Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andrej Ograjenšek

Uporaba metode Kanban pri razvoju programske opreme

diplomsko delo

univerzitetni študij računalništvo in informatika

Mentor: dr. Viljan Mahnič

Ljubljana, 2013

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01942 / 2013
Datum: 4.9.2013

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

Kandidat: **ANDREJ OGRAJENŠEK**

Naslov: **UPORABA METODE KANBAN PRI RAZVOJU PROGRAMSKE OPREME**
APPLYING KANBAN TO SOFTWARE DEVELOPMENT

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Proučite metodo Kanban in možnosti za njeno uporabo pri razvoju programske opreme. Izdelajte pregled razpoložljivih orodij, ki podpirajo uporabo te metode, in analizirajte njihove značilnosti. Na podlagi tega izdelajte specifikacijo zahtev in podatkovni model za čim bolj popolno orodje te vrste. Specifikacijo zahtev pripravite v obliki uporabniških zgodb, skupaj z ustreznimi sprejemnimi testi.

Mentor: Dekan:

izr. prof. dr. Viljan Mahnič prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andrej Ograjenšek, z vpisno številko **63080154**, sem avtor diplomskega dela z naslovom:

Uporaba metode Kanban pri razvoju programske opreme

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 25. novembra 2013 Podpis avtorja:

Zahvaljujem se sošolcem in družini za podporo med študijem. Posebna zahvala gre mentorju za pomoč pri izdelavi tega diplomskega dela.

Kazalo

Povzetek

V diplomskem delu je predstavljena agilna metodologija Kanban ter orodja, ki podpirajo razvoj programske opreme v skladu z njo. Namen diplomske naloge je analiza metodologije Kanban ter specifikacija popolnega računalniškega orodja za podporo delu z njo.

V prvem delu je predstavljena metodologija - njene značilnosti, prednosti in slabosti. Sledi analiza obstoječih orodij, ki so trenutno najbolj uporabljena v praksi. V zadnjem delu smo specificirali popolno računalniško orodje za podporo metodologiji Kanban v obliki uporabniških zgodb ter izdelali logični podatkovni model.

Ključne besede

Kanban, agilne metodologije.

Abstract

Kanban - an agile software development methodology - is presented in the thesis along with an analysis of existing tools that support software development in compliance with it. On the basis of this analysis a specification for a new software tool which supports software development in compliance with Kanban is proposed.

The thesis starts with an introduction of Kanban - its features, advantages and drawbacks. An analysis of existing tools that are currently most used on the market follows. The thesis is concluded with a specification for a new software development tool along with a logical data model of the tool's underlying database.

Keywords

Kanban, agile methodologies.

Poglavje 1

Uvod

1.1 Zakaj agilne metodologije

Agilne metodologije so v zadnjih letih najbolj aktualne pri razvoju programske opreme. Ker se osredotočajo na zadovoljstvo naročnikov in hitro izdelavo delujoče programske opreme v iteracijah, so priljubljene tako pri naročnikih kot tudi pri razvijalcih. Za razliko od klasičnih pristopov, kjer je poudarek na natančnem načrtovanju in obsežni dokumentaciji je pri agilnih metodologijah poudarek na delujoči programski opremi, ki jo naročniku dostavljamo v iteracijah. Velika prednost agilnih metodologij je tudi odprtost k sprejemanju sprememb tekom projekta - tako naročniki kot razvijalci niso več obremenjeni z obsežnim in rigoroznim planiranjem, saj se lahko spremembe uvedejo tekom razvoja opreme. Raziskave med podjetji kažejo, da pri razvoju programske opreme prevladuje uporaba agilnih metodologij, zato sem se v svojem diplomskem delu usmeril na to področje.

1.2 Zakaj Kanban

Kanban so sprva uporabljali v proizvodnih sistemih, kjer se ravnajo po načelu "Just-In-Time" (JIT). Najboljše rezultate so dosegli pri Toyoti, kjer so z uporabo Kanbana zmanjšali povprečen čas izdelave in tako povečali zmogljivost

in produktivnost. Zaradi uspeha, ki ga je prinesel proizvodni industriji, je prišlo do ideje, da bi Kanban lahko uporabili pri razvoju programske opreme.

V reviji IEEE Software smo zasledili zanimiv članek, ki je primerjal proces razvoja programske opreme z uporabo metodologij Scrum in Kanban. Rezultati so pokazali, da se je v podjetju, kjer so presedlali iz Scrum-a na Kanban, povprečni potrebni čas izdelave (angl. average lead time) razpolovil, količina hroščev je padla za 10 %, produktivnost pa se je povečala [8].

V prispevku podjetja VersionOne, "7th Annual State of Agile Development Survey", ki vsako leto naredi podrobno raziskavo med uporabniki agilnih metodologij, navajajo, da se je uporaba Kanbana in njegovih izpeljank v zadnjem letu podvojila [1] .

Glede na stanje v svetu se nam je zdela uporaba Kanbana pri razvoju programske opreme zanimiva tema, ki si zasluži podrobno obravnavo. V drugem poglavju je na začetku predstavljen Kanban in njegove prednosti, sledi opis implementacije metodologije, na koncu pa so podani načini optimizacije sistema. Celotno poglavje je povzeto po delu Davida A. Andersona, "Kanban, Successful Evolutionary Change for Your Technology Business" [2]. Tretje poglavje vsebuje opis in primerjavo obstoječih orodij. Na podlagi analize orodij nato v četrtem poglavju podamo funkcionalne specifikacije popolnega orodja za Kanban v obliki uporabniških zgodb. V zadnjem poglavju so zapisane sklepne ugotovitve in ideje za nadaljnje delo.

Poglavje 2

Kanban

2.1 Kaj je Kanban?

Kot sem že v uvodu omenil, so Kanban sprva uporabljali v proizvodnih sistemih. Proizvod se iz ene postaje v liniji na drugo premakne takrat, ko je delavec, ki prevzema delo, prost in pripravljen na sprejem. Delo poteka po principu "Pull"(prevzemi delo nase, ko si pripravljen) in ne "Push"(delo ti nalagajo drugi), kot smo običajno navajeni. Pri razvoju programske opreme smatramo Kanban kot agilno metodologijo, ki podpira proces razvoja ter omogoča postopno uvajanje sprememb v sistem. Metodologija ni tako stroga kot sorodne, npr. Scrum, saj ne zahteva, da se naš razvojni proces spremeni čez noč, da bi zadostil nekim vnaprej določenim pravilom. Sama vpeljava poteka tako, da povzroča čim manj odpora pri zaposlenih in tistih, ki se bodo ravnali po Kanbanu. Ideja prevzemanja dela ostaja enaka kot pri različici, ki se uporablja v proizvodnih procesih, ena izmed dodatnih pa je omejitev količine dela v teku (angl. work in progress). Bolj podroben opis načel in načina implementacije sledi v nadaljnjih poglavjih.

2.2 Značilnosti Kanbana

2.2.1 Osredotočenost na kakovost

Ena izmed osnovnih sestavin Kanbana je osredotočenost na kakovost. Nekatera podjetja porabijo tudi do 90 % časa za popravilo in odpravljanje napak, ki so posledica slabe kakovosti. Z izboljšanjem kvalitete se lahko propustnost poveča za faktor 2 ali 4, v resnično slabih primerih tudi za faktor 10. Očitno je, da lahko velik korak k izboljšavi poslovnega sistema naredimo že s povečanjem kakovosti.

Tako agilni kot tradicionalni pristopi h kakovosti imajo pozitivne učinke. Uporabljati jih moramo v kombinaciji. Uporaba testerjev za odkrivanje napak je odličen način, kako preprečiti uhajanje hroščev v produkcijo. Pisanje testov enot(angl. unit tests) prav tako prikazuje dobre rezultate. V praksi se je izkazalo, da pisanje testov pred samo funkcionalno implementacijo problema izboljšuje kvaliteto rešitve.

Pregledi kode so ključni za ohranjanje kakovosti tako kode, kot tudi celotne rešitve problema. Lahko je to programiranje v parih, pregled sodelavca ali skupinski pregledi kode - jasno je, da pregledovanje kode povečuje kvaliteto. Najbolje je, če se pregledi izvajajo pogosto in v manjših sklopih.

Skupinska analiza in načrtovanje ter uporaba načrtovalnih vzorcev pripomorejo k povečanju kakovosti. Podobno kot za pregled kode velja tudi za analizo in načrtovanje, da je najboljše, če se izvajata pogosto in v manjših sklopih.

Uporaba modernih razvijalskih orodij je zaželena. Večina jih ima možnost statične in dinamične analize kode. Le-te opozorijo programerja na pogoste napake, kot so varnostne luknje ipd. Funkcionalnost analize bi morala biti vključena in prilagojena vsakemu projektu posebej. Bolj napredna orodja za razvoj, kot so “Software Factories”, še dodatno zmanjšajo število napak. Načrtovalski vzorci so pogosto zapakirani v koščke kode, ki preverjeno delujejo - tako na teh delih kode ni potreben dodaten pregled.

2.2.2 Zmanjševanje količine dela v teku in pogoste izdaje

V knjigi "Kanban, Successful Evolutionary Change for Your Technology Business"avtor David J. Anderson navaja, da je v praksi prišel do spoznanja, da je količina dela v teku neposredno povezana s povprečnim časom izdelave - več kot je dela v teku, daljši je povprečni potrebni čas izdelave, ta pa botruje slabši kvaliteti. Tako ugotavlja, da najhitreje skrajšamo povprečni potrebni čas izdelave in povečamo kakovost z omejitvijo količine dela v teku. Skrajševanje dolžine iteracij prav tako pripomore.

Z zmanjšanim povprečnim potrebnim časom izdelave si lahko privoščimo pogostejše izdaje programske opreme. Majhne in pogoste izdaje so bolj zaželene kot redke in velike. Te nam prinesejo zaupanje pri drugih organizacijskih enotah znotraj podjetja, predvsem pri finančni in prodajni enoti. S tem povečujemo svoj družbeni kapital. Prav tako nam kvalitetna izdelava prinese zaupanje pri sodelavcih, ki bodo vzdrževali in prodajali našo rešitev.

2.2.3 Uravnavanje ravnovesja med povpraševanjem in propustnostjo

Uravnavanje ravnovesja med povpraševanjem in propustnostjo pomeni, kako pogosto sprejemamo nove zahteve v naš razvojni proces v razmerju s tem, kako hitro lahko dostavimo delujočo kodo. S tem posledično omejimo količino dela v teku na določeno število zahtev. Nove zahteve sprejemamo v sistem šele takrat, ko dostavimo implementirano zahtevo.

Ta omejitev ima takojšen in globok učinek na delovanje našega procesa. Pred tem verjetno nismo vedeli, kje v našem procesu leži ozko grlo. Sedaj, ko smo omejili količino dela v teku in sprejemanje novih zahtev, smo le-to razkrili. Samo viri, ki so dejansko ozka grla, bodo ostali 100 % zasedeni. S tem razvojna ekipa ne bo več zasuta z delom kot prej, nekateri bodo imeli celo čas še za kaj drugega.

S tem, ko smo namenili nekaj prostega časa udeležencem v procesu, smo

jih razbremenili stresa, ti pa se bodo zato lahko bolje posvečali kvaliteti in natančnosti pri svojih opravilih. Zaposlene spodbujamo k izboljševanju delovnega procesa, izobraževanju, izpopolnjevanju svojih spretnosti in veščin. S tem smo v organizaciji spodbudili stalno izboljševanje, zanj pa potrebujemo proste vire. Čeprav v današnji družbi velja nepisano pravilo, da je treba proste vire izkoristiti in obremeniti, se izkaže, da to negativno vpliva na razvoj t. i. "Kaizen"ali stalno izboljšujoče se kulture.

2.2.4 Določanje prioritet

Določanje prioritete posamezni nalogi oz. funkcionalnosti je delo lastnika produkta, pokrovitelja ali tržnega oddelka, ne razvojne ekipe. Ta lahko le opozarja in svetuje, ne sme pa postavljati prioritet. Namen določanja prioritet je optimiziranje dostavljene vrednosti produkta in ne števila vrstic kode, ki smo jo dostavili. Prioritetizacije se lotimo šele, ko imamo dovolj visoko kakovost dostavljene kode, omejeno količino dela v teku ter pogosto izdajamo kvalitetno programsko opremo. Pred tem postavljanje prioritet ni smiselno.

2.2.5 Uravnavanje virov variabilnosti za povečanje predvidljivosti

Viri variabilnosti se največkrat pojavljajo v obliki slabih poslovnih pravil. Ta poslovna pravila nato vnašajo negotovost v delovni tok, ki se izrazi v nezanesljivem dostavnem času. K odpravljanju nekaterih manjših virov negotovosti pripomore tudi uporaba Kanbana, saj metodologija teži k vzpostavljanju ravni storitev ter definiranju tipov nalog. Tako lahko zahtevo umestimo v raven storitve ter tako lažje in bolje ocenimo porabo virov. Zmanjševanja negotovosti se lahko lotijo le najbolj zrele organizacije, saj zahteva spremembo načina razmišljanja ključnih udeležencev delovnega toka.

2.2.6 Kaizen kultura

Kaizen v japonščini pomeni "stalno izboljševanje". Delovna kultura, kjer se vsi zaposleni osredotočajo na stalno izboljševanje kvalitete, produktivnosti in zadovoljstva strank, se imenuje "kaizen kultura".

Pri kaizen kulturi so zaposleni pooblaščeni, da ukrepajo - naredijo tisto, kar se jim zdi prav. Ob nastanku problema se zberejo skupaj, predebatirajo možnosti in implementirajo popravke. Nimajo strahu pred ukrepanjem. To zahteva od vodstva, da je pripravljeno sprejeti neuspeh zavoljo stalnega napredka in inovacij. Uslužbenci se organizirajo sami, kaj delajo in kako bodo to naredili. Za posamezne naloge se raje prostovoljno javijo, kot pa da bi jim jih dodelili drugi. Stopnja kolegialnosti in sodelovanja znotraj kaizen kulture je visoka.

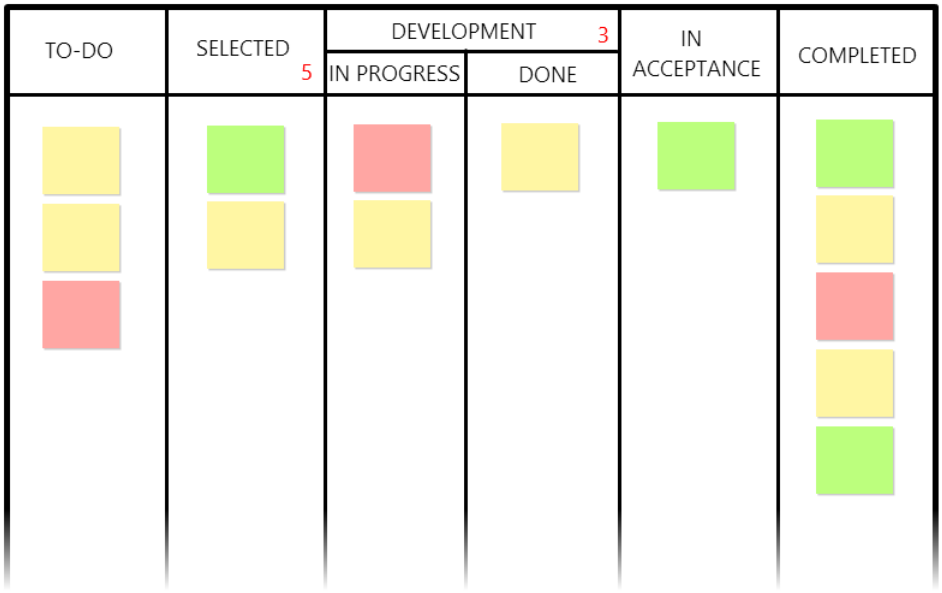
Podjetje, v katerem vlada kaizen kultura, ima velik družbeni kapital. Gre za kulturo, kjer je stopnja zaupanja visoka. Zaposleni, ne glede na njihov položaj, spoštujejo drug drugega in sprejete odločitve. Visoka stopnja zaupanja omogoča izničevanje odvečnih ravni odločanja v organizaciji, kar pomeni prihranek pri stroških in času. Ravno zaradi tega je taka kultura popolnoma drugačna od tega, kar nas učijo v zahodnem svetu. Otroke že od malih nog vzgajamo v tekmovalnem duhu, izpostavljamo najboljše posameznike in iz njih ustvarjamo heroje. Ravno zaradi tega je na zahodu doseganje kaizen kulture toliko težje.

Na področju razvoja programske opreme obstaja model CMMI (Capability Maturity Model Integration) za ocenjevanje zrelosti organizacije. Najvišja stopnja je "optimizing", ki pove, da je podjetje osredotočeno na stalno izboljševanje procesov, ki tečejo znotraj podjetja. Doseči to stopnjo zrelosti je najlažje, če v podjetju vlada kaizen kultura.

2.3 Implementacija Kanbana

Kanban je pristop, ki spodbuja spremembe z optimizacijo obstoječega procesa. Bistvo začetka uporabe Kanbana je predpostavka, da spremenimo kar

se da malo stvari ter vizualiziramo tok dela skozi sistem s pomočjo fizične ali virtualne table s karticami. Primer table je prikazan na sliki 2.1. Upreti se je treba željam po spreminjanju delovnega procesa, spremembi nalog posameznih delovnih mest in modifikacijam načina dela. Vse, iz česar zaposleni in udeleženci črpajo svojo samozavest in ponos, moramo pustiti pri miru. Glavni cilj je omejiti količino dela v teku in način komunikacije s partnerji.



Slika 2.1: Primer table

2.3.1 Modeliranje delovnega toka

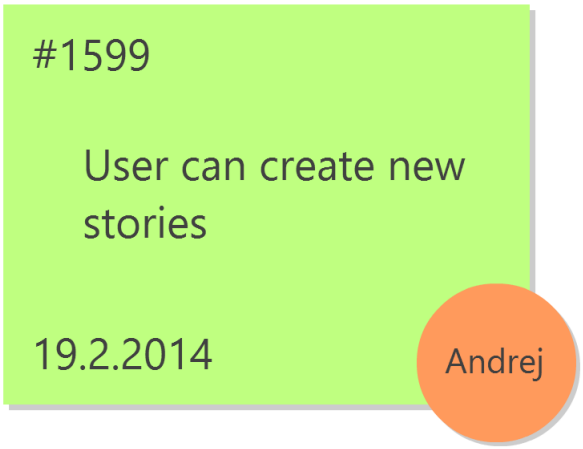
Pri modeliranju delovnega toka moramo paziti, da opisujemo dejanske korake delujočega procesa in ne tistega, ki naj bi veljal po pravilih, ki jih nihče ne upošteva. Tako bomo lahko vizualizirali dejanski proces, kjer se bodo udeleženci prepoznali in z lahkoto znašli, saj bo odražal realno stanje v podjetju.

Prvi korak pri modeliranju je postavitev meja procesa - kje se začne del procesa, v katerem smo udeleženi, in kje se konča. Bistvenega pomena je, da se tega koraka lotimo z občutkom, saj ne smemo vsiljevati svojega načina dela ljudem, ki niso del naše ekipe.

Ko smo enkrat določili meje, moramo identificirati tipe nalog, ki prihajajo v naš sistem. Tipe nalog lahko poimenujemo glede na njihov izvor, velikost ali tip implementacije. Tako bi recimo lahko tip naloge bil zahteva regulatorja, če bi šlo za poimenovanje v skladu z izvorom, ali pa odpravljanje hrošča, če bi jo poimenovali v skladu s tipom implementacije. Tak način poimenovanja omogoča večjo transparentnost ter vnaša dodatne informacije in kontekst v proces.

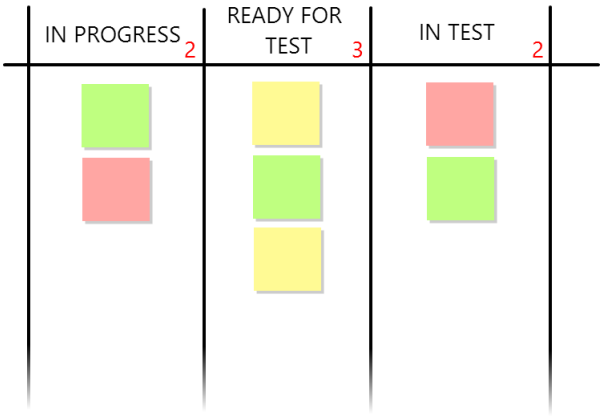
Naslednji korak je risanje table s karticami. Kartice predstavljajo nalogo, ki se nahaja v našem razvojnem procesu. Najpogosteje na kartici najdemo šifro naloge, ki jo enolično identificira ali pa jo povezuje v nek elektronski sistem sledenja, opis naloge, datum vstopa v sistem, lahko tudi rok, do katerega moramo zadevo implementirati. Oseba, ki se trenutno ukvarja z nalogo je napisana na tabli zraven kartice z nalogo ali pa ima pripet kak listek, ki označuje to osebo. Pisanje oseb na kartico z nalogo ni zaželeno, saj se skozi proces razvoja in testiranja z nalogo ukvarja več ljudi. Primer kartice je prikazan na sliki 2.2.

Kartice z nalogami se premikajo po stolpcih na tabli. Stolpci predstavljajo aktivnosti v našem procesu, nanizane v zaporedju, v katerem se izvajajo. Na začetku je te meje smiselno določiti z markerjem, saj se znajo nekatere s časom premakniti ali celo izbrisati. Ko smo enkrat z mejami zadovoljni, jih je najbolje označiti s plastičnim trakom ali čim bolj obstojnim. Med posamezne aktivnosti lahko dodamo čakalne vrste. Čakalne vrste uporabljamo za izravnavanje toka skozi sistem. Postavljamo jih pred aktivnosti, za katere si ne moremo privoščiti, da bi ostale brez dela. To seveda ni pravilo, saj obstaja več priporočil, kdaj in kam postaviti čakalno vrsto. Lahko določimo proces brez čakalnih vrst in tako počakamo, da se ozko grlo v sistemu samo pokaže, nato pa uvedemo vrsto na ustrezno mesto. Drugi



Slika 2.2: Primer kartice z oznako odgovorne osebe

pristop pravi, naj postavimo ohlapne omejitve količine dela v teku ter pred vsako aktivnost uvedemo čakalno vrsto. Opazujemo, kako hitro se vrste napolnijo, in tako zelo hitro opazimo ozko grlo sistema. Z majhnimi spremembami zmanjšujemo dolžine čakalnih vrst in nepotrebne sčasoma tudi ukinemo. Primer čakalne vrste na sliki 2.3 bi bil stolpec "Ready for test", ki označuje stanje čakanja na naslednjo aktivnost. Tipe nalog, ki smo jih



Slika 2.3: Primer table s čakalno vrsto

določili v enem izmed prejšnjih korakov, je potrebno podrobno analizirati.

Če imamo na voljo zgodovino opravljenega dela, lahko iz nje razberemo, kako pogosto in v kakšnih količinah prihajajo posamezni tipi delovnih nalog. Tako razpoložljive vire ustrezno razporedimo med posamezne tipe nalog. Na tabli s karticami lahko to najlažje prikažemo z uporabo t. i. "stez", kjer vsaka vsebuje le en tip delovne naloge.

Eno izmed pomembnih orodij je tudi elektronski način sledenja dela. Omogoča nam, da ljudje na različnih geografskih lokacijah sodelujejo na istem projektu, saj so vse naloge in opisi na voljo v elektronski obliki. Naprednejša orodja za podporo Kanbana omogočajo tudi virtualizacijo kartičnih tabel, tako da je "virtualna tabla"na voljo vedno in povsod, ne glede na lokacijo. Ker so naloge zabeležene v elektronskem orodju, je ponavadi mogoče na podlagi zgodovine analizirati in izboljšati delovni proces.

2.3.2 Koordinacija

Koordinacija pri Kanbanu poteka s pomočjo table s karticami, elektronskega orodja za sledenje dela in sestankov.

Tabla s karticami omogoča, da razvijalci iz nabora nalog sami izberejo tisto, ki se jim zdi najbolj primerna. To razberejo iz informacij, ki jih ponujajo kartice na tabli kot tudi zgradba same table. Pomemben je tip dela, ki ga predstavlja kartica, opis naloge in tudi sama pozicija na tabli. Naloge, ki zahtevajo pozornost ekipe, so na tabli posebej označene. Elektronsko orodje dopolnjuje tablo z različnimi metrikami in opomniki, ki nas opozarjajo na problematične zahteve.

Prvi izmed sestankov, namenjenih koordinaciji, je t. i. "Daily Standup". Odvija se vsak dan ob dogovorjeni uri. Udeleženci sestanka se zberejo okoli table s karticami in pregledajo, če katera izmed nalog zahteva dodatno pozornost. Pozornost zahtevajo naloge, ki že več dni stojijo v istem stolpcu oz. fazi razvoja ali pa so označene kot blokirane, saj čakajo na razrešitev kake druge naloge. Ostale naloge preletijo in ocenijo, kdaj bodo končane. Sestanek poteka stoje in ne traja več kot 15 minut.

Takoj po dnevnem sestanku pride do t. i. "After Meeting-a". Udeleženci

dnevnega sestanka se razdelijo na manjše podskupine z namenom, da bi na kratko predebatirali in uskladili potek dela. Na tem sestanku pogosto pride do največ predlogov za izboljšave in je tako eden bolj učinkovitih sestankov pri Kanbanu.

Ko enkrat na vhodu našega procesa zmanjka delovnih nalog, se izvede sestanek, kjer se ponovno napolni vhodna čakalna vrsta. Udeležijo se ga predstavniki naročnika ter vodstveni del ekipe, ki razvija programsko opremo. Sestanek naj bi se izvajal ob rednih intervalih, npr. enkrat tedensko, lahko pa se izvede tudi po potrebi, ko je nabor nalog skoraj prazen. S tem prihranimo stroške usklajevanja in sestankovanja.

Izdaja nove različice programske opreme se načrtuje na sestanku za načrtovanje izdaje, kjer se zberejo strokovnjaki s posameznega področja razvojne ekipe. Pregledajo, kaj je primerno za naslednjo izdajo, kakšna so tveganja in možne posledice. Ob koncu sestanka je jasno določeno, kako bo potekala izdaja nove različice.

Tekom razvoja prihaja do t. i. triažnih sestankov. Na teh sestankih se izvaja čiščenje seznama zahtev. Če je določena zahteva na seznamu že dolgo časa in še vedno ni prišla na vrsto za razvoj, potem je velika verjetnost, da lahko to zahtevo zbrišemo s seznama zaradi nepomembnosti. Sestanki tega tipa se odvijajo redkeje, enkrat na dva do tri mesece ali pa še redkeje. Udeležijo se ga isti ljudje kot sestanka za polnjenje čakalne vrste.

Pri implementaciji posamezne naloge oz. zahteve lahko pride do ovir. Te ovire se včasih lahko rešijo samo s posredovanjem nadrejenega ali poznavalca vsebine. Ključen je jasno definiran način stopnjevanja zahteve do nadrejenega, saj se le tako ovire hitro odstrani. Probleme, ki nastanejo pri geografsko porazdeljenih razvojnih ekipah, rešujemo s pomočjo sodobnih (tele)komunikacijskih orodij.

2.3.3 Frekvenca izdaj

Kanban tako kot ostale agilne metodologije zagovarja redne izdaje delujoče programske opreme. Časovni interval izdaj je potrebno določiti z upoštevanjem

transakcijskih in koordinacijskih stroškov. Če so stroški veliki, je smiselno izbrati daljši interval, in obratno, če so stroški majhni, izberemo krajši interval. V skladu z načelom vitkosti Kanban teži k zmanjševanju teh stroškov z uporabo prostih virov v sistemu. Poleg rednih izdaj imamo tudi možnost izdaje na zahtevo. Ta pride v poštev, kadar so stroški majhni ali ko gre za neko nujno zahtevo.

2.3.4 Frekvenca sprejemanja zahtev v sistem

Ko govorimo o frekvenci sprejemanja zahtev v sistem, gre za podobno stvar kot pri frekvenci dostave programske opreme. Kanban zagovarja redne in pogoste sestanke, kjer se določa prednosti zahtevam na vhodu sistema. Podobno tudi tu nastajajo transakcijski in koordinacijski stroški, zato je interval potrebno določiti v skladu z njimi.

2.3.5 Omejevanje količine dela v teku

Eno temeljnih načel Kanbana je omejevanje količine dela v teku. Z omejitvijo zagotovimo udeleženim v procesu, da ne bodo zasuti z nalogami v pričakovanju, da bodo vse dokončane pravočasno. Pomembno je, da se omejitve določijo v sodelovanju z ljudmi, ki nam postavljajo naloge, ter ljudmi, ki jih od nas prejemajo. Tako se lahko v primeru nesoglasij in zamud obrnemo na skupni dogovor kot izhodišče pogajanja.

Določanje meje količine dela v teku je odvisna od vsakega primera posebej. V nekaterih primerih bo smiselno, da lahko vsak zaposleni dela samo na eni nalogi hkrati, medtem ko bo nekje ta številka višja. Ko postavljamo omejitve moramo imeti v mislih, da se delo na posamezni nalogi lahko ustavi zaradi drugih dejavnikov, pri čemer bodo udeleženi začasno ostali brez dela. Zato je smotrno, da pri postavljanju omejitev dodamo še nalogo ali dve viška in tako izravnamo obdobje, ko bo katera izmed nalog blokirana. Vredno si je zapomniti, da postavljanje omejitve ni dokončno, vedno jo lahko prilagodimo našim razmeram.

Ko vpeljujemo Kanban, hitro naletimo na koncept čakalnih vrst. Kdaj in kje postaviti čakalno vrsto in kako dolga naj bo? Idealno bi bilo, če čakalnih vrst ne bi potrebovali in bi naloge čez naš sistem tekle gladko. Če temu ni tako in se v našem sistemu udeleženci pogosto znajdejo brez dela, ker čakajo na naslednjega v procesu, je smiselno postaviti čakalno vrsto. Tako izravnamo tok skozi sistem. Čakalno vrsto v večini primerov postavimo pred aktivnost v procesu, ki velja za ozko grlo v sistemu.

Velikost čakalne vrste na vhodu našega sistema določimo v skladu s pretočnostjo in frekvenco polnjenja vhodne vrste. Idealno je, da se vhodna čakalna vrsta izprazni tik pred sestankom za polnjenje vhodne vrste. Če se nam dogaja, da sistem stoji dan ali dva pred sestankom za polnjenje zaradi prazne čakalne vrste, potem moramo povečati velikost vhodne vrste. Obratno, če je na sestanku v vrsti še vedno nekaj nalog od prejšnjega cikla, je smiselno zmanjšati velikost vrste.

Pri postavljanju omejitve moramo biti previdni, saj ne smemo preobremeniti sistema. Če postavimo zelo stroge in nizke omejitve ter so blokirane naloge v sistemu pogoste, bo veliko zaposlenih brez dela, saj bodo čakali, da se ovira odstrani. To se najpogosteje dogaja v podjetjih z nižjo stopnjo zrelosti.

Postavljanja omejitev nas ne sme biti strah. Omejitev moramo postaviti in jo prilagajati našemu procesu. Implementacija Kanbana brez postavljanja omejitev na količino dela v teku se je v praksi izkazala za slabo, saj so učinki zelo majhni in pogostokrat nevidni. Omejitve hitro pokažejo dobre in predvsem slabe lastnosti procesa in tako spodbujajo debato, ki ponavadi prinese izboljšave.

2.3.6 Ravni storitve

Ko govorimo o ravneh storitve, govorimo o tem, kako obravnavamo nek tip naloge, ko vstopi v naš sistem. Od ravni storitve je odvisno, kako hitro bomo nalogo spravili v obdelavo, koliko virov ji bomo namenili in kako hitro jo bomo opravili. Tipično definiramo do šest različnih ravni storitve, več ni

smiselno, saj je potem težko organizirati delo in klasificirati oziroma umestiti posamezno nalogo. Primeri ravni storitev iz prakse:

- **Najvišja raven storitve** - nalogo je potrebno dokončati do določenega roka, ki je zelo kratek. Naloga je iz finančnega vidika zelo pomembna za podjetje in se jo zato splača obravnavati drugače. Število teh nalog v sistemu mora biti omejeno, saj ta raven storitve moti in ovira normalno poslovanje sistema.

- **Naloga s fiksnim rokom** - nalogo je potrebno opraviti do fiksno določenega datuma, sicer lahko pride do dodatnih stroškov ali do izpada poslovanja. Gre za raven storitve, ki se uporablja pri nalogah, kjer je rok fiksno določen in je vsako odstopanje od tega roka kaznovano z globo ali pa zaradi tega izgubimo posel. V obdelavo se jo sprejme ob pravem času, tako da bo dokončana malo pred rokom zapadlosti. Če je naloga v nevarnosti, da ne bo dokončana do roka, jo lahko povzdignemo v višjo raven storitve.

- **Standardna raven storitve** - sem spadajo naloge, ki jih obravnavamo po normalnem postopku. Ponavadi imajo nek strošek zamude, ki pa ni tako velik kot pri nalogah s fiksnim rokom.

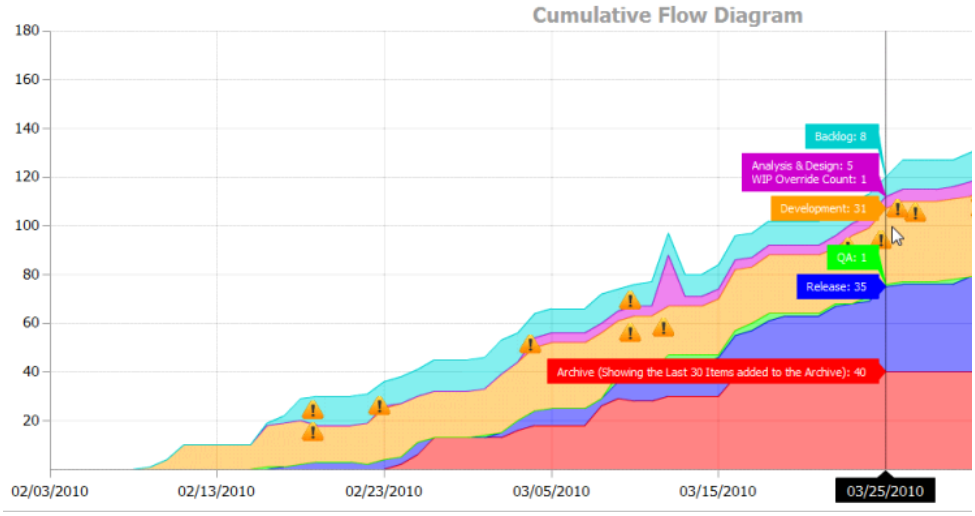
- **Najnižja raven storitve** - naloge, ki imajo rok daleč v prihodnosti ali so manj pomembne izboljšave. Ta tip naloge ponavadi izpodrinejo naloge z višjo ravnjo storitve.

Ko govorimo o ravneh storitve, povemo strankam, v kakšnem času in s kakšno verjetnostjo bo določena naloga dokončana. Primer - naloge s standardno ravnjo storitve bodo obdelane v roku 23-ih dni z 90 % verjetnostjo. To pomeni da bo 9 od 10-ih nalog opravljenih v roku 23-ih dni. To je samo cilj in ne obveza in tega se morajo zavedati tudi stranke.

2.3.7 Metrike in poročila

Pri Kanbanu so poročila drugačna od tistih, ki smo jih navajenih pri ostalih agilnih metodologijah. Ne zanima nas, ali je projekt na pravi poti in če bo končan po planu. Bolj nas zanima, ali Kanban deluje v skladu z našimi cilji, ki naj bodo izboljšanje kulture v podjetju in pot k stalnemu napredku. Zanima nas tudi, ali smo dovolj odzivni in koliko časa potrebujemo, da dano nalogo obdelamo v skladu z ravnjo storitve.

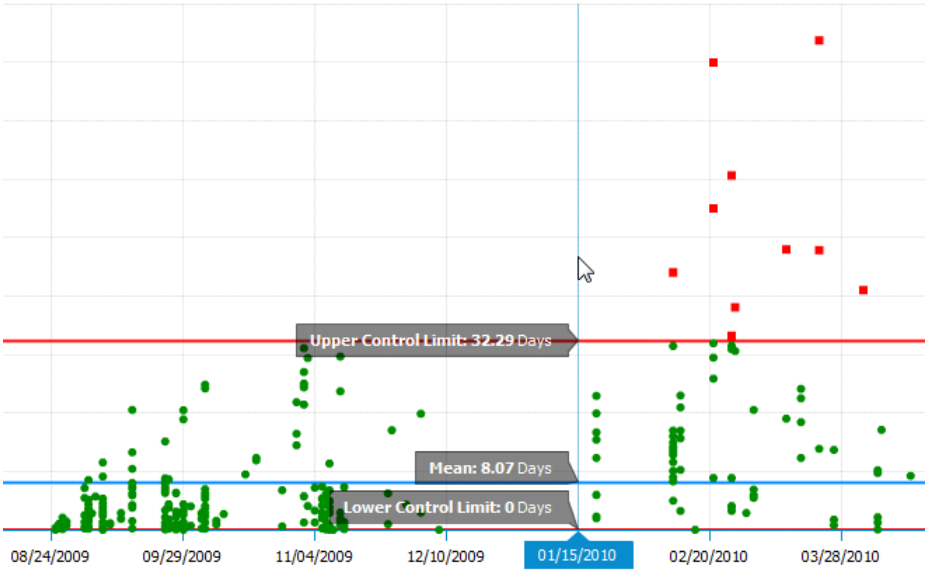
Eno izmed osnovnih poročil je kumulativni diagram delovnega toka (angl. cumulative flow diagram), iz katerega je razvidno, kako dobro se držimo omejitev količine dela v teku ter koliko časa v povprečju potrebujemo, da dano nalogo obdelamo. Primer je prikazan na sliki 2.4.



Slika 2.4: Primer kumulativnega diagrama delovnega toka [5]

Vendar nas vedno ne zanima samo povprečni čas rešitve naloge, zanima nas tudi, v koliko odstotkih nam je uspelo dokončati nalogo v zastavljenem roku. To lahko izvemo iz diagramov, ki prikazujejo odstopanje izdelavnega časa nalog od povprečnega potrebnega časa izdelave. Tako lahko vidimo, koliko nalog je čisto malo zamudilo rok in koliko jih je drastično odstopalo od pričakovanj. Tako lahko analiziramo naloge, ki odstopajo od ustaljenega ritma in ugotovimo, kaj smo naredili narobe in kako lahko to v prihodnje

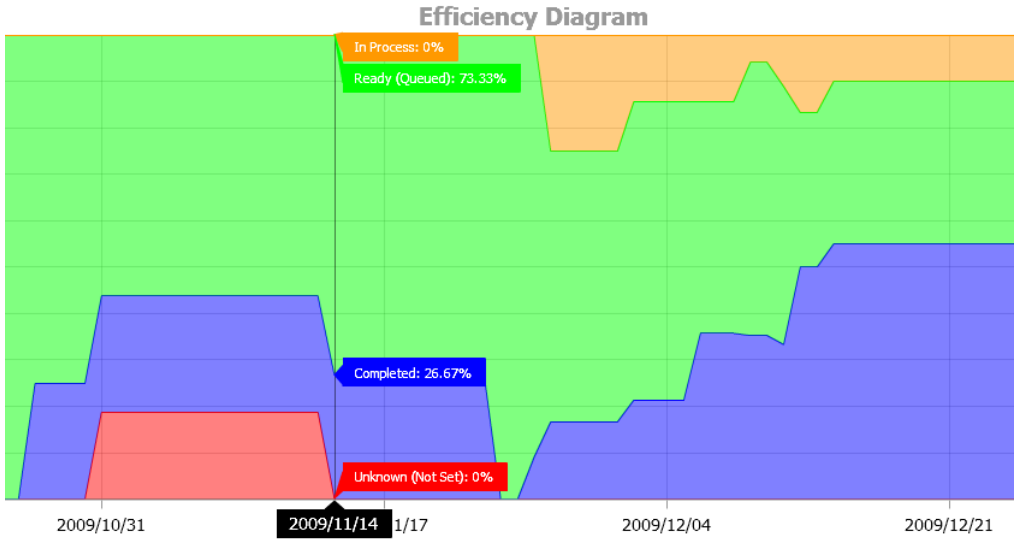
izboljšamo. Primer najdemo na sliki 2.5.



Slika 2.5: Primer diagrama odstopanja od povprečnega potrebnega časa [6]

Za naloge s fiksnim rokom uporabljamo metriko, ki primerja oceno časa, ki bi ga porabili za nalogo, in dejansko količino časa, ki smo ga porabili za izdelavo. S tem ugotavljamo, kako dobro je naša ekipa sposobna podajati ocene za zahtevnost nalog. Če smo dokaj nenatančni, potem moramo z delom na določeni nalogi pričeti prej, kot bi to želeli, kar pa ni optimalno.

Pretočnost je pri Kanbanu še ena metrika, ki pove, kako učinkovit je naš sistem in kako dobro stremimo k stalnemu napredku. Poročamo jo v številu nalog ali zgodb, ki ga obdelamo v določeni enoti časa. Poleg pretočnosti je pomembna tudi analiza ovir v sistemu, ki pove, kako dobro je podjetje pri odpravljanju ovir, ki nastopijo tekom razvoja. To veliko pove o sposobnosti organizacije podjetja. Analizo najlažje prikažemo na kumulativnem diagramu, kjer prikažemo, koliko ovir je v določenem trenutku v sistemu in koliko nalog je zaradi tega oviranih. Učinkovitost našega sistema lahko lepo prikažemo tudi na diagramu učinkovitosti, kjer je na časovni osi prikazano, kakšni so deleži nalog v obdelavi in tistih, ki nanjo čakajo. Iz njega lahko sklepamo o pretočnosti sistema. Primer diagrama je prikazan na sliki 2.6.



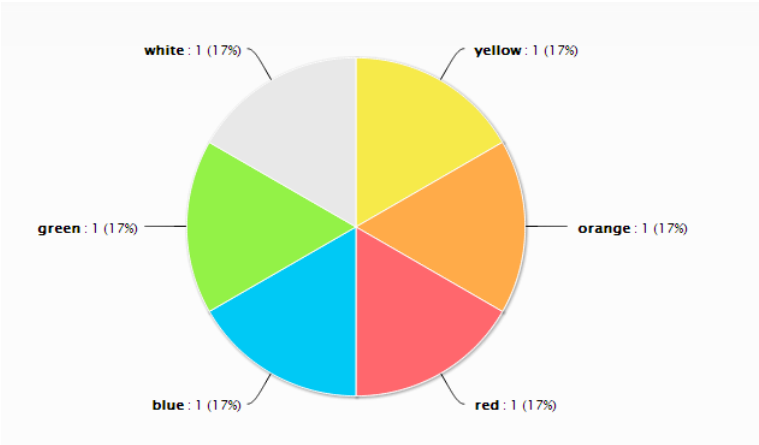
Slika 2.6: Primer diagrama učinkovitosti [7]

Pametno je tudi analizirati, koliko napak naredimo pri razvoju. Lahko merimo, koliko napak se odkrije pri testu neke zgodbe ali pa, koliko napak uide v produkcijo - to merimo s številom prijavljenih hroščev. Z metriko lahko "izmerimo"kvaliteto našega razvoja.

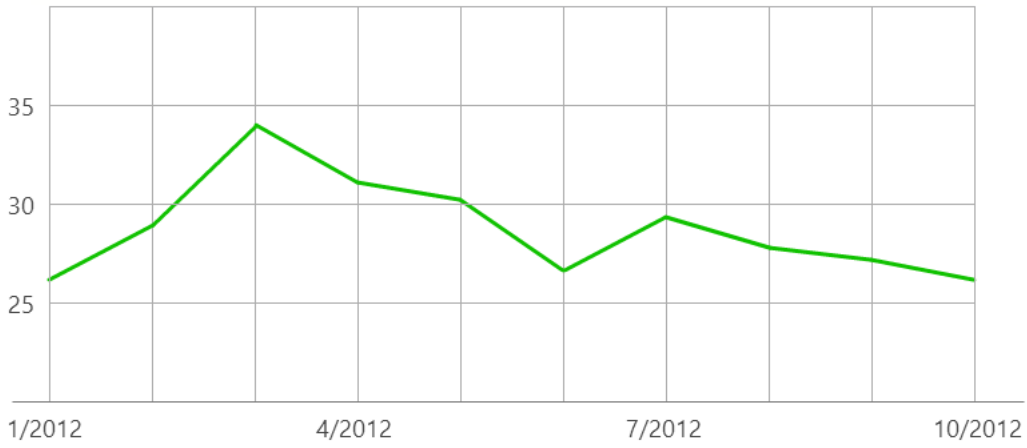
Čeprav nista bistvena za Kanban, je dobro imeti tudi diagram deležev in diagram potrebnega povprečnega časa izdelave. Prvega uporabljamo za lažje razporejanje virov med posamezne tipe nalog ali ravni storitev, saj z njim analiziramo zgradbo našega sistema. Primer je prikazan na sliki 2.7. Pri diagramu potrebnega povprečnega časa gre za bolj jasno sliko, ki jo sicer lahko razberemo iz kumulativnega diagrama. Na časovni osi je prikazano, kako se s časom spreminja povprečni potrebni čas izdelave. Prikazan je na sliki 2.8.

2.3.8 Skaliranje Kanbana

Kanban je mogoče uporabiti tudi na večjih projektih, kjer imamo večjo ekipo in razvijamo neko novo rešitev. Osnove so enake, omejimo količino dela v teku ter vizualiziramo delovni tok, v katerega nato sprejemamo nove naloge



Slika 2.7: Primer diagrama deležev [9]



Slika 2.8: Primer diagrama povprečnega potrebnega časa izdelave

na zahtevo. Dogovorimo se za frekvenco sprejemanja novih nalog v sistem in kako se bomo lotili novih izdaj.

Problem, ki nastane z uporabo Kanbana na večjih projektih, je hierarhija nalog. Naloge so različnih velikosti in velike razbijemo na več manjših zato, da bi lažje določali predviden čas izdelave za posamezno raven storitve. Kako vsemu temu slediti? V praksi se je izkazalo, da je v tem primeru smiselno imeti dvostopenjski pristop, kjer vodimo in vizualiziramo stopnje nalog posebej. Večje naloge vodimo v enem delu table in ko eno potegnemo v razvoj, jo označimo kot v teku razvoja ali kaj podobnega. Za njo nato obdelamo vse pripadajoče manjše naloge in ko so vse končane, tudi večjo označimo kot obdelano. Smiselno je omejiti število nalog na obeh nivojih, saj tako lažje kontroliramo tok skozi sistem.

Pri takem načinu vizualizacije moramo biti previdni, da na tablo ne damo preveč informacij na zapleten način. Pogost način prikaza na tabli je uporaba t.i. "plavalnih stez"ali s tujko "Swim lanes". Vsaka večja naloga, ki predstavlja neko smiselno in zaključeno funkcionalnost, dobi svojo stezo, ki se razteza čez vse obstoječe faze. Znotraj te steze nato prikazujemo in vodimo manjše naloge, ki so del te večje naloge. Po želji lahko za vsako stezo posebej določimo omejitev količine dela v teku.

Težava, ki lahko nastopi pri večjih projektih, so tudi deljeni viri. Tu govorimo o virih, ki si jih deli več ekip hkrati, in so pogosto vzrok za ovire pri delovnem toku. Take naloge, ki so ovirane zaradi nerazpoložljivega deljenega vira, je potrebno posebej označiti. S tem jasno izpostavimo nadrejenim, da se v deljenem viru lahko skriva ozko grlo našega sistema. Probleme lahko rešujemo tudi tako, da deljenemu viru dodelimo svoj Kanban sistem, s pomočjo katerega ta nato izvaja svoje delo.

2.3.9 Pregled delovanja

Pregled delovanja (angl. operations review) se izvaja periodično. Na njem naj bi se zbrali vsi udeleženi v našem procesu, da bi v nekaj minutah predstavili ključne metrike in probleme, do katerih je prišlo v času od zadnjega

pregleda. Predstavitev mora biti osredotočena na podatke in čim bolj transparentna, tako da se vsi zavedajo, kako in na kak način sistem deluje. Komentarji in predlogi so dobrodošli in spodbujajo napredek v organizaciji. S tem, ko na sestanek povabimo tako vodstvo kot tudi navadne zaposlene, dosežemo, da cenijo delo drug drugega in da dobijo vpogled v potek dela.

Pri sestankih moramo biti pozorni na frekvenco - če so prepogosti postanejo potratni z viri, če so preredki, pa izgubijo svoj smisel. V literaturi pogosto zasledimo mesec dni kot priporočljivo obdobje pregleda.

2.4 Optimizacija in izboljšave

2.4.1 Ozka grla in omejeno razpoložljivi viri

O pojavu ozkega grla govorimo, ko je pretočnost celotnega sistema omejena s pretočnostjo ene same komponente tega sistema. To komponento zato tudi imenujemo ozko grlo [3]. Primer ozkega grla v procesu razvoja programske opreme bi lahko bilo testiranje, ki zaradi velike količine nalog dela s 100 % svoje zmogljivosti in je tako pretočnost nalog skozi sistem omejena s pretočnostjo nalog skozi omenjeno fazo razvoja. Zaradi tega ostale faze stojijo, ker čakajo na fazo testa. Ozka grla lahko odpravimo na več načinov - eden izmed njih je povečanje virov. Vir povečamo v primeru, ko je obstoječ vir sto odstotno zaseden in iz njega ne moremo dobiti dodatnega pretoka. V tem primeru vir nadgradimo in tako povečamo pretok. Še en način bi bil, da vir razbremenimo dela, ki nima dodane vrednosti, kot so birokratske in logistične zadeve. Te nato preložimo na vire, ki niso polno zasedeni.

Kadar vir ni popolnoma izkoriščen, je smiselno, da ga pred nadgrajevanjem bolje izkoristimo. Če pride do situacije, ko je vir oviran in ne more nadaljevati z delom, je nujno, da to oviro čim prej odstranimo. To dosežemo z osveščanjem ostalih, da v ozkem grlu ne sme priti do ovir oziroma jih moramo rešiti kar se da hitro. Tu lahko veliko naredimo z dobro organizacijo dela.

Ozko grlo ne sme nikoli ostati brez dela. V ta namen uporabljamo čakalne

vrste, ki jih postavimo pred vir, ki predstavlja ozko grlo, tako da v primeru zamud predhodno v procesu le-ta ne ostane brez dela. Popravke, s katerimi povečamo izkoristek ozkega grla, pogosto izvajamo na drugih delih sistema in ne na samem ozkem grlu.

Omejeno razpoložljivi viri so na pogled zelo podobni ozkim grlom. Gre za vire, ki so na voljo le omejeno količino časa. Če so takrat, ko so na voljo, polno zasedeni in se čakalna vrsta zanje povečuje, lahko rečemo, da je ta omejeno razpoložljiv vir postal ozko grlo. V primeru IT podjetja bi tak vir lahko bil arhitekt programske opreme, ki bi deloval na več projektih hkrati. Ko bi delal na enem projektu, bi bil za ostale nedosegljiv. Če bi večino časa vsi čakali na arhitekta, bi ta postal tudi ozko grlo sistema.

Z omejeno razpoložljivimi viri se soočamo tako, da povečamo čas, ko so na voljo, ali pa jih poizkusimo spremeniti v stalno razpoložljive vire.

2.4.2 Potratne dejavnosti

Med potratne dejavnosti uvrščamo tiste, ki ne prispevajo dodane vrednosti. To so npr. sestanki za planiranje, sestanki za usklajevanje dela, ocene časa potrebnega za razvoj, postavitev testnih okolij ipd. Med potratne dejavnosti spadajo tudi napake ob razvoju, ki so bile vrnjene v popravo. So dejavnosti, ki bi se jim lahko ob primerni kakovosti razvoja izognili.

2.4.3 Viri spremenljivosti

Viri spremenljivosti (angl. sources of variability) v naš sistem vnašajo negotovost in odstopanja od povprečnih potrebnih časov izdelave. Zaradi teh virov časi obdelav posameznih nalog odstopajo od povprečnega potrebnega časa, tega pa ne želimo, saj s tem zmanjšujemo zaupanje udeležencev v predvidljivo delovanje našega sistema. V literaturi se viri spremenljivosti delijo na notranje in zunanje.

- **Notranji viri** izvirajo iz našega sistema in jih lahko spreminjamo in upravljamo s pomočjo poslovnih pravil in orodij za razvoj.

- **Zunanji viri** izvirajo izven našega sistema in sami nimamo vpliva na njih. To so lahko nezgode pri dobavitelju ali stranki, razne naravne katastrofe, ki lahko poškodujejo razvojno opremo, izpadi elektrike ipd. Ne moremo jih spreminjati, lahko pa definiramo postopke, kako se z njimi spopasti.

2.4.4 Upravljanje s problemi in pravila stopnjevanja

V sistemu potrebujemo učinkovit sistem sledenja problemom in postopke, kako jih reševati. Ni dovolj samo to, da vemo, da problem obstaja. Znati ga moramo tudi rešiti kar se da hitro, da bi ohranili tok skozi sistem. V zrelejših organizacijah se zaposleni sami organizirajo in skupaj rešujejo probleme, v manj zrelih je za to potreben nadrejeni, ki za rešitev problema zadolži zaposlene.

Nastane lahko situacija, ko se delo na nalogi ustavi in je za njegovo nadaljevanje potrebno mnenje ali pomoč nadrejenih. V ta namen morajo biti vnaprej jasno določeni postopki in pravila, kako za razrešitev ovire prositi nadrejene. Nadrejeni se morajo teh postopkov zavedati in biti pripravljeni odpraviti nastalo težavo.

Poglavje 3

Orodja

3.1 Kriteriji analize orodij

Analizo posameznega orodja smo razbili v šest delov. V prvem delu smo na kratko opisali podjetje, ki je aplikacijo razvilo, našteli njegove reference ter opredelili, ali gre za brezplačen ali plačljiv program.

Temu splošnemu opisu sledi razdelek, kjer analiziramo sposobnosti orodja pri vizualizaciji delovnega toka. Osredotočili smo se na to, kako fleksibilno je pri modeliranju table s karticami in kaj vse nam omogoča.

Naslednja točka analize je bila, kako dobro orodje podpira omejitev dela v teku ter kako se sooča z morebitno prekoračitvijo te omejitve. Tudi tu smo upoštevali fleksibilnost pri postavljanju omejitev na stanja ter njegova podstanja in plavalne steze.

V četrtem delu smo preverili, če orodje pozna koncept tipov nalog in ravni storitev. V kolikor omenjena koncepta podpira, smo testirali tudi, kako dobro zna prikazati različne tipe nalog in ravni storitev na virtualni tabli. Zanimala nas je tudi svoboda, ki jo ima uporabnik pri določanju novih tipov nalog in ravni storitev.

Sledi analiza poročil in metrik, ki jih je orodje sposobno izračunati in prikazati. Vključili smo seznam uporabnih poročil ter komentirali kvaliteto implementacije.

V zadnjem delu smo ocenili, kako težavna je namestitev aplikacije in kako prijetna je uporabniška izkušnja. Prav tako smo preverili, ali je nastavitev aplikacije za pričetek uporabe zahtevna ali ne, in ali je potrebno uporabnike dodatno uvajati.

3.2 Zakaj ravno ta orodja?

Orodja smo izbirali iz seznama orodij, ki so bila v grobem analizirana v članku raziskovalcev iz Univerze v Cagliariju [4]. Iz tega seznama smo kot prvega izbrali orodje podjetja VersionOne zaradi kompleksnosti in velikega deleža na tržišču. Naslednjega, RadTrack, smo izbrali, ker gre za odprtokodno rešitev, LeanKit in KanbanTool pa sta se nam zdela zanimiva, ker se osredotočata na podporo metodologiji Kanban in sta na voljo v časovno omejeni različici. Poleg vsega naštetega so orodja lahko dostopna posamezniku.

3.3 VersionOne

URL: http://www.versionone.com/

VersionOne je spletna aplikacija za podporo agilnim metodologijam, velik del je namenjen metodologiji Scrum. Že od leta 2002 jo razvija istoimensko podjetje, ki ima sedež v ZDA. Orodje je eno najbolj uporabljenih med uporabniki agilnih metodologij. Na dolgem seznamu referenc zasledimo podjetja, kot so Oracle, SAP, Siemens, Sony in druga. Osnovna različica orodja je na voljo brezplačno za omejeno število uporabnikov, medtem ko so vse nadgradnje plačljive v obliki licenc.

- **Vizualizacije delovnega toka** - orodje omogoča vizualizacijo v obliki virtualne table. Sama predstavitev je za uporabo v Kanbanu skopa. Posamezne stolpce implicitno določimo z definiranjem statusov nalog v delovnem toku (na voljo imamo tudi druge možnosti, ne le status, a so za potrebe Kanbana neprimerne). Tudi steza ne moremo definirati

poljubno. Če jih vključimo, loči naloge po stezah le glede na določen atribut po stezah - naloge z različno vrednostjo tega atributa ne morejo biti v isti stezi. Deljenje stolpcev na podstanja ni možno. Na tabli lahko uporabljamo filtre, s katerimi lahko prikažemo samo tiste naloge, ki nas zanimajo v določenem trenutku.

- **Omejitev dela v teku** - na posamezna stanja lahko dodajamo omejitve tako kot tudi na skupine stanj. Nastavitev je enostavna v primerjavi z ostalimi nastavitvami aplikacije. Ko omejitev določenega stanja presežemo, se na virtualni tabli stolpec obarva z rdečo barvo, s čimer nas opozarja na nepravilnost. Pogrešamo dokumentiranje razloga za prekoračitev omejitve.

- **Tipi nalog in ravni storitve** - tipe nalog lahko poljubno definiramo. Ravni storitve aplikacija sama ne pozna, lahko pa jih vnesemo kot poljubno polje za posamezni tip naloge.

- **Metrike in poročila** - orodje nudi več kot 50 različnih vrst poročil, ki se uporabljajo pri agilnem razvoju programske opreme. Na žalost za Kanban manjka kar nekaj poročil, ki jih najdemo pri ostalih aplikacijah tega tipa, saj se VersionOne osredotoča na podporo metodologiji Scrum. Orodje podpira kumulativni diagram delovnega toka ter diagram povprečnega časa izdelave.

- **Namestitev in uporaba** - aplikacija je na voljo kot gostovana pri proizvajalcu ali kot samostojna programska oprema, ki jo lahko prenesemo in namestimo v našem okolju. Zasebne namestitve žal nismo uspeli narediti, saj je na voljo samo pri plačljivih različicah. V gostovanem okolju nam proizvajalec dodeli dostop do aplikacije, od koder imamo proste roke pri nastavitvah aplikacije, ki pa so kar kompleksne. Za urejanje nastavitev je potrebno prebrati kar nekaj navodil in člankov v proizvajalčevi bazi pomoči. Ko je aplikacija enkrat nameščena in nastavljena, je njena uporaba na začetku kar zapletena. Za začetek

je priporočljivo uvajanje in izobraževanje uporabnikov, kako pravilno uporabljati programsko opremo. Na začetku se ta zdi nepregledna in zapletena, vendar se pri nadaljnji uporabi izkaže za odlično, saj nudi dobro podporo delu ter veliko različnih poročil in metrik, ki jih lahko nato uporabimo v prid optimizaciji procesov.

3.4 RadTrack

URL: http://www.radtrack.com/

RadTrack je spletna aplikacija za podporo Kanbanu. Je zelo osnovna aplikacija, nabor funkcionalnosti je zelo skop. Gre za odprtokodno rešitev, ki je trenutno še v BETA fazi in je brezplačna. Ker je zelo osnovna, je neprimerna za večje in resne projekte.

- **Vizualizacija delovnega toka** - vizualizacija je zelo osnovna, stanja lahko prikažemo le na enem nivoju. Ni možnosti dodajanja podstanj kot tudi ne t. i. plavalnih stez. Sam izgled uporabniškega vmesnika je za časom, vendar je vseeno pregleden.

- **Omejitev dela v teku** - na stanja lahko postavimo omejitev količine nalog. Ob dosegu te omejitve je iz vmesnika jasno razvidno, da je do tega prišlo. Pogrešamo eksplicitno opozorilo in beleženje razloga za prekoračitev omejitve.

- **Tipi nalog in ravni storitve** - Tipi nalog so vnaprej določeni in jih ne moremo spreminjati. Vizualno so določeni z majhno ikono, ki je slabo vidna, tako da tipe nalog med seboj zelo težko razločimo. Ravni storitve aplikacija ne pozna. Pogrešamo uporabo barv za klasifikacijo tipa nalog ali morebitnih ravni storitev, če bi bile te implementirane.

- **Metrike in poročila** - metrik in poročil aplikacija v tej fazi implementacije nima. Vsekakor bi jih želeli imeti.

- **Namestitev in uporaba** - namestitev ni potrebna, ker aplikacija teče na strežniku proizvajalca. Registriramo se na spletni strani, s tem pa si omogočimo dostop do aplikacije. Nastavitev ni težka, ker je orodje zelo osnovno. Sama uporabniška izkušnja je slaba zaradi zastarelega in slabo preglednega uporabniškega vmesnika.

3.5 LeanKit

URL: http://leankit.com/

LeanKit je produkt majhnega startup podjetja iz ZDA. Med njihovimi referencami lahko najdemo podjetja kot so Adobe, Rolls Royce, NBC, GlaxoSmithKline in druga. Aplikacija deluje na spletu, tako da je dostopna preko spletnih brskalnikov. Osnovna različica je na voljo brezplačno, vse nadgradnje, ki omogočajo napredne metrike in poročila pa so plačljive v obliki letnih uporabniških licenc.

- **Vizualizacija delovnega toka** - orodje omogoča vizualizacijo delovnega toka, tako kot bi to naredili na fizični tabli. Dovoljuje dodajanje in odstranjevanje stolpcev za stanja ter deljenje stolpcev za namen dodajanja podstanj. S pomočjo enostavnih filtrov lahko omejimo, kaj se prikaže na tabli, in tako izpostavimo naloge, ki nas zanimajo. Vse skupaj je zapakirano v tekoče delujočo spletno aplikacijo z intuitivnim uporabniškim vmesnikom.

- **Omejitev dela v teku** - vsakemu stanju lahko omejimo količino nalog znotraj njega. V kolikor želimo to omejitev prekoračiti, nas aplikacija na to opozori in zahteva od nas, da argumentiramo naše dejanja, ta pa se nato zapiše v dnevnik dogodkov. Tako lahko ostali udeleženci vedo, zakaj je do take odločitve prišlo.

- **Tipi nalog in ravni storitve** - aplikacija omogoča ustvarjanje poljubnih tipov nalog kot tudi ravni storitve. Te so nato na tabli prikazane s

pomočjo različnih barv in ikon, ki jih prav tako določimo sami.

- **Metrike in poročila**
  - **Diagram kumulativnega toka**
  - **Diagram povprečnega cikla obdelave** - iz tega diagrama je razvidno, koliko časa v povprečju porabimo za obdelavo ene naloge določenega tipa ali prioritete. S tem si pomagamo pri določanju ravni storitve.
  - **Diagram porazdeljenosti nalog** - iz diagrama ugotavljamo, kolikšen odstotek dela je opravil nek uporabnik ali kolikšen odstotek vseh nalog predstavljajo tiste z visoko prioriteto.
  - **Diagram učinkovitosti** - s pomočjo tega diagrama lahko ugotovimo, koliko časa dejansko porabimo za delo na neki nalogi, in koliko časa naloga čaka v vrsti za obdelavo.
  - **Diagram odstopanja od povprečnega cikla obdelave** - tu lahko analiziramo, katere naloge odstopajo od povprečnega časa izdelave in se poizkusimo iz tega kaj naučiti in nato to znanje uporabiti za naprej, da bi izboljšali povprečen čas izdelave.

- **Namestitev in uporaba** - nameščanje aplikacije ni potrebno, izvaja se na strežniku pri proizvajalcu. Potrebno se je registrirati na njihovi spletni strani in vnesti potrebne podatke, oni pa nam nato dodelijo spletni naslov, preko katerega dostopamo do aplikacije. Administracija uporabnikov in tabel je urejena preko centralne nadzorne plošče. Je dokaj enostavna, saj so nastavitve dokaj osnovne, nobene niso posebno napredne ali zapletene. Uporaba aplikacije je enostavna in ne zahteva pretiranega učenja ali uvajanja. Glavni ukazi ter pomoč so vedno na voljo, bližnjice so postavljene na primernih mestih.

3.6 Kanban Tool

URL: http://kanbantool.com/

Kanban Tool je spletna aplikacija namenjena izključno podpori Kanbana. Izdelalo jo je podjetje Shore Labs iz Poljske. Na voljo je brezplačno za 2 uporabnika, nato pa se licencira glede na število uporabnikov. Med referencami zasledimo uspešna podjetja, kot so Disney, Rovio, Skyscanner, Pay Global itd.

- **Vizualizacija delovnega toka** - od vseh orodij je to najbolj fleksibilno pri vizualizaciji delovnega toka. Omogoča skoraj vse, kar bi lahko naredili na fizični tabli - stanja, podstanja, plavalne steze. Uporabniški vmesnik je privlačen in uporaben, enostaven za uporabo.

- **Omejitev dela v teku** - orodje omogoča omejitev dela v teku. V primeru, da želimo omejitev prekoračiti, nas orodje na to opozori in od nas zahteva, da dokumentiramo svojo odločitev. Ta se zabeleži v dnevnik dogodkov za potrebe revizije.

- **Tipi nalog in ravni storitve** - v aplikaciji lahko nastavimo poljubno število tipov nalog, ki se razlikujejo po barvi kartice na virtualni tabli. Žal orodje ne pozna ravni storitev, pozna le prioriteto naloge, kar pa vendarle ni enako. Lahko definiramo dodatna polja po želji, vendar ta polja nimajo posebne označbe na tabli. Želeli bi, da bi bile ravni storitve implementirane, saj so ena ključnih sestavin Kanbana.

- **Metrike in poročila**
  - **Kumulativni diagram toka**
  - **Diagram deležev** - tu lahko analiziramo sestavo nabora naših nalog, kolikšen je odstotek določenega tipa naloge, prioritet, zadolžencev itd. Z uporabo lahko izluščimo veliko uporabnih informacij, ki nam pripomorejo k stalnemu napredku.

  - **Diagram povprečnega časa izdelave** - z njim si pomagamo pri ponujanju ravni storitev, ocenjevanju potrebnega dela.

- **Namestitev in uporaba** - sama namestitev aplikacije ni potrebna. Gostuje pri proizvajalcu, vse, kar je potrebno, je registracija na njihovi spletni strani in že lahko začnemo. Dodajanje uporabnikov je enostavno, kot tudi izdelava table in opis delovnega toka. Tablo lahko osnujemo na obstoječih primerih in urejamo skozi uporabniku prijazen čarovnik. Uporabniška izkušnja je prijetna, gumbi naredijo to, kar se od njih pričakuje na prvi pogled, vse je intuitivno. Vmesnik je minimalističen in pregleden.

3.7 Primerjava orodij

V tabeli 3.1 so našteta orodja ocenjena po posamezni kategoriji.

Orodje	Vizualizacija delovnega toka	Omejitev dela v teku	Tipi nalog in ravni storitve	Metrike in poročila	Namestitev in uporaba
VersionOne	•	••	••	•	•
RadTrack	•	•	•		•
LeanKit	•••	•••	•••	•••	•••
Kanban Tool	•••	•••	••	••	•••

Tabela 3.1: Primerjava orodij

Legenda:

(brez pik) - ne podpira te funkcionalnosti

• - osnovna podpora

•• - dobra podpora

••• - odlična podpora

Poglavje 4

Funkcionalne specifikacije popolnega orodja za Kanban

4.1 Glavne karakteristike orodja

Orodje vključuje vse večje funkcije, ki bi jih popolno orodje za Kanban moralo imeti. Glavna funkcija je zagotovo virtualna tabla s karticami. Podpira stolpce za stanja ter podstanja, steze in postavljanje omejitev količine dela v teku. Kartice lahko ustvarjamo, brišemo, prestavljamo in označujemo po želji. Velik poudarek dajemo svobodi uporabnika pri načrtovanju in oblikovanju kartic, saj ta omogoča natančnejše modeliranje delovnega toka. Posebno poglavje specifikacije orodja je namenjeno uporabniškim pravicam. Operacije nad karticami lahko omejimo z uporabniškimi pravicami. Definiramo lahko razvojne ekipe za posamezno tablo, ji določimo pravice ter vanjo vključimo uporabnike. Izpustili nismo niti metrik in poročil - orodje podpira kumulativni diagram, diagram povprečnega potrebnega časa izdelave, diagram deležev kartic, diagram odstopanja od potrebnega časa izdelave in diagram učinkovitosti. Dogodki v aplikaciji se beležijo v dnevnik dogodkov.

V aplikaciji poznamo naslednje uporabniške vloge:

- **Naročnik** - lahko ustvarja kartice in jih postavi na vhod razvojnega procesa.

- **Razvijalec** - lahko ustvarja in premika kartice, komentira ...
- **Upravljalec** - nadgradnja vloge "Razvijalec", odgovoren za kreiranje in administracijo tabel.
- **Adminstrator** - nadgradnja vloge "Upravljalec", odgovoren za kreiranje uporabnikov, dodeljevanje pravic in administracijo aplikacije.

4.2 Virtualna tabla

- Upravljalec lahko ustvari novo prazno tablo.
  - Preveri, da vlogi razvijalec in naročnik nimata dostopa do te funkcije.
- Upravljalec lahko v določeno tablo dodaja stolpce (na poljubno specificirano mesto).
  - Preveri, da uporabnik lahko definira, za kateri tip stolpca gre - vrsta, obdelava ali končano.
  - Preveri, da uporabnik lahko doda stolpec na prvo, zadnje in poljubno vmesno mesto.
  - Preveri, da uporabnik lahko postavi omejitev količine dela v teku na stolpec.
  - Preveri, da uporabnik ne more vnesti negativne omejitve ali necelega števila.
  - Preveri, da vlogi razvijalec in naročnik nimata dostopa do te funkcije.
- Upravljalec lahko stolpec razdeli na več podstolpcev.
  - Preveri, da uporabnik lahko razdeli poljuben stolpec.
  - Preveri, da uporabnik lahko določi omejitev na podstolpec.

  - Preveri, da uporabnik ne more določiti take omejitve na podstolpec, da bi seštevek podstolpcev presegal omejitev nadrejenega stolpca.
  - Preveri, da vlogi razvijalec in naročnik nimata dostopa do te funkcije.

- Upravljalec lahko na tabli ustvari plavalno stezo.
  - Preveri, da vlogi razvijalec in naročnik nimata dostopa do te funkcije.
  - Preveri, da aplikacija ob kreiranju prve plavalne steze premakne obstoječe kartice v prvo stezo.

- Upravljalec lahko premika stolpce.
  - Preveri, da uporabnik lahko premakne stolpec na poljubno mesto.
  - Preveri, da uporabnik ne more prestaviti stolpca, ki ima v sebi kartice.

- Upravljalec lahko premika plavalne steze.
  - Preveri, da uporabnik lahko premakne stezo na poljubno mesto.

4.3 Kartice

- Razvijalec/naročnik lahko ustvari kartico.
  - Preveri, da razvijalec/naročnik lahko ustvari novo kartico v skladu s svojimi pravicami.
  - Preveri, da razvijalec/naročnik lahko določi tip naloge.
  - Preveri, da razvijalec/naročnik lahko določi raven storitve nalogi.

- Razvijalec/naročnik lahko spreminja kartico.

  - Preveri, da razvijalec/naročnik lahko spreminja kartico v skladu s svojimi pravicami.

- Uporabnik lahko prestavlja kartice znotraj razvojnega procesa v skladu s svojimi pravicami.
  - Preveri, da uporabnik ne more prestaviti kartice v stanje, za katerega nima pravic dodajanja.
  - Preveri, da aplikacija zazna prekoračitev omejitve količine dela v teku.
  - Preveri, da aplikacija ob prekoračitvi omejitve od uporabnika zahteva pisno obrazložitev.
  - Preveri, da aplikacija brez obrazložitve ne dovoli prekoračitev omejitve.
  - Preveri, da je dokumentirana prekoračitev ustrezno zabeležena v dnevniku dogodkov.

- Upravljalec lahko doda poljubno polje na kartico, ki se bo nato uporabljalo na karticah.
  - Preveri, da upravljalec lahko doda polje in mu določi diskretno zalogo vrednosti (uporabnik na kartici izbira vrednosti iz spustnega polja).
  - Preveri, da upravljalec lahko doda polje, ki sprejema številsko vrednost in mu določi interval sprejemljivih vrednosti.
  - Preveri, da upravljalec lahko doda polje, ki sprejema poljuben vnos besedila.

- Upravljalec lahko barvo posamezne kartice poveže s poljubnim poljem na kartici (ki ima omejeno diskretno zalogo vrednosti).
  - Preveri, da upravljalec ne more povezati barve kartice s poljem, ki nima diskretne zaloge vrednosti.

  - Preveri, da mora upravljalec za vsako vrednost polja definirati barvo kartice.

- Upravljalec lahko poveže ikono posamezne kartice s poljubnim poljem na kartici (ki ima omejeno diskretno zalogo vrednosti).
  - Preveri, da upravljalec ne more povezati ikone s poljem, ki nima diskretne zaloge vrednosti.
  - Preveri, da mora upravljalec za vsako vrednost polja definirati ikono.

- Upravljalec lahko določi, katere lastnosti posamezne naloge bodo prikazane na kartici (omejena količina).
  - Preveri, da upravljalec ne more prekoračiti meje štirih polj na kartico.

- Razvijalec/naročnik lahko pregleduje podrobnosti kartice.
  - Preveri, da se na pregledu podrobnosti kartice prikažejo vsa razpoložljiva polja.

4.4 Administracija

- Administrator lahko ustvari uporabnika.
  - Preveri, da uporabnika lahko ustvari le administrator.
  - Preveri, da administrator ne more vnesti 2 uporabnikov z istim e-poštnim naslovom.

- Administrator lahko spreminja lastnosti uporabnika.
  - Preveri, da uporabnika lahko ureja le administrator.
  - Preveri, da administrator lahko ponastavi geslo uporabniku.
  - Preveri, da administrator ne vidi gesla uporabnikov.

- Administrator lahko dodeljuje pravice uporabnikom.
  - Preveri, da sistem opozori administratorja, ko nekomu dodeli vlogo administratorja.
  - Preveri, da administrator ne more odvzeti pravic administratorja samemu sebi.
- Upravljalec lahko ustvarja uporabniške skupine za posamezno tablo.
  - Preveri, da upravljalec lahko ustvari uporabniško skupino za poljubno tablo.
  - Preveri, da upravljalec lahko ureja pravice uporabniške skupine.
- Upravljalec lahko za vsak stolpec definira, katera uporabniška skupina lahko dodaja in odvzema kartice iz njega.
  - Preveri, da upravljalec lahko uporabniški skupini dodeli samo pravico dodajanja ali samo odvzemanja.
  - Preveri, da upravljalec lahko uporabniški skupini dodeli pravici dodajanja in odvzemanja.
- Upravljalec lahko dodaja uporabnike v uporabniške skupine za posamezno tablo.
  - Preveri, da upravljalec ne more onemogočiti dostopa do table administratorju.
  - Preveri, da upravljalec ne more onemogočiti dostopa do table samemu sebi.

4.5 Poročila in metrike

- Razvijalec lahko izpiše kumulativni diagram delovnega toka.
  - Preveri, da aplikacija izračuna in prikaže diagram.

  - Preveri, da aplikacija izračuna in prikaže diagram za prvi dan uporabe table.
  - Preveri, da aplikacija pravilno izračuna in prikaže diagram za prvi in vse naslednje dni uporabe table.
  - Preveri, da aplikacija pri izračunu diagrama za kartico na posamezen dan upošteva le zadnji stolpec, v katerem je bila na ta dan.

- Razvijalec lahko izpiše diagram povprečnega časa izdelave.
  - Preveri, da aplikacija izračuna in prikaže diagram.
  - Preveri, da aplikacija ne prikaže diagrama, v kolikor še ni bilo kartice, ki bi opravila pot skozi sistem.
  - Preveri, da aplikacija pravilno izračuna in prikaže diagram.

- Razvijalec lahko izpiše diagram deležev kartic.
  - Preveri, da aplikacija izračuna in prikaže diagram.
  - Preveri, da aplikacija pravilno izračuna in prikaže deleže kartic glede na izbran delilnik.
  - Preveri, da aplikacija pravilno izračuna in prikaže deleže kartic, ko je v sistemu le ena kartica.

- Razvijalec lahko definira, po čem želi deliti kartice na diagramu deležev.
  - Preveri, da razvijalec lahko deli kartice glede na tip naloge.
  - Preveri, da razvijalec lahko deli kartice glede na raven storitve.
  - Preveri, da razvijalec lahko deli kartice glede na prioriteto naloge.

- Razvijalec lahko izpiše diagram odstopanja od povprečnega časa izdelave.
  - Preveri, da aplikacija izračuna in prikaže diagram.

  - Preveri, da razvijalec lahko pregleduje podrobnosti katerekoli kartice, ki je prikazana na diagramu.
  - Preveri, da aplikacija izračuna povprečni čas izdelave enako kot pri diagramu povprečnega časa izdelave.

- Razvijalec lahko izpiše diagram učinkovitosti (koliko časa kartica preživi v vrsti, obdelavi in končano).
  - Preveri, da aplikacija izračuna in prikaže diagram.
  - Preveri, da aplikacija pravilno izračuna in prikaže diagram.

- Razvijalec lahko omeji, katere kartice želi zajeti v diagramih (s pomočjo filtra po lastnostih).
  - Preveri, da razvijalec lahko filtrira kartice po polju, ki ima diskretno zalogo vrednosti.
  - Preveri, da razvijalec lahko filtrira kartice po polju, ki nima diskretne zaloge vrednosti.

4.6 Splošne funkcionalnosti orodja

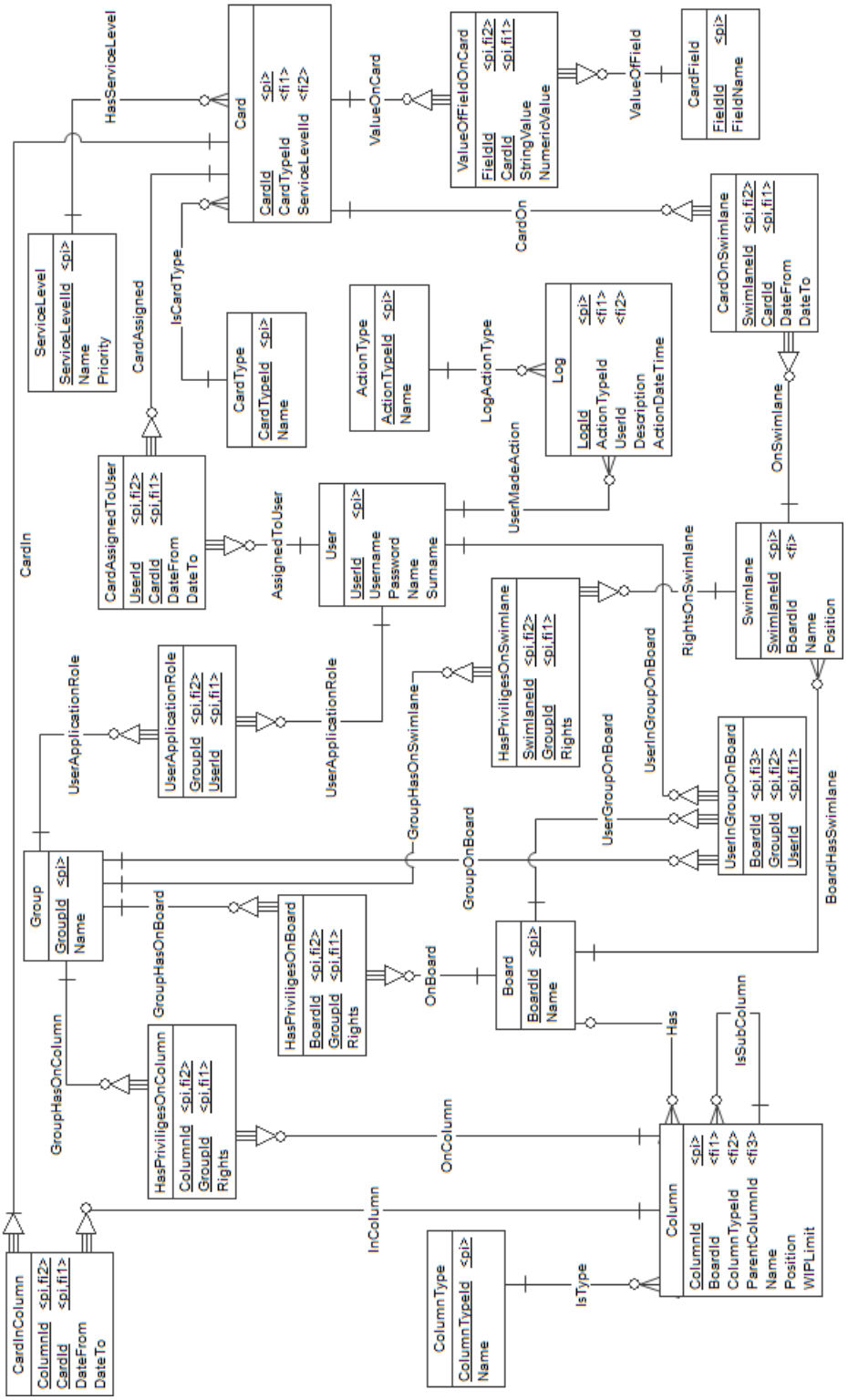
- Aplikacija ob vnosu vrednosti v obrazce preveri pravilnost vnosa.
  - Preveri, da ob izdelavi nove kartice aplikacija preveri obvezna polja.
  - Preveri, da ob vnosu omejitve količine dela na stolpec aplikacija preveri smiselnost vnosa.

- Aplikacija vsako akcijo in dogodek zabeleži.
  - Preveri, da ob izdelavi nove kartice aplikacija zabeleži dogodek.
  - Preveri, da ob definiranju novega stolpca aplikacija zabeleži dogodek.

- Preveri, da aplikacija za dani dogodek zabeleži datum in čas, ko se je ta zgodil.
- Preveri, da aplikacija za dani dogodek zabeleži uporabnika, ki ga je povzročil.

4.7 Podatkovni model

Izdelali smo logični podatkovni model za aplikacijo, ki bi podpiral implementacijo zgoraj navedenih uporabniških zgodb. Na sliki 4.1 je prikazan celotni diagram s tabelami in njihovimi medsebojnimi povezavami. V nadaljevanju smo model razdelili na vsebinsko smiselne sklope ter vsak del podrobneje opisali.

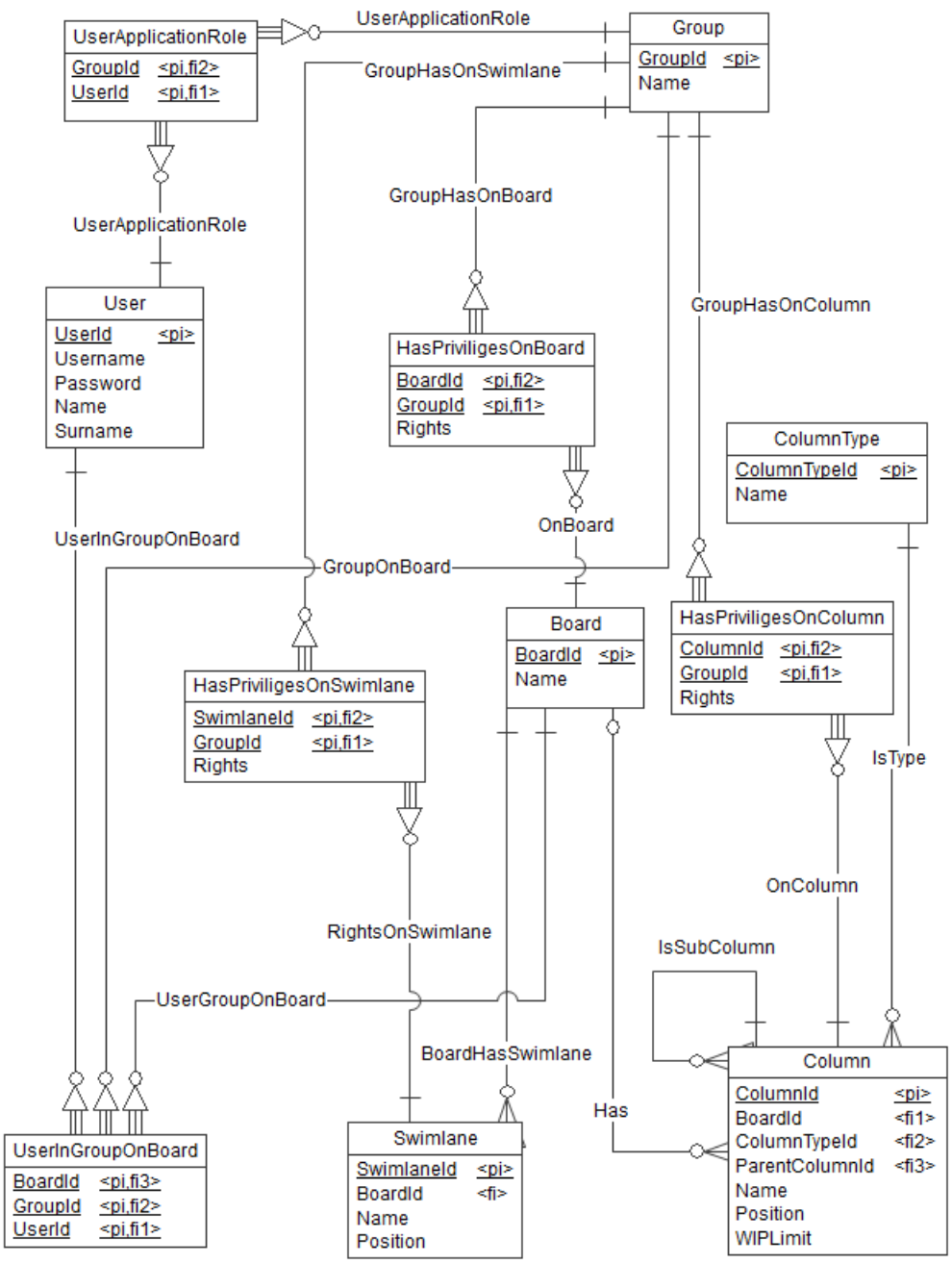


Slika 4.1: Diagram konceptualnega podatkovnega modela

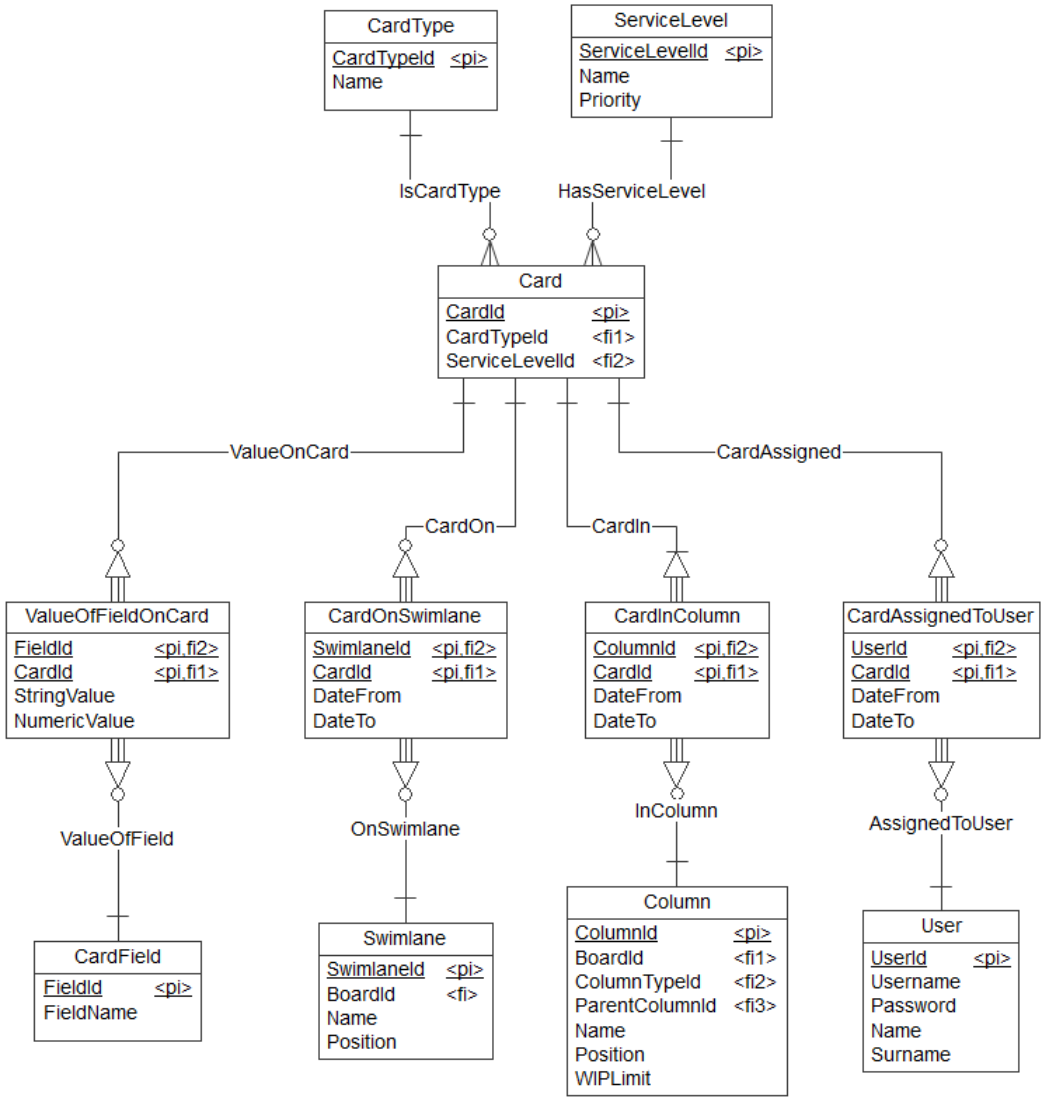
Na izseku logičnega podatkovnega modela na sliki 4.2 so prikazane tabele, ki podpirajo definiranje virtualne table, uporabnikov, uporabniških skupin ter pravic nad tablami. Tabela `Board` hrani informacije o virtualnih tablah. Vsaka tabla ima definirane stolpce in plavalne steze, ki so zapisane v tabelah `Column` in `Swimlane`. Vsak zapis v tabeli `Column` ima definiran tip, ki ga najdemo v tabeli `ColumnType`. Tabela `User` hrani informacije o uporabnikih, te pa nato združujemo v uporabniške skupine, ki so zapisane v tabeli `Group`. Vsak uporabnik ima dva tipa pripadnosti skupinam. Ena od njih je globalna uporabniška skupina (skupina na nivoju celotne aplikacije), ki velja na splošno za uporabnika (npr. administrator aplikacije, upravljalec itd.). Ta podatek razberemo iz tabele `UserApplicationRole`, ki pove, v katerih globalnih uporabniških skupinah je uporabnik. Vsak uporabnik lahko pripada tudi več uporabniškim skupinam za določeno tablo. Informacijo o tem najdemo v tabeli `UserInGroupOnBoard`, ki povezuje entitete `User`, `Group` in `Board`. Uporabniške pravice nad tabelami `Board`, `Column` in `Swimlane` so zapisane v `HasPriviligesOnBoard`, `HasPriviligesOnColumn` in `HasPriviligesOnSwimlane`.

Na sliki 4.3 vidimo izsek logičnega podatkovnega modela, ki prikazuje tabele ter razmerja, ki so povezana s karticami na tabli ter premikanjem le-teh. Tabela `Card` predstavlja posamezno kartico. Povezana je s tabelo `CardType`, ki definira tip kartice, in tabelo `ServiceLevel`, ki definira raven storitve naloge. `CardInColumn`, `CardOnSwimlane` in `CardAssignedToUser` hranijo informacije o poziciji kartice na tabli ter kdo je odgovoren zanjo. Odgovornost določa `CardAssignedToUser`, ki povezuje kartico z uporabnikom v tabeli `User`. `CardInColumn` in `CardOnSwimlane` določata položaj kartice v stolpcih ter plavalnih stezah. Vse tri informacije so časovno spremenljive, saj se kartica premika po tabli. Polja na kartici so definirana v `CardField`, njihove vrednosti na kartici pa v `ValueOfFieldOnCard`.

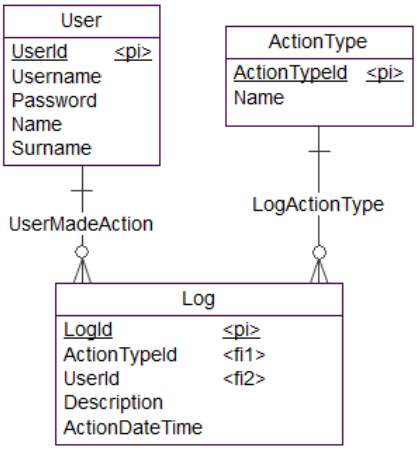
Na sliki 4.4 so prikazane tabele `Log`, `User` in `ActionType` ter razmerja med njimi. Podpirajo shranjevanje dnevnika dogodkov aplikacije. V tabelo `Log` se zapiše čas in opis dogodka, poveže se ga s tipom dogodka, ki je za-



Slika 4.2: Izsek diagrama logičnega podatkovnega modela



Slika 4.3: Izsek diagrama logičnega podatkovnega modela



Slika 4.4: Izsek diagrama logičnega podatkovnega modela

Tabela Board		
Ime atributa	**Podatkovni tip**	**Opis polja**
BoardId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime table.

Tabela 4.1: Opis tabele Board

pisan v `ActionType`. Dogodek se poveže z uporabnikom zapisanim v tabeli `User`, ki je dogodek povzročil. Primer zapisa v dnevnik dogodkov bi bila prekoračitev omejitve dela v teku, kjer aplikacija zabeleži, da je do nje prišlo, ter doda uporabnikovo obrazložitev dogodka. Zapis v `Log` bi bil povezan z zapisom uporabnika v tabeli `User` ter zapisom v `ActionType`, ki opredeljuje tip "Prekoračitev omejitve dela v teku". V polje `Description` bi zapisali uporabnikovo obrazložitev. V tabelah 4.10 in 4.11 so natančneje opisani atributi tabel `Log` in `ActionType`. Tabela `User` je bila podrobneje opisana v prejšnjem poglavju.

Tabela Column		
Ime atributa	**Podatkovni tip**	**Opis polja**
ColumnId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime stolpca.
Position	Integer	Zaporedna številka stolpca na tabli. V kolikor zapis opredeljuje podstolpec, velja zaporedna številka stolpca znotraj nadrejenega stolpca in ne na tabli.
WIPLimit	Integer	Omejitev količine dela v teku, ki velja za stolpec.
BoardId	Integer	Tuji ključ, ki kaže na zapis v `Board`. Pove, na kateri tabli se nahaja stolpec.
ColumnTypeId	Integer	Tuji ključ, ki kaže na zapis v `ColumnType`. Opredeljuje tip stolpca.
ParentColumnId	Integer	Tuji ključ, ki kaže na zapis v `Column`. Polje je napolnjeno, v kolikor gre za podstolpec, polje pa v tem primeru kaže na nadrejeni stolpec.

Tabela 4.2: Opis tabele Column

Tabeka ColumnType		
Ime atributa	**Podatkovni tip**	**Opis polja**
ColumnTypeId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime tipa stolpca.

Tabela 4.3: Opis tabele ColumnType

Tabela Swimlane		
Ime atributa	**Podatkovni tip**	**Opis polja**
SwimlaneId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime plavalne steze.
Position	Integer	Zaporedna številka plavalne steze na tabli.
BoardId	Integer	Tuji ključ, ki kaže na zapis v `Board`. Pove, na kateri tabli se nahaja plavalna steza.

Tabela 4.4: Opis tabele Swimlane

Tabela User		
Ime atributa	**Podatkovni tip**	**Opis polja**
UserId	Integer	Primarni ključ tabele.
Username	Variable characters	Uporabniško ime, ki je enako e-poštnemu naslovu uporabnika.
Password	Variable characters	Uporabnikovo geslo.
Name	Variable characters	Ime uporabnika.
Surname	Variable characters	Priimek uporabnika.

Tabela 4.5: Opis tabele User

Tabela Group		
Ime atributa	**Podatkovni tip**	**Opis polja**
GroupId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime uporabniške skupine.

Tabela 4.6: Opis tabele Group

Tabela HasPriviligesOnColumn		
Ime atributa	**Podatkovni tip**	**Opis polja**
Rights	Integer	Številka, ki predstavlja pravice. Vrednost 1 je izvzemanje kartice iz stolpca, 2 je dodajanje v stolpec. S seštevanjem številk kombiniramo pravice.
ColumnId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis v tabeli `Column`, za katerega veljajo pravice.
GroupId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis v tabeli `Group`, za katerega veljajo pravice.

Tabela 4.7: Opis tabele HasPriviligesOnColumn

Tabela HasPriviligesOnSwimlane		
Ime atributa	**Podatkovni tip**	**Opis polja**
Rights	Integer	Številka, ki predstavlja pravice. Vrednost 1 je izvzemanje kartice iz plavalne steze, 2 je dodajanje vanjo. S seštevanjem številk kombiniramo pravice.
SwimlaneId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis v tabeli `Swimlane`, za katerega veljajo pravice.
GroupId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis v tabeli `Group`, za katerega veljajo pravice.

Tabela 4.8: Opis tabele HasPriviligesOnSwimlane

Tabela HasPriviligesOnBoard		
Ime atributa	**Podatkovni tip**	**Opis polja**
Rights	Integer	Številka, ki predstavlja pravice. Vrednost 1 je dostop do table, 2 je ustvarjanje kartic. S seštevanjem številk kombiniramo pravice.
BoardId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis v tabeli `Board`, za katerega veljajo pravice.
GroupId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis v tabeli `Group`, za katerega veljajo pravice.

Tabela 4.9: Opis tabele HasPriviligesOnBoard

Tabela Log		
Ime atributa	**Podatkovni tip**	**Opis polja**
LogId	Integer	Primarni ključ tabele.
Description	Variable characters	Opis dogodka.
ActionDateTime	Date & Time	Časovni žig, ki označuje trenutek dogodka.
ActionTypeId	Integer	Tuji ključ, ki kaže na tip dogodka v `ActionType`.
UserId	Integer	Tuji ključ, ki kaže na zapis uporabnika v `User`, ki je dogodek povzročil.

Tabela 4.10: Opis tabele Log

Tabela ActionType		
Ime atributa	**Podatkovni tip**	**Opis polja**
ActionTypeId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime tipa akcije ali dogodka.

Tabela 4.11: Opis tabele ActionType

Tabela Card		
Ime atributa	**Podatkovni tip**	**Opis polja**
CardId	Integer	Primarni ključ tabele.
CardTypeId	Integer	Tuji ključ, ki določa tip kartice. Polje kaže na zapis v tabeli `CardType`.
ServiceLevelId	Integer	Tuji ključ, ki določa raven storitve kartice. Polje kaže na zapis v tabeli `ServiceLevel`.

Tabela 4.12: Opis tabele Card

Tabela CardType		
Ime atributa	**Podatkovni tip**	**Opis polja**
CardTypeId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime tipa kartice.

Tabela 4.13: Opis tabele CardType

Tabela ServiceLevel		
Ime atributa	**Podatkovni tip**	**Opis polja**
ServiceLevelId	Integer	Primarni ključ tabele.
Name	Variable characters	Ime ravni storitve.
Priority	Integer	Prioriteta ravni storitve, določa vrstni red ravni.

Tabela 4.14: Opis tabele ServiceLevel

Tabela CardField		
Ime atributa	**Podatkovni tip**	**Opis polja**
FieldId	Integer	Primarni ključ tabele.
FieldName	Variable characters	Ime polja.

Tabela 4.15: Opis tabele CardField

Tabela ValueOfFieldOnCard		
Ime atributa	**Podatkovni tip**	**Opis polja**
FieldId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis polja v tabeli `CardField`, za katerega velja vrednost.
CardId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis kartice v tabeli `Card`, za katero velja vrednost polja.
StringValue	Variable characters	Besedilna vrednost polja na kartici.
NumericValue	Decimal	Številska vrednost polja na kartici.

Tabela 4.16: Opis tabele ValueOfFieldOnCard

Tabela CardInColumn		
Ime atributa	**Podatkovni tip**	**Opis polja**
ColumnId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis stolpca v tabeli `Column`, v katerem je (bila) kartica.
CardId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis kartice v tabeli `Card`, za katero opredeljuje pozicijo v stolpcu.
DateFrom	Date & Time	Časovni žig, ki označuje trenutek, ko je kartica vstopila v stolpec.
DateTo	Date & Time	Časovni žig, ki označuje trenutek, ko je kartica izstopila iz stolpca. Če je polje prazno, je kartica še vedno v tem stolpcu.

Tabela 4.17: Opis tabele CardInColumn

Tabela CardOnSwimlane		
Ime atributa	**Podatkovni tip**	**Opis polja**
SwimlaneId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis plavalne steze v tabeli `Swimlane`, na kateri je (bila) kartica.
CardId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis kartice v tabeli `Card`, za katero opredeljuje pozicijo na plavalni stezi.
DateFrom	Date & Time	Časovni žig, ki označuje trenutek, ko je kartica vstopila na plavalno stezo.
DateTo	Date & Time	Časovni žig, ki označuje trenutek, ko je kartica izstopila s plavalne steze. Če je polje prazno, je kartica še vedno na tej stezi.

Tabela 4.18: Opis tabele CardOnSwimlane

Tabela CardAssignedToUser		
Ime atributa	**Podatkovni tip**	**Opis polja**
UserId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis uporabnika v tabeli `User`, ki je (bil) odgovoren za kartico.
CardId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis kartice v tabeli `Card`.
DateFrom	Date & Time	Časovni žig, ki označuje trenutek, ko je bila kartica dodeljena uporabniku.
DateTo	Date & Time	Časovni žig, ki označuje trenutek, ko uporabnik ni več odgovoren za kartico. Če je polje prazno, je uporabnik še vedno odgovoren za kartico.

Tabela 4.19: Opis tabele CardAssignedToUser

Tabela UserApplicationRole		
Ime atributa	**Podatkovni tip**	**Opis polja**
UserId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis uporabnika v tabeli `User`, za katerega velja vloga.
GroupId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis uporabniške skupine v tabeli `Group`.

Tabela 4.20: Opis tabele UserApplicationRole

Tabela UserInGroupOnBoard		
Ime atributa	**Podatkovni tip**	**Opis polja**
UserId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis uporabnika v tabeli `User`.
GroupId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis uporabniške skupine v tabeli `Group`.
BoardId	Integer	Tuji ključ in del sestavljenega primarnega ključa. Opredeljuje zapis table v tabeli `Board`.

Tabela 4.21: Opis tabele UserInGroupOnBoard

Poglavje 5

Zaključek

5.1 Sklepne ugotovitve

Pri izdelavi diplomskega dela sem se spoznal z metodologijo Kanban in orodji, ki podpirajo delo v skladu z njo. Iz analize same metodologije sem prišel do izredno uporabnih informacij, ki jih bom lahko uporabil v svojem poklicnem delu. Ugotovil sem, da vizualizacija delovnega toka pripomore k organizaciji dela ter da omejitev količine dela pozitivno vpliva na delovno vzdušje in produktivnost zaposlenih.

S pomočjo informacij, pridobljenih pri analizi Kanbana in pregledu obstoječih orodij, sem lahko specificiral funkcionalnosti popolnega orodja za podporo metodologiji. Na tržišču sicer obstaja že kar nekaj zelo dobrih orodij, vendar nobeno ne izpolni vseh zahtev podanih v specifikaciji v poglavju 4. S tem, ko sem se lotil analize obstoječih orodij, sem pridobil tudi pomembno znanje in izkušnje pri uporabi, ki mi bodo koristile v nadaljnjem poklicnem življenju.

Menim, da je Kanban metodologija, katere uporaba se bo v prihodnje še povečala, saj trenutne raziskave kažejo obetavne rezultate. Analize trenda uporabe v zadnjih letih moje mnenje potrjujejo.

5.2 Nadaljnje delo

V nadaljevanju bi lahko bolj podrobno zajeli rezultate raziskav in uporabe Kanbana v praksi. S pomočjo teh informacij bi nato lahko boljše ocenili, kdaj je uporaba metodologije primerna in kje ležijo potencialne nevarnosti in slabosti. Prav tako bi se lahko lotili vpeljave v realno okolje, ob tem pa bi natančno dokumentirali potek in s tem pridobili koristne informacije za nadaljnje raziskovalno delo.

Smiselna bi bila tudi implementacija orodja v skladu z uporabniškimi zgodbami definiranimi v poglavju 4. Ob tem bi lahko uporabili pridobljene informacije iz vpeljave v realno okolje in dopolnili nabor uporabniških zgodb. Orodje bi nato lahko uporabili v praksi ter ocenili, ali je orodje izpolnilo pričakovanja in dopolnili morebitne manjkajoče funkcije. Prav tako bi bilo zanimivo opazovati, kako uporaba orodja vpliva na vsakdanje delo in izsledke dokumentirati.

Literatura

[1] 7th Annual State of Agile Development Survey. Dostopno na: http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf

[2] Anderson, D. J. Kanban, Successful Evolutionary Change for Your Technology Business. Sequim: Blue Hole Press, 2010.

[3] Bottleneck. Dostopno na: http://en.wikipedia.org/wiki/Bottleneck

[4] Corona E., Eros Pani F. "A Review of Lean-Kanban Approaches in the Software Development". WSEAS Transactions On Information Science and Applications, Issue 1, Volume 10, January (2013). Dostopno na: http://www.wseas.org/multimedia/journals/information/2013/5709-110.pdf

[5] Glabman, A. (2012) "Cumulative Flow Diagram". Dostopno na: https://support.leankit.com/entries/21914317-Cumulative-Flow-Diagram

[6] Glabman, A. (2012) "Process Control (SPC) Diagrams". Dostopno na: https://support.leankit.com/entries/21917006-Process-Control-SPC-Diagrams

[7] Glabman, A. (2012) "The Efficiency Diagram". Dostopno na: https://support.leankit.com/entries/21914367-The-Efficiency-Diagram

[8] Johnsen A., Sjøberg D.I.K., Solberg J. "Quantifying the Effect of Using Kanban versus Scrum: A Case Study". IEEE Software. 9/10 (2012), str. 47-53.

[9] KanbanTool, metrika aplikacije. Aplikacija dostopna na: http://kanbantool.com/