UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Murn

Razvoj poslovnih aplikacij po metodi Scrum

DIPLOMSKO DELO

UNIVERZITETNI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Murn

Razvoj poslovnih aplikacij po metodi Scrum

DIPLOMSKO DELO

UNIVERZITETNI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Viljan Mahnič

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Proučite značilnosti agilnega razvoja programske opreme s poudarkom na metodi Scrum. Z uporabo te metode realizirajte aplikacijo za vodenje rezervacij športne dvorane. V ta namen definirajte ustrezne uporabniške zgodbe in izpeljite ocenjevanje njihove zahtevnosti po metodah "planning poker" in "team estimation game". Nato podrobno opišite potek posameznih iteracij. V vsaki iteraciji posvetite posebno pozornost zbiranju podatkov o količini vloženega in preostalega dela, da bi lahko na koncu izdelali analizo točnosti ocenjevanja posameznih uporabniških zgodb. Za izračun točnosti ocen uporabite uravnoteženo relativno napako in analizirajte, kako se je točnost ocen spreminjala po posameznih iteracijah.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Matej Murn, z vpisno številko **63090117**, sem avtor diplomskega dela z naslovom:

Razvoj poslovnih aplikacij 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 26. februarja 2015 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Viljanu Mahniču za izdatno pomoč pri izdelavi diplomskega dela. Ravno tako bi se zahvalil Saši Rozman, družini in sošolcem za podporo in spodbudne besede med študijem.

Kazalo:

Kazalo slik:

Kazalo grafov:

Seznam uporabljenih pojmov:

Agilnost	Fleksibilnost, prilagodljivost
Daily Scrum meeting	Dnevni sestanek
Document Object Model (DOM)	Document Object Model, specifikacija predstavitve objektov na spletni strani
Extreme Programming (XP)	Extreme Programming metoda za razvoj aplikacij
Framework	Ogrodje, ki nam omogoča boljšo strukturo, hitrejši razvoj ter boljšo konsistentnost na nekem razvojnem jeziku
Funkcionalnost	Ena izmed zahtev, ki jih poda naročnik in jo želi avtomatizirati oz. podpreti ali izboljšati v aplikaciji
Lastnik/i projekta	Oseba, ki zastopa interese naročnika in vzdržuje seznam zahtev
Model View Controller (MVC)	Model View Controller način strukturiranja kode
Planning Poker (PP)	Planing poker, metoda za ocenjevanje uporabniških zgodb
Product backlog	Seznam zahtev, kjer imamo zbrane vse zahteve v obliki uporabniških zgodb
Scrum master	Oseba, ki usklajuje delo na projektu, nadzira in poroča nadrejenim
Sprint	Izraz, ki ga Scrum uporablja za iteracijo, v kateri se razvija uporabniške zgodbe
Sprint backlog	Prostor, kjer imamo shranjene vse zgodbe, ki so v posameznem sprintu
Sprint planning meeting	Sestanek, kjer se določi vsebino sprinta
Sprint retrospective meeting	Sestanek, za pregled dela na koncu sprinta
Sprint review	Sestanek, za pregled dela na koncu sprinta
Team Estimation Game (TEG)	Team estimation game metoda za ocenjevanje uporabniških zgodb
Uporabniška zgodba	Oblika zapisa posamezne zahteve uporabnikov

Povzetek

Diplomsko delo obravnava tematiko agilnega razvoja programske opreme. V sklopu dela so najprej opisane tradicionalne in agilne metode s poudarkom na metodologiji Scrum. Nato sledita dve osrednji poglavji. Prvo izmed njiju opisuje uporabo metodologije Scrum na realnem projektu implementacije rezervacij športne dvorane, v sklopu katerega smo podrobneje opredelili zahteve, oblikovali uporabniške zgodbe, izvedli njihovo ocenjevanje po metodah *planning poker* in *team estimation game* ter se nenazadnje dotaknili tudi uporabljenih tehnologij. Drugo pa obsega analizo točnosti ocen zahtevnosti posameznih uporabniških zgodb, v kateri primerjamo čas, ocenjen po metodah *planning poker* in *team estimation game*, z dejansko porabljenim časom. V zaključnem delu je podan še pregled težav, s katerimi smo se srečali med implementacijo in predlogi za njihovo rešitev.

Ključne besede: scrum, planning poker, team estimation game, uporabniške zgodbe, agilne metode, razvoj programske opreme, ocenjevanje, točnost ocenjevanja

Abstract

The thesis deals with the agile software development. In thesis we first describe traditional and agile methodologies for software development with emphasis on Scrum methodology. Then two main chapters follow. First of them describes how to use Scrum methodology on a real project for reservation of sport hall. In accordance with the Scum guidelines user stories were formed and estimated. Estimation was made by using *planning poker* and *team estimation game* methods. Within chapter used technology for implementing application is described. In the second chapter we analyse accuracy of all estimations based on methods *planning poker*, *team estimation game* and actual spent time for implementing user stories. At the end we review the problems and their solutions, which were encountered during implementation.

Keywords: scrum, planning poker, team estimation game, user stories, agile methodologies, software development, estimating, accuracy of estimation

1 Uvod

Spletni informacijski sistemi so ena izmed ključnih tematik in trendov v zadnjem obdobju. Slednji omogočajo lažje delo, lažji nadzor ter predvsem lažjo komunikacijo s potencialnimi strankami. Posredno se pojavi tudi dodatna priložnost oglaševanja, ki ga nekateri od lastnikov storitev pridoma izkoriščajo, spet drugi pa se potenciala svojega sistema še ne zavedajo v celoti.

Prednost spletnih poslovnih aplikacij je predvsem v tem, da do njih lahko dostopamo preko svetovnega spleta. Za njihovo uporabo ne potrebujemo posebne programske opreme, ampak zgolj spletni brskalnik ter internetno povezavo, s katero je dandanes opremljen že skoraj vsak namizni računalnik, mobilna naprava ali tablični računalnik. Spletni sistem je dostopen tako ključnim ljudem, kot tudi navadnim uporabnikom, ki uporabljajo zgolj nekatere funkcionalnosti, ki so jim v sistemu dodeljene. V ta namen bom tekom diplomskega dela ustrezno implementiral spletno rešitev rezervacij za športno dvorano za OŠ Stopiče.

Pri trenutnem porastu razvoja in potreb spletnih aplikacij je skorajda nujno, da se za nadzor implementacije ter vodenje ekip uporablja metoda, ki predpisuje zaporedje korakov, po katerih poteka razvoj. V diplomskem delu se bomo osredotočili na razvoj poslovnega informacijskega sistema rezervacij športne dvorane za OŠ Stopiče, po metodi Scrum.

Med razvojem pričakujemo dodatne izzive s specifikacijo uporabniških zgodb, saj se velikokrat izkaže, da ima uporabnik neko funkcionalnost popolnoma drugače zamišljeno kot oseba, ki jo implementira, zato bomo že na samem začetku poskušali priti do kar se da natančnih opisov in primerov uporabe posamezne uporabniške zgodbe.

Pred samim začetkom implementacije bomo ocenili posamezne zgodbe po metodah *team estimation game* in *planning poker*, katere bodo služile za natančnejšo določitev časovne zahtevnosti uporabniških zgodb. Ocene bomo primerjali z dejansko porabljenim časom ter na koncu naredili pregled natančnosti ocenjevanja in vzrokov za njihova odstopanja. Do sprememb ocen uporabniških zgodb lahko pride tudi pri predaji aplikacije in med samim razvojem funkcionalnosti.

V diplomski nalogi se bomo v drugem poglavju najprej seznanili z nekaterimi metodologijami, ki se uporabljajo pri razvoju programske opreme. V danem sklopu si bomo tako ogledali nekaj izmed tradicionalnih in agilnih metod, ki se uporabljajo. Pri tem bomo največ pozornosti namenili opisu agilne metodologije Scrum. V nadaljevanju bomo v poglavju tri opisali izvajanje metodologije Scrum na realnem projektu rezervacij. Tekom izvajanja bomo zbrali veliko število podatkov, ki jih bomo v poglavju štiri analizirali. Izvedli bomo analitični pregled po sprintih, na koncu pa si bomo ogledali še vzroke in težave, na katere smo naleteli med implementacijo.

2 Metodologije za razvoj programske opreme

2.1 Tradicionalno vodenje projektov

Tekom poglavja si bomo ogledali dve izmed osnovnih tradicionalnih metod, ki se še vedno uporabljajo za nadzor in razvoj programske opreme. Opisali bomo njihov življenjski cikel in ključne korake, ki so temelj vsake metodologije.

2.1.1 Waterfall / Linearno sekvenčni model

Model je bil definiran leta 1970 s strani Winston W. Royca [10]. Osnovna ideja modela je, da mora biti pred prehodom v naslednjo sekvenco predhodna končana (Slika 1). Da se zagotovi skladnost z zahtevami, je ob koncu vsake projekt pregledan.

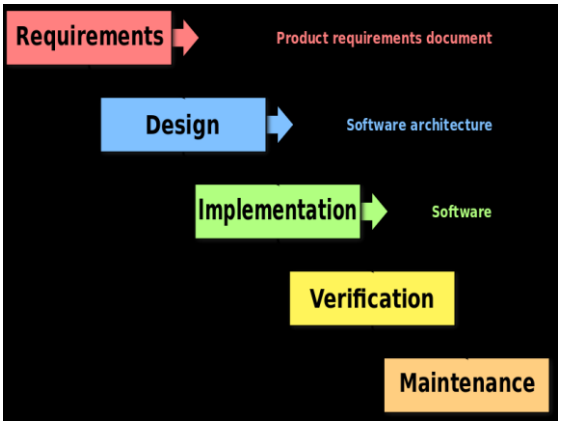
Modelu lahko pripišemo naslednje prednosti:

- dokumentacija in struktura sta prednost pri vključitvi novih članov tima,
- enostavno za razumevanje in uporabo,
- enostavna koordinacija.

Vendar ima tudi nekaj pomanjkljivosti:

- slaba fleksibilnost,
- novi zahtevki posredno vodijo k dodatnim stroškom,
- težko je ocenjevati čas in vire,
- ni prototipov, dokler življenjski cikel ni zaključen,
- pri testiranju in pojavu problemov se je težko vrniti v stopnjo načrtovanja …

Z uporabo metode dobimo najboljše rezultate pri projektih, ki so zelo dobro definirani, pri projektih, kjer ne prihaja do novih zahtev za implementacijo ter je vsa razvojna ekipa natančno seznanjena s tehnologijo, ki se uporablja, tako, da med implementacijo ne prihaja do učenja in spoznavanja zmožnosti uporabljene tehnologije [15].



Slika 1: Življenski cikel Waterfall modela [15]

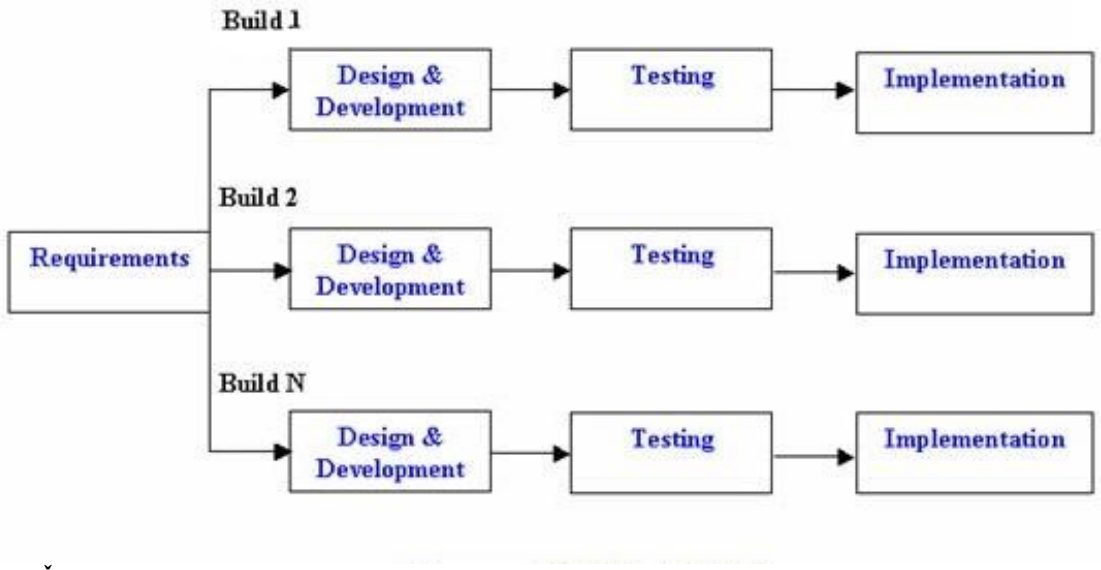
2.1.2 Kaskadni model

Model bi z drugimi besedami lahko imenovali tudi kaskadni (angl. Multi-waterfall) [13] model. Osnovni princip metode je, da zahteve razdrobimo v manjše obvladljivejše množice zahtev. Drobitev ponavljamo, dokler ne pridemo do posameznega lahko obvladljivega problema. Vsaka množica mora iti preko naslednjih stopenj:

- implementacija,
- analiza,
- oblikovanje,
- testiranje.

V vsaki stopnji se obstoječa funkcionalnost dogradi in doda nove (slika 2), vendar se pri tem ni mogoče vračati na predhodno stopnjo. Cikel se ponavlja, dokler ni sistem končan. Prednost modela je v tem, da po vsaki stopnji dobimo delujoč produkt, ki ustreza zahtevam stranke, prav tako dobimo večji odziv uporabnikov, ob pojavitvi novih zahtev pa slednje lahko vključimo v nov prototip. Med vsemi prednostmi najdemo tudi nekaj slabosti:

- težje odpravljanje napak,
- stranka lahko vidi, kaj je možno narediti in zahteva več,
- končni stroški so večji,
- slabo definirana celotna rešitev lahko vodi do velikega števila novih zahtev, kar preloži roke in predvsem poveča stroške [13].



Slika 2: Življenjski cikel kaskadnega modela razvoja [16]

2.2 Agilni pristop k razvoju programske opreme

V slednjem podpoglavju bomo pogledali, kaj pomeni agilen pristop k razvoju programske opreme ter, kateri so njegovi ključni dejavniki, po katerih lahko sklepamo, da razvoj poteka agilno. Ogledali si bomo specifične značilnosti, prednosti in slabosti metod, ki jih označujemo za agilne. Na tem mestu naj poudarimo, da poleg opisanih metod obstaja še mnogo drugih, ki imajo bodisi svojevrsten pristop ali pa se uporabljajo v kombinaciji s katero že znano metodo. Metode, ki jih bomo opisali so bile izbrane na podlagi popularnosti njihove uporabe [13].

Podlaga za razvoj vsake izmed metodologij so bile pomanjkljivosti nekega procesa, ki so ga udeleženci želeli nadzorovati podrobneje ali pa so s pomočjo uporabljene metodologije želeli izboljšati sodelovanje v razvojni ekipi. V ta namen so se oblikovale v naslednjih podpoglavjih opisane metode, ki so zaradi svoje vsebine, dinamike in prilagajanja zelo primerne za projekte, kateri so nepredvidljivi ter pri katerih začetne zahteve niso dokončno definirane.

Dinamika življenja in poslovanja se je razvila do te stopnje, da je komunikacija med razvojno ekipo ter ostalimi zaposlenimi v podjetju nujna, kar je tudi ena izmed glavnih značilnosti agilnih metodologij. Mednje uvrščamo tudi:

- kaskadni razvoj,
  Zaradi vnaprej določenih dolžin sprintov ter implementiranih zahtev se implementirane rešitve hitreje predajajo v uporabo.
- intenzivno sodelovanje z uporabniki,
  Med uporabniki, razvijalci in ljudmi, ki bodo v stiku z rešitvijo, ki se implementira, poteka nenehna komunikacija in izmenjava zahtev ter rešitev.
- enostavnost,
  Metoda je zelo enostavna za uporabo. Kot bomo lahko videli v nadaljevanju, po podobnem principu deluje že veliko ekip, vendar ga poimenujejo drugače,
- prilagodljivost,
  Eden izmed pomembnejših dejavnikov je vključevanje novih zahtevkov in sprememb v trenuten cikel razvoja [3, 11].

Prednosti agilnih metod vidimo predvsem v fleksibilnosti, odzivnosti ter v bolj zadovoljnih končnih uporabnikih. Metode nam omogočajo celovit nadzor nad razvojem, težavami, zaostanki ter implementiranimi funkcionalnostmi, kar nam omogoča, da lahko v določenem trenutku sprejmemo pravilne odločitve glede na sam potek implementacije funkcionalnosti. Hkrati nam omogočajo tudi visoko stopnjo prilagodljivosti glede na nove zahteve ter želje morebitnih nadrejenih za spremembe. Nenazadnje je to najpomembnejši del samega nadzora razvoja, saj, kot bomo videli v nadaljevanju, veliko idej še nima popolnoma izoblikovanje končne funkcionalnosti. To posledično privede do sprotnega popravljanja in izboljševanja. Med slednje metode spada tudi metoda Scrum [8], kateri sem posvetil podrobnejši opis v poglavju 2.2.4.

Pri vseh prednostih in širini uporabe metodologij je ključno, da izpostavimo tudi slabšo stran. Agilne metode odlično poskrbijo za vključitev novih zahtev, ki jih lastnik projekta definira. Vendar se moramo istočasno odločiti, katere zahteve imajo večjo poslovno vrednost ter prioriteto, da jih bomo v naslednjih sprintih implementirali. Posledično to pomeni, da bodo določene uporabniške zgodbe ostale ne implementirane, saj imajo zgodbe z večjo prioriteto prednost, v sprint pa lahko umestimo samo uporabniške zgodbe, katerih vsota ocenjene zahtevnosti ne presega predvidene hitrosti sprinta. Slednje lahko preloži implementacijo zahtev, ki imajo manjšo pomembnost, vendar so ravno tako ključnega pomena za aplikacijo. Vse to vpliva na končni rok, do katerega mora biti rešitev pripravljena za predajo naročniku.

2.2.1 Ekstremno programiranje

Ekstremno programiranje (angl. Extreme programming – XP) [3, 12] je metodologija razvoja programske opreme, primerna predvsem za majhne, do srednje velike skupine. Metodologija se je razvila na osnovi že znanih praks in principov razvoja programske opreme. Ideja slednje je, da v razvojni ekipi ni deljenih del, temveč se določi samo lastnika, ki odstopa po njegovi pomembnosti in določa funkcionalnosti ter prioritete. Vsi sodelujoči v ekipi lahko k razvoju pripomorejo enako, po svojih zmožnostih in izkušnjah. Dobre prakse sodelujočih članov ekipe pa se uvede v razvojni proces. Metoda predvideva dva ključna koraka:

1. napovedati, kaj bo dokončano do nekega datuma,
2. določiti, kaj narediti naslednje.

Hkrati pa je dober primer pravila 20-80, ki pravi naslednje: *80% koristi implementacije pride iz 20% dela*, iz česar lahko razberemo, da bomo za implementacijo ključnih funkcionalnosti predvidoma porabili 20% predvidenega časa. Metoda predvideva naslednjih 12 ključnih praks, katerim naj bi sledili pri razvoju po metodi XP [2]:

1. uporabniške zgodbe (angl. User stories),

Stranka izrazi željo po določenih funkcionalnostih, ki jih oblikujemo v obliki uporabniških zgodb, katerim se določi poslovna vrednost ter prioriteta. Slednje bo razvojna ekipa uporabila za ocenjevanje ter vodenje projekta.

2. majhne izdaje (angl. Small releases),

Ob vsaki implementirani uporabniški zgodbi, se slednjo doda k že delujoči celoti. S tem zagotovimo konstantno nadgrajevanje sistema.

3. metafora (angl. Metaphor),

Pojem metafore metoda XP uporablja za razlago tehnične vsebine projekta naročniku ter vsem sodelujočim na način, ki je vsem enostavno razumljiv, brez specifičnega predhodnega znanja o tehnični zasnovi projekta.

4. skupinsko lastništvo kode (angl. Collective ownership),

Vsi razvijalci si lastijo kodo, zato so vsi člani zadolženi za pregledovanje ter popravljanje morebitnih napak. Vsak izmed članov razvojne ekipe mora razumeti vso kodo, v njegovi domeni pa je tudi poprava napak ostalih članov oz. odprava pomanjkljivosti, ki jih opazi. S tem doseženo konstantno pregledovanje sistema ter preprečimo podvajanje kode.

5. standardiziran način kodiranja (angl. Coding standards),

Vsi člani razvojne ekipe naj bi imeli isti način pisanja kode. Slednje je zelo pomembno zaradi skupnega lastništva, saj je koda vsem bolj razumljiva. Prednosti pa se pokažejo tudi pri uvajanju novih razvijalcev.

6. enostaven načrt (angl. Simple design),

Najboljši načrt je najenostavnejši, ki še omogoča funkcionalnosti, ki jih stranka želi. Pri tem mora prestati vse teste, ki smo jih napisali.

7. prestrukturiranje kode (angl. Refactoring),

Pri dodajanju novih funkcionalnosti k obstoječim, se pojavijo nove nekonsistentnosti ter možnosti za optimizacijo. Zaradi skupnega lastništva kode ima vsak član razvojne ekipe dovolj znanja, da lahko naredi optimizacijo ali uskladi nekonsistentnosti.

8. testiranje (angl. Testing),

Za določeno funkcionalnost se najprej napiše test, na podlagi katerega razvijemo funkcionalnost. Slednje poskrbi za dolgoročno lažje odpravljanje napak, predno aplikacija postane prevelika in neobvladljiva za odpravo napak brez napisanih testov.

9. programiranje v parih (angl. Pair programming),

Programerji razvijajo funkcionalnosti v parih. Pri tem eden izmed programerjev kodira, medtem ko drugi pregleduje kodo ter opozarja na napake ter možne izboljšave. Kot rezultat programiranja v parih, dobimo boljšo kodo.

10. sprotna integracija (angl. Continuous integration),

Aplikacijo se dnevno nadgrajuje. Slednje poskrbi, da nihče izmed razvijalcev ne ostane ujet v svojem delu, ampak so vsi obveščeni o trenutnem stanju ter napredku, ki ga naredijo.

11. 40-urni delavnik (angl. 40-hour workweek),

Za uspešnost metode XP je ključno, da so razvijalci spočiti in visoko motivirani. Z omejitvijo delavnika dosežemo manj napak pri razvoju ter optimalnejše rešitve problemov.

12. prisotnost stranke (angl. On-site customer)

Ena izmed najpomembnejših praks XP je nenehna dostopnost stranke. Njena dostopnost je ključnega pomena, saj določi prioriteto uporabniških zgodbam, posreduje nove zahteve ter odgovarja na vprašanja.

Pri metodi lahko zasledimo tudi nekaj slabosti, izmed katerih bi izpostavili uporabo metode v velikih skupinah, uporabo tehnologij, ki ne dopuščajo hitrih sprememb ter nenazadnje tudi prisilno uvajanje posameznikov v uporabo posameznih praks. Slednje se še posebno lahko zelo razlikuje od trenutnih, ki prinašajo dobre rezultate in so za razvijalce udobne.

2.2.1.1 Življenjski cikel metode XP

Vsaka metoda ima določen življenjski cikel, po katerem se sodelujoči ravnajo in upoštevajo karakteristike posameznega. XP ima naslednje življenjske korake [8] (slika 3):

1. faza raziskovanja,

Stranka napiše zahteve (lahko v obliki uporabniških zgodb), za katere želi da so vključene v aplikacijo.

2. faza planiranja,

Člani razvojne ekipe ocenijo posamezne uporabniške zgodbe, določijo prioriteto in število iteracij in jih uvrstijo v posamezen sprint.

3. faza sprintov,

V vsakem posameznem sprintu se implementira določena vsebina funkcionalnosti in izvede testiranje.

4. faza predajanja rešitve v produkcijo,

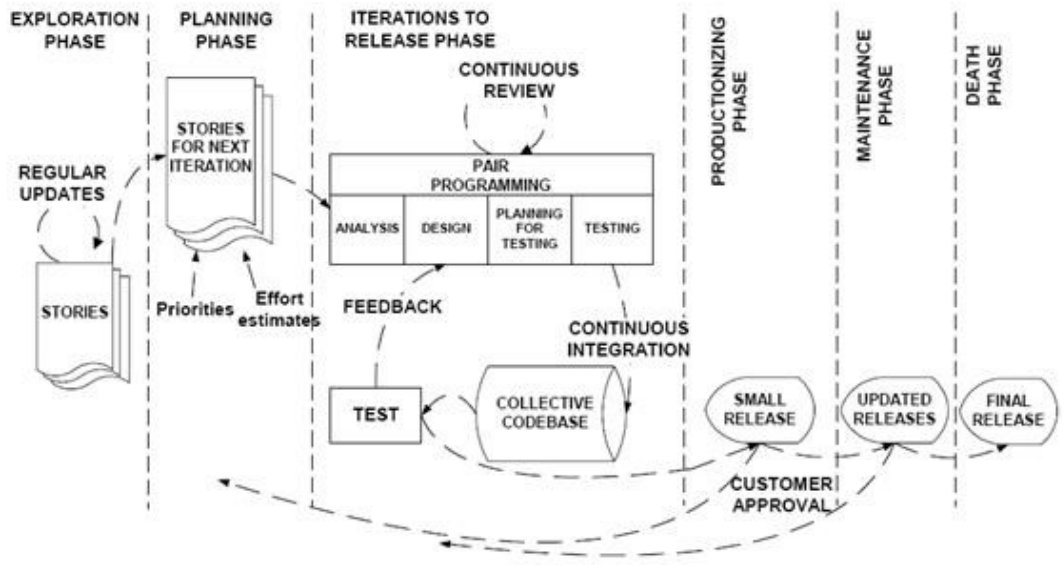
V fazi predaje se izvede zadnje testiranje s prej pripravljenimi testi, nato se sistem preda stranki.

5. faza vzdrževanja,

Vsaka ideja, predlog ali nova želja po dodatnem modulu po zaključeni fazi predaje rešitve v produkcijo se implementira v sklopu vzdrževanja.

6. faza smrti projekta

Stranka nima nobene nove zahteve za implementacijo, dokumentacija za obstoječe pa je napisana. V kolikor ne bo prišlo do nobene spremembe na nivoju arhitekture same aplikacije, lahko fazo označimo kot smrt projekta.



Slika 3: Cikel metode XP [17]

Kot smo lahko videli metoda XP nekako povzema najboljše pristope razvoja programske opreme ostalih, do sedaj omenjenih metod na način, da jih združuje v smiselno celoto. Samo s slednjim lahko namreč zagotovi njihovo temeljito izvajanje ter medsebojno dopolnjevanje različnih praks.

2.2.2 Kanban

Metodologija izhaja iz Japonskih besed, ki pomenijo naslednje: Kan: *signal* in ban: *kartica*. Kanban je razvil Taiichi Ohno pri Toyoti [18]. Metodologija uporablja nabor stanj, v katerih se lahko nahaja določena funkcionalnost v izdelavi in so prikazana v obliki table (slika 4) [6]. Layman je metodologijo Kanban opisal kot:

- pot za organizacijo kaosa, ki obkroža razvojne ekipe, s katerim dosežemo boljši fokus ter prikažemo prioritete,
- pot, s katero odkrijemo probleme, da jih lahko uspešno rešimo in dostavimo bolj konsistentni končni produkt k stranki.

Metodologijo lahko v grobem razbijemo na štiri osnovne principe, ki jih je potrebno upoštevati, če jo želimo dosledno izvajati. Principi so:

1. začni s tem, kar delaš sedaj,

Kanban ne predpisuje posebne vzpostavitve oziroma procedure za uvajanje na trenuten projekt. Ravno tako ne posega v spremembe obstoječega načina dela, vendar zgolj vzpostavlja mehanizem za nadgradnjo trenutnega načina dela. Tako je uvajanje metode zelo enostavno.

2. postopen razvoj: dogovor o naknadnih zahtevkih za izboljšavo,

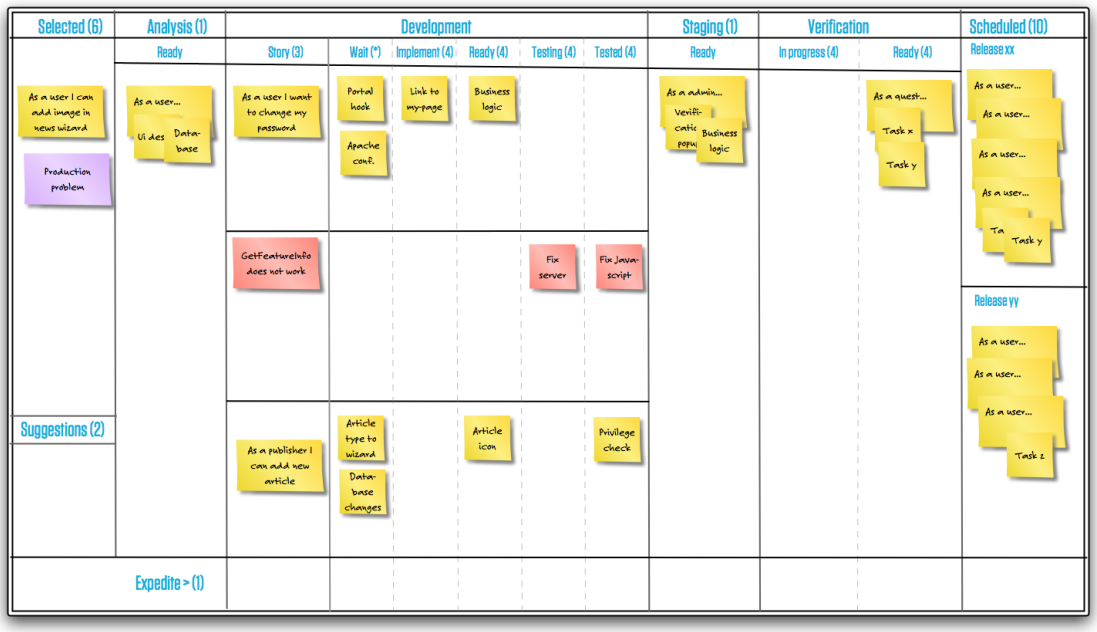
Metoda je zasnovana na principu minimalno porabljenega časa za naknadne spremembe, pri tem pa spodbuja neprekinjene sprotne spremembe v obstoječem sistemu. Večje spremembe so nezaželene, ker pri razvijalcih in uporabnikih spodbujajo strah po odpovedih delujoče kode ter vnašajo negotovost.

3. spoštuj trenutni razvoj, pravice in odgovornosti,

Kanban prepoznava pomembnost določenih vlog, odgovornosti ter nazivov. Ravno tako ne vsiljuje svojih sprememb po določanju hierarhije, pušča pa odprte možnosti za postopni razvoj, ki smo ga opisali v prejšnji točki.

4. spodbujaj nove ideje,

Poznamo veliko projektov, kjer najboljše ideje niso prišle od vodij projektov ter strank, ampak od samih razvijalcev ter inženirjev, ki so se s problemom ukvarjali. Metodologija zato spodbuja odprtost in izmenjavo mnenj. Nenazadnje se sama vpeljava metodologije lahko izvede tudi preko inženirja in ne samo preko vodje projekta [19].



Slika 4: Življenjski cikel Kanban metodologije [29]

David Anderson v knjigi *Kanban – Successful Evolutionary Change for your Technology Business* izpostavlja naslednje ključne postavke pri uvedbi kanban metodologije [1]:

1. vizualizacija poteka dela,

Najenostavnejši način je vizualizacija z že prej omenjenimi signalnimi karticami na pravi ali virtualni deski, kjer so označena posamezna stanja in dovoljeni koraki dela. Glede na problem, nad katerim želimo izvajati kanban sestavimo poljuben potek dela, ki ga metodologija ne definira.

2. omejitev dela, ki je v izdelavi,

Vsako stanje, v katerem se lahko znajde določena zahteva lahko omejimo na število slednjih, ki se lahko v nekem času v njem nahajajo. Ukrep je hitri pokazatelj zaostanka, novih nepredvidenih izzivov ter morebitnih problemov, na katere moramo biti pozorni oziroma jim nameniti več časa.

3. nadzorovanje delovnega toka,

Predno lahko delo uredimo na posamezna stanja je dobro, da trenuten potek razvoja preučimo s stališča poteka razvoja. Nato postopoma uvedemo določene spremembe iz metodologije kanban in spremljamo rezultate. Proces se ne zaključi nikoli, kar je tudi eden izmed pokazateljev, da moramo biti vedno usmerjeni k izboljšavam.

4. definiraj cilj, potek dela in seznani ekipo,

Ko je proces definiran, ga je potrebno prestaviti in vpeljati tudi med ostale člane razvojne ekipe, saj procesa ne moremo izboljšati, če ga ne razumemo oziroma ne poznamo. Po uspešni prestavitvi lahko začnemo s samo vpeljavo in spremljanjem sprememb.

5. izboljšaj potek dela v sodelovanju z vsemi deležniki,

Na koncu, ko vsi člani razvojne ekipe razumejo potek dela, vsebino metodologije kanban se ekipa lahko osredotoči na nove izboljšave.

2.2.3 Dynamic system development

Metodologija Dynamic System Development [20] je bila razvita v Angliji, leta 1994. Značilnost metodologije je, da fiksira čas ter vire, na podlagi česar se slednje prireja glede na potrebo za uspešno implementacijo neke funkcionalnosti. Glavne faze metodologije so naslednje (slika 5):

1. študija izvedljivosti (angl. Feasibility study),

V fazi se sodelujoči odločijo ali bodo metodologija uporabili ali ne glede na tip projekta, nad katerem bi se metoda izvajala ter njegovo fleksibilnostjo.

2. študija poslovanja (angl. Business study),

Da sodelujoči bolje razumejo samo problematiko področja, se naredi študija poslovanja. S slednjo se zagotovi tudi osnovno definicijo sistemske arhitekture in plan prototipnih rešitev.

3. funkcijska iteracija modela (angl. Functional model),

V tej iteraciji se naredijo prvi primeri, testi in prototipi funkcionalnosti. Rezultati in dognanja med samim reševanjem se uporabijo za nadaljnje izboljšave in posodobitve ostalih delov določene funkcionalnosti.

4. izgled in postavitev projekta na predogled uporabnikom (angl. Design & Build),

V tej fazi se funkcionalnost postavi v stanje, ko ga lahko uporabniki uporabljajo. Ti podajo svoje mnenje, glede na katerega se potem organizira nadaljnji razvoj.

5. implementacija (angl. Implement),

Zadnja faza služi predaji sistema uporabnikom. Po potrebi se zagotovi izobraževanje, pregled poteka same implementacije, izroči pa se tudi uporabniški priročnik. Na tem mestu se namesto zaključka metodologija lahko nadaljuje v katerekoli fazi, z namenom, da se izboljša funkcionalnost, ki je bila narejena [1].

Slika 5: Slikovni cikel metode DSDM [20]

2.2.4 Scrum

Pojem Scrum izhaja iz športa, imenovanega *rugby*. Z njim je poimenovana ekipa osmih igralcev, katerih cilj je jasen: delovati kar se da celovito, povezano in kot skupina strmeti k doseganju skupnega cilja. Ko prenesemo idejo s športa na razvoj programske opreme, bi jo predstavili na naslednji način: *Scrum je orodje/ogrodje za nadzorovanje ter usmerjanje razvojne ekipe, kjer vsak član točno pozna svojo vlogo ter skupen cilj* [1]. Metoda ne uporablja nobenega posebnega/novega pristopa, saj so vsi pristopi že dobro poznani pri samem vodenju in razvoju programske opreme. Njeno moč lahko še posebno opazimo pri projektih, kjer zahteve v naprej niso najbolje definirane ter, kjer se pričakuje veliko sprememb zahtev ali vodstva samega projekta [9]. Metodologija za sprotni pregled opravljenega dela predpisuje določene sestanke, ki jih bomo v nadaljevanju opisali. Ravno tako bomo natančneje opredelili principe metodologije, ki so ključnega pomena za razumevanje in njeno pravilno izvajanje.

2.2.4.1 Opredelitev osnovnih pojmov

Za natančnejše razumevanje metodologije bomo opredelili nekaj pojmov, ki se tekom izvajanja redno pojavljajo in so ključnega pomena za dobro razumevanje. Izvajanje metodologije se izvaja v tako imenovanih iteracijah imenovanih **Sprint**. Sprint je časovno omejeno obdobje omejeno med enim ali štirimi tedni, med katerim vsak član ekipe razvija svoje zahteve, ki so bile uvrščene v posamezen sprint. Med tem časom spremembe zahtev, ki so v trajajočemu sprintu niso dovoljene. Zahteve, ki so bolj obširne in vsebujejo večji faktor tveganja naj bi bile uvrščene v zgodnejše sprinte, saj imajo tekom naslednjih sprintov več

časa za morebitno odpravo napak. Konec vsakega sprinta imamo zaključeno delujočo celoto, ki je na voljo za uporabo.
Med samim opisom sprinta smo že omenili pojem razvijalcev, ki v skupini tvorijo skupino razvijalcev (angl. **Team**), ki so zadolženi za implementacijo rešitev. Skrbnik metodologije (angl. **Scrum master**) skrbi za pravilno izvajanje metodologije Scrum, o njenem poteku poučuje ekipo razvijalcev, medtem ko ima lastnik projekta (angl. **Product owner**) nalogo podajanja uporabniških zgodb (angl. **User stories**). Skupek uporabniških zgodb se imenuje seznam zahtev (angl. **Product backlog**) in se tekom razvoja aplikacije lahko dopolnjuje z novimi zahtevami. Pred začetkom vsakega sprinta člani ekipe izberejo posamezne uporabniške zgodbe, za katere menijo, da jih bodo lahko realizirali. Slednje so zbrane v seznamu nalog za določen sprint (angl. **Sprint backlog**). Predstavljene pojme bomo natančneje opisali v nadaljevanju dela. [9, 10]

2.2.4.2 Uporabniške vloge

Metodologija Scrum predvideva tri uporabniške vloge, ki so ključnega pomena za izvajanje razvoja v skladu s predvidenimi smernicami metodologije. Slednje poskrbijo za pravilno izvajanje metodologije, uspešno implementacijo in pojasnjevanje slabše definiranih uporabniških zgodb. Definicijo in zadolžitve vsake si bomo ogledali v nadaljevanju.

- Skrbnik metodologije (angl. Scrum master)

Skrbnik metodologije je oseba, ki vodi dnevne/tedenske sestanke, skrbi za njihovo dosledno in pravilno izvajanje in nenazadnje skrbi za pretok informacij med ekipo ter lastnikom projekta oziroma, če imamo več ekip z ostalimi skrbniki. Ena izmed njegovih zadolžitev je tudi spremljanje in izločanje morebitnih dejavnikov tveganja, ki bi lahko povzročili nepredvideno prepreko pri predaji in implementaciji zahtev. Poleg tega nadzoruje delovanje ekipe, da delo poteka v skladu s pravili, ki jih predpisuje metodologija Scrum. V okviru dolžnosti je zadolžen za predstavitev metodologije lastniku projekta (angl. Project Owner), v namene maksimiranja donosnosti naložbe (angl. ROI – return of investment).

- Lastnik projekta (angl. Product Owner)

Lastnik projekta je predstavnik vseh vlagateljev v projekt in zastopa njihove interese. Slednji je zadolžen za določitev funkcionalnosti projekta v obliki uporabniških zgodb (angl. User Stories), na podlagi katerih se oblikuje seznam zahtev (angl. Product Backlog). Pred vsakim izmed sprintov lastnik določi prioriteto preostalih uporabniških zgodb za implementacijo. S slednjim želi doseči najvišji možen ROI (angl. Return of investment) in največjo možno dodano vrednost implementiranih zgodb ob končanem sprintu. Njegova vloga obsega tudi testiranje implementiranih zgodb ter pojasnjevanje glede njihove nejasnosti, ki se pojavijo razvojni ekipi. Njegovo neprestano sodelovanje je zelo pomembno, saj uporabniške zgodbe nikoli niso dokončno definirane in se odražajo glede na pogoje in zahteve uporabnikov in naročnika.

- Razvojna skupina (angl. Team)

Glavna zadolžitev razvojne skupine je učinkovita implementacija uporabniških zgodb. Razvojna ekipa je kolektivno odgovorna za uspešno implementacijo in delitev dela znotraj posameznega sprinta. Sodelujejo tudi pri uvrščanju uporabniških zgodb v posamezen sprint, saj imajo največ izkušenj z implementacijo, na podlagi katerih natančneje izberejo uporabniške zgodbe, ki jih bo moč implementirati med posameznim sprintom.

2.2.4.3 Končni izdelki

Metodologija Scrum nudi celovit način vodenja projektne ekipe. Pri slednjem imamo ob koncu projekta kar nekaj končnih izdelkov, iz katerih lahko naredimo analizo in poročila. Končne izdelke bomo opisali v nadaljevanju.

- Seznam zahtev (angl. Product backlog)

Zahteve se zbere v tako imenovani seznam zahtev (angl. Product backlog), ki zajemajo vse funkcionalnosti aplikacije. Seznam ni nikoli končno definiran, kar omogoča lastniku projekta nenehno dodajanje in spreminjanje že definiranih uporabniških zgodb. Ocenjevanje zahtev je zadolžitev razvojne ekipe, ki lahko s svojimi izkušnjami zagotovi kar najnatančnejšo oceno. Slednja je temelj pri izbiranju zgodb za določen sprint ter za planiranje izdaj.

- Seznam nalog (angl. Sprint backlog)

Seznam nalog se sestavi na podlagi zahtev, ki so uvrščene v posamezen sprint. Razvojna ekipa je zadolžena, da posamezno zgodbo razbijejo na manjše dele. Med sprintom je razvojna ekipa zadolžena za vzdrževanje atributov, kar nam v vsakem trenutku ponudi celovit pregled nad trenutnim stanjem sprinta. Med atribute tako štejemo čas, ki je že bil vložen v realizacijo, predviden čas za dokončanje realizacije, podatek o osebi, ki je zadolžena za implementacijo itd...

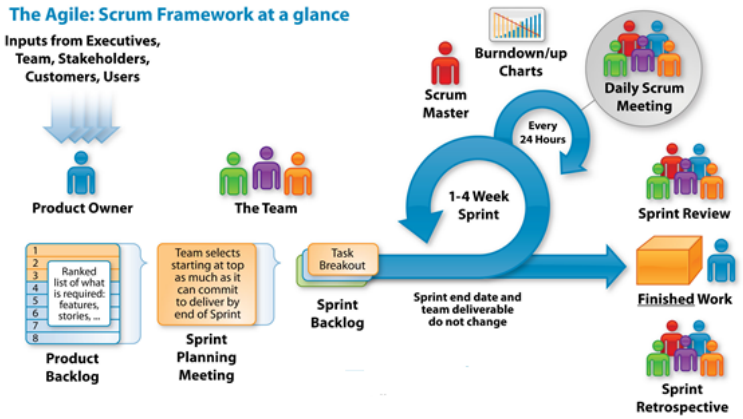
- Inkrement delujoče programske kode

Ob koncu vsakega sprinta je bistveno, da imamo delujočo programsko kodo. Slednja naj bi vsebovala popolno implementacijo uporabniških zgodb, ki smo jih umestili v posamezen sprint. Pri tem je metodologija zelo striktna, saj morajo zgodbe prestati vse sprejemne teste, na koncu pa jih mora potrditi tudi lastnik projekta. Implementirane in potrjene uporabniške zgodbe se označi kot zaključene. Vse nedokončane zgodbe se uvrsti v naslednji sprint in ponovno oceni potreben čas preostanek implementacije. Šele, ko zadostijo vsem že omenjenim pogojem se jo označi kot opravljeno.

Poleg redne implementacije uporabniških zgodb so člani razvojne ekipe zadolženi tudi za sprotno pisanje dokumentacije, ki je namenjena uporabnikom. Slednja mora biti napisana nedvoumno za laične uporabnike sistema, kateri morajo v njej najti večino odgovorov, ki se jim zastavijo med samo uporabo sistema.

2.2.4.4 Potek Scrum metodologije

Za uspešno izvajanje metodologije Scrum moramo slediti določenim principom uporabe, ki jih metodologija predvideva. Slika 6 prikazuje življenjski cikel metodologije. Iz nje lahko razberemo, da metoda zahteva več vrst sestankov v določenih trenutkih razvoja. V nadaljevanju si bomo ogledali, kateri so ti sestanki in, čemu so namenjeni.



Slika 6: Življenjski cikel metodologije Scrum [28]

- Dnevni pregledni sestanek (angl. Daily scrum meeting)
  Namen dnevnega sestanka (15-30min) je, da se pregleda, v kakšnem stanju je projekt. Pri sestanku so prisotni vsi člani razvojne ekipe in vsak izmed njih mora odgovoriti na naslednja vprašanja:
  - Kaj sem naredil včeraj?
  - Kaj delam danes?
  - Kaj me upočasnjuje/kje imam probleme?

  Pri tem se zelo hitro vidi, kje so odstopanja in kje nastanejo dodatni izzivi, kar je zelo dobro, saj na ta način hitro ugotovimo napake ter jih poskušamo odpraviti. Glavne naloge dnevnega sestanka so:
  - fokusiranje razvijalcev na zahteve v trenutnem sprintu,
  - boljši pretok informacij,
  - obveščanje o napredku,
  - reševanje novo odkritih izzivov na kar se da hiter način
  - minimizacija neželenih preprek, ki se lahko pojavijo.

- Sestanek za načrtovanje sprinta (angl. Sprint planning meeting)

Sestanek, ki je namenjen načrtovanju prihajajočega sprinta imenujemo *Sprint planning meeting*. Na njem so prisotni člani razvojne ekipe, skrbnik metodologije in lastnik projekta. Naloga lastnika je, da predstavi željen napredek ter prioritetne zgodbe razvojni ekipi, katera lahko ob morebitni nejasnosti zastavi dodatna vprašanja. Ključna naloga razvojne ekipe je izbira primerne količine uporabniških zgodb, ki jih bodo realizirali v tekočem sprintu. Med slednje naj bi izbrali najprej tiste, ki doprinesejo največ vrednosti projektu in jih lastnik projekta posebno predstavi. Kot rezultat sestanka dobimo oblikovan seznam zahtev.

- Sestanek za pregled opravljenega dela (angl. Sprint review meeting)

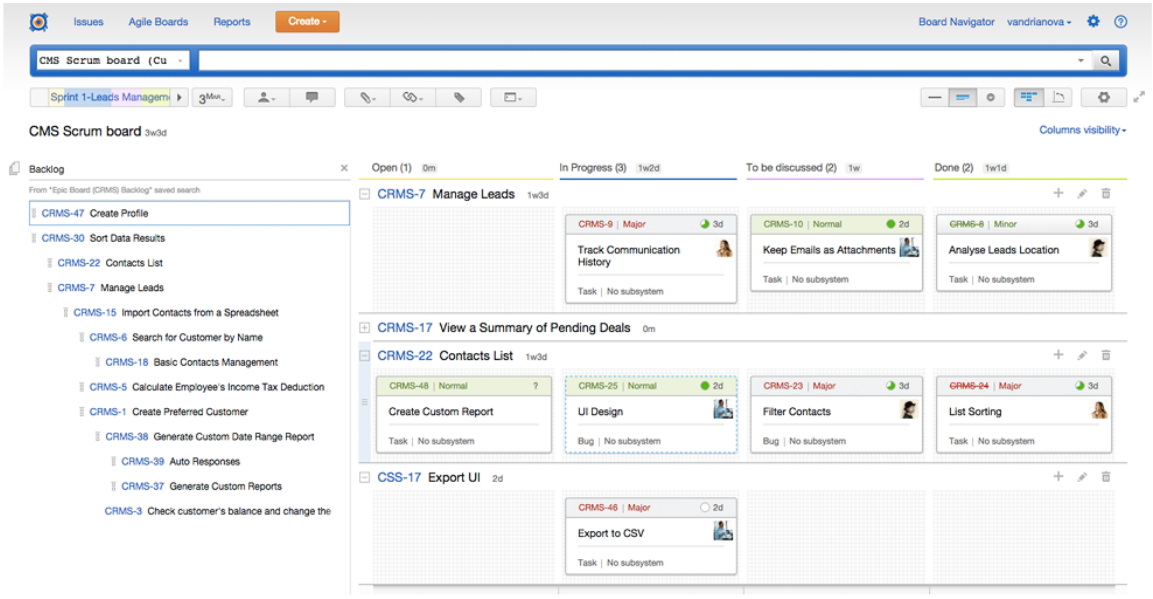
Sestanek se izvaja na koncu vsakega sprinta, katerega dolžina je predvidena na štiri ure. Na njem so prisotni razvojna ekipa, lastnik projekta in vsi zainteresirani, katerim se predstavi opravljeno delo in analizira aktivnosti trenutnega sprinta. Naloga lastnika projekta je, da oceni realizacijo posameznih uporabniških zgodb na podlagi sprejemnih testov. Vse uspešno zaključene se označijo z oznako končane, ostale pa se pustijo za nadaljnji razvoj. Sestanek ima vlogo seznanitve sodelujočih z opravljenim delom in okvirnega dogovora o nadaljnjem razvoju.

- Sestanek za pregled sprinta (angl. Sprint retrospective meeting)

Pred vsakim začetkom naslednjega sprinta razvojna ekipa skupaj s skrbnikom metodologije izvede sestanek v trajanju treh ur. Razvojna ekipa pregleda, kako uspešno je uporabila metodo Scrum v preteklem sprintu. Sodelujoči lahko na podlagi pridobljenih izkušenj predlagajo izboljšave ter dobre prakse, ki so jih uporabili. S tem pregledajo vzroke nastalih težav, s katerimi nenehno izboljšujejo razvoj in produktivnost razvojne skupine.

2.2.4.5 Orodja za Scrum

Za samo implementacijo metode Scrum ne potrebujemo naprednih aplikacij. Dovolj je že osnovno usklajevanje, dogovor o sestankih, vsebini le teh, dnevna komunikacija in že lahko pričnemo z grobo uporabo same metode. Vendar v današnji informacijski dobi obstaja veliko programov, ki nam olajšajo vodenje, pregled ter vpogled v trenutno stanje razvoja programske opreme. Sam sem za namene sledenja uporabil sistem *YouTrack* podjetja JetBrains [21].



Slika 7: Youtrackov pogled v osnovno nadzorno ploščo [21]

Obstaja še veliko ostalih alternativ, ki omogočajo podobne funkcionalnosti, vendar vse omogočajo vodenje projektov po metodologiji Scrum. Med njimi bi izpostavili naslednje, ki poleg vodenja metodologije Scrum omogočajo tudi povezavo z Git repozitorji, določenimi integracijskimi strežniki, imajo pa še mnogo drugih možnih razširitev in možnosti, ki lahko pri večjih aplikacijah zelo poenostavijo postavitev kode na produkcijski strežnik oziroma olajšajo nadzor, testiranje in razvoj aplikacij:

- https://www.atlassian.com/software/jira
- https://trello.com/
- https://asana.com/
- http://trac.edgewall.org/.

Sama izbira sistema je predmet osebne odločitve uporabnikov in skrbnika projekta ter ni omejena na izpostavljene. Slednje se lahko odloči na podlagi željene uporabe posamezne

metodologije, izgleda orodja, združljivosti orodja z drugimi orodji, ki jih ekipa uporablja za razvoj ter nenazadnje uporabniške izkušnje in celovitosti nadzora nad potekom projekta.

2.3 Specifikacija zahtev uporabniških zgodb

Vsako zahtevo naročnika lahko opišemo na več načinov. Pri agilnih metodah, posebno pri metodi Scrum je delo velikokrat opisano v obliki uporabniških zgodb, ki jih bomo uporabili. Uporabniška zgodba opisuje samo delovanje in možnosti posamezne funkcionalnosti iz zornega kota uporabnika. Sam opis je razumljiv tako lastniku projekta, razvijalcem, kot tudi uporabnikom, ki pri tem sodelujejo. Vsaka zgodba je sestavljena iz več delov, ki naj bi si sledili v naslednjem zaporedju:

- ime zgodbe,
- besedilo (kratek opis same funkcionalnosti),
- sprejemni testi (opis primerov, ki so za funkcionalnost pomembni in primerov, ki jih moramo prestreči ter obvestiti uporabnika o nedovoljenih vnosih).

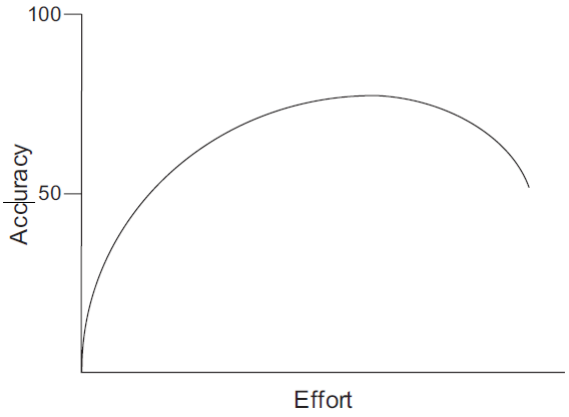
Vsaki uporabniški zgodbi se določi tudi prioriteta, ki med drugim služi za boljšo organizacijo glede porabe resursov, načrtovanja, umestitve zgodb v posamezen sprint in časovnega načrtovanja med sodelujočimi.
V našem projektu implementacije sistema za rezervacije poslovnih prostorov prioriteta ni bila pomembna. Sistem smo gradili od začetka tako, da vrstni red realizacije posameznih funkcionalnosti ni bil ključnega pomena za končnega uporabnika, kajti sistem je uporaben zgolj kot celota uporabniških zgodb, kar pa za nadaljnji razvoj ne bo veljalo več. Vseeno smo posamezen sklop predajali lastniku projekta, ki je delo pregledal ter podal mnenje.

2.3.1 Ocenjevanje uporabniških zgodb

Vsako uporabniško zgodbo želimo oceniti kar se da dobro, saj bomo s tem bolje organizirani, bolje bomo lahko načrtovali potek razvijanja in vključevanja ostalih komponent, potrebnih za uspeh nekega projekta. Pri vsem tem pa se moramo zavedati, da nam še tako natančna ocena ne more preprečiti nepredvidljivih dejavnikov. Z dobro postavljeno oceno pa se vendarle lahko pripravimo na dodatne posege in daljši razvojni čas. Posebno z agilnimi metodami, ki omogočajo hiter odziv na nove spremembe in želje naročnika ali lastnika projekta.

Za ocenjevanje naših zgodb bomo uporabili pojem točke, ki ponazarja eno uro dela na implementaciji. Tako vsaka točka pri oceni zahtevnosti pomeni dodatno uro dela na uporabniški zgodbi.

Iz slike 7 lahko razberemo, da v ocenjevanje ni primerno vložiti veliko časa, saj točnost ocenjevanja narašča zgolj do neke mere, nato pa se začne slabšati. Zato je dobro najti nekakšno razmerje, ki nam doprinese kar se da dobro in natančno oceno za določeno uporabniško zgodbo.



Slika 8: Natančnost ocene glede na vložen čas za ocenjevanje. [28]

Splošno najbolj uporabljene metode za ocenjevanje so:

- mnenje eksperta,

Najhitrejša ocena je mnenje eksperta. Slednji se pri ocenjevanju zanaša na svoje izkušnje, do sedaj izpeljane projekte in intuicijo. Vendar je pri agilnih metodah ta način ocenjevanja malo uporabljen, saj so uporabniške zgodbe oblikovane na način, ki zahteva več funkcionalnosti in lahko, da vse niso v primarni domeni eksperta, saj lahko pri njeni implementaciji sodeluje več razvijalcev. Metoda se lažje vključi v tradicionalne projekte, saj je vsaka izmed ocen povezana z zahtevo, ki jo bo implementiral en razvijalec.

- ocenjevanje po analogiji,

Pri metodi ocenjevanja se zanašamo na primerjavo posamezne zgodbe z ostalimi. Preprosto primerjamo posamezne zgodbe med seboj ter jih poskušamo smiselno oceniti, glede na kriterij ali je od neke zgodbe večja ali manjša. Metoda uporablja pojem *triangulacija,* kar pomeni, da vsako zgodbo ocenimo na podlagi primerjave z več podobnimi zgodbami (zgodbo, ki smo jo ocenili z več točkami kot trenutno in zgodbo, ki ima manj točk), ne samo z eno.

- členitev

Problem ocenjevanja lahko nastane, če imamo uporabniško zgodbo, ki vključuje veliko funkcionalnosti. Slednjega se lotimo s členitvijo zgodbe na manjše, bolj obvladljive dele, za katere lahko z večjo gotovostjo napovemo natančnejšo oceno časa za implementacijo. Vendar moramo paziti, da deli niso premajhni, saj na ta način pokvarimo celotno oceno zahteve.

Poleg vseh opisanih načinov ocenjevanja, najboljše ocene dobimo, ko jih združimo v celoto. Dve izmed metod, ki nam to omogočata sta *Planning poker* [14] ter *Team estimation game* [5, 3]. Naše uporabniške zgodbe smo ocenili po metodi *Planning poker* in po zadnji iteraciji, prav tako po metodi *Team estimation game,* katere ocene smo v poglavju 4.2 uporabili za primerjavo. Za izhodiščno oceno uporabniških zgodb smo uporabili aritmetično sredino ocen, pridobljenih po metodi *Planning poker*.

2.3.1.1 Določanje razpona ocen

Za najpogostejše časovne ocene za ocenjevanje zgodb se uporablja naslednja zaporedja:

1. 1, 2, 3, 5, 8, 13, 21…
2. 1, 2, 4, 8, 16, 32

Prvo od zaporedij je zelo znano Fibonaccijevo zaporedje, ki si sledi po naslednjem zaporedju:

$$F_n = F_{n-1} + F_{n-2}$$

Njegova uporabnost se vidi predvsem v presledku med števili, pri naraščanju le teh. Slednje ponazarjajo določeno nepredvidljivost, s katero ocenimo določeno zgodbo. Če bo zgodba imela veliko oceno, lahko vsebuje tudi več nepredvidljivih okoliščin, ki lahko vplivajo na sam razvoj. Tudi drugo zaporedje deluje na podoben način, le da si slednje ocene sledijo po enačbi:

$$n_1 = 1$$
$$n_{i+1} = 2 * n_i$$

Seveda se lahko za ocenjevanje uporabi tudi drugačno vrsto lestvice ocen, ki jo je potrebno preudarno določiti, skupaj z razvojno ekipo, da bodo slednje ocene kar se da dobro izražale zahtevnost posamezne zgodbe v trenutnem ciklu [14].

Ker je sam projekt rezervacij obsegal zelo podobne uporabniške zgodbe, za katere smo menili, da jih ne moremo oceniti z isto časovno zahtevnostjo, smo se z ocenjevalci odločili, da bo bolje, če za lestvico uporabimo množico pozitivnih realnih števil, kjer vsako število predstavlja število ur, ki jih predvidevamo za izdelavo posamezne uporabniške zgodbe.

2.3.1.2 Planning poker

Metoda *Planning poker,* za ocenjevanje uporabniških zgodb, vključuje zgoraj omenjene principe ocenjevanja zahtevnosti. Pri ocenjevanju sodelujejo vsi člani razvojne ekipe. Prisoten je lahko tudi lastnik, vendar slednji ne ocenjuje. Originalna metodologija predvideva uporabo kartic za ocenjevanje, na katerih so napisana števila, ki ponazarjajo zahtevnost zgodbe, ki jo ocenjujemo (opisano v prejšnjem poglavju). Ocenjevanje poteka po naslednjem načinu:

1. lastnik prebere opis zgodbe,
2. vsak izmed razvijalcev ima možnost postaviti vprašanje o implementaciji, zasnovi, na podlagi katerega se lahko razvije diskusija, ki ni predolga. Še vedno se poskušamo držati diagrama, ki ga ponazarja Slika 8,
3. vsak izmed ocenjevalcev izbere oceno iz kartice; kartice se obrnejo, ko imajo vsi izmed ocenjevalcev izbrano oceno,
4. pri prvem ocenjevanju lahko sklepamo, da bodo podane ocene zelo variirale. Zato sledi soočenje ocenjevalca z najvišjo in najnižjo oceno, ki predstavita vzroke za njuno oceno, skupina pa se vključi v diskusijo naknadno,
5. ponavljamo koraka 3 in 4 vse dokler se ocene ne poenotijo (če predpostavimo število ocenjevalcev na 4 in dobimo njihovo oceno 5,5,5,3, se lahko sporazumno odločimo za oceno 5 in ni potrebno ponovne iteracije, saj je osnova ocenjevanja na pridobivanju razumne ocene).

Prednosti, ki jih prinese *planning poker* so pred vsem v izmenjavi različnosti mnenj, ki lahko izpostavijo stvari, na katere niso vsi ocenjevalci pozorni. Na ta način dosežemo dobro oceno morebiti bolj specifičnega dela zgodbe (za katerega imamo v ekipi osebo, ki je na tem področju bolj domača kot ostali), kot tudi boljšo končno oceno. Z izmenjavo mnenj hitreje

odkrijemo skrite pasti, ki se pojavljajo v sami vsebini zgodbe in pri površinskem opisu zgodbe niso takoj opazne.

Vsekakor lahko vidimo, da metoda uspešno združuje različno globino znanja in izkušenj sodelujočih ter vzpodbuja dodatno diskusijo o morebitnih dvomih, ki izboljša končno oceno ter izpostavlja morebitne izzive, ki sprva niso bili opaženi [14, 36].

2.3.1.3 Team estimation game

Ena izmed največkrat uporabljenih alternativ metodi *Planning poker* je tudi *Team estimation game*. Metoda ocenjevanja temelji na naslednjih korakih:

1. prvi izmed ocenjevalcev vzame prvo kartico iz skupine zbranih kartic in jo položi na površino, kjer bo potekalo ocenjevanje,
2. naslednji ocenjevalec vzame naslednjo kartico iz skupine zbranih kartic in jo glede na zahtevnost zgodb, ki so že na ocenjevalni površini primerno umesti (levo, če misli, da je časovna zahtevnost manjša; desno, če misli, da je večja ter pod kartico na površini, če misli, da je zahtevnost enaka; kartico lahko položi tudi med dve že postavljeni, če meni, da je po zahtevnosti med njima),
3. točko 2 ponavljamo, dokler ne razporedimo vseh kartic iz skupine,
4. ko kartic iz skupine zmanjka, ocenjevalci nadaljujejo s premikanjem vrstnega reda že postavljenih kartic, v kolikor se s postavitvijo ne strinjajo (ocenjevalec lahko premikanje preskoči, če meni, da si kartice sledijo naraščajoče po časovni zahtevnosti),
5. točko 4 ponavljamo toliko časa, dokler se vsi izmed ocenjevalcev ne strinjajo z vrstnim redom položenih kartic,
6. nato točke 1 in 2 ponovimo, a tokrat namesto kartic z uporabniškimi zgodbami razvrščamo izbrano lestvico ocen nad že postavljene kartice z zgodbami (največkrat razvijalci uporabijo lestvico *Fibonaccijevega* zaporedja, po dogovoru pa lahko naredijo svoje kartice, na katerih uporabijo svojo lestvico ocen),
7. po razvrščenih karticah z ocenami imajo ocenjevalci ponovno možnost razvrstiti kartice z uporabniškimi zgodbami pod razporejene ocene, če menijo, da slednja bolj ustreza časovni zahtevnosti zgodbe,
8. točko 7 ponavljamo, dokler se vsi ocenjevalci ne strinjajo z vrstnim redom, ter ocenami težavnosti uporabniških zgodb.

Metoda je zaradi manjšega izmenjavanja mnenj med samim premikanjem kartic hitrejša kot *planning poker*. Pri ocenjevanju je potrebno paziti na izkušnje posameznega sodelujočega, saj se v primeru slabo izkušenega ocenjevalca kakovost ocene začne zanašati na bolj izkušene, kar lahko privede do ocen, ki so jih podali izkušeni sodelujoči, neizkušene pa se ne upošteva, kar ne govori v prid skupinskemu delu in ocenjevanju [4, 12, 36].

3 Uporaba metodologije Scrum na projektu

Tekom poglavja bomo predstavili uporabo metodologije Scrum na realnem projektu. Opisali bomo tehnično osnovo projekta in tudi vsebinsko namembnost. Kot edini član projektne ekipe sem bil postavljen v vlogo dveh izmed treh oseb, ki jih metodologija Scrum predvideva (skrbnik metodologije, razvijalec). Vlogo lastnika projekta je prevzel idejni oče rezervacij Murn Andrej. V nadaljevanju si oglejmo podrobnosti.

3.1 Opis projekta

V poslovnem objektu OŠ Stopiče so nedavno dobili popolnoma novo zgrajeno športno dvorano za namene predmeta športna vzgoja, ki se izvaja v vseh razredih osnovne šole (1-9. razreda). Dandanes pa ni objekta, ki ga ne bi porabili tudi za izven šolske dejavnosti, saj na ta način lastniki hitreje amortizirajo vrednost naložbe in s tem pridobijo dodatne prihodke, s katerimi pokrijejo nakup novih pripomočkov ter stroške vzdrževanja. V ta namen se je na šoli ponudila možnost oddaje telovadnice rekreativcem, športnim klubom za treninge in raznim svečanim slovesnostim. S tem pa je prišla potreba po večji organiziranosti, sledenju ter nadzoru zasedenosti dvorane. Trenutni postopki rezervacije potekajo tako, da je uporabnik primoran obiskati OŠ Stopiče najmanj 3x (preverjanje termina, rezervacije termina, podpis pogodbe). Pristojni so za sledenje uporabili Excel, ki pa je sčasoma postal precej nasičen ter nepregleden. Poleg tega, pristojni vseh potrebnih podatkov niso imeli vedno na voljo. Zato so določene rezervacije/spremembe/odpovedi preprosto zapisali na list papirja. Že sami lahko vidimo, da je sčasoma nadzor zasedenosti ter obveščanje postal prava nočna mora. Pojavili so se izzivi obveščanja uporabnikov, pri odpovedih zaradi pomembnejših prireditev. Ob spremembah je bilo potrebno najti vse uporabnike ter jih obvestiti po telefonu. Uporabniki dostikrat niso bili dosegljivi, kar je botrovalo k iskanju kontaktnih podatkov (iz Excela ali pa iz naključnega lista). Že sami lahko opazimo, da je skrbnik za določene korake porabil veliko dragocenega časa.

Z razvojem aplikacije smo poskušali razbremeniti, poenostaviti in avtomatizirati potek rezervacije športne dvorane ter poenostaviti sam pregled nad zasedenostjo, tako za uporabnike, ki želijo samo rezervirati telovadnico za športno aktivnost, kot tudi za ključne osebe, ki bodo sledile, usklajevale in obračunavale sam najem prostorov.

Implementiran projekt je dosegljiv na naslovu rezervacije.murn.eu.

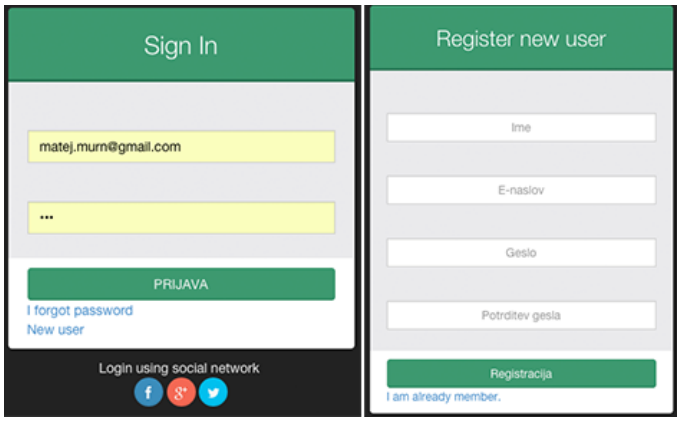
3.2 Uporaba aplikacije glede na uporabniške vloge

Sama aplikacija je predvidevala tri ključne vloge pri uporabi sistema rezervacij. Prva vloga je bila administrator, ki ima omogočene vse funkcionalnosti in lahko ureja, briše in pregleduje vse podatke. Vloga navadnega uporabnika služi za vnos rezervacij in pregledovanje uporabnikovih rezerviranih terminov ter njegovih osebnih podatkov. Nenazadnje imamo tu tudi vlogo tajnice, kateri so dodeljene pravice pregledovanja novih rezervacij ter kreiranje računov. Zaradi boljše preglednosti nad celotnim sistemom bomo v nadaljevanju opisali tipične primere uporabe po uporabniških vlogah, ki hkrati zajemajo tudi vsebino posameznih uporabniških zgodb, ki smo jih implementirali.

3.2.1 Uporabnik

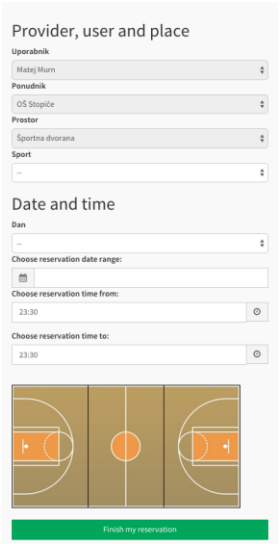
Kot neregistrirani uporabnik lahko pregleduje prostore, ki so na voljo v OŠ Stopiče. Pri opisu prostorov dobi vse potrebne podatke (Slika 12), da lahko rezervacijo opravi na star način (s

telefonskimi klici, dogovarjanjem in osebnim obiskom), v nasprotnem primeru pa se mora registrirati v aplikacijo, da lahko opravi rezervacijo preko spleta. Pri postopku registracije lahko uporabnik uporabi prijavo z enim od socialnih omrežij ali pa izpolni kratek obrazec ter ustvari uporabniški račun (Slika 9).



Slika 9: Obrazec za prijavo in registracijo

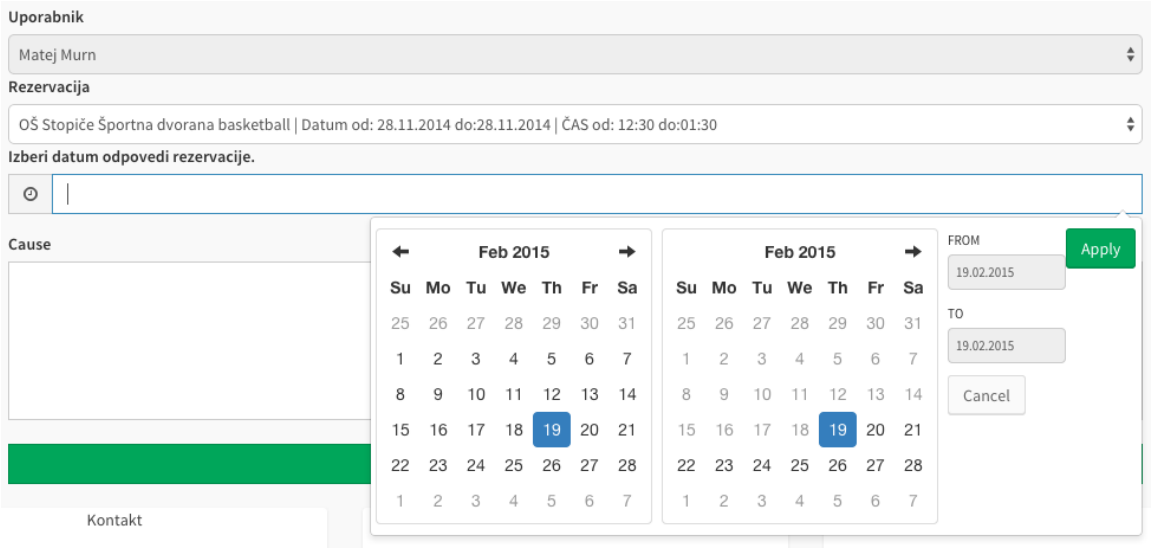
Po uspešni registraciji se uporabniku pošlje potrditvena povezava, preko katere aktivira svoj račun. S tem smo mu dodelili osnovne pravice uporabnika, ki so opisane v nadaljevanju. Sedaj lahko prostor rezervira s klikom na gumb rezerviraj (Slika 12), pri podrobnem opisu prostora. Na tem mestu se mu prikaže rezervacijski obrazec, ki ga mora pravilno izpolniti (Slika 10).



Slika 10: Obrazec za vnos podatkov pri rezervaciji prostora

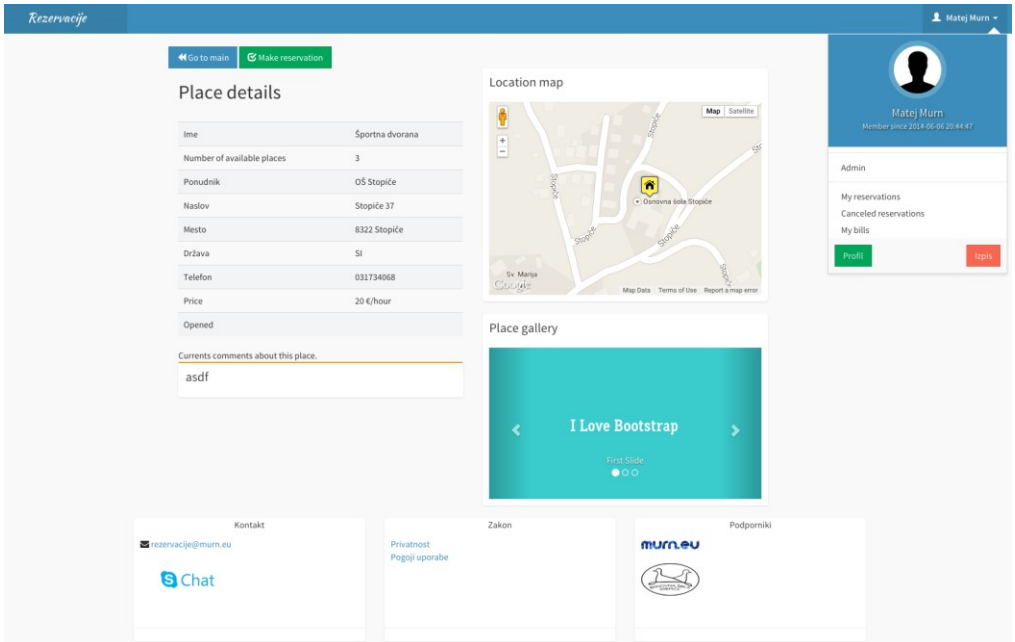
V kolikor je med izpolnjevanjem prišlo do napak, bo sistem uporabnika na to opozoril. Sistem pri vsaki oddaji preverja zasedenost kapacitete rezerviranega prostora in v kolikor se izkaže, da želi uporabnik rezervirati prostor na termin, ki je že zaseden je na to opozorjen in mora spremeniti datum ali uro rezervacije. Lastnik projekta je presodil, da prikaz razpoložljivih terminov trenutno še ni ključnega pomena, zato smo implementirali zgolj preverjanje prostega termina. Ob uspešni rezervaciji uporabnika obvestimo o njegovi aktivnosti. Rezervacije je vnesena v sistem, administrator pa jo mora potrditi, da postane

veljavna in uporabnik lahko prične z njeno uporabo. Velikokrat se zgodi, da moramo rezervacijo odpovedati. V ta namen smo naredili obrazec odpovedi (Slika 11), ki nam omogoča vnos odpovedi posameznega rezerviranega termina v roku 7 dni pred željeno odpovedjo. Vsi registrirani uporabniki lahko kadarkoli pogledajo in uredijo svoj uporabniški profil, kjer vnesejo svoje kontaktne podatke.



Slika 11: Odpoved določene rezervacije

Uporabnik lahko iz svojega menija (Slika 12) izbere možnost pregleda aktualnih računov, pregled in urejanje svojega profila ter nenazadnje pregled rezervacij in vnešenih odpovedi rezerviranih prostorov. V kolikor ima uporabnik pravice administratorja ali tajnice, se v meniju pojavi možnost prehoda v administratorski pogled o katerem bomo več povedali v nadaljevanju.



Slika 12: Vmesnik navadnega uporabnika pri pregledu prostora pred rezervacijo

3.2.2 Tajnica

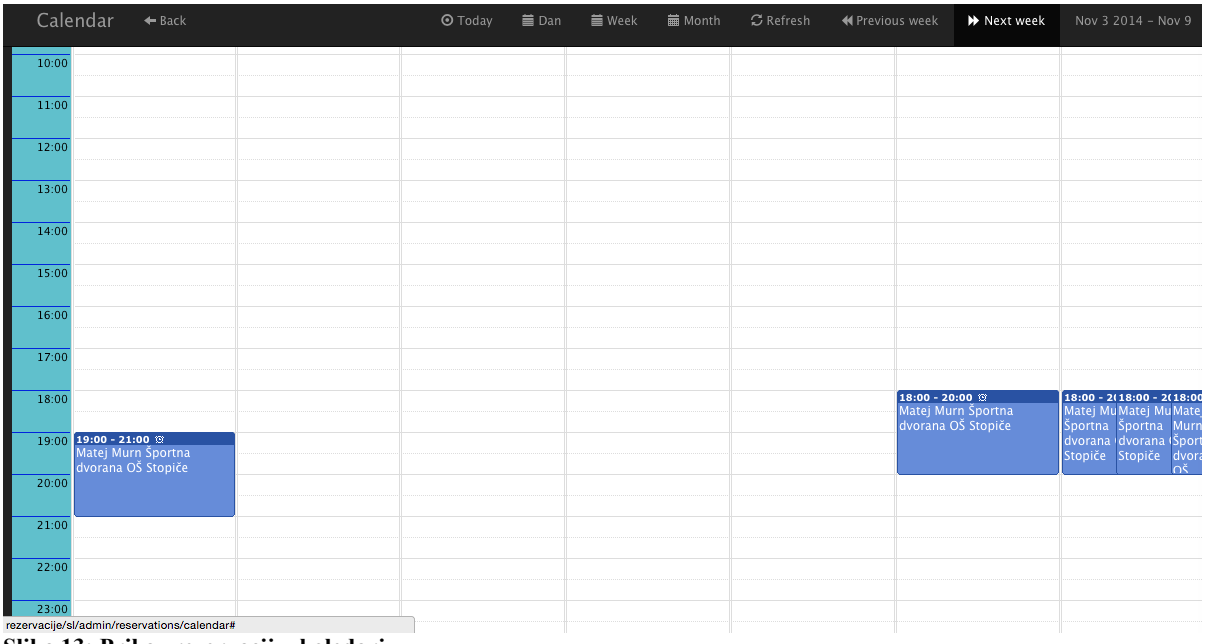
Pravice tajnice smo razširili na podlagi uporabnika. Njena glavna naloga je, da usklajuje računovodske zadeve in poskrbi za pravilno izdajanje najemnih pogodb, generiranje računov in aneksov. Dodatno omogočene pravice so:

- možnost uporabe administratorskega vmesnika,
- pregledovanje zapisov,
- generiranje računov in potrjevanje podpisa rezervacij.

Slednje pravice zadoščajo za dostop do primerne količine podatkov, da lahko tajništvo uskladi podpis najemne pogodbe, za kar se je uporabnik primoran zglasiti na mestu ponudnika rezerviranega prostora. Ob vsaki izmed opravljenih akcij se uporabnika obvesti. Še vedno pa ima tudi tajnica pravice navadnega uporabnika tako, da lahko rezervira prostore, ki so na voljo ali pa svoje trenuno aktivne rezervacije odpove.

3.2.3 Administrator

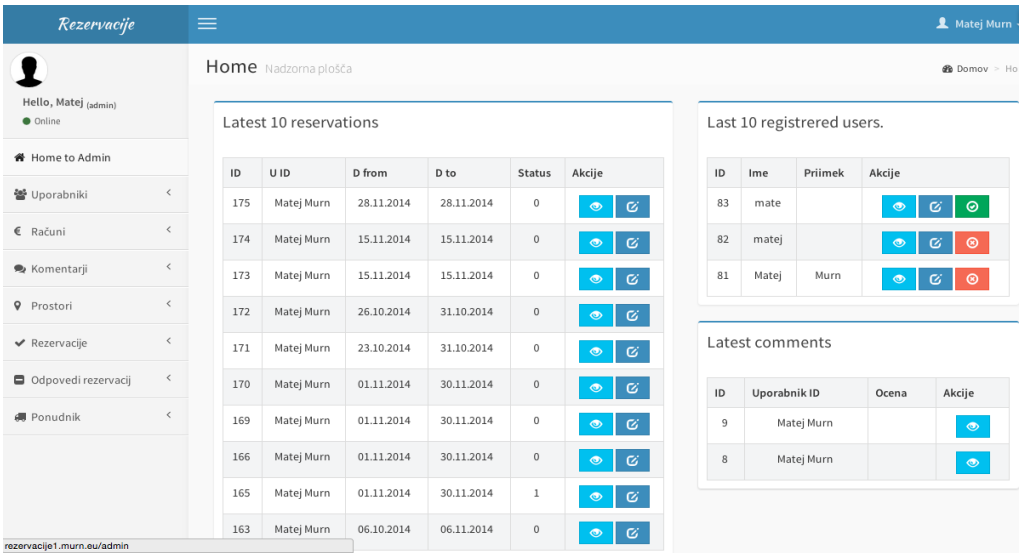
Administratorju so omogočene vse že zgoraj omenjene funkcionalnosti uporabnika in tajnice, vendar ker je njegova vloga ključna ima v skladu s pomembnostjo tudi dodatne funkcionalnosti, ki jih lahko opravlja. Administrator lahko posamezen zapis; ki je bodisi tipa rezervacija, uporabnik, prostor; izbriše iz sistema, v kolikor presodi, da je zapis nepravilen. Vsakega izmed zapisov ima možnost popraviti, lahko vnaša nove zapise, pregleduje ter aktivira posamezne rezervacije, uporabnike, komentarje in prostore. Na voljo ima tudi grafični pregled rezervacij v obliki koledarja (Slika 13) in vnos pomembnih rezervacij.



Slika 13: Prikaz rezervacij v koledarju

Pojem pomembnejše rezervacije je bistvenega pomena, saj je dvorana na voljo vsem, ki jo želijo najeti. Na tem mestu moramo v primeru celodnevnega dogodka poskrbeti za odpovedi vseh rezervacij v trajanju tega dogodka. O vstavljeni odpovedi so uporabniki obveščeni

preko njihovih e-mail naslovov. Slika 14 prikazuje administratorski vmesnik, ki je na voljo administratorju in tajnici, v okrnjeni obliki. Ko uporabnik odda rezervacijo prostora je slednji administratorju viden v domačem zaslonu in pod menijem Rezervacije/Vse rezervacije v njegovem administratorskem vmesniku. Poleg vsake izmed rezervacij ima poleg že omenjenih možnosti urejanja tudi možnost podpisa, aktivacije ter pintanja pogodbe in aneksa. Z akcijo *aktivacije* lahko administrator uporabniku aktivira/deaktivira rezervacijo zaradi pomankljivo oddanih podatkov ali višje sile. Gumb podpis rezervacije služi za virtualni podpis, ko stranka vrne podpisano pogodbo. Od tega trenutka naprej se rezervacija označi za aktivno, uporabnik lahko prične z uporabo prostorov, tajnica pa lahko izda račun za termine, katere je uporabik uporabljal najeti prostor.



Slika 14: Administratorski uporabniški vmesnik

3.3 Uporabljene tehnologije

V sklopu želja in pričakovanja glede funkcionalnosti aplikacije je obvezen pogovor tudi o sistemskem zaledju, ki bo temelj same aplikacije. Predstavljenih je bilo nekaj možnosti, izmed katerih se je zaradi denarnih sredstev za vzdrževanje in predvsem čim manjšega vložka v projekt izbralo naslednje tehnologije:

- YouTrack

YouTrack je eden izmed sistemov za nadzor in pregled nad stanjem in izdelavo aplikacij ali pa zgolj sistem za boljše organiziranje. V sklopu diplomske naloge sem ga uporabil za trenutni časovni pregled nad razvojem aplikacije, časovni nadzor ter za vpogled v začrtane, pričakovane cilje.

- Git (Bitbucket) + Sourcetree

Git je odprtokoden sistem za nadzor sprememb v kodi. Večinoma se uporablja predvsem v večjih ekipah, kjer je sodelujočih več, kar močno poenostavi hkraten razvoj posameznih uporabniških zahtev ter združevanje kode posameznega razvijalca. Sam sem ga uporabil za nadzor lastnih sprememb, hranjenje zgodovine itd., kajti zavedam se, da v osnovni verziji sistem ne bo dokončno narejen in bo potrebno dodelati še kakšen popravek ali posodobitev, pri katerem lahko pride tudi do morebitne vsebinske napake, zato je dobro beležiti zgodovino sprememb [23].

Bitbucket je oblačna storitev, ki omogoča neomejeno hrambo git repozitorjev. Iz slednje lahko vsak razvijalec posodablja svojo lokalno verzijo kode in shranjuje svojo dokončano kodo, ki postane dostopna ostalim razvijalcem [24].
Sourcetree je prosto dostopni program, ki nam omogoči grafično uporabo git sistema in preglednejši nadzor nad narejenimi funkcionalnostmi. S slednjim se izognemo uporabi terminala/konzole, vendar je za reševanje večjih izzivov pri združevanju še vedno priporočljivo uporabiti konzolne ukaze.

- Php

Eden izmed najbolj popularnih objektno orientiranih skriptnih jezikov na strani strežnika, s katerim poskrbimo za odgovore ter procesiranje uporabnikovih zahtev [25].
K odločitvi zanj je botrovala predvsem osnova odprte kode ter prosto dostopnih vsebin ter posledično manjših stroškov, ki bi bili namenjeni za razvojna orodja ter strežnike.

- MySql

MySql je eden izmed najbolj široko uporabljenih in popularnih sistemov za nadzor in upravljanje podatkovnih baz. Odlikuje ga velika fleksibilnost, skalabilnost in število dodatnih funkcionalnosti, ki jih lahko uvedemo s pomočjo dodatne kode ali razširitev [26].

- HTML

HTML (Hypertext Markup Language) je programski jezik, ki pove, kako naj se predstavitvena stran prikaže. Slednjo dosežemo z uporabo različnih značk, ki jih uokvirimo v <> oklepaje [31].

- jQuery

Je lahka javascript knjižnica, ki se izvaja na strani uporabnika in nam omogoča, da napišemo manj javascript kode [30]. S slednjo pridemo do večjega učinka in hitrejšega delovanja strani. Njeno namembnost lahko vidimo predvsem v naslednjih primerih:

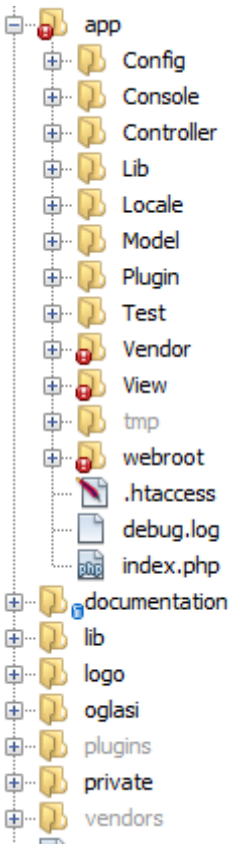
  - izbiri različnih DOM (Document Object Model) elementov,
  - ustvarjanju animacij,
  - prestreganju dogodkov,
  - delo z Ajax zahtevki in še bi lahko našteval.

- Bootstrap

Bootstrap je knjižnica, ki vsebuje elemente, oblike in stile, s katerimi poteka osnovna interakcija z uporabnikom in preko katerih uporabnik dobi vizualno podobo aplikacije. Elementi so že zasnovani na principu odzivnega načrtovanja (angl. responsive design*)*, kar pomeni, da se prilagajajo glede na velikost zaslona, preko katerega uporabnik uporablja aplikacijo (mobilni telefon, tablica, računalnik) [27].

- CakePHP framework

Je eno izmed mnogih php ogrodij (angl. framework), ki razvijalcu poenostavijo ter strukturirajo razvoj aplikacije po principu MVC (Model View Controller). Sama datotečna struktura ogrodja je naslednja:



Slika 15: Struktura projekta v CakePHP ogrodju

Za uporabnika so najpomembnejše datoteke naslednje:

- config: vsebuje konfiguracijske datoteke za bazo, mail, in prevezovanje,
- controller: vsebuje vse kontrolerje, iz katerih prikličemo določen pogled ali obdelavo prejetih podatkov,
- model: ključni del aplikacije, kjer je strukturirana baza ter polja posamezne tabele. V njem poteka tudi obdelava podatkov ter interakcija z bazo,
- view: vsebuje raznorazne poglede in prikaze podatkov, ki so združeni po imenih posameznega modela,
- plugin: vsebuje razne dodatke, ki jih želi uporabnik uporabiti in zunanje php knjižnice (primer: Opauth za prijavo s socialnimi omrežji),
- webroot: vsebuje vse css, slike in javascript datoteke, ki poskrbijo za boljšo uporabniško izkušnjo aplikacije.

Ostalih datotek ogrodja po priporočilih ni primerno spreminjati, saj obstaja velika verjetnost, da nekdo, ki ga uporablja zgolj za razvoj aplikacij in poseduje premalo znanja o samem jedru ogrodja nevede pokvari določene funkcionalnosti, ki pa so navsezadnje nujne za normalno delovanje ogrodja.

3.4 Specifikacija uporabniških zgodb

Pred začetkom projekta smo lastnika seznanili z metodologijo Scrum ter ključnimi pojmi, ki se v njej pojavljajo. V skladu z metodologijo smo izdelali specifikacijo zahtev, ki je zajemala

vse funkcionalnosti. Te smo oblikovali v obliki uporabniških zgodb. V prilogi A si lahko ogledate vsebino posamezne zgodbe, kjer so vključeni tudi prejemni testi in podrobnejši opis kot ga predvideva sama metodologija Scrum.

3.5 Načrtovanje izdaj

Pri samem razvoju programske opreme je potrebno pred samo implementacijo podrobneje pregledati vse funkcionalnosti ter doreči roke posameznih izdaj. Velikokrat se funkcionalnosti razvijajo na velikih že obstoječih projektih, kar pomeni, da so lahko zahteve za čimprejšnjo implementacijo zelo nujne. Da so vsi člani ekipe in uporabniki obstoječe/nove funkcionalnosti obveščeni o izdaji, se pred implementacijo zahteve pregleda in določi urnik izdaj, kar omogoča boljši pregled in načrtovanje dela ostalega razvojnega tima ter ostalih sodelujočih, ki čakajo na neko novo funkcionalnost.
Pri konkretnem projektu rezervacij nismo imeli množice uporabnikov, ki bi sistem uporabljali. Zato sem organiziral naključno skupino ljudi, ki je simulirala samo uporabo in delo na sistemu po zaključeni izdaji.

- **Urnik izdaj**

Sam sem izbral dolžino sprinta 14 dni. Za slednjo sem se odločil na podlagi uporabniških ocen ter mojega prostega časa. V razvojni skupini sem bil sam, med prostim časom sem delal preko študentskega servisa. Poleg obstoječih obveznosti mi je ostalo še nekaj časa, v katerem sem predvidel, da bom lahko implementiral funkcionalnosti, ki smo jih planirali.
Vsebino v vsaki izdaji, sem sem določil na podlagi smiselno grupiranih funkcionalnosti, ki po posamezni izdaji tvorijo neko smiselno celoto. Ta omogoča določene nove funkcionalnosti bodisi za uporabnika, bodisi za administratorja. V nadaljevanju bom na kratko povzel vsebino vsake izmed izdaj. Trenuten projekt je imel 6 izdaj:

- izdaja 1: 4.5.2014

Izdaja je obsegala osnovne funkcionalnosti kot so kreiranje baze, registracija, osnovna prijava, osnoven pogled, administratorski vmesnik, osnovna obveščanja.

- izdaja 2: 19.5.2014

V naslednji izdaji sem izboljšal uporabniški vmesnik, dodal sem osnovne elemente za uporabniško podporo, generiranje pdf-jev, omogočil sem nekatere funkcionalnosti, ki so pomembne predvsem za nadaljnji razvoj ter trženje samih poslovnih objektov, ki jih bo ponudnik ponudil preko sistema rezervacij.

- izdaja 3: 2.6.2014

Dodane so bile možnosti urejanja določenih sklopov, urejeno je bilo soodvisno nalaganje parametrov ter dodajanje uporabniškega mnenja.

- izdaja 4: 17.6.2014

Izdaja je obsegala primarni del rezervacij, izdelan je bil postopek same rezervacije ter njeno urejanje. Med drugim je bilo poskrbljeno tudi za nekaj obstranskih zadev kot so opozorilo za uporabo piškotkov, shranjevanje nekaterih operacij uporabnika (za hitrejše razhroščevanje (angl. Debugging)) ter prijaznejše opozarjanje uporabnika pri nedosegljivih straneh.

- izdaja 5: 30.6.2014

Izdaja je obsegala integracijo koledarja in prijavo v sistem z uporabo socialnih omrežij.

- izdaja 6: 16.7.2014

V zadnji izdaji je ostalo še nekaj zgodb, v katerih sem poskrbel za boljšo uporabniško izkušnjo med samim postopkom rezervacije.

3.6 Ocenjevanje uporabniških zgodb projekta

Uporabniške zgodbe smo ocenili po metodi *planning poker*. Ker sem bil v razvojni ekipi sam, sem za pomoč poprosil kolege, ki so z velikim entuziazmom priskočili na pomoč. Predstavili smo jim problematiko aplikacije in definirane uporabniške zgodbe. Samo ocenjevanje je potekalo tako, kot metoda predvideva. Tukaj sem opazil prvi izziv o premoščanju časovnih razlik pri ocenjevanju posamezne zgodbe, saj sem uspel izvesti samo dve iteracije metode *planning poker*. Metoda sicer predvideva poljubno število iteracij vse dokler, se člani razvojne ekipe sporazumno ne odločijo za natančno oceno posamezne zgodbe. Ker sem po nekem času pridobil samo delne rezultate 3. iteracije, sem se odločil, da izvedem ocenjevanje tudi po metodi *team estimation game*, ki bo izvedeno hitreje. Iz podatkov pridobljenih metod za ocenjevanje sem kljub nepopolno izvedeni metodi *planning poker* uspel izvleči nekaj karakteristik, o katerih bomo več spregovorili v časovni analizi poteka dela vsakega sprinta.

3.7 Opis poteka izvajanja iteracije

Vsaka izmed iteracij se je začela z zaključkom prejšnje. Slednjo smo začeli s sestankom Sprint planning meeting, v katerem sem pregledal seznam uporabniških zgodb in iz slednjih, na podlagi planirane hitrosti izbral nedokončane zgodbe ter jih umestil v naslednji sprint. Slednje sem ocenil ponovno, saj metodologija predvideva, da so ocene s pomočjo izkušenj, pridobljenih iz predhodnih sprintov, natančnejše.

Med samim sprintom nisem imel možnosti izvajati sestanka Daily Scrum meeting, saj sem bil v razvojni ekipi sam. Kot približek sestanku sem izvedel dnevni pregled implementacije, kjer sem odgovoril na tri ključna vprašanja, ki smo jih omenili v poglavju 2.2.4 ter ponovno ocenil potrebno število ur za dokončanje uporabniških zgodb, na katerih sem delal. Odstopanje izvajanja sestankov v skladu s samo metodologijo lahko opazimo v natančnosti ocen zgodb ter samem izvajanju posameznega sprinta. Kot bomo lahko videli v poglavju 0, smo med sprinti velikokrat povečali oceno predvidenih ur za dokončanje. Slednje bi lahko zmanjšali, v kolikor bi projekt izvajala skupina razvijalcev, katerih znanja, izkušnje in mnenje bi lahko izmenjevali.

Zadnji dan sprinta smo skupaj z lastnikom izvedli sestanek Sprint review meeting, kjer smo pregledali implementirane funkcionalnosti in v skladu z metodologijo označili ustrezno končane zgodbe s statusom *done*.

Sprint smo zaključili s sestankom Sprint retrospective meeting. Na slednjem sem kot edini razvijalec zbral primere dobrih praks celotnega sprinta, ki bi bili primerni za uporabo v naslednjih sprintih.

4 Analiza pridobljenih rezultatov po metodologiji Scrum

Za uporabo metodologije Scrum smo uporabili projekt rezervacij, na katerem smo dosledno spremljali porabljen čas za implementacijo posamezne funkcionalnosti ter izvajali korake v skladu s samo metodologijo. Posamezne uporabniške zahteve si bralec lahko ogleda v prilogi 1. V poglavju bomo predstavili, kje je prihajalo do novih izzivov in kje so bili določeni odmiki od same metodologije.

4.1 Analiza ocen predvidenega ter izmerjenega časa po sprintih

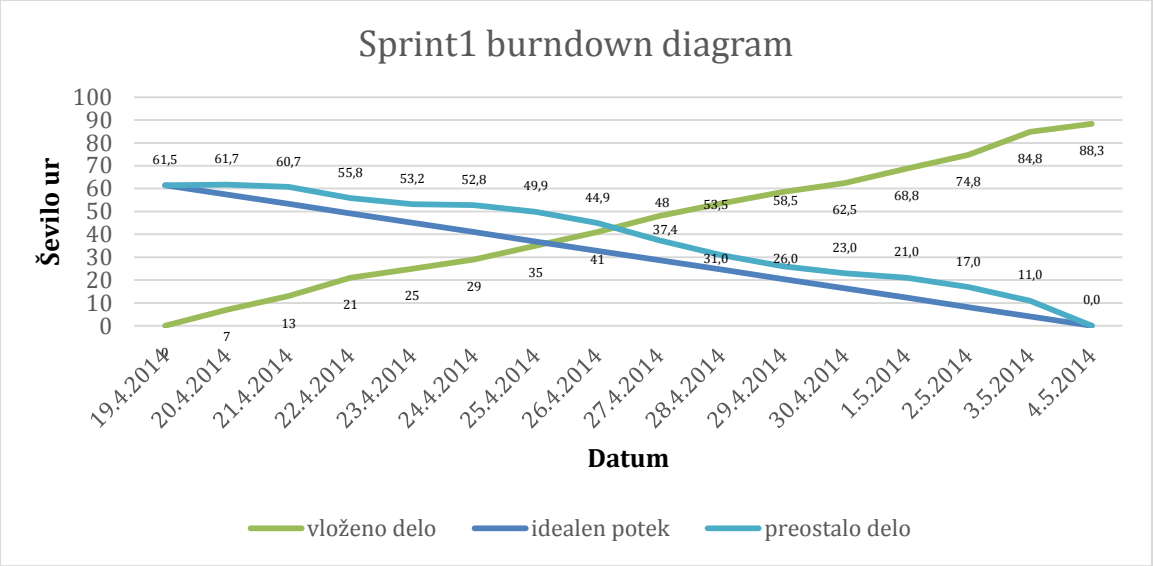
V naslednjih podpoglavjih bomo podrobneje opisali posamezen sprint in njegovo vsebino. Dotaknili se bomo analize ocenjenega ter predvidenega časa ter pogledali vpliv dejavnikov, ki so kljubovali k sami določitvi hitrosti sprinta. Vsak graf (Sprintx; kjer x predstavlja številko zaporednega sprinta) predstavlja pogoriščni, *burn down* diagram in predstavlja naslednje podatke na oseh:

- X os: dan – točen datum,
- Y os: št točk – ur dela.

Iz slednjega je moč razbrati količino opravljenega dela na dan, v posameznem sprintu, količino preostanka planiranega časa in presežek med planiranim in dejansko porabljenim časom. Grafi z naslovom *Ocenjen vs. Porabljen čas* prikazujejo odstopanja planiranega časa uporabniških zgodb med prvim in ponovnim ocenjevanjem. Časovne ocene so zapisane z decimalnimi mesti, saj smo bili primorani za oceno uporabniških zgodb vzeli povprečno oceno vseh razvijalcev. Pred vsakim začetkom sprinta smo ponovno ocenili uporabniške zgodbe, ki so bile uvrščene v posamezen sprint na podlagi dosedanjih izkušenj. Izjema je bil sprint 1.

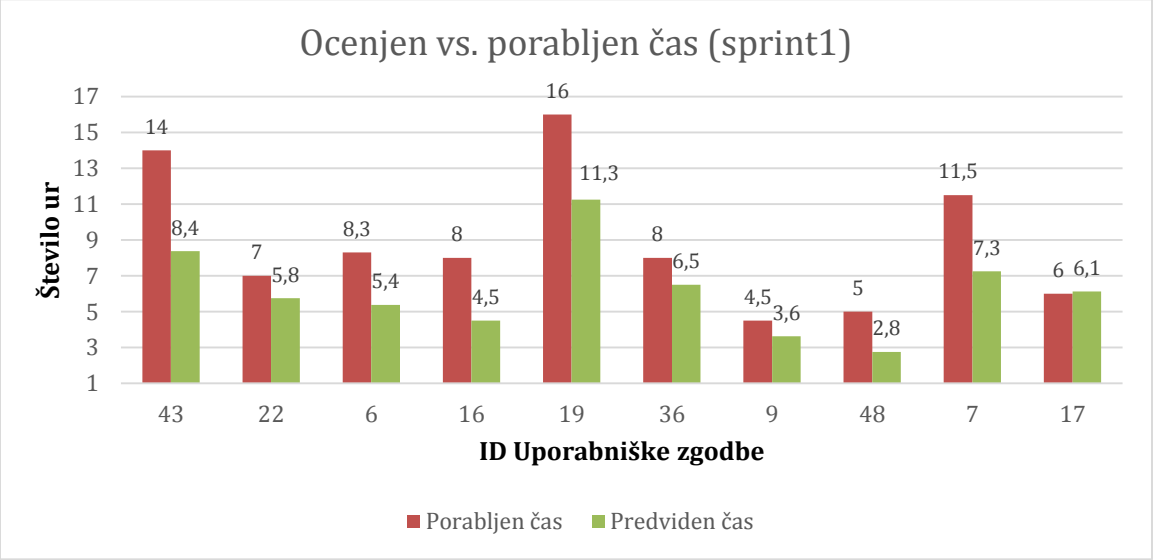
4.1.1 Sprint 1

V prvem sprintu sem privzel oceno uporabniških zgodb glede na pridobljene ocene preko metod *Poker planning* ter T*eam estimation game*. V prvi sprint sem uvrstil zgodbe, ki so v prilogi A označene z rumeno barvo. Ponovnega ocenjevanja slednjih nismo izvedli, saj pred tem nismo končali še nobenega sprinta, da bi lahko zgodbe primerjali in iz pridobljenih izkušenj ocenjevali nove uporabniške zgodbe. Iz diagrama Graf 1 je razvidno, da smo za realizacijo uporabniških zgodb porabili več časa, kot smo sprva predvideli. Najvišja prioriteta je bila določena prijavi (RP-36) in registraciji (RP-06) uporabnika v aplikacijo.



Graf 1: Analiza časa sprinta 1

Hitrost sprinta sem ocenil na 61,5 ur. Kot bomo lahko opazili v poznejših sprintih, je hitrost razmeroma velika, k čemur so botrovali tudi prazniki, saj sem s sprintom pričel 20. aprila ter ga končal 4. maja. Kot je razvidno iz grafa, sem za realizacijo vseh zgodb porabil 87,8 ur, čemur pripisujem vpliv manjšega poznavanja ogrodja ter večjemu številu ur, ki sem jih namenil za pravilno implementacijo, spoznavanje ogrodja in pisanju konsistentne kode, ki je ponekod zelo odvisna od načina uporabe pri sledenju MVC strukture.



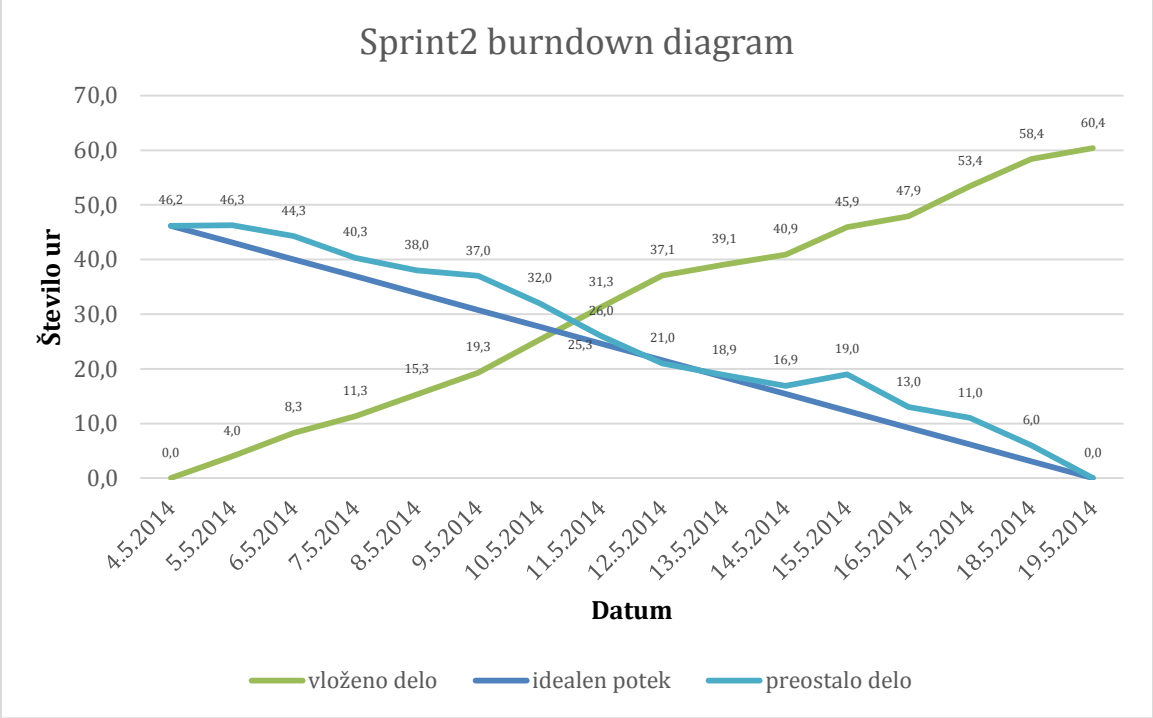
Graf 2: Primerjava med začetno oceno in dejansko porabljenim časom v Sprintu 1

Kot je razvidno iz grafa (Graf 2) so bile ocene zgodb preveč optimistične, saj naše poznavanje ogrodja še ni bilo na primernem nivoju, kar bomo opazili tudi v naslednjih sprintih. Ob posamezni implementaciji uporabniške zgodbe smo naleteli tudi na nekaj manjših nepredvidenih nalog, pri katerih je bilo potrebno poskrbeti za pravilno delovanje

funkcionalnosti, a vendar naloge niso bile take veličine, da bi jih lahko oblikoval v obliki nove uporabniške zgodbe, ki bi se implementirala v enem izmed naslednjih sprintov.

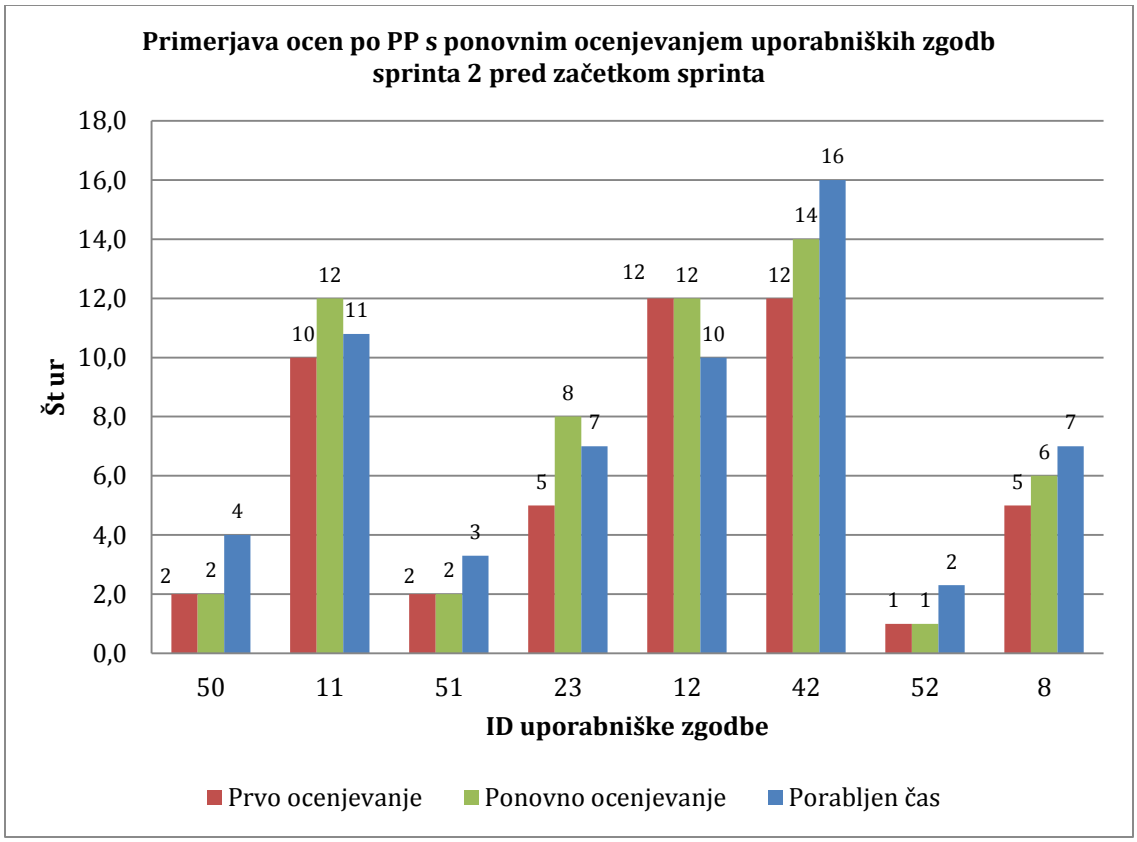
4.1.2 Sprint 2

Vsebina drugega sprinta je obsegala izdelavo uporabniškega vmesnika (RP-12) za uporabniško interakcijo in nekaterih enostavnejših uporabniških zgodb, ki so ključnega pomena za sistem. V prilogi A so zgodbe označene z oranžno barvo. Ker med sprintom ni bilo dela prostih dni (med katerimi bi lahko več časa posvetil implementaciji), sem hitrost sprinta znižal na 46,1 ur. Za dosego cilja in popolne realizacije sem porabil 60,4 ur, kar lahko razberemo tudi iz diagrama Graf 3. Prioriteta je bila postavljena na izboljšavi uporabniškega izgleda (RP-42) ter izdelavi pdf poročil (RP-23).



Graf 3: Analiza časa sprinta 2

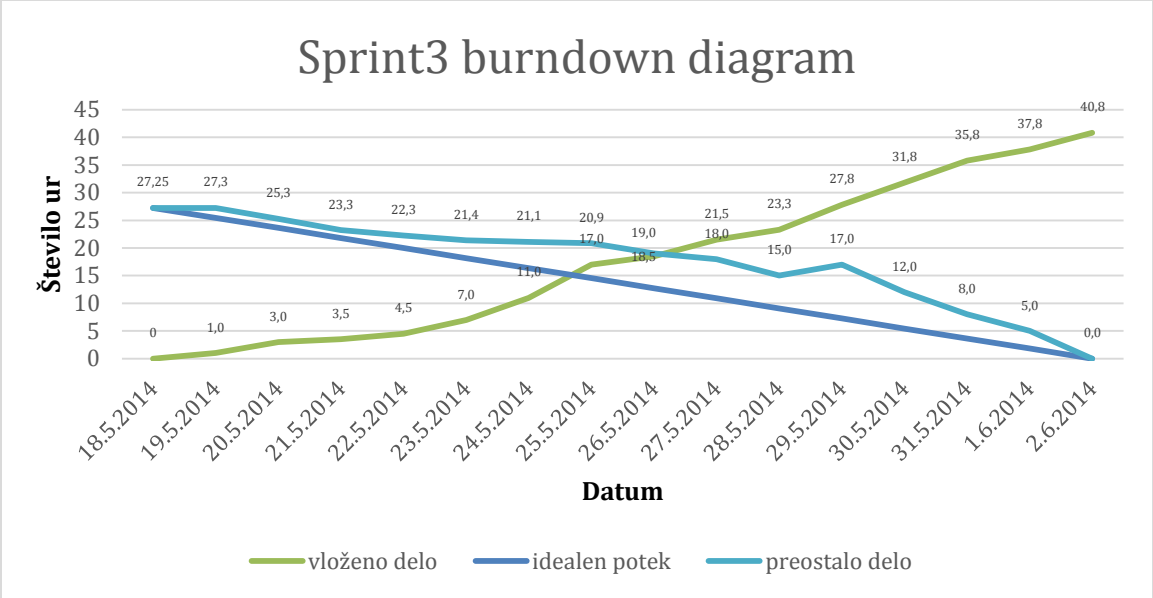
S pridobljenimi izkušnjami iz prvega sprinta sem ponovno ocenil uporabniške zgodbe, ki sem jih implementiral v sprintu 2. Iz diagrama Graf 4 lahko vidimo primerjavo med ocenami, pridobljenimi po začetnem ocenjevanju, ocenjevanju po končanem prvem sprintu ter dejansko porabljenim časom. Metodologija Scrum predvideva, da so ocene pri ponovnem ocenjevanju točnejše, kar lahko potrdimo tudi v našem primeru.



Graf 4: Primerjava med začetno oceno, ponovno oceno pred sprintom 2 in dejansko porabljenim časom

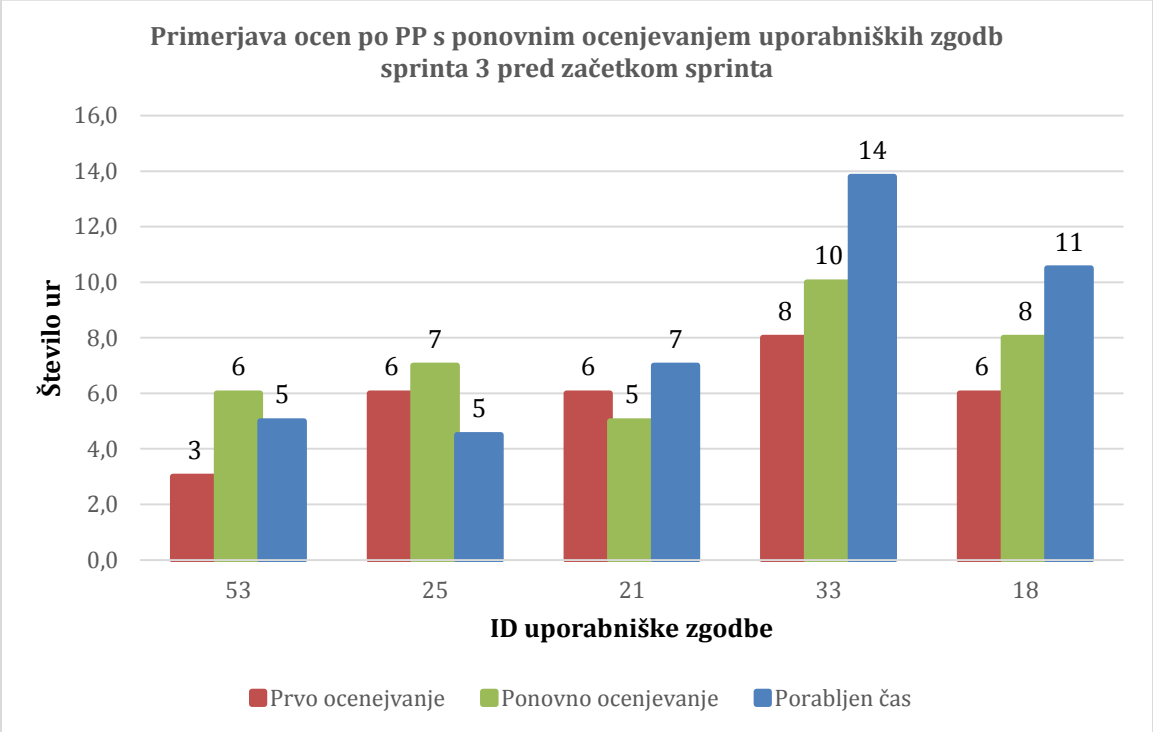
4.1.3 Sprint 3

Pri ponovnem ocenjevanju zgodb, ki smo jih vključili v sprint 3 smo znižali hitrost, saj smo v predhodnih sprintih kljub natančnejši oceni še vedno naleteli na izzive, ki so zahtevali več časa za pravilno realizacijo. Hitrost sprinta smo zato določili na 27,3 ur (Graf 5). Prioritetni sta bili dve zgodbi: priprava računov (RP-25) ter priprava najemne pogodbe (RP-33).



Graf 5: Analiza časa sprinta 3

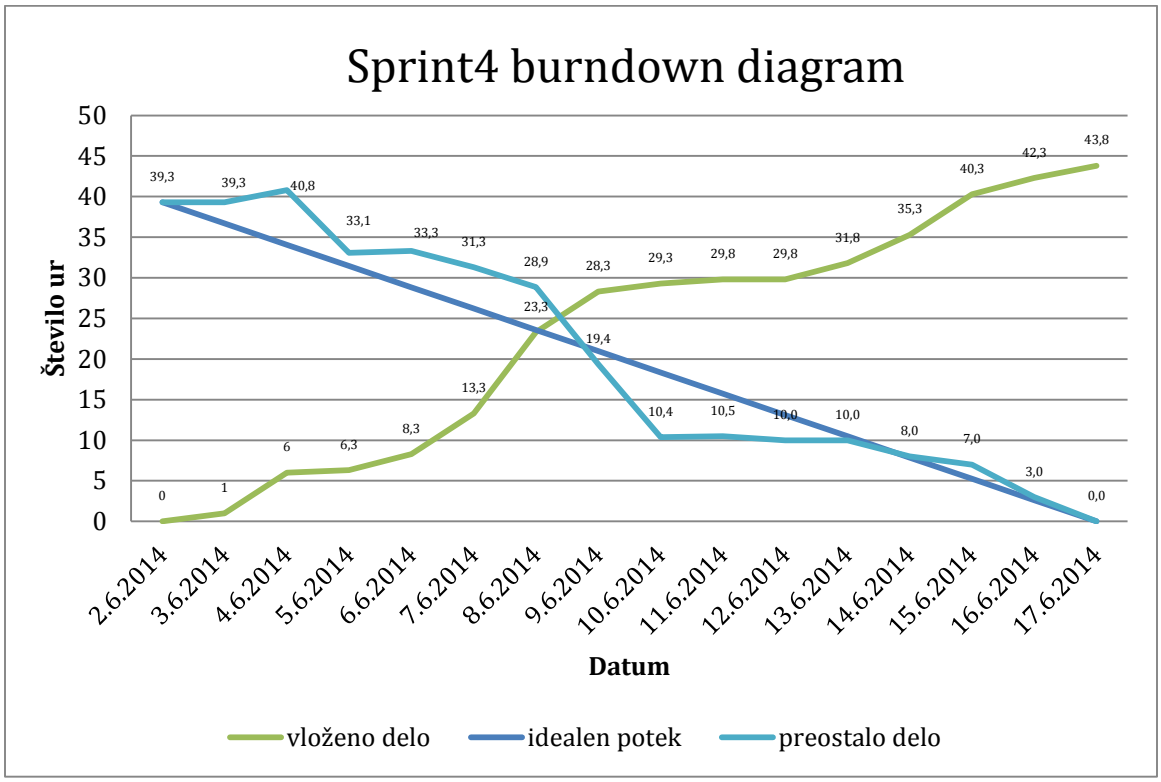
Pred začetkom sprinta smo ocenili zahtevnost uporabniških zgodb v sprintu. Natančnost ocene smo izboljšali predvsem pri obsežnejših uporabniških zgodbah, kar je razvidno iz diagrama Graf 6, medtem ko smo pri enostavnejših precenili količino dela, ki ga je bilo potrebno opraviti. Slednje pripisujemo (še vedno) slabšemu poznavanju ogrodja, ki smo ga uporabili.



Graf 6: Primerjava med začetno oceno, ponovno oceno pred sprintom 3 in dejansko porabljenim časom

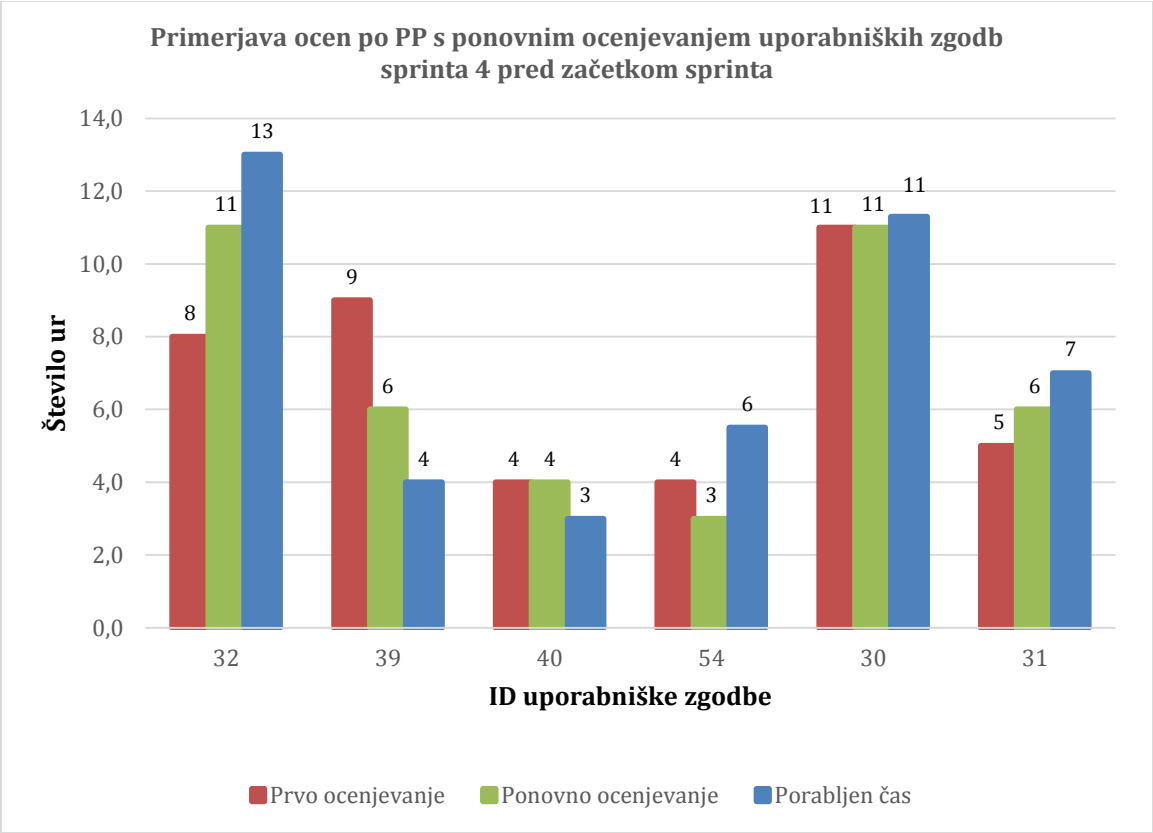
4.1.4 Sprint 4

V sprintu 4 so bile uvrščene uporabniške zgodbe, ki so v prilogi A označene z modro barvo. Hitrost sprinta smo ocenili na 39,3 ur, saj smo umestili v sprint obsežnejše uporabniške zgodbe. Hitrost sprinta je višja od predhodne tudi na podlagi planiranega dopusta v okviru delovnega mesta, zaradi česar sem menil, da bom lahko implementaciji namenil več časa (Graf 7). Najvišjo prioriteto je imela zgodba rezervacij terminov (RP-30), saj smo v tej fazi že imeli implementirane temeljne funkcionalnosti, na katerih smo lahko pričeli z implementacijo konkretne vsebine. Iz grafa 7 lahko opazimo, da smo v dnevih med 7.6. ter 11.6. implementirali veliko količino funkcionalnosti. Slednje izhaja iz večje količine dela, ki sem ga opravil med vikendom.



Graf 7: Analiza časa sprinta 4

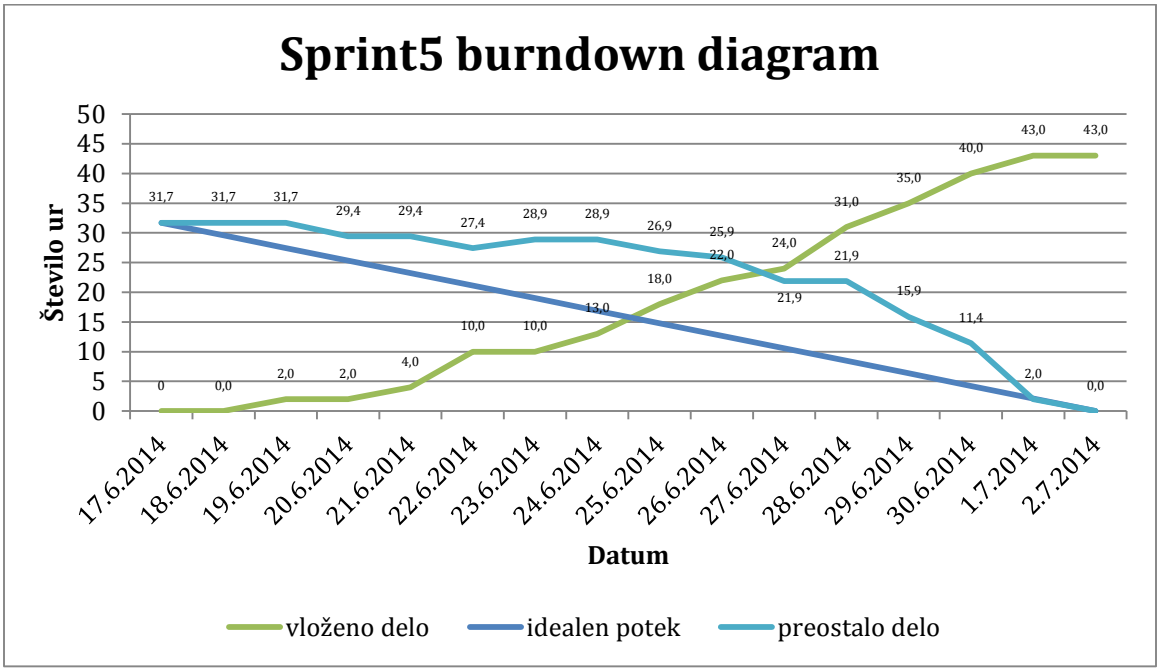
Sprint je vseboval zelo raznolike uporabniške zahteve, zato so bile zgodbe in njihovo ocenjevanje odlični pokazatelj uspešnosti ponovnega ocenjevanja (Graf 8). Glede na njihovo zahtevnost in glede na izkušnje prejšnjih implementiranih funkcionalnosti, sem oceno nalog izboljšal, kar samo potrjuje principe, ki jih metodologija Scrum predvideva.



Graf 8: Primerjava med začetno oceno, ponovno oceno pred sprintom 4 in dejansko porabljenim časom

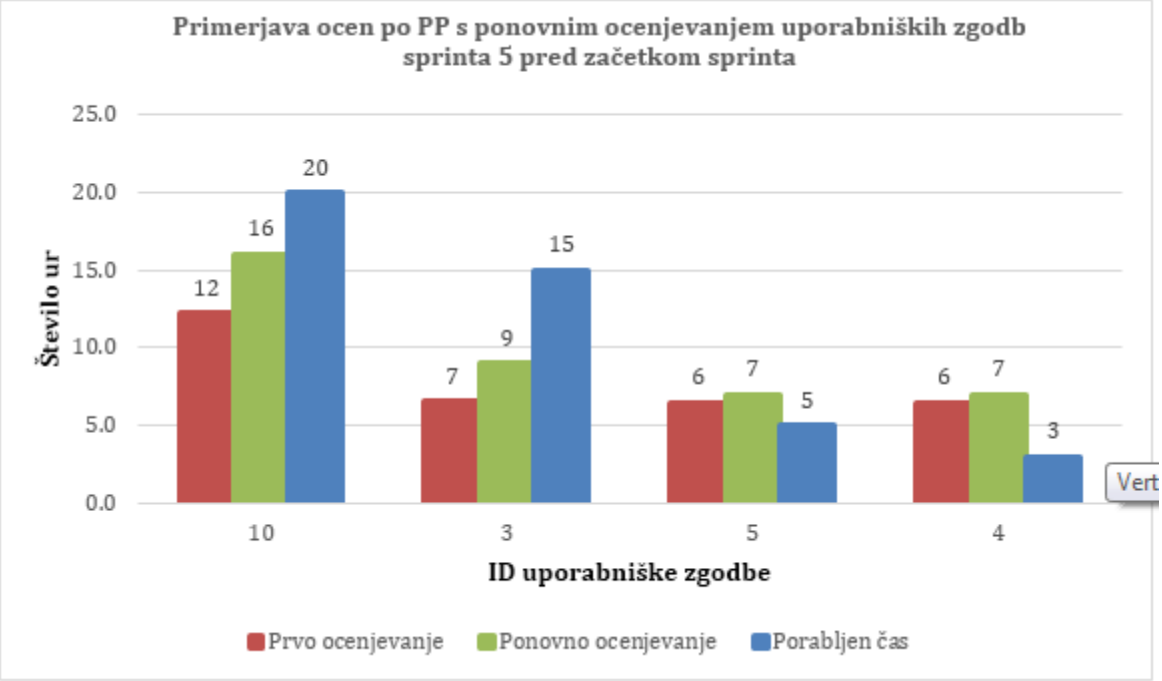
4.1.5 Sprint 5

Sprint je obsegal zgodbe, ki so v prilogi A označene z vijolično barvo. Med njimi je imela najvišjo prioriteto uporabniška zgodba integracije koledarja (RP-10). Hitrost sprinta smo določili na 31,7 ur, kar je manj kot v predhodnem sprintu (Graf 9). Na višino ocene je vplival konec dopusta in vključitev novih projektov, za katere sem bil zadolžen na delovnem mestu, zaradi česar sem lahko manj časa porabil za implementacijo aplikacije rezervacij.



Graf 9: Analiza časa sprinta 5

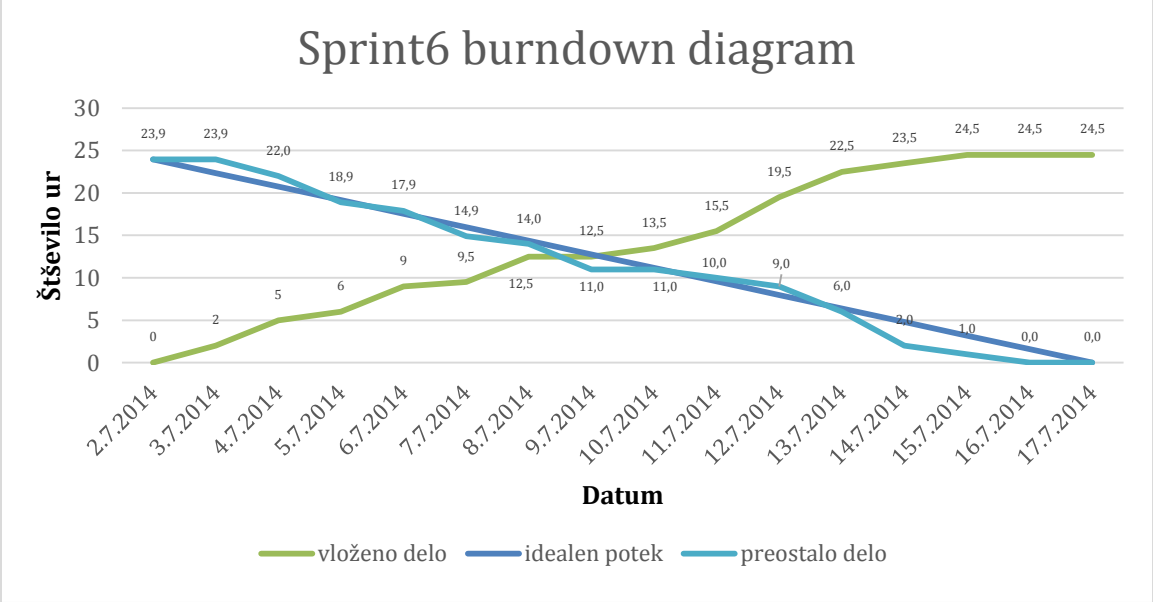
Pri ponovnem ocenjevanju sem nekaj izmed zahtev ocenil natančneje (Graf 10). Opazimo lahko, da se pri nekaterih uporabniških zgodbah ocena ni izboljšala ali pa se je celo poslabšala. To pripisujemo načinu ocenjevanja, saj sem kot edini razvijalec ponovno ocenjevanje izvedel sam. Uporaba knjižnice za koledar in implementacija prijave v aplikacijo preko sistema Facebook (RP-03) sta zahtevali več prilagoditev ter preoblikovanja obstoječe kode, kakor smo prva predvideli. Posledično lahko opazimo povečan obseg vloženega dela na diagramu (Graf 9), ki pa smo ga uspešno opravili, saj smo pridobljeno znanje pridoma uporabili pri podobnih uporabniških zgodbah prijave v sistem z uporabo Googla (RP-04) in Twitterja (RP-05).



Graf 10: Primerjava med začetno oceno, ponovno oceno pred sprintom 5 in dejansko porabljenim časom

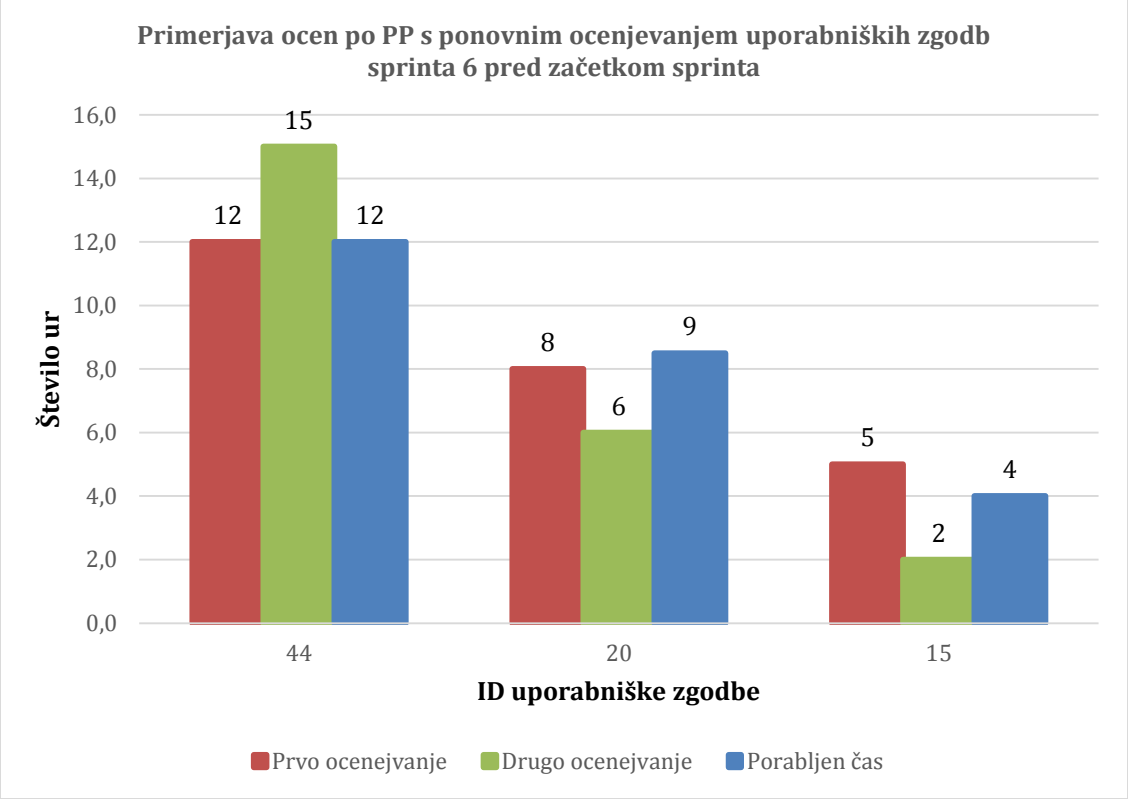
4.1.6 Sprint 6

V zadnji sprint so bile uvrščene zgodbe, ki so še ostale in so v prilogi A označene z rdečo barvo. Število ocenjenih ur dela znaša 23,9. Dejansko število ur potrebnih za realizacijo je ponovno preseglo predvidene. Graf 11 prikazuje oceno preostanka dela na zgodbah po dnevih. Na podlagi slednjega lahko sklepamo, da smo iz predhodnih sprintov dobili primerne izkušnje, katere so nam pomagale pri hitrejši implementaciji zgodb, kar lahko opazimo v natančnejšem ocenjevanju preostanka potrebnega časa za realizacijo uporabniških zgodb med samim sprintom.



Graf 11: Analiza časa sprinta 6

Pri ocenjevanju se tako kot v nekaterih izmed predhodnih sprintov pokaže vpliv pomanjkanja skupine razvijalcev, ki bi sodelovali tudi pri ocenjevanju, saj so začetne ocene, ki so bile določene s skupino razvijalcev natančnejše od moje ocene (Graf 12), ki je bila podana pred začetkom sprinta, z vsemi pridobljenimi izkušnjami. K nenatančnosti je doprinesla tudi vsebina uporabniških zgodb, ki so se od ostalih razlikovale po vsebinski težavnosti in dokaj specifični umestitvi programske kode v projekt. Vsi zgoraj omenjeni dejavniki samo potrjujejo primernost metodologije Scrum, saj slednja predvideva veliko pogovarjanja, izmenjevanja mnenj ter predlogov, kar posledično prepreči pomanjkljivosti, na katere so razvijalci pozabili (v kolikor bi v skupini bilo več razvijalcev).



Graf 12: Primerjava med začetno oceno, ponovno oceno pred sprintom 6 in dejansko porabljenim časom

4.2 Analiza predvidenega ter ocenjenega časa

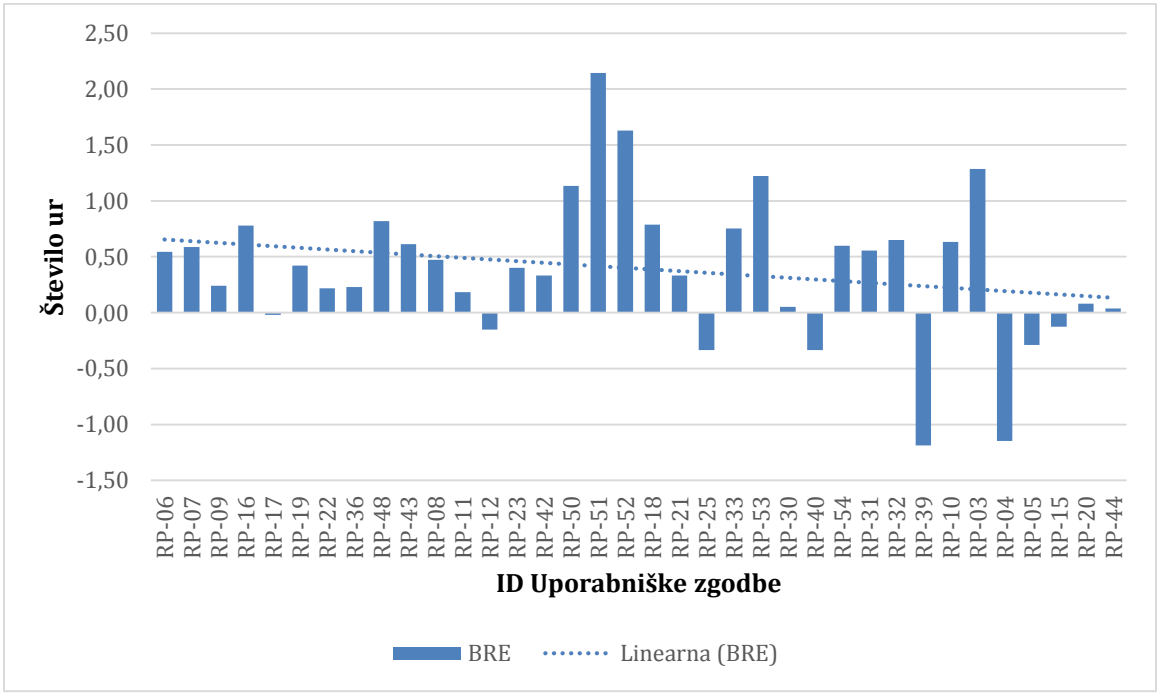
Graf 13 prikazuje primerjavo med porabljenim časom za realizacijo posamezne uporabniške zgodbe in uravnoteženo vrednostjo relativne napake (angl. Balanced measure of relative error - BRE) za vse uporabniške zgodbe v projektu. Slednje so podane v vrstnem redu njihove implementacije. Mero BRE smo izračunali po spodnji formuli, kjer oznaka A pomeni dejanska zahtevnost uporabniške zgodbe, oznaka E pa ocenjena zahtevnost uporabniške zgodbe.

$$BRE = \frac{A - E}{\min(A, E)}$$

Poleg mere BRE smo izračunali tudi povprečno vrednost uravnotežene relativne napake za vsak sprint (angl. Balanced mean of relative error) po naslednji formuli:

$$\overline{BRE} = \frac{1}{n}\sum_{i=1}^{n} BRE_i$$

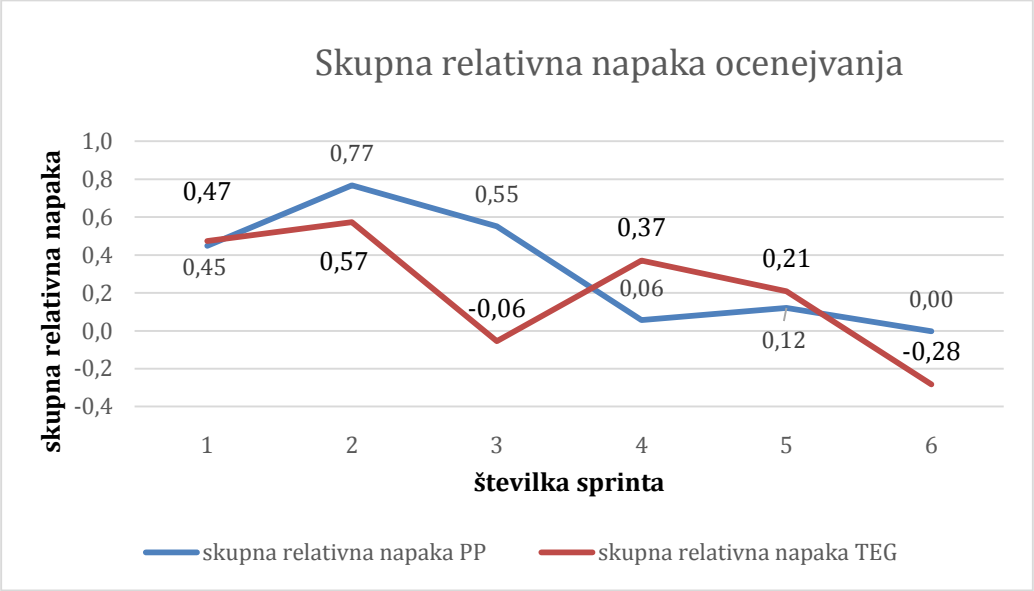
Iz diagrama (Graf 13) lahko vidimo, da smo kljub nekaterim ocenam, ki so bile slabše določene, na podlagi aktualnih izkušenj in prakse ocenjevanja svojo oceno izboljševali. Presežek dela smo uspešno nižali in se približevali dejanski oceni. Na žalost je bil projekt prekratek, da bi lahko naredili podrobnejši nadzor, vendar če sklepamo po trendu, lahko zatrdimo, da bi se čas realizacije uporabniških zgodb zelo približal njihovi časovni oceni. K natančnosti ocen je vplivalo tudi ocenjevanje samo enega člana razvojne ekipe in celotna zasnova projekta, ki predhodno ni temeljila ali izhajala iz že obstoječega sistema. Na tem mestu lahko predvidevamo, da bi skupina razvijalcev prej izboljšala točnost ocenjevanja, saj več razvijalcev hkrati deluje na različnih uporabniških nalogah in pridobiva različne izkušnje, katere sem sam spoznaval dalj časa.



Graf 13: Prikaz odstopanj ocen porabljenem času ter oceno BRE

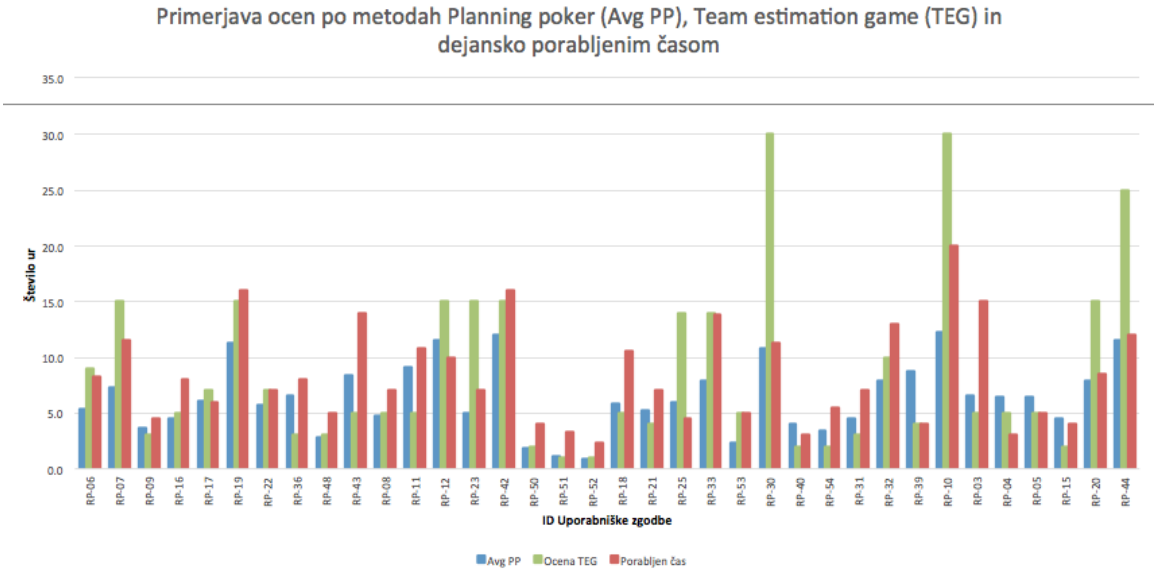
Iz diagrama Graf 15 lahko vidimo grafično primerjavo ocen po metodi *Planning poker* ter *Team estimation game*. Na tem mestu lahko potrdimo, da se vsem poznani rek *Več glav več ve* potrjuje tudi pri našem projektu. Uporabniške zgodbe, ki so bile implementirane v začetnih sprintih so imele v sprintih 2 in 3 natančnejšo oceno pridobljeno po metodi *Team estimation game* (Graf 14). Iz slednjega lahko razberemo, da izkušenost posameznega ocenjevalca ter razna znanja, ki smo jih posredovali med diskusijo ocenjevanja botrujejo k natančnejši oceni in spodbujajo ostale člane k aktivnejšemu razmišljanju. Predvidevamo lahko, da bi oceno močno izboljšali tudi pri ponovnem ocenjevanju uporabniških zgodb

vsakega sprinta, v kolikor bi delovali v skupini razvijalcev, kjer bi lahko vsi izrazili svoje mnenje in izpostavili dejavnike tveganja s svojega zornega kota.



Graf 14: Skupna relativna napaka ocen pridobljenih po metodah planning poker in team estimation game

Metodologija Scrum se je izkazala za odlično metodo za dotični projekt, saj nam je nudila celovit vpogled nad narejenim delom ter postopek za zbiranje podatkov na podlagi katerih sem lahko izvedel analizo po zaključku projekta. Hkrati smo postavili tudi dobre temelje za nadaljnji razvoj. V kolikor pa se bo projekt predal v roke drugega razvijalca, ima slednji vse potrebne podatke za natančnejše ocenjevanje novih zahtev.



Graf 15: Primerjave ocen po metodah *Planning poker* ter *Team estimation game*

4.3 Težave pri načrtovanju in implementaciji

Tako kot vsak razvijalec sem tudi sam naletel na nekaj dodatnih izzivov. Nekatere med njimi plitkejše ter druge globje. V nadaljevanju bom opisal najpogostejše, ki so se zgodili ter predlagal ukrepe, s katerimi bi jih lahko preprečil.

- Časovna ocena uporabniške zgodbe je bila prenizka,

Kot razvijalec, sem se o namembnosti ter za specifikacijo samih uporabniških zgodb pogovoril z lastnikom projekta. Pri ocenjevanju časovne zahtevnosti, ki sem jo opravil po metodi *planning poker* ter *team estimation game* je prišlo do manjših razhajanj, vendar je metoda ocenjevanja naravnana tako, da smo na koncu ocene uskladili. Izziv je bil, da se je med razvojem pojavilo nekaj dodatnih nenačrtovanih preprek ter potrebnih implementacij, ki sprva niso bile predvidene. Čeprav je bil posamezen sklop uporabniških zgodb pravočasno razvit, je med tem prišlo do več porabljenega časa za sam razvoj.

  - Vzroki:
    - nepoznavanje ogrodja, v katerem sem razvijal aplikacijo (tekom naslednjih sprintov se je pokazalo, da se časi vedno bolj ujemajo, ker se tudi ogrodje pozna bolje),
    - preveč široko določene funkcionalnosti (uporabniške zgodbe), posledično slabše načrtovanje potrebnih implementacij znotraj zgodbe.
  - Predlogi za reševanje problema:
    - izbira ogrodja, s katerim imajo razvijalci izkušnje,
    - po specifikacijah uporabniških zgodb z lastnikom projekta, predstaviti slednje drugemu razvijalcu, ki ne bo deloval na tem projektu ter predebatirati posamezne funkcionalnosti (saj ima drugačen pogled na razvoj ter implementacijo gleda iz drugačne perspektive).

- potreba po dodatni dodelavi funkcionalnosti med implementacijo,

Pri samem razvoju se vsak razvijalec poglobi v del, ki ga implementira, kar posledično lahko pripelje do spoznanja o novih funkcionalnostih ter boljši implementaciji za nadaljnji razvoj, kot je sprva bila načrtovana oziroma ravno narejena.

  - Vzroki:
    - bolj poglobljeno razmišljanje o implementaciji,
    - ob implementiranih zgodbah se nam odpre drugačen pogled na samo funkcionalnost.
  - Predlogi za reševanje problema:
    - pred implementacijo pregledati že narejene podobne projekte, kontaktirati njihove razvijalce in poskušati dobiti kakšen dober nasvet,
    - pregledati dobre prakse implementacije določene funkcionalnosti in ter jo poizkusiti vključiti v sam razvoj.

- manj je več (preveč funkcionalnosti za trenutno aplikacijo)

Med razvojem smo zasledili, da funkcionalnost večjezičnosti ni tako pomembna, kot je bilo sprva načrtovano, saj se bo aplikacija uporabljala lokalno, zato je dovolj, da omogoča slovenski jezik.

  - Vzroki:
    - želja po dobri zasnovi implementacije za nadaljnji razvoj,
    - omogočiti uporabniku izbiro željenega jezika.
  - Predlogi za reševanje problema:

- podrobneje pregledati področje uporabe same aplikacije ter njen razvojni cikel po koncu osnovne implementacije v smeri trženja, povpraševanja

Dotaknil pa bi se tudi problematike ocenjevanja. Pri projektu sem sam opravljal več vlog, ki sem jih omenil v metodologiji Scrum, saj projekta nisem imel možnosti izpeljati v realnih okoliščinah. Naj omenim, da bi po predvidevanjih ocene v realnem okolju prišle bolj točne. Tudi rezultati bi bolje podpirali metodologijo Scrum, saj lahko več različnih znanj, ki jih ocenjevalci imajo pri projektu natančneje oceni določene specifične zahteve in izpostavi dele, na katere sam morebiti nisem bil tako pozoren, vendar sem na njih naletel tekom implementacije.

4.4 Možnosti za izboljšanje

Trenutna aplikacije je namenjena za osnovno uporabo rezervacij, pregled ter izpis računov za trenutno aktivne rezervacije. V bodoče je možnost, da se pri trenutnem razvoju socialnih omrežij doda še naslednje funkcionalnosti:

- možnosti prijave (z več aktualnimi socialnimi omrežji),
- možnost večjezičnosti, ki je bila zaradi lokalnosti ter specifičnosti uporabe izključena,
- elektronski podpis, ki se ga lahko implementira na več načinov (ali kot storitev preko pripadajoče tablice, ali pa s preprostim podpisom z miško),
- dodatne izboljšave lahko uvedemo s pomočjo avtomatiziranih testov, ki se v kasnejšem razvoju pokažejo kot zelo dobra praksa, saj na zelo enostaven način preverimo delovanje naše kode (primerna metoda dela se imenuje TDD Testno vodeno razvijanje),
- ker se zavedam, da je ažurnost ena izmed ključnih lastnosti novega sistema je ena izmed možnih posodobitev tudi obveščanje strank preko SMS-ov ter tesnejša integracija z družabnimi omrežji,
- nenazadnje bi lahko knjižnico jQuery.js zamenjali z Angular.js in s tem še dodatno razbremenili strežnik ter izboljšali uporabniški vmesnik, aplikacija pa bi bila bolje pripravljena na prihajajoče trende razvoja programske opreme.

Nedvomno se bo tekom uporabe aplikacije pokazalo še kar nekaj prostora za izboljšave. O implementaciji slednjih se bomo z OŠ Stopiče postopoma dogovorili za morebitno implementacijo po uporabljeni metodi Scrum.

Poleg prpedlaganih izboljšav bi lahko metodologijo Scrum združili z metodo XP tako, da bi uvedli naslednje prakse:

- programiranje v parih,
- testno voden razvoj (angl. Test driven development),
- postopen razvoj izgleda aplikacije,
- centraliziran sistem za preverjanje kode in funkcionalnosti,
- informativno delovno okolje,
- (bolje) definiran standard kode,

s katerimi bi pripomogli k uspešnejšemu, hitrejšemu ter bolje opravljenemu delu.

5 Zaključek

Tekom diplomskega dela smo uspešno razvili sistem rezervacij za Športno dvorano Stopiče, pri tem pa smo samo aplikacijo že zasnovali v smeri uporabe več tako imenovanih ponudnikov poslovnih prostorov. Bralec je bil seznanjen z vsebino projekta in razlogi, zaradi katerih smo se odločili, da projekt razvijemo. Aplikacija ima na tej stopnji razvite osnovne funkcionalnosti, katerim se lahko v bodoče doda nove ter s tem še izboljša uporabniško izkušnjo uporabe sistema ali pa dodatno poenostavi proces rezervacije.

Na projekt smo uspešno aplicirali agilno metodo Scrum, s katero smo izvedi uspešen nadzor ter sledenje nad delom. Poleg uporabljene metode smo opisali tudi nekaj alternativnih, ki jih uvrščamo med agilne ter njihove predhodnike, ki spadajo v sklop tradicionalnih metod. Na tem mestu bi omenil, da se tradicionalne metode še vedno uporabljajo in ne bojo izumrle. Vse je odvisno od samega projekta, zahtev, specifikacij in željenih rezultatov. Upam si trditi, da se na podlagi dosedanje razvitih metod razvijajo nove, fleksibilnejše, ki bodo posameznim skupinam razvijalcev odgovarjale bolje. Slednje lahko že opazimo v združevanju metodologije Scrum z metodama kanban in XP. V kolikor bi bil projekt podprt s finančnimi sredstvi, bi bilo zanimivo razširiti analizo in spremljati porabo denarnih sredstev na posameznih uporabniških zgodbah in na koncu analizirati porabo in narediti vzporednice z ocenami.

Največji izziv je bilo zagotavljanje transparentnosti ocen ter njihovo pridobivanje. Kot sem že omenil, sem bil za namene implementacije projekta v ekipi sam, zato nekatere ocene odstopajo od ocen, ki bi bile podane v skupini razvijalcev, kar smo lahko opazili tudi v analizah posameznega sprinta. Hkrati pa sem imel možnost izkusiti vlogo vseh akterjev, ki so predvideni po metodologiji. Med njimi po mojem mnenju zahteva največ spoštovanja scrum master, saj je oseba velikokrat tudi razvijalec, zato nosi veliko odgovornosti usklajevanja, organiziranja in vodenja ekipe. Podzavestno je metoda tudi gonilna sila, saj se na dnevnih sestankih jasno vidi, kdo je kaj naredil, kdo ima kje težave, s tovrstnimi sestanki pa se spodbuja tudi delavnost in pomoč.

Metodo sem predstavil vsem sodelujočim med izdelavo projekta. Slednji so ugotovili, da določene principe metode že sedaj uporabljajo in, da so ti nepogrešljiv del vsakdana. Pri diskusiji smo se strinjali, da je za uspešno izvedbo projekta in primerno ozaveščenost odgovornih, uvedba slednje v neko projektno ekipo nujna.

V prihodnje gojim upanje, da se bo projekt razširil po smernicah, ki sem jih navedel. Odgovorim bom tudi predlagal, da se bo pri vsem uporabljala metodologija Scrum, saj slednja na zelo enostaven način ponudi celovit nadzor nad potekom dela.

Literatura

[1] D. J. Anderson, Kanban: *Successful Evolutionary Change for Your technology Business*, Blue Hole Press, 2010.
[2] K. Beck, *Extreme programming explained*, Addison-Wesley, 2004.
[3] A. Časar, V. Mahnič, *Programsko orodje za podporo projektnemu delu študentov*, Didakta, vol. 163, maj 2013.
[4] M. Cohn, *Agile estimating and planning*. Upper Saddle River: Prentice Hall, 2011.
[5] M. Cohn, *User stories applied: for agile software development*, Boston: Addison Wesley, 2011.
[6] (2014) C. Keith, *Beyond Scrum: Lean and Kanban for Game Developers*. Dostopno na: http://www.gamasutra.com/view/feature/3847/beyond_scrum_lean_and_kanban_for_.php?print=1
[7] V. Mahnič, *Tehnologija programske opreme in agilne metode*, prosojnice za predmet Tehnologija programske opreme v štud. letu 2013/14.
[8] R. C. Martin, *Agile software development: principles, patterns and practices*. Upper Saddle River: Pearson Prentice Hall, 2012.
[9] (2000) L. Rising, *The Scrum Software Development Process for Small Teams*. Dostopno na: http://web.lindarising.info/uploads/IEEEScrum.pdf
[10] K. Schwaber, *Agile project management with Scrum*. Redmond: Microsoft press, 2004.
[11] (2014) H. Shojaee, Axosoft Customer survey. Dostopno na: "http://www.axosoft.com/blog/tag/methodology/"
[12] C. Sims, Agile Learning labs. Dostopno na: "http://www.agilelearninglabs.com/2012/05/how-to-play-the-team-estimation-game/"
[13] F. Solina, Projektno vodenje razvoja programske opreme. Ljubljana: Fakulteta za računalništvo in informatiko, 1997.
[14] (2014) B. Swanson, Dynamic Team Estimation in Action. Dostopno na: "http://properosolutions.com/2010/05/dynamic-team-estimation-in-action/"
[15] (2015) Wikipedija Waterfall model. Dostopno na: "http://commons.wikimedia.org/wiki/File:Waterfall_model.svg"
[16] (2015) ISTQB Exam Certification. Dostopno na: http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/"
[17] (2014) www.umsl.edu. Dostopno na: "http://www.umsl.edu/~sauterv/analysis/6840_f09_papers/Nat/Agile_clip_image004_0000.jpg"
[18] (2014) Wikipedija. Dostopno na: "http://en.wikipedia.org/wiki/Kanban"
[19] (2014) http://www.everydaykanban.com. Dostopno na: "http://www.everydaykanban.com/what-is-kanban/"
[20] (2014) Metz Consultancy. DSDM & Scrum. Dostopno na: " http://www.metzconsultancy.eu/?page_id=49"
[21] (2014) Jetbrains YouTrack. Dostopno na: "https://www.jetbrains.com/youtrack/"
[22] (2014) Planning poker vs Table sorting. Dostopno na: "http://agile-management.com/wordpress/planning-poker-vs-table-sorting/"
[23] (2014) Git. http://git-scm.com/. Dostopno na: "http://git-scm.com/"
[24] (2014) Bitbucket. Dostopno na: "https://bitbucket.org/"

[25] (2014) PHP. Dostopno na: "http://php.net/"
[26] (2014) MySql. Dostopno na: "http://www.mysql.com/"
[27] (2014) Bootstrap. Dostopno na: "http://getbootstrap.com/javascript/"(2014) Dostopno na: http://redletterday.ch/?p=550
[28] (2014) Dostopno na: http://jesusgilhernandez.com/2013/02/19/estimation-effort-vs-accuracy/
[29] (2014) Dostopno na: http://samuliheljo.com/blog/reflections-on-kanban-vs-scrum-development/
[30] (2014) jQuery. Dostopno na: http://jquery.com/
[31] (2014) W3 schools. Dostopno na: http://www.w3schools.com/html/html_intro.asp

Priloga A

Seznam vseh uporabniških zgodb sistema rezervacij

ID uporabniške zgodbe	Ime	Kratek opis	Sprejemni test
RP-06	Ragistracija uporabnika	VPis novega uporabnika v aplikacijo, pobiranje ključnih podatkov, potrjevanje pristonosti uporabnika ter vpis.	registracija uporabnika na ime matej@mailinator.com 123 klikniti na potrditveni mail ter se logirati v aplikacijo
RP-7	email sistem obveščanja	urediti sistem email obvestil pošiljanje mailov	vnesi novo rezervacijo, spremeni rezervaciji status, vnesi pomembnejšo rezervacijo, uporabnik mora biti obveščen o spremembah, odpovedih..
RP-09	obveščanje uporabnika ob posamezni akciji	obveščanje uporabnika o posebnih akcijah vstavlanje, izbris, neveljaven vnos..	uporabnik vnese, rezervacijo, jo zbriše, vstavi neveljavne podatke -> biti mora obveščen o nerpavilnih potezah
RP-16	menjava gesla	uporabnik mora imeti možnost spremembe gesla/povrnitev pozabljenega gesla	Registrirani uporabnik zahteva novo geslo. Po menjavi gesla se prijavi v aplikacijo z novim geslom
RP-17	registracija / urejanje uporabnika	registrirani uporabnik mora imeti možnost urejanje svojega uporabniškega profila, uporabnike lahko ureja tudi admin	test registracija z ter nato prijava, kjer ime in priimek username: matej@mailinator.com password: 123
RP-19	admin vmesnik	potrebno je oblikovanje vmesnika, načrtovanje postavitve polij in urejanje layouta	admin dostopi do svojega admin vmesnika preko naslove /admin
RP-22	Uporabniške vloge(admin, registrered, free user)	Določiteve uporabniških vlog, ki bodo imele določene funkcionalnosti v aplikaciji.	normalni - pregleduje spletno stran, uporabnik ni logiran registrirani - spreminja svoje podatke in ureja rezervacije, vpisuje mnenja ali pusti povratni informacijo, administrator - spreminja vse in določa vse ... tajnica - dostop do računov in pogodb v admin vmesniku
RP-36	Okrno za prijavo uporabnika v aplikacijo	pripraviti je potrebno uporabniški pogled prijavljenega nornalnega uporabnika	dostopno na /users/login logiranje naj bo omogočeno za testnega userja: matej.murn@gmail.com 1234
RP-48	Varnostni mehanizem pri resetiranju gesla	ob resetiranju gesla uporabnik dobi potrditveni mail za potrditev.	TEST: matej.murn@gmail.com 123 simuliram da sem pozabil geslo; geslo spremenim na 1234 se poizkušam prijaviti -> dostop mora biti onemogočen! preko povezave ki sem jo prejel na mail aktiviram račun ponovno se poizkušam prijaviti v aplikacijo tokrat prijava uspe
RP-43	Oblikovanje baze	Definirati je potrebno obliko baze tabel in polij ki se bodo uporabljala, nastaviti je potrebno omejitve na nivoju baz.	/
RP-08	priprava prostora za oglaševanje	pripravi prostor za morebitne oglase	ko vključim uporabo oglasov se na strani pojavijo primerna standardizirana okna, kjer lahko vstavimo oglase za oglaševanje
RP-11	podpora večjezičnosti	podpora večim jezikom ang si...	ob izbiri jezika se jezičnost strani zamenja.
RP-12	mobile first oblikovanje	responsive design, iskanje vseh prostih terminov, callendar, user interface admin vmesnik urejanje nalogov, rezervacij, obveščanje uporabnikov	kot uporabnik se prijavim v rezervacijo, kjer imam lepo urejene forme za pregeld, urejanje, editiraje, vnos, ravno tako kot admin in navadni uporabnik
RP-23	kreiranje pdf poročil	urediti je potrebno možnosti tiskanja pogodb vsem rezervacijam.., možnost spreminjanja pogodbe	rezervacijam se kreira pdf in aneks s pritiskom na primeren gumb v admin vmesnuku, ravno tako lahko konec meseca pogledamo račun
RP-42	Uporabniški pogled	navadnemu registriranemu uporabniku omogočimo osnovne funkcionalnosti	uporabnik lahko rezervira prostor ter pogleda svoje: rezervacije odpovedi račune
RP-50	Coutry picker	Dodati možnost izbire držav	kot admin dodam novega ponudnika in mu nastavim državo na Slovenija.
RP-51	preverjanje telefonskih številk	Preveri veljavnost vnesene telefonske številke glede na mobilno ali stacionarno omrežje	Pri prijavi providerja vnesem neveljavno telefonsko številko, na katero sistem opozori.
RP-52	Skype contact	Dodaj kontakt skype za hiter dostop strank do tehnične podpore	kot uporabnik lahko zahtevam tehnično pomoč, preko skypa ali kakega drugega implementiranega obrazca za pomoč tako da kliknem na ikono ter sledim navodilom.
RP-18	dodajanje in urejanje prostorov	dodajanje novih poslovnih prostorov ponudnika	TEST: admin lahko doda in ureja prostore ki pripadajo nekemu providerju preko pripravljenih obrazcev
RP-21	upoarbniški odziv	registrirani uproabnik lahko komentira prostor.	TEST: za določeno rezervacijo lahko podamo comment. Admin lahko comment pogleda ga uredi, pokaže pa se ko normalni uporabnik pregleduje prostore.
RP-25	urejanje, tiskanje računov	admin ima možnost tiskanja pošiljanja obveščanja oseb o plačilu računov neplačilov... z uporabnikom se sklene pogodba..	admin pogleda račune, ob kliku na uporabnika se odprejo detaili uporabnika za morebiten klic.
RP-33	pdf pogodba novega najema	uporabnik dobi pogodbo	TEST: ob vsaki rezervaciji se pojavi gumb za download pogodbe s predizpolnjenimi podatki o uporabniku. in najemu prostora
RP-53	Ajax loading dropdown	zagotoviti je potrebno ajax nalaganje odvisnega dropdowna kjer je to potrebno	TEST: Pri rezervaciji se polje places izpolni glede na izbranega providerja s prpostori, ki smo jih vnesli pod določenim providerjem Zadeva se uporabi tudi tam, kjer se ne sme zgoditi redirect!
RP-30	Rezervacija termina	rezervacija poljubnega termina in obveščanje o poteku same rezervacije	uporabnik rezervira prostor admin potrdi novo rezervacijo, uporabnik je obveščen admin pa dobi možnost pregleda pogodbe.
RP-40	404 error page	obvestilo ob nedovoljenem dostopu	pojdi na neveljavno stran, kjer mora biti obvestilo o napaki ter povezava do delujoče strani
RP-54	ureditev shranjevanja cookijev	uporaba cookijev ter obveščanje uporabnikov	uporabnik obišče stran, in potrdi zakon cookijev
RP-31	vstavlanje odpovedi rezervacije	registrirani uporabnik lahko odpove svoj najeti termin 7 dni pred uporabo	uporabnik ki imarezervirano rezervacijo jo odpove, vstavi se odpoved ter odšteje od mesečne najemnine; odpoved mora biti podana 7 dni pred terminom drugače odpovedi ni mogoče vnesti
RP-32	Premik rezervacije	registrirani uporabnik lahko premakne svojo rezervacijo; sistem ga opozori če je premik možen in če je prostor prazen	uprabnik kateremu rezervacija še ni bila potrjena spremeni trajanje ali čas ene rezervacije
RP-39	ureditev logiranja napak	urejanej logiranja napak za debugiranje in odkrivanje napak	vse napake naj se logirajo v datoteko logs v primeren direktorij
RP-10	prokaz rezervacij v koledarju	Pregled obstoječih rešitev implementacije koledarja za prikaz zasedenosti dvoran, razmisliti o alternativnih prikazih	admin ima možnost pregleda rezervacij v preglednem javascript urniku, kjer so vidne rezervacije ter zasedenost določenega providerja
RP-03	prijava uporabnika s facebook računom	realizacija prijave uporabnika s pomočjo njegovega facebook računa	uporabnik se v aplikacijo prijavi preko fb računa
RP-04	prijava uproabnika z google računom	prijava preko google računa uporabnika	uporabnik se prijavi preko google računa
RP-05	prijava uporabnika s twitter računom	praijava uporabnika z njegovim twitter računom	uporabnik se lahko prijavi v račun preko twitterja
RP-15	paging iskanja	ob veliki količini zapisov potrebno urediti pregled	vnesi več kot 20 zapisov pojaviti semora paging, delovati mora sortiranje po pagingu
RP-20	preverjanje prostih terminov pri rezervaciji	uporabnik pogleda proste termine med rezervacijo.	pri rezervaciji izberemo datum ter pritisnemo pokaži proste termine kjer se izpišejo prosti termini ki so na voljo za rezervacijo.
RP-44	Vstavlanje pomembnega dogodka	admin vstavi pomemben dogodek	admin vstavi pomembnejši dogodek na termin kjer so že ostale rezervacije, uporabniki so obveščenio njihovih odpovedih., odpovedi se odštejejo od njihovih nejemnin.

Priloga 1: Opis posameznih uporabniških zgodb

Priloga B

Pridobljene ocene po metodi *Team estimation game*

ocena PP	ID Uporabniške zgodbe
1	RP-51
1	RP-52
2	RP-40
2	RP-54
2	RP-15
2	RP-50
3	RP-31
3	RP-48
3	RP-09
3	RP-36
4	RP-39
4	RP-21
5	RP-43
5	RP-08
5	RP-16
5	RP-18
5	RP-53
5	RP-03
5	RP-04
5	RP-05
5	RP-11
7	RP-22
7	RP-17
9	RP-06
10	RP-32
14	RP-25
14	RP-33
15	RP-07
15	RP-19
15	RP-23
15	RP-42
15	RP-20
15	RP-12
25	RP-44
30	RP-10
30	RP-30

Priloga C

Primerjava ocen po metodah planning poker, team estimation game in dejansko porabljenega časa