavljene zgodbe je *kompleksna zgodba* dolga zgodba, ki pa jo težje razdelimo na več manjših zgodb. Če je zgodba kompleksna zaradi negotovosti, povezane z njo, lahko tako zgodbo razdelimo na dve zgodbi: eno preiskovalno in eno razvojno. Za primer vzemimo zgodbo *Podjetje lahko plača za oglas s kreditno kartico*. V primeru, ko noben od razvijalcev še ni delal ničesar s kreditnimi karticami, lahko izberemo naslednje razbitje zgodbe:

- Raziskovanje procesiranja kreditnih kartic preko spleta.
- Podjetje lahko plača za oglas s kreditno kartico.

V takih primerih je dobro raziskovalno zgodbo, ki nastane iz kompleksne zgodbe, uvrstiti v iteracijo pred iteracijo, v kateri so ostale zgodbe nastale iz te iste kompleksne zgodbe.

4.2.6 Primernost za testiranje

Uporabniške zgodbe morajo biti napisane na takšen način, da se jih da testirati. Uspešno opravljeni testi potrjujejo uspešnost implementacije zgodbe. Zgodbe, ki jih je težko testirati, se pojavljajo pri nefunkcionalnih zahtevah, kot so:

- *Aplikacija mora biti prijazna uporabniku.*
- *Uporabnik ne sme nikoli čakati predolgo na odzivnost aplikacije.*

Če je le mogoče, morajo biti testi avtomatizirani. To je predvsem pomembno pri inkrementalnem razvoju. Produkt se skozi inkremente spreminja, tako se lahko zgodi, da nekaj, kar je v preteklosti že delovalo, zdaj ne deluje več. Tako lahko z avtomatiziranimi testi hitro odkrijemo tovrstne napake.

Obstaja pa podmnožica testov, ki se jih ne da avtomatizirati. Vzemimo na primer zgodbo *Nov uporabnik lahko izpolni enostaven vnos brez predhodnega usposabljanja*. Ta zgodba je lahko testirana, ne moremo pa za njo napisati avtomatskih testov. Testiranje take zgodbe bo najverjetneje vključevalo druge ljudi. Taki tipi testov so dragi in časovno potratni.

4.2.7 Povzetek

- Zgodbe naj bodo neodvisne med seboj (če je le možno). Če to ni možno, naj bodo vsaj napisane tako, da se jih lahko razvija v poljubnem vrstnem redu.
- Podrobnosti zgodbe so pridobljene skozi pogajanja razvijalca in stranke.
- Zgodbe naj bodo napisane na tak način, da so dragocene za stranko. Najboljši način za dosego tega je, da stranka piše uporabniške zgodbe.
- Zgodbam so lahko dodane opombe. Vendar nas lahko preveč opomb zavede, da pogovor s stranko ni potreben.
- Najboljši način za dodajanje podrobnosti zgodbi je pisanje testov.

- Sestavljene in kompleksne zgodbe lahko razdelimo na več manjših zgodb.
- Premajhne zgodbe lahko združimo v večje.
- Zgodbe morajo biti primerne za testiranje.

4.3 Testiranje sprejemljivosti uporabniških zgodb

Eden glavnih razlogov za pisanje testov je način, kako zapisati vse podrobnosti ob pogovoru s stranko. Testiranje najbolje predstavimo v dveh korakih:

- Zapis opomb o prihodnjih testih, ki jih ponavadi dobimo med pogovorom s stranko.
- Ti testi so nato zapisani do podrobnosti in jih lahko uporabimo za dokaz, da je bila uporabniška zgodba uspešno realizirana.

Drugi razlog za pisanje testov je, kot smo že omenili, določitev kriterija uspešnosti uporabniške zgodbe. Tako imamo nekaj, kar nam pove, kdaj smo z delom zaključili. Na ta način se najbolje izognemo primerom, ko naredimo preveč oziroma premalo.

4.3.1 Pisanje testov pred začetkom kodiranja

Sprejemni testi posredujejo veliko informacij, ki jih lahko programer uporabi v svojo korist. Vzemimo na primer *Test z različnimi zneski nakupa (vključno z enim čez mejo kartice)*. Če ima programer test na voljo, preden začne s kodiranjem, bo opomnjen obravnavati primere, ko je nakup zavrnjen zaradi nezadostnega kredita na kartici. Nekateri programerji bi lahko brez predhodnega videnja testa na to pozabili.

Da lahko ti testi koristijo programerju, morajo biti napisani, preden ta začne z delom na zgodbi. Navadno so testi napisani ob enem od naslednjih dogodkov:

- Ko imata programer in stranka pogovor o uporabniški zgodbi in hočeta zapisati podrobnosti le tega.
- Kot del prizadevanj na začetku iteracije, vendar pred začetkom programiranja.
- Kadar so odkriti novi testi, med ali po začetku dela na zgodbi.

Ko stranka in razvijalci razpravljajo o zgodbi, se pridobljene podrobnosti o zgodbi odražajo kot testi. Vendar mora stranka na začetku iteracije iti skozi zgodbe, ki so izbrane za trenutno iteracijo, in napisati morebitne dodatne teste. Pomaga si lahko z naslednjimi vprašanji:

- Kaj še morajo razvijalci vedeti o tej zgodbi?
- Kaj predpostavljam o tem, kako bo zgodba implementirana?
- Ali obstajajo okoliščine, ko se zgodba obnaša drugače?
- Kaj gre lahko narobe med razvojem zgodbe?

4.3.2 Testiranje je del procesa

Ko programer razvije novo funkcionalnost aplikacije, jo razloži testerju in ta nato pove, če aplikacija deluje po opisu. V takih primerih se velikokrat zgodi, da aplikacija prestane test, vendar se pojavijo napake, ko končni uporabnik začne z uporabo aplikacije. Težava je seveda bila v tem, da je tester testiral, če je programer naredil to, kar je rekel, da je naredil. Nihče pa ni testiral, če aplikacija naredi to, kar je stranka oziroma uporabnik pričakoval.

Za uporabniške zgodbe je ključno, da je testiranje videno kot del razvojnega procesa, ne pa nekaj, kar se zgodi, ko je kodiranje zaključeno. Specifikacija testov je mnogokrat skupna zadolžitev *produktnega vodje* in testerja.

4.3.3 Vrste testiranja

Pri večini sistemov gre večinoma za funkcionalno testiranje, ki zagotavlja, da aplikacija deluje po pričakovanjih. Seveda pa obstajajo tudi druge vrste testov:

- Testiranje uporabniškega vmesnika, ki zagotavlja, da se vse komponente vmesnika obnašajo po pričakovanjih.
- Testiranje uporabnosti, ki zagotavlja preprosto uporabo aplikacije.
- Performančno testiranje. S tem testiramo, kako se aplikacija odziva ob različnih obremenitvah.
- Testiranje izjemnih situacij.

4.3.4 Povzetek

- Sprejemne teste uporabljamo za zapis podrobnosti o zgodbi, ki jih dobimo med pogovorom s stranko.
- Sprejemni testi predstavljajo osnovne kriterije, ki določajo, če je zgodba končana.
- Sprejemne teste naj napiše stranka, ne pa razvijalec.
- Sprejemni testi naj bodo zapisani, preden razvijalec prične s kodiranjem.
- Če z dodajanjem novih testov ne dosežemo dodatnega pojasnjevanja podrobnosti zgodbe, to opustimo.

4.4 Zahtevane funkcionalnosti aplikacije

Kot del diplomske naloge smo morali narediti aplikacijo za pomoč pri vodenju projektov po metodi Scrum. Zahtevane funkcionalnosti te aplikacije bomo tako podali v obliki uporabniških zgodb.

Uporabniške zahteve so bile podane v obliki *Kot <vloga>, želim <cilj> tako da bom imel <koristi>* (angl. *As a <role>, I want <goal/desire> so that <benefit>*). Taka oblika podajanja zahtev se je izkazala za zelo učinkovito, ne glede na obliko in velikost projekta. V podpoglavjih bomo najprej našteli funkcionalnosti, ki naj bi jih imeli: *vsi uporabniki*, *skrbnik metodologije*, *produktni vodja* in *član razvojne skupine*.

4.4.1 Vsi uporabniki

Prijava v sistem Kot uporabnik se želim prijaviti v sistem z uporabniškim imenom in geslom, da bi ga lahko uporabljal pri razvoju projektov po metodi Scrum. Nato moram imeti za delo na projektu eno od vlog opisanih v naslednjih sekcijah.

4.4.2 Skrbnik metodologije

Vnos novega projekta Kot skrbnik metodologije želim vnesti podatke o novem projektu, da bi lahko pričeli s spremljanjem dela na tem projektu.

Spreminjanje podatkov o projektu Kot skrbnik metodologije želim spremeniti podatke o projektu, da bi lahko vnesel morebitne spremembe, ki so nastale po začetnem vnosu.

Brisanje projekta Kot skrbnik metodologije želim odstraniti projekt, ki je bil pomotoma vnesen ali se ne bo izvajal. Brisanje projekta, ki že teče (ali je končan) in zanj obstajajo podatki, ni dovoljeno.

Vnos novega razvijalca Kot skrbnik metodologije želim vnesti podatke o sodelavcu, ki lahko sodeluje pri razvoju projekta, da bi jih potem uporabil pri sestavljanju razvojnih skupin za posamezne projekte.

Spreminjanje podatkov o razvijalcu Kot skrbnik metodologije želim spremeniti podatke o razvijalcu, da bi lahko vzdrževal podatke o spremembah, ki se pojavijo v času, ko je sistem v uporabi.

Brisanje razvijalca Kot skrbnik metodologije želim odstraniti razvijalca, ki je bil pomotoma/narobe vnesen. Brisanje podatkov o razvijalcu, ki je že delal na nekem projektu, ni dovoljeno.

Vnos nove iteracije Kot skrbnik metodologije želim vnesti podatke o novi iteraciji, da bi lahko pričeli s spremljanjem podatkov, ki se nanašajo na to iteracijo.

Spreminjanje podatkov o iteraciji Kot skrbnik metodologije želim spremeniti podatke o iteraciji, da bi lahko vnesel morebitne spremembe, ki so nastale po začetnem vnosu.

Brisanje iteracije Kot skrbnik metodologije želim odstraniti iteracijo, ki je bila pomotoma/narobe vnešena. Brisanje iteracije, za katero že obstajajo podatki, ni dovoljeno.

Dodajanje zgodbe iteraciji Kot skrbnik metodologije želim dodati zgodbo v iteracijo, da bi izdelal plan naslednje iteracije. Če zgodba v prejšnji iteraciji ni bila realizirana, se lahko uvrsti v eno izmed naslednjih iteracij z novo oceno zahtevnosti in novim statusom (nedokončana zgodba, popravki sprejete zgodbe).

Odvzemanje zgodbe iz iteracije Kot skrbnik metodologije želim izločiti zgodbo iz iteracije, da bi izdelal plan naslednje iteracije. Odvzemanje zgodbe, za katero že obstajajo podatki (naloge, opravljeno delo), ni dovoljeno.

Uvoz iz Agila Kot skrbnik metodologije želim uvoziti podatke iz Excelove preglednice (ki predstavlja izhod iz Agila), da bi lahko analiziral delo na študentskih projektih.

Diagram za analizo trenda iteracije Kot skrbnik metodologije želim izrisati diagram za analizo trenda iteracije, da bi lahko ugotovil, ali iteracija poteka v skladu s planom.

Diagram za analizo trenda produkta (izdaje) Kot skrbnik metodologije želim izrisati diagram za analizo trenda produkta (izdaje), da bi lahko ugotovil, ali projekt poteka v skladu s planom.

WE, SPI in CPI diagram Kot skrbnik metodologije želim izrisati diagrame indeksa učinkovitosti, terminskega indeksa in stroškovnega indeksa, da bi lahko ugotovil, ali iteracija poteka v skladu s planom.

Delež posameznih razvijalcev v izbrani iteraciji/vseh iteracijah Kot skrbnik metodologije želim izrisati diagram, ki bo prikazoval planiran in dejanski delež posameznih razvijalcev v posameznih iteracijah, da bi lahko zasledoval obremenitve in učinkovitost posameznih razvijalcev.

Planirano in dejansko opravljeno delo v izbrani iteraciji/vseh iteracijah Kot skrbnik metodologije želim izpis seznama vseh zgodb skupaj z oceno zahtevnosti in dejansko vloženim delom, da bi lahko analiziral točnost ocenjevanja.

4.4.3 Produktni vodja

Vnos nove uporabniške zgodbe Kot produktni vodja želim vnesti novo uporabniško zgodbo, da bi lahko sestavil seznam zahtev projekta.

Spreminjanje uporabniške zgodbe Kot produktni vodja želim spremeniti podatke o uporabniški zgodbi, da bi lahko vnesel spremembe, ki so se pojavile po začetnem vnosu. Ocene zahtevnosti in komentarje lahko vnaša samo razvijalec. Paziti je treba na vzdrževanje podatka, ali je zgodba veljavna.

Brisanje uporabniške zgodbe Kot produktni vodja želim brisati uporabniško zgodbo, ki se izkaže kot nepotrebna. Brisanje zgodbe, za katero že obstajajo podatki (razporeditev v iteracijo, razbitje na naloge, opravljeno delo), ni dovoljeno.

Razdelitev uporabniške zgodbe na več manjših Kot produktni vodja želim razdeliti preveč obsežno zgodbo na več manjših. Razbitje zgodbe je možno, samo preden je zgodba uvrščena v iteracijo.

4.4.4 Član razvojne skupine

Vnos ocene zahtevnosti uporabniške zgodbe Kot član razvojne skupine želim vnesti oceno zahtevnosti uporabniške zgodbe, da bi lahko (upoštevajoč predvideno hitrost) izdelal plan izdaje in plan iteracije. Začetne ocene zahtevnosti ni moč spreminjati potem, ko je izdelava plana končana.

Vnos komentarjev k uporabniški zgodbi Kot član razvojne skupine želim vnašati komentarje k izbrani uporabniški zgodbi, da bi tako zabeležil dodatna pojasnila in dogovore glede realizacije, do katerih sem prišel v razgovorih s *produktnim vodjo*.

Vnos opravljenega in preostalega dela Kot član razvojne skupine želim vnesti število ur vloženega in število ur preostalega dela (za vse svoje naloge), da bi lahko izdelali diagram za analizo trenda in razne analize poteka dela na projektu.

Spreminjanje vloženega in preostalega dela Kot član razvojne skupine želim spremeniti število ur vloženega in število ur preostalega dela na neki nalogi, da bi popravil podatke, ki sem jih predhodno vnesel. Spreminjanje podatkov je možno samo na dan dnevnega Scrum sestanka (za nazaj ni dovoljeno).

Brisanje vloženega in preostalega dela Kot član razvojne skupine želim brisati (pomotoma narobe vnesene) podatke o številu ur vloženega in preostalega dela na neki nalogi.

Vnos nove naloge Kot član razvojne skupine želim vnesti novo nalogo, ki je potrebna za realizacijo zgodbe.

Spreminjanje naloge Kot član razvojne skupine želim popraviti podatke o nalogi, če so se le-ti spremenili. Začetna ocena se ne sme več spremeniti, ko je izdelava seznama nalog končana.

Brisanje naloge Kot član razvojne skupine želim brisati nalogo, ki se izkaže kot nepotrebna. Brisanje naloge, za katero že obstajajo podatki o vloženem delu, ni dovoljeno.

Razdelitev naloge na več manjših Kot član razvojne skupine želim razdeliti preobsežno nalogo na več manjših, da bi lahko natančneje ocenil in načrtoval svoje delo. (Mogoče samo spreminjanje obstoječe, z oznako, da odslej ni več veljavna, in vnos več novih nalog z veljavnostjo od tega dne dalje).

Poglavje 5

Podatkovni model

Naj še predstavimo podatkovni model, ki smo ga uporabili za shranjevanje podatkov. Podatkovni model izhaja iz modela iz članka [3]. Prikazan je na sliki 5.1. Ta model je zelo splošen in ne predpisuje vrste in števila meritev, ki jih hrani.

Pri načrtovanju podatkovnega modela smo se omejili in tako ta ne podpira zapisa rezultatov vseh meritev, naštetih v poglavju 3, ki se začne na strani 15. Lahko pa model, z dodatkom kakšne tabele, zlahka nadgradimo, tako da bi podpiral širšo množico meritev. Osredotočili smo se na meritve, ki nam omogočajo pravočasne informacije o uspešnosti projekta. To so *indeks učinkovitosti* (angl. *Work effectiveness*), *terminski indeks* (angl. *Schedule performance index - SPI*) oziroma *stroškovni indeks* (angl. *Cost performance index - CPI*). Podatovni model je predstavljen na sliki 5.2.

S kreiranjem novega projekta naredimo nov zapis v tabeli *PROJECT*. Shranimo samo ime projekta in morebiten opis. Pred začetkom dela na projektu pa moramo uporabnikom aplikacije še določiti uporabniške vloge na tem projektu (*Skrbnik metodologije, Produktni vodja ali Član razvojne skupine*).

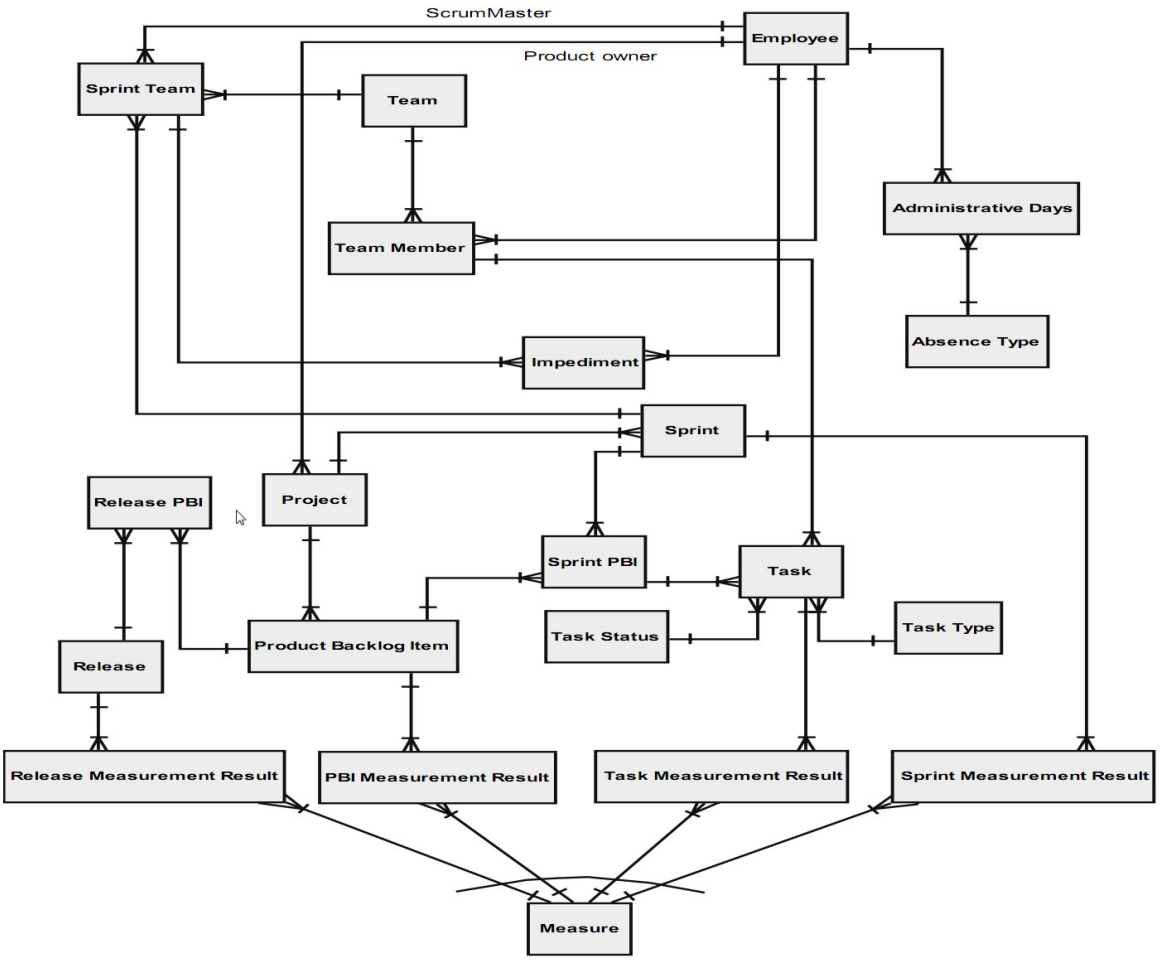
Ko smo to naredili, lahko pričnemo z določanjem uporabniških zgodb. Te shranjujemo v tabeli *PBI* (*Product backlog item*). Vnesemo ime uporabniške zgodbe, njen opis in njeno prioriteto. Lahko določimo tudi točke (zahtevnost uporabniške zgodbe), vendar to ponavadi naredimo kasneje, ob prenosu zgodbe v iteracijo. Naj še povemo, zakaj je tabeli ime *PBI*. Kratica *PBI* je, kot smo že omenili, kratica za *Product backlog item*, kar prevedeno pomeni *element seznama zahtev*. V našem primeru je seznam zahtev množica uporabniških zgodb.

Ob vsakem začetku cikla določimo podatke iteracije, njen začetek, konec in dolžino. Ti podatki so vidni v tabeli *SPRINT*. Pred pričetkom iteracije določimo zgodbe, nad katerimi bomo v tej iteraciji delali in jim določimo njihovo zahtevnost (točke). Ti podatki se zapišejo v tabeli $SPRINT\_PBI$.

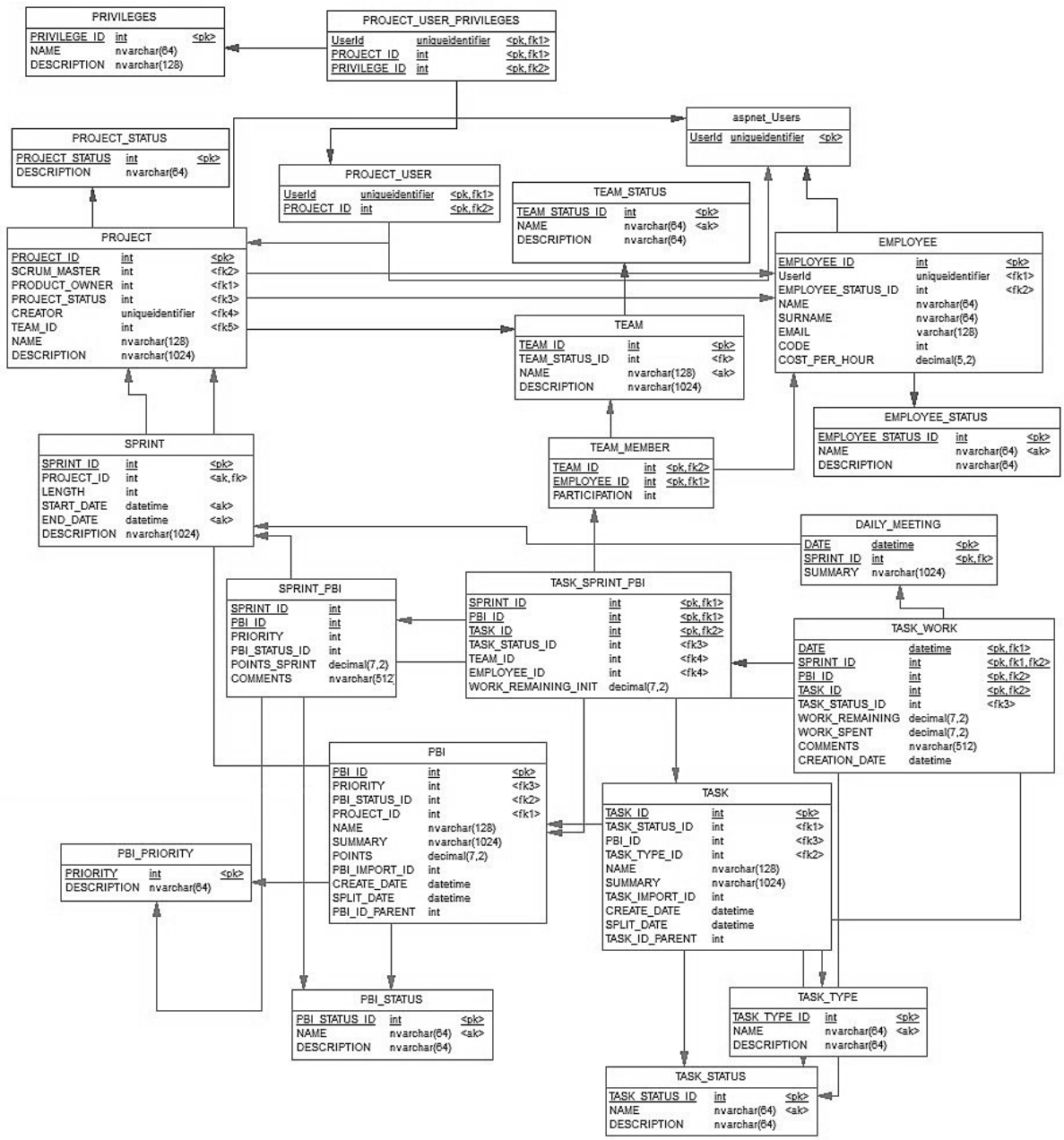
Nato vsako izbrano zgodbo razdelimo na naloge. Določimo ime in opis naloge ter število ur, ki bodo potrebne za dokončanje le-te (tabele *TASK* in $TASK\_SPRINT\_PBI$).

Ob delu na nalogah iteracije dnevno zapisujemo, koliko smo ta dan porabili na tej nalogi in koliko dela nam je še preostalo na tej nalogi. To se zapiše v tabelo $TASK\_WORK$.

S tem smo na kratko opisali glavne poslovne tabele naše aplikacije. Več podrobnosti je videnih na sliki 5.2, kjer je predstavljen celoten podatkovni model.



Slika 5.1: Podatkovni model, na katerega smo se sklicevali pri kreiranju našega podatkovnega modela [3]



Slika 5.2: Podatkovni model

Poglavje 6

Aplikacija

Glavni del diplomske naloge se nanaša na aplikacijo za pomoč pri vodenju projektov po metodi Scrum. Poleg upravljanja s podatki o projektu (iteracije, zgodbe, naloge, delo na nalogah) naj bi aplikacija omogočala še prikaz rezultatov v obliki tabel ter stolpičnih in tortnih diagramov. Podatki za prikaz izhajajo iz meritev, predstavljenih v poglavju 3, ki se začne na strani 15.

Odločili smo se narediti spletno aplikacijo, in sicer v tehnologiji *ASP.NET MVC* [7]. Odločitev za to tehnologijo je temeljila predvsem na dejstvu, da smo imeli že predhodno izkušnje z izbrano tehnologijo in smo tako poznali ogrodje.

Zahtevane funkcionalnosti aplikacije smo že opisali v poglavju 4, na strani 32.

6.1 Uporabniški vmesnik

Na kratko bomo predstavili uporabniški vmesnik. Tekst ni mišljen kot navodilo za uporabo aplikacije, ampak kot predstavitev funkcionalnosti aplikacije.

Za uporabo aplikacije se je potrebno predhodno v aplikacijo prijaviti, oziroma registrirati, če uporabniškega računa še nimamo.

6.1.1 Kreiranje projekta

Za kreiranje novega projekta moramo imeti zadostne privilegije v sami aplikaciji (vloga *ProjectCreator*). Ta vloga ni povezana z vlogami v metodi Scrum. Navezuje se na samo aplikacijo in nam kot prijavljenemu uporabniku dovoli kreiranje novega projekta.

Na ekranu pregleda projektov izberemo povezavo *Kreiranje novega projekta* in izpolnimo formo. Ko ta korak uspešno naredimo, določimo vloge (privilegije)

uporabnikom, ki bodo sodelovali na tem projektu. Te vloge so vezane samo na ta projekt. Poznamo štiri vloge, in sicer *Skrbnik metodologije*, *Produktni vodja*, *Član razvojne skupine* in *Dodaj*. Pomeni prvih treh naštetih vlog so taki, kot so specificirani v metodi *Scrum*. Več v poglavju 2.3.1, ki se začne na strani 10. Z vlogo *Dodaj*, pa lahko ostalim uporabnikom dodajamo in odvzemamo vloge na projektu. To vlogo avtomatsko dobi kreator projekta. Po tem koraku moramo projektu še določiti skupino, ki bo na izbranem projektu delala.

Brisanje projekta

Projekta iz podatkovne baze fizično ni mogoče izbrisati (preko odjemalca). Je pa možno projektu spremeniti status na *Izbrisan* (angl. *Deleted*) in tako ta projekt ni več viden pri prikazu le-teh. Če smo se zmotili, lahko projektu ponovno spremenimo status. Do brisanega projekta pridemo tako, da pod iskalnimi pogoji izberemo še status projekta *izbrisan*.

6.1.2 Kreiranje skupin

Vsakemu projektu, ki ga kreiramo, moramo določiti tudi skupino, ki bo delala na njem. Potrebni podatki so samo njeno ime in po želji še opis. Nato sledi dodajanje razvijalcev skupini. Če ti predhodno niso vnešeni, moramo storiti še ta korak. O razvijalcu nas zanima njegovo ime in priimek, poleg tega pa je pomemben še podatek o njegovi ceni na uro. Ta se kasneje uporablja pri izračunu *stroškovnega indeksa*.

6.1.3 Dodeljevanje vlog na projektu

Po kreiranju projekta in dodelitvi projekta skupini, lahko pa tudi kasneje med delom na njem, lahko na ekranu za urejanje privilegijev, uporabnikom, ki bodo sodelovali na projektu, dodelimo določene vloge. To lahko počnemo, če imamo mi na tem projektu vlogo *Dodaj* (samodejno jo dobi kreator projekta). Vloge na projektu so lahko naslednje:

- **Dodaj** - uporabnik s to vlogo lahko drugim uporabnikom (in tudi sebi) dodeljuje vloge.
- **Skrbnik metodologije** - vloga je namenjena skrbniku metodologije projekta.
- **Produktni vodja** - vloga je namenjena produktnemu vodji projekta.

- **Član razvojne skupine** - to vlogo imajo vsi ostali razvijalci na projektu (Člani razvojne skupine).

Funkcionalnosti, ki so na voljo posameznim vlogam, smo že opisali v specifikaciji zahtev.

6.1.4 Kreiranje uporabniških zgodb

Ob začetku projekta in pa tudi med samim projektom moramo kreirati uporabniške zgodbe (slika 6.1). Vloga, ki jo moramo imeti, da lahko to počnemo, je vloga *Produktni vodja*. Zgodbo lahko kreiramo brez določitve iteracije in jo kasneje dodamo določeni iteraciji ali pa ob že izbrani iteraciji. Na drugi način, ob že izbrani iteraciji, se tej zgodba avtomatsko doda.

Brisanje uporabniških zgodb

Uporabniško zgodbo lahko v primeru napake pri vnosu tudi izbrišemo. To nam aplikacija dovoli, če ta zgodba še ni bila dodeljena nobeni iteraciji. V kolikor je bila zgodba dodeljena iteraciji, moramo to zgodbo iz nje predhodno odstraniti. Šele nato jo lahko izbrišemo.

6.1.5 Kreiranje iteracije

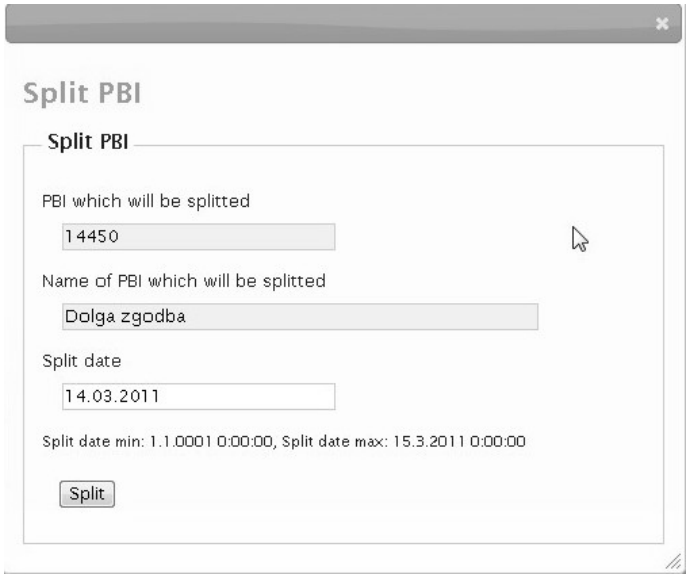
Ob začetku vsake iteracije moramo le-to kreirati. Pri njenem kreiranju moramo poznati njeno dolžino in njen začetni ter končni datum. Iteracije v istem projektu se ne smejo časovno prekrivati.

Dolžina iteracije je mišljena kot število delovnih dni v njenem obdobju, na katerih bomo delali na izbranem projektu. Pomembna je, saj vpliva na meritev terminskega indeksa in zato, ker določa, koliko dnevnih Scrum sestankov bomo imeli v tej iteraciji. Na slednjih dnevno določamo naše delo na nalogah projekta in pa preostalo planirano delo na teh nalogah.

Dodeljevanje zgodb iteraciji

Kot smo že omenili, lahko že obstoječo zgodbo dodamo izbrani iteraciji. S tem dodamo tudi vse naloge, ki pripadajo izbrani uporabniški zgodbi. Ko zgodbo dodamo iteraciji, ji moramo določiti njeno zahtevnost (v točkah). Ena točka naj bi predstavljala približno en idealen delovni dan (6 ur dela). Tako število točk na zgodbi pomeni oceno števila delovnih dni, ki jih planiramo za njeno implementacijo.

Slika 6.1: Kreiranje uporabniške zgodbe



Slika 6.2: Delitev uporabniške zgodbe

Delitev zgodbe

Velikokrat so zgodbe na začetku napisane preobširno in se šele s podrobnejšim pogovorom vidi njena dejanska obsežnost. V takih primerih lahko zgodbo razdelimo na več manjših zgodb (slika 6.2). Pogoj, da lahko to naredimo je, da uporabniška zgodba ne pripada še nobeni iteraciji, kar posredno pomeni, da uporabniška zgodba še ni razdeljena na podnaloge.

6.1.6 Kreiranje nalog zgodbe

Ob začetku vsake iteracije, lahko pa tudi med samo iteracijo, je potrebno uporabniške zgodbe natančneje definirati. Ob tem naredimo tudi razčlenitev teh zgodb na naloge. Vsaka dobljena naloga naj ne bi trajala več kot šestnajst ur. Ob kreiranju naloge moramo določiti tudi njeno začetno oceno v urah (koliko časa naj bi razvijalec porabil na tej nalogi).

Delitev naloge

Ko ob začetku iteracije razpravljamo o uporabniški zgodbi, velikokrat nimamo časa točno definirati vseh nalog, oziroma želimo raje kot razpravljati o vseh nalogah, začeti z delom na že definiranih. Tako ostalih funkcionalnosti ne

razbijemo na dokončne naloge, ampak kreiramo večje naloge. Te večje naloge nato, ob pravem času, razbijemo na podnaloge.

6.1.7 Vpisovanje dela na nalogi

Med samim delom na nalogi moramo dnevno (ponavadi na dnevnem Scrum sestanku) vpisovati število ur, ki jih porabimo za delo na njej. Poleg datuma in števila ur, ki smo jih porabili z delom na nalogi in ocene preostanka dela na nalogi (v urah), ki so obvezni podatki, lahko tukaj zabeležimo še dodaten *komentar*, ki nam je lahko v pomoč kasneje.

Urejamo lahko samo zadnji zapis in še tega samo, če je bil kreiran največ n dni nazaj. Število n je v aplikaciji nastavljivo kot parameter. Privzeto ima vrednost tri. Tri dni pa zato, da lahko uporabnik v ponedeljek, ko pride na delo, še popravi zadnji zapis, ki ga je ustvaril v petek.

Mogoče bi bilo tukaj tudi smiselno omejiti vnos za več kot n dni nazaj, saj bi s tem razvijalca, ki je dolžan izpolnjevati te podatke, prisilili v sprotno izpolnjevanje. Tako bi prišli do bolj verodostojnih podatkov za meritve *terminskega indeksa* in ostalih meritev, ki jih izvajamo, saj lahko v obratnem primeru razvijalec z vnašanjem podatkov počaka do konca naloge, nato pa vnese podatke tako, da je njegovo planiranje prikazano kot idealno. Po drugi strani pa naj bi skupina imela dnevne Scrum sestanke, kjer bi se vnašali ti podatki, tako da smo se odločili, da te omejitve v aplikacijo ne vgradimo.

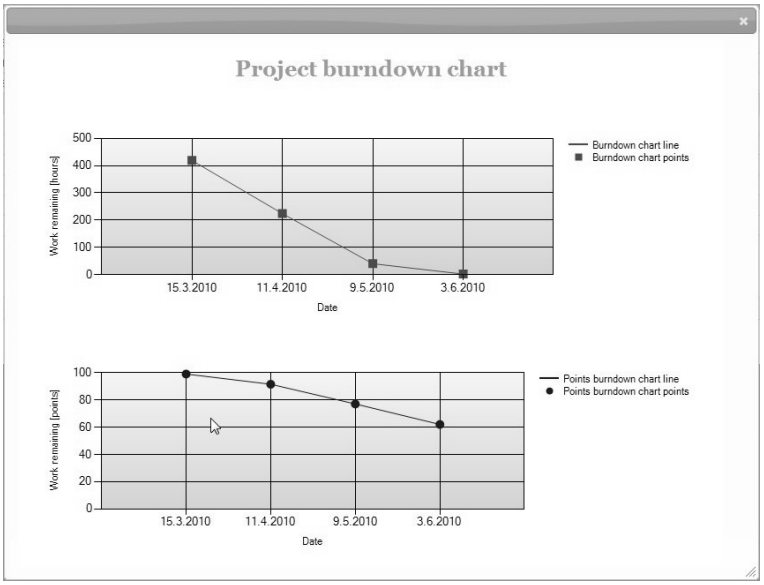
Za brisanje zapisa veljajo enaki pogoji kot za njegovo urejanje. Brišemo lahko samo zadnji zapis in še ta ne sme biti starejši kot n dni. Seveda lahko nato izbrišemo še predzadnji zapis in tako naprej. Pogoj je le, da noben od njih ni starejši od n dni.

6.1.8 Prikaz meritev

Glavni cilj diplomske naloge je bil, da iz podatkov, ki jih imamo o projektu (iteracije, uporabniške zgodbe, naloge in predvsem delo na nalogah), prikažemo različne uporabne rezultate. Ti rezultati naj bi bili v obliki, ki je hitro in lahko berljiva. Najprej bomo opisali prikaz rezultatov na nivoju projekta, nato pa še rezultate na nivoju iteracije, uporabniške naloge oziroma naloge.

Rezultati na nivoju projekta

Z dvoklikom na želen projekt v pregledu prikaza projektov se nam prikaže modalno okno s podrobnostmi projekta. Na tem oknu imamo tri povezave, ki nas vodijo do različnega prikaza rezultatov o projektu.

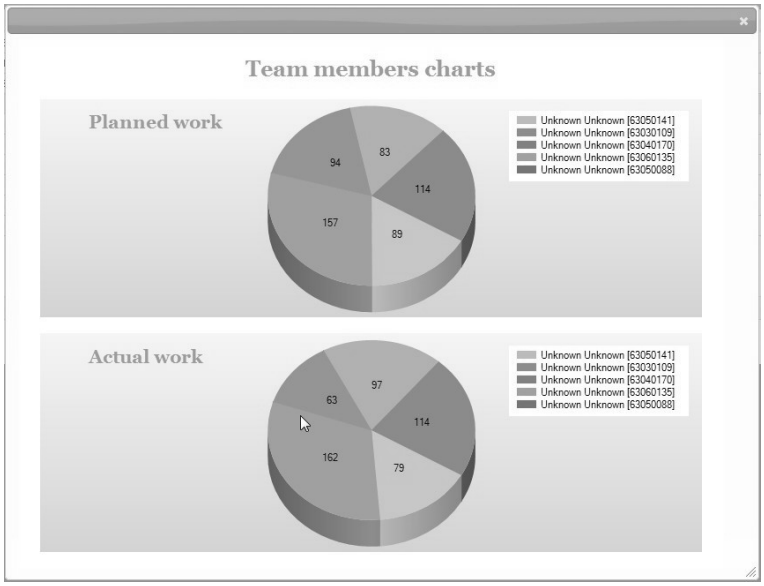


Slika 6.3: Diagram za analizo trenda projekta - na prvem diagramu je prikazano padanje števila preostalih ur na projektu, na drugem pa padanje števila točk

Diagram za analizo trenda projekta V tem prikazu gre dejansko za dva diagrama (slika 6.3). Prvi prikazuje padanje preostanka dela na projektu v urah. Vzorci so vzeti samo na začetku vsake iteracije (oziroma na koncu, če gre za zadnjo iteracijo). Drugi pa prikazuje padanje preostanka točk. Točke so definirane na uporabniški zgodbi. Ko zgodbo zaključimo in je potrjena s strani *produktnega vodje*, se preostanek dela na projektu zmanjša za to število točk. Tako je ta podatek samo seštevek točk še ne zaključenih uporabniških zgodb. Privilegij za spreminjanje statusa uporabniške zgodbe ima samo *produktni vodja*, saj mora o uspešnosti realizacije zgodbe presoditi sam.

Diagram članov skupine Ta prikaz je sestavljen iz dveh tortnih diagramov (slika 6.4). Prvi predstavlja delež načrtovanega dela na projektu, v številu ur, po razvijalcih, drugi pa predstavlja delež dejanskega dela na projektu po razvijalcih. S primerjavo obeh diagramov lahko vidimo kako dobro so posamezni razvijalci planirali svoje delo na projektu in pa koliko je bilo skupaj planiranega in dejanskega dela na projektu.

Izvrševanje plana uporabniških zgodb Ob kreiranju oziroma dodajanju uporabniške zgodbe iteraciji smo ji morali določiti število točk. Kot smo že opisali, naj bi število točk predstavljalo zahtevnost zgodbe. In sicer na način,



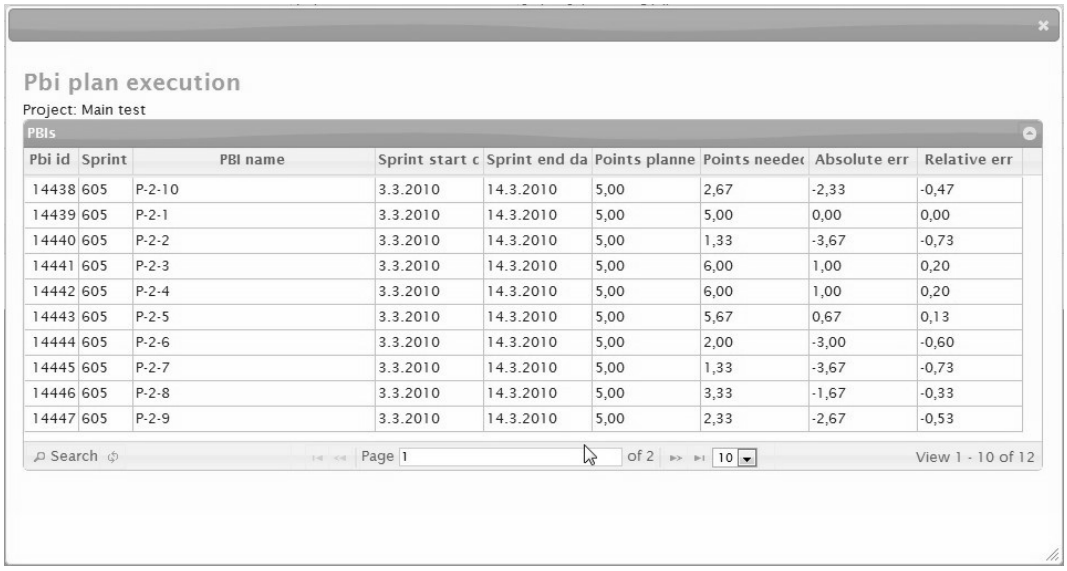
Slika 6.4: Diagram članov skupine - na prvem diagramu je prikazano planirano delo po razvijalcih, na drugem pa dejansko delo

da če točke pomnožimo s šest (efektivnih ur v delovnem dnevu), dobimo koliko ur naj nam bi vzelo delo na zgodbi. Tako točke direktno pomenijo, koliko dni naj bi delali na nalogi, če se posvečamo samo njej.
Če se vrnemo na to, kaj izvrševanje plana uporabniških zgodb pomeni. Predstavlja nam absolutno in relativno napako med planiranim delom in dejanskim delom (slika 6.5). Slednje dobimo tako, da dejansko število ur, ki smo jih porabili na vseh nalogah uporabniške zgodbe, delimo s šest. Napaka (absolutna in relativna) pod vrednostjo nič pomeni, da smo za nalogo porabili manj kot smo planirani, nad vrednostjo nič pa obratno.
Za prikaz te meritve se nismo odločili uporabiti grafične oblike. Predstavili smo jo kar v obliki tabele.

Rezultati na nivoju iteracije, uporabniške zgodbe, naloge

Naslednje meritve lahko opravimo na različnih nivojih strukture procesa. Ti nivoji so *iteracija*, *uporabniška zgodba* in *posamezna naloga*. Razlika je samo v tem, katere naloge vključimo v izračun kazalcev. Samo eno nalogo, vse naloge uporabniške zgodbe ali kar vse naloge iteracije.
Na podmnožici nalog tako prikažemo naslednje meritve (slika 6.6):

- Delež prislužene vrednosti (angl. *Earning rule*)

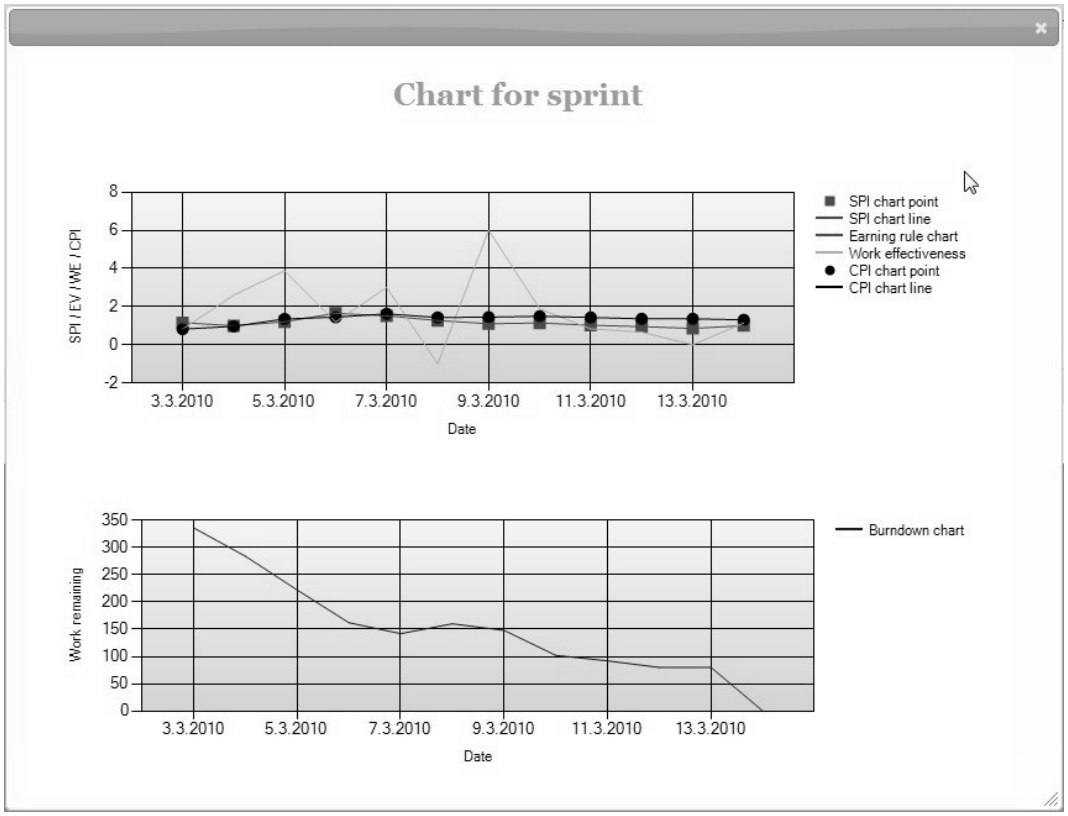


Pbi id	Sprint	PBI name	Sprint start c	Sprint end da	Points planne	Points neede(	Absolute err	Relative err
14438	605	P-2-10	3.3.2010	14.3.2010	5,00	2,67	-2,33	-0,47
14439	605	P-2-1	3.3.2010	14.3.2010	5,00	5,00	0,00	0,00
14440	605	P-2-2	3.3.2010	14.3.2010	5,00	1,33	-3,67	-0,73
14441	605	P-2-3	3.3.2010	14.3.2010	5,00	6,00	1,00	0,20
14442	605	P-2-4	3.3.2010	14.3.2010	5,00	6,00	1,00	0,20
14443	605	P-2-5	3.3.2010	14.3.2010	5,00	5,67	0,67	0,13
14444	605	P-2-6	3.3.2010	14.3.2010	5,00	2,00	-3,00	-0,60
14445	605	P-2-7	3.3.2010	14.3.2010	5,00	1,33	-3,67	-0,73
14446	605	P-2-8	3.3.2010	14.3.2010	5,00	3,33	-1,67	-0,33
14447	605	P-2-9	3.3.2010	14.3.2010	5,00	2,33	-2,67	-0,53

Slika 6.5: Izvrševanje plana uporabniških zgodb

- Terminski indeks (angl. *Schedule performance index*)
- Stroškovni indeks (angl. *Cost performance index*)
- Indeks učinkovitosti (angl. *Work effectiveness*)
- Diagram za analizo trenda (angl. *Burndown chart*)

Kaj posamezna meritev pove in kako priti do nje, smo napisali že v enem od prejšnjih poglavij.



Slika 6.6: Rezultati na nivoju iteracije - prikazane so meritve terminski indeks, stroškovni indeks, indeks učinkovitosti in diagram za analizo trenda

Poglavje 7

Sklepne ugotovitve

V nalogi smo opisali Scrum, eno od najbolj razširjenih metodologij razvoja programske opreme. Po raziskavi, ki jo je naredilo podjetje *VersionOne* leta 2010, vidimo, da že več kot 58 odstotkov uporabnikov agilnih metodologij razvoja uporablja ravno metodo Scrum. Ena od prednosti te metodologije pred ostalimi metodologijami agilnega razvoja je prav njena enostavnost, kar omogoča hitro uvajanje v delo z njo. Poleg tega Scrum spodbuja timsko delo in pomaga pri razbitju hierirahije v skupini.

Z vpeljavo meritev v metodo Scrum naj bi *produktni vodja* in *skrbik metodologije* imela dosti več sprotnih informacij o planiranem poteku dela na projektu. Tako lahko hitreje ukrepata v primeru odstopanja od plana dela. Druga prednost pa je tudi ta, da ima lahko tudi naročnik vpogled v potek dela na projektu na zanj prijazen način, kar mu predstavlja neke oprijemljive informacije, kar ima velikokrat za naročnika velik pomen. Meritve, ki smo jih uvedli v metodologijo Scrum so: *terminski indeks*, *stroškovni indeks*, *indeks učinkovitosti* in *diagram za analizo trenda*. S tem se zmanjšuje tudi tveganje, ki nastane ob razvoju projekta.

Za predstavitev zahtev projekta smo podrobneje opisali uporabniške zgodbe, ki so se izkazale za zelo učinkovito sredstvo za prenos informacij. So razumljive naročniku, hkrati pa razvijalcu ne dovoljujejo preširoke interpretacije naloge. Rezultat so zadovoljni naročniki, ki dobijo zagotovilo, da bo programerski čas, ki ga zakupijo, optimalno uporabljen za naloge, ki so najpomembnejše. Prednost je v tem, da jih naročnik ponavadi piše sam in se tako čuti bolj vključenega v razvoj samega projekta, kar nedvomno pomaga pri reševanju problemov.

Naredili smo tudi orodje za vodenje projektov po metodi Scrum. Orodje je v obliki spletne aplikacije, in sicer v tehnologiji *ASP.NET MVC*. Pri njeni realizaciji nismo naleteli na kakšne posebne težave, saj so bile zahteve, ki so

bile podane kot *uporabniške zgodbe*, dobro definirane in se skozi čas razvoja niso bistveno spreminjale. Bilo je dodanih samo nekaj zgodb, kar pa ni predstavljalo težav.

Dodatek A

Namestitev aplikacije

Kot smo že omenili, je aplikacija napisana v tehnologiji *ASP.NET MVC*. Za njeno delovanje potrebujemo *Windows* strežnik z nameščenimi naslednjimi komponentami:

- IIS 7 (Internet Information Services)
- .NET Framework 4.0
- ASP.NET MVC 3
- SQL Server (eno od izdaj)

V SQL Server-ju kreiramo novo podatkovno bazo oziroma uporabimo že obstoječo. Kreiramo uporabnika, ki bo imel ustrezne privilegije v shemi, ki jo bo aplikacija uporabljala (vloge *db_datareader*, *db_datawriter*). Tega uporabnika bo kasneje aplikacija uporabljala za dostop do podatkovne baze. S pomočjo orodja *aspnet_regsql.exe*, ki je priloženo .NET Framework-u kreiramo tabele v izbrani podatkovni bazi, ki jih rabimo za avtentikacijo in avtorizacijo. Nato s pomočjo priložene skripte *crebas.sql* kreiramo še preostale tabele aplikacije. Uporabniku, ki smo ga kreirali za dostop do podatkovne baze, moramo določiti še naslednje vloge:

- $aspnet\_Membership\_FullAccess$
- $aspnet\_Roles\_FullAccess$

Nato znotraj IIS-a kreiramo spletno aplikacijo, ki uporablja *framework* verzije 4.0. Nastavimo ji ustrezen direktorij, v katerega nato postavimo našo aplikacijo. V nastavitvah aplikacije (*Web.config*) je potrebno še nastaviti:

- Nastavitve povezave do podatkovne baze (*Connection string*)
- Direktorij kamor se piše dnevnik aplikacije
- Elektronski naslov skrbnika aplikacije
- Nastavitve poštnega strežnika, ki se bo uporabljal za pošiljanje elektronske pošte

Aplikacija bi morala v tej točki že delovati. Vse kar moramo še storiti je dodeliti registriranim uporabnikom, tistim ki jim bomo dovolili kreiranje projektov, vlogo *ProjectCreator*. To v aplikaciji ni podprto, lahko pa to storimo ročno v sami podatkovni bazi, oziroma uporabimo orodje *ASP.NET Configuration*, ki je priloženo *Visual studiu*.

Literatura

[1] K. Schwaber, Agile Project Management with Scrum, Microsoft Press, 2004

[2] M. Cohn, User Stories Applied: For Agile Software Development, Addison Wesley, Marec 2004

[3] V. Mahnic, N. Zabkar, Introducing CMMI Measurements and Analysis Practices into Scrum-based Software Development Process, International journal of mathematics and computers in simulation, Issue 1, Volume 1, 2007

[4] CMMI Product Team, CMMI for Development: Version 1.2, Carnegie Mellon University, Avgust 2006

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

[6] State of Agile Development Survey Results, Dostopno na `http://www.versionone.com/state_of_agile_development_survey/10/`

[7] J. Palermo, B. Scheirman, J. Bogard, E. Hexter, M. Hinze, ASP.NET MVC 2 in Action, Manning, Junij 2010