Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tamara Žlender

Vpeljava načel vitkosti v razvoj programske opreme po metodi Kanban

diplomsko delo

univerzitetni študijski program računalništva in informatike

Mentor: izr. prof. dr. Viljan Mahnič

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Proučite značilnosti agilnega in vitkega razvoja programske opreme s poudarkom na metodah Scrum in Kanban. Analizirajte stične točke obeh metod in pokažite, kako lahko njune prakse izboljšajo postopek razvoja programske opreme. Ker je za uporabo metode Kanban bistvenega pomena vzdrževanje Kanban table, izdelajte pregled računalniških orodij, ki podpirajo vzdrževanje omenjene table, in analizirajte njihove funkcionalnosti. Na podlagi pridobljenih spoznanj izdelajte predlog, kako bi vpeljali Kanban v projekt, na katerem ste že delali v praksi.

Izjava o avtorstvu diplomskega dela

Podpisana Tamara Žlender z vpisno številko 63020084, sem avtorica diplomskega dela z naslovom:

Vpeljava načel vitkosti v razvoj programske opreme po metodi Kanban

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, 21. julija 2015 Podpis avtorice:

Zahvaljujem se mentorju, izr. prof. dr. Viljanu Mahniču, za strokovno svetovanje, potrpežljivost in spodbudo pri nastajanju diplomskega dela.

Iskrena hvala tudi vama, Rok in Nik, ki me sprejemata tako, kot sem. Ob vseh mojih vzponih in padcih verjameta vame, me optimistično spodbujata ter mi nesebično pomagata.

Hvala mami Marti in očetu Andreju, ki sta me vselej podpirala, mi pomagala in svetovala ter Rokovim staršem in prijateljem, ki so skrbeli za mojega Nika, mi s tem podarili čas in prostor za dokončanje študija.

Zahvaliti si želim še Ireni, ki me je spodbujala s prijaznimi besedami in sporočili.

Mojemu najboljšemu prijatelju in možu Roku ter vsem dekletom in fantom, ki se trudijo za ravnovesje med spoloma v računalništvu.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
TPS	Toyota Production System	Toyota proizvodni sistem
VSM	Value Stream Mapping	Vrednost toka procesa
TDD	Test Driven Development	Testno voden razvoj
XP	Extreme Programming	Metodologija ekstremnega programiranja
JIT	Just-in-Time	Ravno ob pravem času
WIP	Work-in-Progress	Število delovnih nalog, ki se lahko izvajajo sočasno
SLA	Service Level Agreement	Dogovor o ravni storitve
TOC	Theory of Constraints	Teorija omejitev
CFD	Cumulative Flow Diagram	Kumulativni diagram delovnega toka
API	Application Programming Interface	Programski vmesnik
RSS	Rich Site Summary	Protokol za objavo in distribucijo spletnih vsebin
PPM	Project Portfolio Management	Upravljanje projektnega portfelja
JQL	Jira Query Language	Jezik za poizvedbe v orodju Jira
CSV	Comma-separated values file	Format datoteke, vrednosti so ločene z vejico
LDAP	Lightweight Directory Access Protocol	Internetni protokol za dostop do imenikov
JSON	JavaScript Object Notation	Enostaven in berljiv standard za zapis podatkov
RPC	Remote procedure call	Klic za oddaljeni postopek
CSS	Cascading Style Sheets	Stilna predloga za spletne strani

Povzetek

Diplomsko delo obravnava vpeljavo načel vitkosti pri razvoju programske opreme po metodi Kanban. Začetno poglavje je namenjeno seznanitvi z agilnim in vitkim procesom razvoja, ki sta v zadnjih nekaj letih vedno bolj priljubljena in kažeta boljše rezultate kot tradicionalno voden razvoj.

Agilni in vitki proces razvoja sta opisana z manifestom agilnega razvoja in načeli vitkosti. Bolj podrobno pa sta opisani agilna metoda Scrum in vitka metoda Kanban. Sledi njuna primerjava ter predstavitev Scrumbana, ki je kombinacija lastnosti obeh. V diplomskem delu je pojasnjena pomembnost sodelovanja med vodstvom in delavci, vizualizacije delovnega toka, omejevanja obsega dela in vzpostavitve načela vlečenja dela.

Sledi analiza obstoječih elektronskih kanban orodij, ki so trenutno najbolj razširjena, in primerjava le-teh z zmogljivostmi fizične kanban table. Na podlagi pridobljenih spoznanj je predstavljen še načrt vpeljave metode Kanban v projekt prenove informacijskega sistema premoženjskih zavarovanj, na katerem je avtorica sodelovala.

Sklepni del je namenjen zapisu ugotovitev in idej za nadaljnje delo.

Ključne besede: agilni razvoj programske opreme, vitek razvoj programske opreme, Scrum, Kanban, Scrumban, vizualizacija delovnega toka, omejevanje obsega dela, primerjava elektronskih Kanban orodij, načrt vpeljave metode Kanban.

Abstract

Thesis describes applying lean principles to software development with Kanban method. The first chapter is an introduction to agile and lean software development processes, which are both gaining popularity and showing better results in the last few years than traditional methodologies.

Agile and lean software development processes are described with the Agile Manifesto and lean principles. Scrum and Kanban methods are then described and compared in more detail. What follows is an introduction of Kanban adaptation to Scrum called Scrumban. Thesis also emphasizes the importance of cooperation between management and employees, workflow visualization, limiting Work-in-Progress and pulling new work into the system.

In the next chapter online kanban tool comparison is presented. Most popular online tools are compared with a physical kanban board capabilities. Followed by a chapter in which accquired learnings are used to prepare a plan for introducing Kanban method to an insurance IT system rebuild project in the area of property insurance on which the author worked on.

In the final chapter findings together with ideas for future research are presented.

Keywords: agile software development, lean software development, Scrum, Kanban, Scrumban, workflow visualization, limit Work-in-Progress, online kanban tool comparison, Kanban method introduction plan.

Poglavje 1

Uvod

Programska oprema je v moderni družbi prisotna zadnjih 60 let. Proces njenega razvoja je sprva potekal na način »kodiraj in popravi« (angl. code and fix), ki pa z večanjem velikosti in zahtevnosti izzivov ni več zadostoval za uresničitev ciljev po kakovostni programski opremi, izdelani v okviru predvidenega časa in želenih stroškov. Kot alternativa so se v 60. letih 20. stoletja, začenši z modelom življenjskega cikla programske opreme (angl. Systems Development Life Cycle), uveljavile tradicionalne metodologije razvoja programske opreme. Podjetje Standish Group je leta 1994 objavilo poročilo o neuspehih IT projektov, imenovano Chaos, ki je pokazalo, da je bilo kar 31,1 % projektov odpovedanih pred njihovim zaključkom, ter da je 52,7 % projektov na koncu stalo 189 % več od začetne ocene [22]. Tradicionalno vodenje na žalost ni dalo ustreznih rezultatov.

Kot alternativa so se v 90. letih 20. stoletja pojavile, začenši z metodo dinamičnega razvoja sistemov (angl. Dynamic Systems Development Method), agilne metodologije razvoja programske opreme, ki so hitro postale zelo priljubljene. Podjetje VersionOne s poizvedbo Annual State of Agile Development Survey v zadnjih letih poroča o konsistentni rasti uporabe agilne metode Scrum [3], ki jo je leta 2009 uporabljalo 50 % vprašanih anketirancev [34]. Z letom 2011, ko je metodo uporabljajo 52 % vprašanih anketirancev [35], je metoda postala de facto standard za izbiro metode razvoja programske opreme in se v letu 2013 še okrepila na 55 % [36] ter letu 2014 na 56 % [37]. Sodeč po poročilu Chaos iz leta 2011 pa je tudi stopnja uspeha agilnih projektov z 42 % kar trikrat višja od stopnje uspeha tradicionalnih projektov, vodenih po slapovni (angl. Waterfall) metodi, ki je 14-odstotna [23].

Danes vemo, da tradicionalni stil vodenja z načinom »ukazuj-in-nadzoruj« za današnjo generacijo delavcev ni več učinkovit, saj je ta popolnoma drugačna od tiste iz 20. stoletja in tudi naloge, ki jih opravlja, so veliko bolj zahtevne in sofisticirane. Raziskave kažejo, da dandanes ljudem na delovnem mestu primanjkuje motivacije, samoiniciative, želje po skupinske delu in sodelovanju [13]. Ljudje na delovnem mestu potrebujemo dru-

gačno obravnavo. Na osebno zadovoljstvo in s tem boljšo produktivnost na delovnem mestu vplivajo trije faktorji: samostojnost, pridobivanje spretnosti in znanja ter motivacija. Raziskave kažejo, da ko podjetja uvidijo, da je treba z zaposlenimi ravnati spoštljivo in se odpovedati principu korenčka in palice ter pri svojem vodenju raje upoštevati sodobne metode, zgradijo take organizacije in organizacijsko klimo, da se delavci počutijo dobro in delajo bolje [14].

Za reševanje strateških problemov, ki vključujejo neko mero negotovosti, tveganja in zahtevnosti, je najboljša strategija sodelovanje med vodstvom in delavci. Vodstvo se spozna na svoje področje, delavci na svoje in šele medsebojno sodelovanje daje najboljše rezultate za doseganje skupnih ciljev. Prav taka načela zagovarja vitek razvoj programske opreme, ki se v zadnjih 10 letih vedno bolj uveljavlja. Primer uspešne metode vitkega razvoja je metoda Kanban [1]. Hitro rast uporabe metode Kanban kažejo tudi prej omenjene VersionOne poizvedbe, ki v letu 2011 prvič umestijo metodo Kanban v svojo poizvedbo in poročajo o 3-odstotni uporabi [35], v letih 2013 in 2014 pa o 5-odstotni uporabi metode Kanban [36,37]. Vitko razmišljanje in agilne prakse je med seboj mogoče kombinirati. Eden od takih pristopov je tudi Scrumban [9], za katerega VersionOne poizvedba kaže dvakratno povečanje uporabe med letoma 2011 in 2014, in sicer s 3 % [35] na 6 % [37]. Uporaba vitkega razvoja po metodi Kanban in njenih različicah je kot kaže zelo aktualna tema, ki se je že pokazala kot uspešna praksa. David J. Anderson poroča [1], da se je povprečni potrebni čas pri projektu, ki so ga razvijali za Microsoft, po uvedbi Kanbana zmanjšal od 125–155 dni na samo 14 dni [1]. Podobno Dag I.K. Sjøberg in drugi opisujejo primer nekega skandinavskega podjetja [19], ki je po uvedbi Kanbana prepolovilo potrebni čas, zmanjšalo število okvar za 10 % in povečalo produktivnost. Poizvedba podjetja Ambysoft IT Project Success Rates Survey iz leta 2013 pa kaže, da so v povprečju bolj uspešne tiste ekipe, ki sledijo vitkemu razmišljanju, kot tiste, ki sledijo agilnosti [42].

Cilj diplomskega dela je preučiti značilnosti nekaterih tipičnih tovrstnih metodologij in analizirati orodja za elektronsko in fizično podporo sistema kanban. Obenem pa vidimo cilj tudi v vpeljavi načel vitkosti v tradicionalen proces razvoja projekta, pri katerem je avtorica diplomskega dela sodelovala.

Diplomsko delo je sestavljeno iz treh delov. V prvem delu (Poglavje 2) predstavimo agilnost in vitkost z opisom manifesta agilnega razvoja programske opreme in načel vitkosti. Pojasnimo njihov nastanek, vrednote ter povezljivost vitkega razmišljanja z že uveljavljenimi agilnimi praksami. Posebej opišemo metodi Scrum in Kanban, ju primerjamo ter predstavimo še kombinacijo lastnosti obeh, kar imenujemo Scrumban. V drugem delu (Poglavje 3) predstavimo merila za primerjavo elektronskih kanban orodij, jih razdelimo v štiri skupine ter predstavimo po eno orodje iz vsake skupine. Rezultate predstavimo v tabeli, kamor za primerjavo dodamo tudi značilnosti fizične kanban table. V zadnjem delu

(Poglavje 4) na podlagi pridobljenih izkušenj izdelamo in predstavimo predlog vpeljave metode Kanban v projekt prenove informacijskega sistema premoženjskih zavarovanj, na katerem je avtorica sodelovala. Diplomsko delo sklenemo z zapisom svojih ugotovitev in idej za nadaljnje delo.

Poglavje 2

Vitkost in povezljivost z agilnimi praksami

2.1 Agilnost in vitkost

2.1.1 Manifest agilnega razvoja programske opreme

Leta 2001 se je sedemnajst strokovnjakov iz skupnosti razvijalcev programske opreme dogovorilo, da bi pri določanju načina razvoja programske opreme uporabili izraz agilnost (angl. agile). Napisali so manifest [44], s katerim so definirali nov pristop, ki je danes znan kot agilni razvoj programske opreme.
Vsebina manifesta je naslednja:
Odkrivamo boljše načine razvoja programske opreme tako, da jo razvijamo in pri tem pomagamo tudi drugim. Naše vrednote so ob tem postale:

- Posamezniki in interakcije pred procesi in orodji
- Delujoča programska oprema pred vseobsežno dokumentacijo
- Sodelovanje z naročnikom pred pogodbenimi pogajanji
- Odziv na spremembe pred togim sledenjem načrtom

Z drugimi besedami, četudi cenimo dejavnike na desni, vseeno bolj cenimo tiste na levi. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

Poleg štirih vrednot so v ozadju agilnega manifesta določili še dvanajst principov [31]:

- Naša najvišja prioriteta je zadovoljiti naročnika s hitro in neprestano dostavo uporabne programske opreme.
- Sprejemamo spremembe zahtev, celo v poznih fazah razvoja. Agilni procesi štejejo upoštevanje sprememb zahtev kot primerjalno prednost za naročnika.
- Delujočo programsko opremo izdajamo čim pogosteje, od nekaj tednov do nekaj mesecev.
- Naročniki in razvijalci morajo pri celotnem projektu vsakodnevno sodelovati.
- Projekte gradimo okrog motiviranih posameznikov. Omogočimo jim ustrezno delovno okolje, nudimo podporo in jim zaupamo, da bodo svoje delo opravili.
- Najprimernejši in najučinkovitejši način posredovanja informacij razvojni ekipi in med sodelavci v ekipi je osebni pogovor.
- Delujoča programska oprema je primarno merilo napredka.
- Agilni procesi promovirajo trajnostni razvoj. Sponzorji, razvijalci in uporabniki morajo biti zmožni konstantnega tempa za nedoločen čas.
- Nenehna težnja k tehnični odličnosti in k dobremu načrtovanju izboljša agilnost.
- Preprostost – umetnost zmanjševanja količine nepotrebnega dela – je bistvena.
- Najboljše arhitekture, zahteve in načrti izhajajo iz tistih ekip, ki so samoorganizirane.
- V rednih časovnih razdobjih ekipa išče načine, kako postati učinkovitejša ob rednem prilagajanju svojega delovanja.

Manifest poleg vrednot in principov ne predpisuje nobenega načina, kako te cilje doseči. V pomoč so nam agilne metode, ki opisujejo prakse, definirajo vloge in pričakovanja. Kljub temu, da so agilne metode dobile svoje ime leta 2001, jih je večina nastala v 80. in 90. letih. Te metode si prizadevajo na preprost, fleksibilen in hiter način izdelati zmogljivo programsko opremo in zagotoviti nenehno spremljanje poteka projekta. Vključujejo iterativni in inkrementalni razvoj, zahtevajo intenzivno komunikacijo ter sodelovanje z uporabniki, omogočajo sprotno vključevanje sprememb zahtev naročnika, so enostavne za uporabo in se jih da hitro naučiti. Nove metode hitro postanejo alternativa težko obvladljivim, tradicionalnim metodologijam. Odpravljajo probleme, kot so dolg življenjski cikel, zahtevno učenje in uporaba, težka prilagodljivost, obsežna dokumentacija, težko vključevanje sprememb v fazi razvoja, preveč zapleten in nepredvidljiv postopek razvoja, redke izdaje, nesamozavestna razvojna ekipa, pomanjkanje interakcije in nesporazumi med končnimi uporabniki in

razvojno ekipo, nezaupanje naročnika ipd., s katerimi so se spoprijemali vodje projektov in razvijalci [8].

Agilnost je postal skupni izraz za vsako metodo ali postopek, ki sledi vrednotam in principom, zapisanim v manifestu. Manifest agilnega razvoja programske opreme je zgodovinski dokument, ki izraža željo po spremembah. Besedilo manifesta se je hitro razširilo po celem svetu in povzročilo revolucijo v razvoju programske opreme.

2.1.2 Načela vitkosti

Začetki idej o vitki (angl. lean) proizvodnji prihajajo iz Japonske. Podjetje Toyota je z idejo Ohno Taiichija v poznih 80. letih razvilo t. i. Toyota proizvodni sistem (angl. Toyota Production System – TPS). TPS je skupek načel in konceptov za doseganje večje konkurenčnosti, cenovne konkurenčnosti na trgu in stroškovne učinkovitosti, kar pomeni manj stroškov in izgub v proizvodnih procesih ter manj napak in izmeta v izdelkih [67].

Vse od takrat so se orodja vitkega poslovanja izpopolnjevala in tako se je koncept vitkosti razširil v proizvodna podjetja, zdravstvene storitve, gradbeništvo in druge panoge povsod po svetu. Vitkost je postal izraz, ki opisuje sistemski pristop za ohranjanje vrednosti z manj dela, s spodbujanjem ekip in podjetij k sprejemanju odločitev in s tem prilagajanju vsakodnevnim problemom in oviram ter zmanjševanju odpadkov. Toyotin način in navada uporabe koncepta vitkosti sta se izkazala uporabna tudi pri razvoju programske opreme.

Ideje agilnosti in vitkosti prihajajo iz različnega okolja, a si delita enako razmišljanje, katerega cilj je čim hitrejši razvoj in zmanjšanje pomanjkljivosti ter napak. Zato se metodologiji zelo dobro dopolnjujeta in vedno več organizacij razvijalcev programske opreme išče načine, kako povezati principe obeh metod za podprtje vseh procesov, od zamisli do predaje izdelka. Izraz vitek razvoj programske opreme se je prvič pojavil v knjigi z naslovom Lean software development: An Agile Toolkit, ki sta jo leta 2003 napisala Mary in Tom Poppendieck [16]. S to knjigo sta želela pripeljati vitko razmišljanje v že uveljavljene agilne prakse.

Agilne metodologije ponujajo ustaljene prakse, metodologija vitkega razvoja pa dodajo v proces razvoja skupek visoko prilagodljivih načel. Načela usmerjajo in nudijo razumevanje o določeni problematiki, prakse pa so načini ravnanja, s katerimi se izvedejo načela. Mary in Tom Poppendieck v svoji knjigi ne opredelita praks, ampak se osredotočita na sedem načel vitkosti ter ponudita ideje, kako jih prevesti v agilne prakse.

Sedem načel vitkosti in ideje, kako jih prenesemo v agilne prakse, so: [8, 15, 16]

1. Odstranjevanje odpadkov (angl. eliminate waste)

   Odpadek (angl. waste, jap. muda) je vse, kar ne prispeva k vrednosti končnega izdelka, torej za naročnika nima dodane vrednosti. Idealno je, da v čim krajšem času

izdelamo točno tisto, kar naročnik zahteva. Kar koli preprečuje, da se naročnikove potrebe ne zadovoljijo dovolj hitro, je odpadek.

V razvoju programske opreme so naslednji odpadki:

- transport (angl. transport): predajanje dokumentacije iz ene ekipe v drugo;
- čakanje (angl. waiting): član ekipe je na voljo, da prevzame novo nalogo, ki pa ni definirana;
- prevelika proizvodnja (angl. over production): ekipa razvije funkcionalnost, ki jo uporabniki ne potrebujejo;
- popravila (angl. defects): ekipa izda verzijo programa, ki ima napake in jih je potrebno odpraviti;
- zaloga (angl. inventory): neuporabljeni izdelki, ki nastanejo v celotnem življenjskem ciklu razvojnega procesa;
- gibanje (angl. motion): zamenjava članov ekipe, ki ne prispeva k nobeni razvojni izboljšavi;
- preveč vrednosti (angl. over processing): protokoli, ki jim morajo člani ekipe slediti, ne prispevajo pa k vrednosti izdelka za končnega naročnika;
- napačno vključeni oz. vodeni ljudje (angl. skills): nevključevanje služb za upravljanje s človeškim viri.

Za prepoznavanje odpadkov se lahko uporablja orodje za analizo vrednosti toka procesa (angl. Value Stream Mapping – VSM), ki omogoča, da analiziramo proces z vidika uporabnika procesa oz. naročnika. To je ključno za odstranjevanje izgub v obstoječem procesu in za doseganje izboljšav. Prepoznavanje odpadkov spodbuja skupinsko delo in pomaga razviti podrobno, s podatki popolno karto VSM ter definirati želeno prihodnje stanje. Ko odpadke identificiramo, jih moramo še odstraniti.

Agilna praksa, ki prepoznava in odpravlja odpadke, je npr. retrospektiva preteklega dela. Retrospektiva (angl. retrospective) je sestanek po vsaki končani iteraciji oz. po rednem ritmu, kjer je omogočena diskusija, kaj je šlo dobro, kaj ne in kaj se lahko naredi drugače pred naslednjo retrospektivo. Kadar le-ta poteka redno, omogoča ekipi, da sproti izboljšuje proces in se pravočasno spoprijema z manjšimi spremembami, ko so še obvladljive.

2. Ustvarjanje znanja (angl. amplify learning)

   Razvoj programske opreme je nenehni proces učenja. Najboljši način za izboljšavo razvoja programske opreme je ustvarjanje znanja. Ekipe se morajo truditi, da pridobivajo nova znanja, ki jih bodo potrebovale za sprejemanje ustreznih odločitev. Znanje povečuje produktivnost in fleksibilnost ekipe.

Tehnike, ki pomagajo ustvarjati znanje, so:

- programiranje v paru (angl. pair programming);
- pregledi kode (angl. code review);
- dokumentacija;
- Wiki, kjer se znanje dopolnjuje inkrementalno;
- temeljito dokumentirana koda;
- prenosi znanj in dobrih praks v ekipi;
- usposabljanja;
- različna orodja za vodenje projektov.

3. Pozno odločanje (angl. decide as late as possible)

   Odločati se pozno je dobro, ker odločitve s tem, ko jih prestavimo na poznejši čas, izboljšamo, saj temeljijo na novih dejstvih in informacijah in ne na špekuliranju. Rešitve morajo biti dobro arhitekturno zasnovane, da so prožne in tako omogočajo sprejetje čim kasnejših odločitev. Paziti je treba, da se ne odločamo predolgo, saj to lahko upočasni ekipo in naredi projekt še težji.

   Primer poznega odločanja v agilnem razvoju programske opreme je planiranje iteracije. Na sestanku za planiranje iteracije se iz seznama zahtev izbere funkcionalnosti, ki bodo dokončane v času iteracije. Posamezna nova funkcionalnost se tako določi pred vsako iteracijo in analizira zadnji trenutek, tik pred razvojem. S tem, ko odločitev o izbiri funkcionalnosti in njenega razvoja sprejmemo istočasno, zmanjšujemo odpadke, saj razvijamo le tiste funkcionalnosti, ki jih uporabniki potrebujejo.

   To ne pomeni, da planiranje ni potrebno. Planiranje je še zmeraj zaželeno, ampak bolj osredotočeno na drugačno izvedbo ter prilagajanje trenutni situaciji in kot dodatna pomoč za razčiščevanje nejasnosti z vnaprej vzpostavljenimi koraki za hitro odzivanje.

4. Hitro izdajanje (angl. deliver as fast as possible)

   Hitrost dostave našega produkta na trg nas dela konkurenčne. Podjetja, ki imajo konkurenčno prednost, niso tista največja, ampak tista, ki so najhitrejša. Hiter razvoj ima veliko prednosti. Če nismo hitri, se ne moremo pozno odločati. Če nismo hitri, tudi nimamo zanesljivih povratnih informacij. Hiter življenjski cikel razvoja programske opreme je ključen za ustvarjanje znanja. Krajši ko je cikel, več se lahko naučimo. Prej ko dostavimo naročniku končni izdelek brez večjih napak, prej bomo dobili nazaj povratne informacije in jih vključili v naslednjo iteracijo. Krajše ko so iteracije, več znanja ustvarimo in več medsebojno komuniciramo. Hitrost jamči

naročniku, da dobi tak produkt, kot si ga želi danes, in ne takega, kot ga je potreboval včeraj. Hitrost naročniku omogoča sprejeti pozno odločitev takrat, ko bo dokončno vedel, kaj potrebuje.

Hiter razvoj dosežemo s pravimi ljudmi v ekipi, takimi, ki zmorejo misliti s svojo glavo, rešujejo probleme, so proaktivni, fleksibilni in znajo sprejemati zahtevne odločitve. Delo mora potekati timsko, člani si morajo pomagati in poskrbeti, da ekipa doseže svoje cilje. Druge možnosti za hitrejši razvoj so tudi, da ne delamo prezapletenih rešitev, poenostavljamo zahteve, najdemo najhitrejše načine za zadovoljitev naročnikovih ciljev, ne razmišljamo predolgo o zahtevah, ki morda ne bodo nikoli uresničene, odstranimo odpadke in vgradimo v svoj proces kakovost.

5. Pooblaščanje ekipe (angl. empower the team)

   Skrivnost brezhibne izvedbe se skriva v podrobnostih, nihče pa ne razume podrobnosti bolje od razvijalcev, ki pišejo programsko kodo. Vključevanje razvijalcev v proces odločanja je ključnega pomena za doseganje prvorazrednosti. Razvijalci za t. i. prvo obrambno črto s skupinskim razmišljanjem združujejo znanje in vse najmanjše podrobnosti. Če jih opremimo z ustreznimi strokovnimi znanji in dobro vodjo, so sposobni sprejemati najboljše tehnične in procesne odločitve. Zaradi narave razvoja, poznega odločanja in hitre izvedbe, je boljše, da procesa ne določa vodstvo. Vloge v vitkem razvoju so torej obrnjene, vodstvo se mora naučiti poslušati svoje razvijalce, saj lahko le-ti bolje razjasnijo, kateri naslednji koraki v procesu so smiselni, in pripravijo predloge za izboljšavo. Vitek razvoj uporablja t. i. tehnike potega oz. vlečenja dela (angl. pull). To pomeni, da ko ima razvojna ekipa proste kapacitete, sama odloča, koliko dela bo prevzela, namesto da delo določa nekdo od zunaj in ga s tem potiska v naslednje stanje (angl. push).

   Načelo vlečenja daje ljudem zaupanje in priložnost, da sprejemajo odločitve o svojem delu. Za dosego tega je treba ustvariti veliko znanja in omogočiti učenje vsem v ekipi z namenom, da postanejo najboljši v tem, kar počnejo. Ljudje, ki so ustrezno in pošteno plačani, so dodatno motivirani s samostojnostjo, pridobivanjem spretnosti in z zaupanjem v smisel svojega dela [14].

6. Vgradnja kakovosti (angl. build integrity in)

   Kakovost je izjemnega pomena, saj brez nje ustvarjamo nepotrebne odpadke. Pomembno je, da jo v razvojni proces vgradimo že v začetnih fazah in jo med celotnim procesom izboljšujemo.

   Če naročnik meni, da je dobil točno tako rešitev, kot jo potrebuje, se sistem obravnava kot celovit. Rešitev mora naročniku dati celotno izkušnjo uporabe sistema, tudi

kako je izdelek oglaševan, razvit, dostopen, intuitiven, koliko stane ter kako dobro rešuje probleme. Celovit sistem mora imeti pregledno arhitekturo, mora biti uporaben, lahek za vzdrževanje, prilagodljiv in razširljiv. Celovitost dosežemo predvsem z modrim vodenjem, ustreznim strokovnim znanjem, učinkovito komunikacijo in zdravo disciplino.

Če pa se v razvojnem procesu redno pojavljajo napake, je proces okvarjen. Napake je treba najti in odstraniti v razvojnem procesu čim bolj zgodaj. Pregledno arhitekturo lahko zagotovimo s sprotnim preoblikovanjem kode (angl. code refactoring), sprotno integracijo (angl. continuous integration) in izčrpnimi testi (angl. software testing). Avtomatizirani testi (angl. test automation) so zaželeni, vendar le tisti, ki imajo dodano vrednost in se ne obravnavajo kot odpadki.

V celoten proces se lahko vgradi testno voden razvoj (angl. Test Driven Development – TDD), ki odpravlja težave s kakovostjo tako, da pripravi teste pred pisanjem kode. Druga možnost za vgraditev kakovosti je programiranje v paru, ki odpravlja težave s kakovostjo tako, da isto nalogo opravljata dva razvijalca hkrati. Z združevanjem izkušenj na dolgi rok večamo produktivnost in povečujemo kakovost. Obe omenjeni metodi spadata med dobre prakse metodologije ekstremnega programiranja (angl. Extreme Programming – XP) [2].

7. Optimizacija celote (angl. see the whole)

   Računalniški sistemi danes niso le skupki posameznih delov, so tudi rezultat interakcije med njimi. Večji ko je sistem, več organizacij, ki sodelujejo pri njegovem razvoju, vsebuje. In s tem združuje tudi več ekip, ki razvijajo posamezne dele. Zato je toliko bolj pomemben odnos, ki ga imamo z vsemi notranjimi in zunanjimi izvajalci. Dober odnos je pomemben za zagotovitev sistema, v katerem vse komponente sodelujejo na lahkoten način. Nujna je optimizacija celotnega procesa razvoja in ne samo posameznih delov ekip. Vsi člani, udeleženi na projektu, morajo dobro razumeti vitko razmišljanje, pomembnost dobre medekipne komunikacije, razmišljanje zunaj okvirjev ter hitro učenje iz neuspeha.

2.2 Scrum

Scrum je skupaj z njenimi hibridi najbolj priljubljena agilna metoda za razvoj programske opreme [37]. Zgodovina Scruma sega v leto 1986, ko sta Hirotaka Takeuchi in Ikujiro Nonaka [11] opisovala ekipe, ki z novim, drugačnim in celostnim pristopom zelo učinkovito razvijajo programske produkte, in ta pristop primerjala z ragbijem. V 90. letih sta Ken Schwaber in Jeff Sutherland ločeno, vendar istočasno uporabila tak pristop za razvoj

programske opreme, leta 1995 pa sta javnosti na konferenci v Austinu v Texasu prvič predstavila Scrum kot metodologijo. Leta 2001 je izšla prva knjiga o Scrumu z naslovom Agile Software Development with Scrum, ki sta jo skupaj napisala Ken Schwaber in Mike Beedle [3].

Scrum ni tipična metoda, ampak je bolj ogrodje za razvoj kompleksnih izdelkov. Scrum je sestavljen iz scrum ekipe in z njo povezanih vlog, dogodkov, izdelkov in pravil. Zasnovan je na teoriji empiričnega nadzora procesov, kar pomeni, da se znanje pridobiva z izkušnjami in odločitvami na podlagi znanega. Scrum uporablja časovno omejen inkrementalni pristop za razvoj programske opreme in vodenje projektov, tako da spodbuja pogosto interakcijo z naročnikom, med vsako iteracijo. Izdelki se razvijajo iterativno in inkrementalno, kar daje več priložnosti za pridobivanje povratnih informacij. Inkrementalni razvoj zagotavlja, da je potencialno uporabna verzija delujočega izdelka vedno na voljo.

Scrum ekipe se samoorganizirajo in so navzkrižno funkcionalne, kar pomeni, da so sestavljene iz strokovnjakov z več področij oz. generalistov (angl. cross-functional team), imajo vse veščine ter strokovno znanje, ki je potrebno za dokončanje dela in se ne zanašajo na druge, ki niso del nje. Ekipo sestavljajo tri pomembne vloge: produktni vodja (angl. product owner), razvojna ekipa (angl. development team) in skrbnik metodologije Scrum (angl. ScrumMaster).

Produktni vodja je predstavnik vseh uporabnikov oz. deležnikov projekta (angl. stakeholders) in je odgovoren za poslovno plat produkta. Njegova primarna naloga je upravljanje seznama zahtev izdelka (angl. product backlog), kar vključuje pripravo t. i. uporabniških zgodb (angl. user stories), določanje njihovih prioritet in umeščanje v seznam zahtev izdelka. Njegove naloge so še določitev časovnega okvira dostave produkta in njegove vsebine, odgovornost za donosnost produkta (angl. Return on Investment) ter sprejemanje ali zavrnitev delovnih rezultatov razvojne ekipe.

Razvojna ekipa mora biti dovolj majhna, da je agilna, in dovolj velika, da lahko dokonča večino dela; praviloma ne presega 10 članov. Mike Cohn opisuje [5] idealno velikost scrum ekipe kot 7 +- 2 člana. Ekipa razvija naloge iz seznama zahtev izdelka v kratkih iteracijah, imenovanih sprinti. Sprinti so časovno omejeni na en mesec ali manj in v času razvoja trajajo vedno enako dolgo. Ko se konča predhodni sprint, se takoj začne nov sprint.

Sprint ima v posebnih okoliščinah pravico prekiniti le produktni vodja. Prekinitev je neobičajen dogodek v primeru, da postane cilj sprinta zastarel, npr. zaradi sprememb v poslovnih zahtevah, prodaje podjetja, prehude konkurence itd.

Pri Scrumu se načrtuje na dveh ravneh. Produktni vodja lahko skupaj z ekipo začne proces s planiranjem izdaje (angl. release planning), pred začetkom vsakega sprinta pa se celotna scrum ekipa sestane na sestanku za planiranje iteracije (angl. sprint planning meeting).

Planiranje izdaje ni predpisano, je pa priporočeno. Cilj takega sestanka je, da razvojna ekipa predvidi izdaje izdelka skupaj z verjetnimi datumi za dokončanje. Ta začetni načrt ni zelo podroben, a vseeno začrta skupno smer in cilje.

Na sestanku za planiranje iteracije se iz seznama zahtev izdelka izberejo tiste uporabniške zgodbe, ki bodo realizirane v sprintu.

Sestanek je časovno omejen in je sestavljen iz dveh delov. V prvem delu se produktni vodja in razvojna ekipa dogovorijo, kaj bo narejeno v sprintu, v drugem pa se razvojna ekipa odloči, kako bo izbrano delo izvedla. Število izbranih uporabniških zgodb s seznama zahtev izdelka je izključno v domeni razvojne ekipe. Obseg dela se lahko med projekti razlikuje, zato lahko le razvojna ekipa napove, kaj lahko naredi v prihajajočem sprintu.

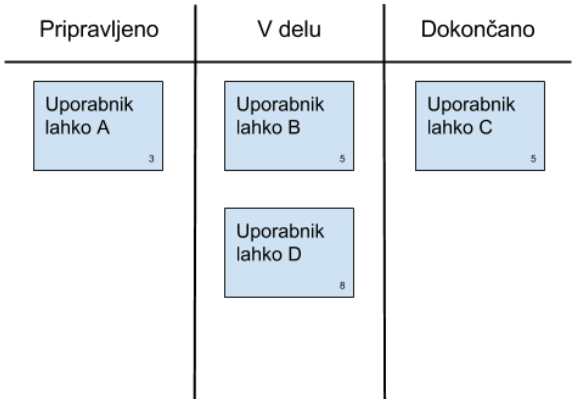
Število točk (angl. story points), ki jih razvojna ekipa določi vsaki uporabniški zgodbi, pove, kako težavna je zgodba. Te točke praviloma niso ure oz. dnevi, ki so potrebni za implementacijo zgodbe, ampak predstavljajo težavnost in mehanizem za medsebojno primerjavo. Vsota točk vseh uporabniških zgodb, ki jih ekipa implementira v sprintu, določa hitrost ekipe (angl. velocity), kar je uporaben podatek pri predvidevanju poteka projekta. Najbolj priljubljena tehnika, ki se uporablja za ocenjevanje težavnosti uporabniških zgodb, se imenuje Planning Poker [12].

Izbrane uporabniške zgodbe predstavljajo seznam nalog sprinta (angl. sprint backlog). Ko razvojna ekipa preučuje načrt in spoznava delo, ki je potrebno za dosego ciljev sprinta, lahko spreminja seznam nalog sprinta tudi med sprintom. Spremembe seznama nalog sprinta so lahko na novo dodana dela, pa tudi odstranitev tistih elementov načrta, ki niso več potrebni za dosego ciljev sprinta. Vendar pa lahko ta seznam med sprintom spreminja le razvojna ekipa sama, produktni vodja nima dovoljenja za spremembe, razen v zelo posebnih okoliščinah, kjer se s spremembo strinja tudi razvojna ekipa [75].

Seznam zahtev izdelka se lahko ves čas spreminja. Na spremembe tega seznama vplivajo spremembe poslovnih zahtev, tržnih razmer ali tehnologije. Seznam tako ni nikoli dokončan, se ves čas spreminja, da določa tisto, kar izdelek potrebuje, da je primeren, konkurenčen in uporaben.

Razvojna ekipa za lažji pregled nad delom uporablja scrum tablo. Najbolj enostavna oblika table, ki je predstavljena na sliki 2.1, vključuje stolpce »pripravljene naloge«, »v delu« ter »dokončano«. V stolpcu »pripravljene naloge« (angl. todo) na začetku vsakega sprinta prikažemo vse uporabniške zahteve, ki jih bomo razvili v sprintu. Zahteve se po tabli pomikajo skozi ta tri stanja. Tabla je v pomoč tudi deležnikom projekta, saj lahko na njej spremljajo napredovanje dela.

Vsi člani ekipe se morajo strinjati, kaj pomeni, da je delo končano (angl. done). Henrik Kniberg [7] definira dokončano delo kot tisto funkcionalnost, ki je pripravljena za izdajo. Poudarja pa, da so v nekaterih primerih sprejemljive tudi drugačne definicije dokončanega



Slika 2.1: Scrum tabla

dela, kot je npr. delo po končanem programiranju in testiranju, delo po končani integraciji, delo po končanem testiranju pred demo verzijo izdelka ali pa delo po končanem sprejemnem testiranju, ki ga opravijo poslovni uporabniki. Kakor koli že, definicijo ekipa oblikuje skupaj in po želji zapiše na vidno mesto, npr. ob scrum tablo. Namen vsakega sprinta je ustvariti nove funkcionalnosti, ki se ujemajo z definicijo dokončanega dela.

Scrum zahteva, da se razvojna ekipa sestaja čim bolj pogosto, najbolje dnevno, da preverja, kako dosega cilje sprinta ter uskladi aktivnosti in izdela načrt za naslednjih 24 ur. Tak časovno omejen 15-minutni dogodek razvojne ekipe se imenuje dnevni scrum sestanek (angl. daily Scrum meeting). Med sestankom vsak član razvojne ekipe pojasni, kaj je naredil od zadnjega sestanka, kaj bo naredil do naslednjega sestanka in kaj ga ovira.

Razvojna ekipa si po koncu vsakega sprinta rezervira nekaj ur za njegovo revizijo (angl. sprint review meeting), kjer produktnemu vodji in nosilcem interesa predstavi svoje rezultate. Na sestanku se pregleda, katere zahteve so dokončane, ter predstavi demo verzijo izdelka (angl. sprint demo). Produktni vodja preveri ustreznost izdelka in posreduje svoje predloge in pripombe. Revizija sprinta je neformalni sestanek, ki ima glavni namen dobiti povratno informacijo od vseh sodelujočih ter uporabiti te podatke pri urejanju seznama zahtev izdelka in pri načrtovanju naslednjega sprinta. Taka zanka s povratnimi informacijami (angl. feedback loop) je osnova za učenje in izboljšanje procesa ter praks.

Po reviziji sprinta in pred njegovim naslednjim načrtovanjem se izvede še retrospektiva sprinta (angl. sprint retrospective). Namen te retrospektive je pregledati potek zadnjega sprinta v zvezi z ljudmi, odnosi, procesi in orodji ter izboljšati svoj razvojni proces in prakse, da bo delo v naslednjem sprintu bolj učinkovito in prijetno. Člani razvojne ekipe se osredotočijo na dve vprašanji, in sicer kaj je bilo pri tej iteraciji dobro narejeno ter kaj bi bilo treba izboljšati pri naslednji iteraciji.

Produktni vodja lahko skupaj z razvojno ekipo med celotnim procesom skrbi še za negovanje seznama zahtev izdelka (angl. backlog grooming), kjer se pregleda uporabniške

zgodbe, dopolni oz. spremeni prioriteto, zamenja vrstni red itd.

Tretja vloga, ki jo definira Scrum, je skrbnik metodologije Scrum (angl. ScrumMaster), ki skrbi, da se scrum ekipa drži teorije Scrum, praks in pravil. Poleg tega, da je odgovoren za razvojni proces, je njegova naloga tudi pomagati produktnemu vodji ter razvojni ekipi odstraniti vse ovire, ki se pojavijo pri projektu ter pomagati razumeti vsem drugim v organizaciji, ki niso člani ekipe, kakšna interakcija s scrum ekipo je koristna in kakšna ne.

Scrum priporoča tudi spremljanje napredka na padajočem diagramu, kjer lahko spremljamo količino preostalega dela na dveh ravneh. Pri spremljanju napredka na ravni izdaje (angl. release burn down chart) produktni vodja na vsaki reviziji sprinta spremlja celotno preostalo delo in primerja to količino s preostalim delom iz prejšnjih revizij, da lahko oceni napredek pri dokončanju projektnega dela v želenem času.

Pri spremljanju napredka na ravni vsake posamezne iteracije (angl. sprint burn down chart) pa razvojna ekipa spremlja celotno preostalo delo na vsakem dnevnem scrumu. Razvojna ekipa dnevno spremlja te seštevke in projicira verjetnost doseganja cilja sprinta. Oba diagrama spremljanja napredka sta ves čas razvoja vidna vsem udeležencem projekta.

Testiranje se ne izvaja več v posebni fazi na koncu razvojnega cikla. Vsak inkrement pri razvoju programske opreme testiramo zgodaj in pogosto. Ker je Scrum ogrodje, ne omenja dobrih inženirskih praks za programiranje in testiranje. Izkušnje različnih scrum ekip pa kažejo, da se z izvajanjem avtomatiziranih testov ter sprotno integracijo hitrost ekipe dolgoročno lahko poveča [12].

2.3 Kanban

V literaturi se kanban pojavi v različnih kontekstih.

Beseda kanban je besedna zveza, ki prihaja iz japonskega jezika, »kan« pomeni vizualno in »ban« pomeni kartica ali znak, skupaj se to lahko razlaga kot vizualna oz. signalna kartica.

Kanban sistem pomeni na področju razvoja programske opreme t. i. sistem vlečenja dela, ki nam pokaže, kaj, kdaj in koliko naj izdelujemo oz. razvijamo. Ta koncept izhaja iz podjetja Toyota, kjer je bil vpeljan kot sistem oskrbe delovnih mest z materialom, in temelji na ideji »ravno ob pravem času« (angl. Just-in-Time – JIT) ob uporabi načela vlečenja dela. Sistem kanban je sestavljen iz kanban table s karticami, omejitvijo dela ter pravili, ki se jih držimo. Ta tabla nam omogoča vizualizacijo delovnega toka in je zemljevid našega dela. Pogled na kanban tablo nam omogoča takojšnje in intuitivno razumevanje našega sistema dela, jasno prikaže vloge, odgovornosti, omejitev obsega dela (angl. Work-in-Progress – WIP), dosežene rezultate, strukture naših procesov ter ovire. Tak natančen pregled omogoča boljše razumevanje dela in procesov ter vzpostavi skupni jezik razvojnih

ekip in poslovnih uporabnikov. Informacije na tabli so vedno aktualne in dostopne vsem zainteresiranim, kadar koli jih potrebujejo, ne da bi morali po njih spraševati. Najpogosteje so to veliki prikazi ali grafi, ki so razumljivi že na prvi pogled, npr. posterji z grafi, ki prikazujejo napredovanje projekta, stene z indeksnimi karticami, ki vsebujejo informacije o delovnih nalogah ali table s stolpci, ki prikazujejo delovni tok, in karticami, ki predstavljajo delovne naloge.

Metoda Kanban (angl. The Kanban method) je evolucijski pristop k razvijanju programske opreme, ki ga je David J. Anderson skupaj s sodelavci po metodologiji vitkega razvoja zasnoval in prvič predstavil leta 2010 v svoji knjigi Kanban: Successful Evolutionary Change for your Technology Bussiness [1].

Open Kanban je prva odprtokodna agilna in vitka metoda razvoja programske opreme za doseganje stalnih izboljšav (angl. continuous improvement, jap. kaizen) [30].

V diplomskem delu bomo predstavili kanban sistem po metodi Kanban.

Kanban ne predpisuje veliko pravil in praks, je metaproces, tj. proces, ki je namenjen izboljšanju drugih, morda že obstoječih procesov. Pri razvoju programske opreme ni pomembna trenutna izbrana metoda, saj Kanban lahko ekipe vpeljejo takoj, ko se z njim seznanijo, in postopno uvajajo izboljšave v delovni proces v kateri koli fazi projekta, ne da bi se le-ta radikalno spremenil.

Glavne prakse Kanbana so:

- vizualizacija delovnega toka;
- omejitev obsega dela;
- upravljanje delovnega toka;
- jasna pravila v procesu;
- vpeljava zanke s povratnimi informacijami;
- izboljšanje sodelovanja, spodbujanje eksperimentiranja z znanstvenimi metodami in modeli.

Temeljni principi [4]:

- začni v kateri koli fazi projekta;
- zaveži se k iskanju evolucijskih sprememb;
- na začetku spoštuj trenutne vloge, odgovornosti in delovne nazive;
- spodbujaj prevzemanje odgovornosti in vodenja na vseh ravneh organizacije, od posameznikov do vodstva.

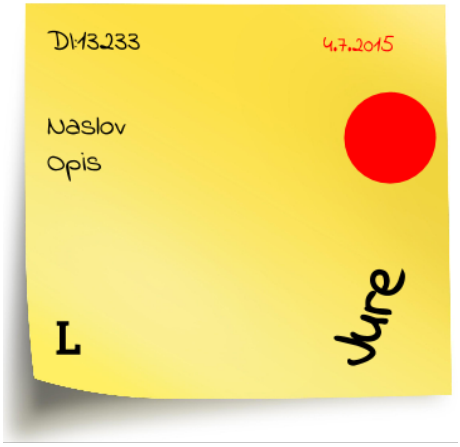
Glavno sporočilo, ki ga nosi Kanban, je, da preden se lotimo novih delovnih nalog, končajmo tiste, ki jih sedaj delamo. Tak način dela nas spodbuja, da omejimo število nalog, ki jih delamo istočasno, skrajša potrebni čas, zagotovi povratne informacije in s tem možnosti za izboljšanje procesa. Hkrati pa nam tako delo pomaga vzpostaviti enakomeren tempo razvoja, kar omogoča vsem sodelujočim, da uravnotežijo razmerje med poslovnim in zasebnim življenjem. Dobro ravnotežje pa lahko naredi podjetje na trgu dela še bolj zanimivo [1].

2.3.1 Predstavitev delovne naloge na tabli

Celotno delo, ki ga je treba opraviti, vse nove funkcionalnosti ter dodatne zahteve, ki prihajajo od nadrejenih, manjša podporna dela, usluge drugim razvojnim ekipam, vzdrževanje sistema itd., razdelimo na posamezne delovne naloge in jih na kratko opišemo na kartici ter kartice postavimo na kanban tablo. Z njihovo pomočjo prehajajo delovne naloge skozi delovni tok in za vsako kartico pokaže tabla njeno napredovanje.

Delovno nalogo predstavimo s kartico (angl. index card ali sticky note), ki lahko vsebuje različne informacije, kot so:

- naslov in opis delovne naloge, ki je lahko kratek povzetek uporabniške zgodbe;
- šifra, pod katero se delovna naloga beleži v elektronskem sistemu za vodenje in sledenje delovnim nalogam (angl. project tracking software), kjer lahko preberemo več o njej;
- dogovorjeni rok za dokončanje, če ga delovna naloga ima;
- velikost naloge, ki je lahko izražena s konfekcijsko velikostjo - S, M, L, XS in XL, številom točk ali pa velikostjo samolepilnega lističa, ki ga uporabimo na tabli;
- informacijo o lastniku kartice, ki je tisti uporabnik, ki je prevzel delovno nalogo. Na kartico lahko zapišemo ime ali uporabimo sliko oz. ikono uporabnika (angl. avatar);
- tip naloge, ki je lahko nova funkcionalnost, odprava napake, vzdrževanje ali kaj drugega;
- razred, ki pove, ali je delovna naloga standardna (angl. standard), nujna (angl. expedite), ima dogovorjen rok za dokončanje (angl. date-driven), okvara (angl. defekt), nima posebne poslovne vrednosti, vendar lahko pripomore k boljši kakovosti (angl. intangible). Razred lahko določa tudi velikost oz. pomembnost naročnika, izvor naročila, ki je lahko npr. marketing, podpora uporabnikom ali zahteva razvijalcev. Lahko pa je razred nekaj tretjega, kar pač ustreza ekipi.



Slika 2.2: Kartica

Primer kartice prikazuje slika 2.2.

Oblikovanje kartice in informacij na njej nam olajša sprejemanje odločitev in pomaga pri samoorganiziranju. Ni potrebno, da kartica vsebuje vse omenjene podatke, pomembno pa je, da so zapisane informacije prikazane jasno in so razumljive celotni ekipi.

2.3.1.1 Druge podrobnosti na kartici

Dodatno preglednost lahko dosežemo s karticami različnih barv, npr. za ločevanje delovnih nalog po tipu, kar pregledno prikaže raznolikost našega dela in razkrije potencialne težave. Prikaže lahko, da imamo v sistemu preveč napak, kar pomeni, da je treba izboljšati kakovost dela, lahko imamo premalo vzdrževalnih del, kar pomeni, da zanemarjamo tehnični del, kar se nam na dolgi rok lahko maščuje, lahko pa tudi razkrije, da imamo premalo novih funkcionalnosti, kar ni nujno vedno dobro.

Ker je dogovorjeni rok za dokončanje izrednega pomena, se ne sme zgoditi, da ga ekipa spregleda, zato ga zapišemo na kartico z drugačno barvo, tako da je bolj viden.

Druga pomembna stvar, ki mora biti poudarjena, so blokade. Ekipe lahko v ta namen uporabijo dodatne samolepilne lističe, kamor dopišejo razlog za blokado in jih prilepijo čez kartice, ali pa uporabijo magnete oz. prestavijo kartice v poseben del table. Na blokadi se lahko zapisuje še njen napredek, npr. število blokiranih dni.

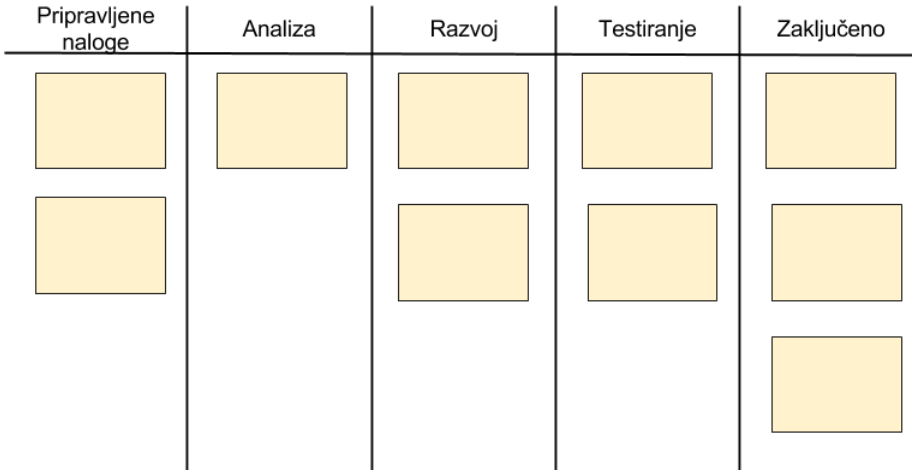
Če želimo slediti, koliko časa je delovna naloga v razvoju, lahko to informacijo prikažemo z indikatorji napredka. To so lahko preproste oznake za vsak dan razvoja, ali pa število dni do dogovorjenega roka za dokončanje.

Na kartico lahko vključimo časovne žige, ki bodo koristni za kreiranje metrik za spremljanje delovnega toka, kot so datum, kdaj je kartica vstopila v delovni tok, prešla v vsako naslednje stanje in izstopila iz delovnega toka. Na podlagi teh podatkov ekipa lahko predvidi, koliko časa potrebuje, da opravi različno velike delovne naloge.

Uporabna informacija, ki jo nosi kartica, je tudi opisovanje čustev, povezanih z delom. Počutje v ekipi, ko zaključi delo, se označi s t. i. smeškom (angl. emoticon), ki je lahko nasmejan, žalosten ali ravnodušen. Ekipa lahko te podatke spremlja in najde morebitne usmeritve oz. vzorce ter naredi takojšne spremembe. Na primer, če se na kartici pojavijo sami žalostni smeški, lahko ekipa nemudoma skliče retrospektivo in se pogovori o težavah.

2.3.2 Vizualizacija delovnega toka

Kanban tabla je tabela z različnimi navpičnimi stolpci, ki predstavljajo posamezna stanja v delovnem procesu. Ta stanja ekipa sama prepozna z analizo svojega delovnega toka od naročnikove zahteve do zaključenih funkcionalnosti v produkciji. Slika 2.3 prikazuje primer take table in vsebuje stolpce »pripravljene naloge«, »analiza«, »razvoj«, »testiranje« in »zaključeno«.



Slika 2.3: Kanban tabla

Ker je to sistem vlečenja dela, predhodno stanje v toku (angl. upstream) ne preda dela naslednjemu stanju (angl. downstream), dokler ta ne vpraša po njem. Delo se premika iz enega stanja v drugo, razvijalci, samostojno ali v paru, pa se posvečajo posameznemu stanju. Kanban tako uporablja posameznikove najmočnejše adute oz. njihovo specializacijo.

Taka vizualizacija dela ponuja vsem mimoidočim zelo pregledne in jasne informacije o delu, s katerim se ekipa in posamezniki trenutno ukvarjajo. Tabla ustvari transparentnost in pomaga razumeti, kako delo poteka ter kako učinkovito sodelujemo. Njen cilj je, da vizualno sporoča ravno prav informacij in s tem ustvari samoorganizirajoč in spodbujajoč sistem, ki omogoča članom ekipe, da prevzamejo delo brez navodil vodje.

Vizualizacija razkrije potencialne probleme v procesu, kot so npr. prepočasno prehajanje delovnih nalog skozi stanja, blokirane delovne naloge, neopravljene naloge, nejasna pravila pri predajanju in prevzemanju nalog. Tabla lahko zelo pomaga pri sprejemanju

odločitev, določanju prioritet, samoorganiziranju in izboljšanju procesa dela.

Kartice v prvem stolpcu »pripravljene naloge« lahko razvrstimo po prioriteti, kar še dodatno izboljša transparentnost. Tudi prikaz stanja »zaključeno«, kjer so zbrane vse zaključene kartice, je dobrodošel, saj nam in drugim v podjetju potrjuje, da dobro delamo.

2.3.2.1 Elektronske in fizične table

Ena od dilem, s katerimi se soočajo uporabniki, je izbira med elektronsko in fizično tablo. Kanban tabla mora biti vedno aktualna in enostavna za posodabljanje. Odločitev o tipu table mora biti prepuščena ekipi, ki naj si oblikuje tako tablo, da odraža dejanski proces dela. Tako bodo tablo laže vzeli za svojo ter spoštovali pravila in procese, ki so jih definirali skupaj [20].

Elektronska kanban tabla je univerzalno dostopna in zato posebej uporabna za tiste ekipe, ki so porazdeljene na različnih lokacijah. Elektronska tabla omogoča avtomatske izračune metrik in njihov prikaz, hrani informacije ter dodatne zapiske o delovnih nalogah. Če želi vodstvo redno spremljati različne metrike, izboljševati proces ter na retrospektivi opravljati analizo temeljnih vzrokov (angl. root-cause analysis), potem je smiselno, da ekipa že od začetka dela beleži podatke elektronsko.

Za fizično belo tablo potrebujemo le steno, samolepilne lističe, trakove in flomastre. Taka tabla omogoča več eksperimentiranja in lažje izboljševanje sistema kanban, da najbolje ustreza trenutnemu kontekstu. Fizična tabla je praviloma večja in stoji na opaznem mestu, spodbuja ekipo, da se redno sestaja v njeni bližini, je bolj enostavna za vzpostavitev in spreminjanje. Kartice na fizični tabli se lažje označi z dodatnimi informacijami, kot so ikona lastnika, zaporedne številke, indikatorji napredka ali blokade. Vendar pa so kartice omejene velikosti in lahko se zgodi, da za vse potrebne informacije zmanjka prostora. Kartice na fizični tabli lahko premikamo in če potrebujemo dodatna pojasnila, jih vzamemo s seboj na sestanke. S tem ko premikamo kartice po fizični tabli, vključimo svoja senzorična čutila in ustvarimo še močnejšo povezavo z delom, ki ga opravljamo in raje prevzamemo odgovornost zanj [20].

Informacije na elektronski tabli se načeloma ne morejo izgubiti, na fizični tabli pa lahko samolepilni lističi padejo na tla ali pa pisava ni berljiva.

Ekipam, ki se prvič spoznavajo s to metodo, se svetuje, da permanentne kanban table ne naredijo prehitro. Z učenjem procesa in preverjanjem metode mora tabla omogočati hitre in enostavne spremembe ter se prilagajati delovnemu procesu.

Ekipa vedno lahko začne zapisovati proces na fizični tabli in kasneje, ko spozna, kaj točno potrebuje, napreduje na elektronsko. Lahko pa uporabi kar obe in poskrbi za dnevno sinhronizacijo.

Ne glede na to, kakšen sistem kanban izberemo, mora biti njegov prikaz čim večji in

postavljen na opaznem mestu. Paziti je treba, da ni preveč zapleten, ker lahko povzroči zmedo in nejasnosti. Tablo je treba nujno redno posodabljati in odstranjevati nepotrebne informacije.

2.3.3 Omejevanje obsega dela

Pri scrum tabli je težava, da nam ne prepreči kopičenja delovnih nalog. Časovni okvir sicer določa mejo, koliko dela imamo lahko hkrati v razvoju, vendar pa je ta meja lahko veliko višja, kot je zaželeno.

Omejitev dela ekipo razbremeni občutka, da je preobremenjena, saj s tem, ko delamo na manj nalogah hkrati, potrebujemo manj koordinacije med njimi in dokončamo posamezno nalogo hitreje [41].

Obseg dela je celotno delo, ki mora biti opravljeno in dostavljeno naročniku, to so vse tiste delovne naloge, ki čakajo, da jih prevzamemo, vse, ki jih izvajamo ter vse, ki se morajo testirati in dostaviti naročniku. Omejitev WIP predstavlja število delovnih nalog, ki se lahko izvajajo sočasno v vsakem stolpcu oz. stanju.

Omejitev zapišemo na tablo na vrh vsakega stolpca. Novo nalogo lahko prevzamemo v naslednji stolpec, če je število kartic v naslednjem stolpcu manjše od njegove omejitve WIP. Ekipa se odloči prevzeti novo delovno nalogo na podlagi vizualizacije, ki jo omogoča tabla, npr. tip ali razred delovne naloge, obljubljen rok za dokončanje, indikatorji napredka.

Pri omejevanju WIP upoštevamo Littlov teorem, ki definira povprečno število predmetov v čakalni vrsti L kot zmnožek povprečnega števila prihodov predmetov λ v ta sistem v nekem času in povprečnega čakalnega časa W v tem sistemu za nek predmet. To pomeni, da če obdelujemo več predmetov hkrati, bomo potrebovali več časa za obdelavo vsakega posameznega predmeta. Teorem $L = \lambda \times W$ je dokazal matematik John D.C. Little.

Da lahko Littlov teorem prilagodimo terminologiji razvoja programske opreme, moramo definirati nekaj novih enot za merjenje časa v sistemu kanban. Prva je potrebni čas (angl. lead time), ki pomeni čas, potreben za realizacijo ene delovne naloge od takrat, ko jo naročnik poda, do takrat, ko jo ekipa dostavi. Naslednja je čas cikla procesa (angl. cycle time), ki označuje čas od prevzema naloge v razvoj do trenutka, ko je naloga pripravljena za izdajo [20]. Tretja metrika je prepustnost (angl. throughput), ki pove, koliko delovnih nalog opravimo v nekem obdobju [29]. Prepustnost ni indikator za napovedovanje časa, potrebnega za izvedbo določenega števila delovnih nalog v izbranem časovnem intervalu, ampak se uporablja kot indikator predvidljivosti, kako dobro gre ekipi, in za prikaz stalnih izboljšav [17].

Littlov teorem lahko prilagodimo s pomočjo prepustnosti, če v sistemu veljajo pogoji:

- zahteve odhajajo iz sistema približno enako pogosto, kot vanj prihajajo, kar pomeni, da je prepustnost enaka λ;

- vse delovne naloge v izvajanju so na koncu zaključene in zapustijo sistem;
- pri izračunu uporabljamo dosledne enote merjenja;
- povprečna starost delovnih nalog v izvajanju se ne povečuje ali zmanjšuje, kar pomeni, da v sistemu ni nujnih, blokiranih ali takih nalog, ki se izgubijo oz. jim je dovoljeno, da ostanejo v sistemu v nedogled, brez posebnega razloga;
- skupna vrednost omejitev WIP mora biti približno enaka na začetku in na koncu intervala, ki ga upoštevamo pri izračunu.

$L = \lambda \times W$
tako lahko prilagodimo v
$omejitev\_WIP = prepustnost \times cas\_cikla\_procesa$ [43].

Enoti čas cikla procesa ter potrebni čas se radi uporabljata izmenično, kar nemalokrat povzroča zmedo. Kljub temu, da obe spremenljivki govorita o času, pa se ne merita v enakih enotah. Enota za merjenje potrebnega časa je pretečeni čas v minutah, urah itd. Čas cikla procesa pa se meri s pretečeno količino časa na neko enoto, npr. minute na naročnika ali ure na funkcionalnost. V diplomskem delu bomo Littlov teorem zapisali s pomočjo potrebnega časa.

$potrebni\_cas = \frac{omejitev\_WIP}{prepustnost}$

Iz teorema izhaja, da čim manjša je omejitev WIP, tem hitreje prehajajo kartice skozi delovni tok, kar pripelje do krajšega potrebnega časa. Krajši potrebni čas pomeni hitrejše končevanje dela in s tem večjo agilnost, saj lahko zahteve dostavljamo eno za drugo in ne več po en velik paket naenkrat. Produkt je zaradi pogostih izdaj bolj konkurenčen in se hitreje prilagaja spremembam na trgu, kar ustvarja večje zaupanje zunanjih partnerjev, še posebej marketinga in naročnika projekta. S pogostim izdajanjem hitreje dobivamo povratne informacije in s tem priložnost, da se učimo iz napak in izboljšamo svoj proces, da bo tekel hitreje. Kadar dobivamo povratne informacije z večjo zakasnitvijo, nam je težje prepoznati, zakaj je prišlo do nastalih težav, in se tako težje učiti iz napak.

Daljši potrebni čas je povezan s slabo kakovostjo dela. David J. Anderson navaja, da šestinpolkratno povečanje potrebnega časa pripelje do več kot tridesetkratnega povečanja okvar [1]. Visoka kakovost izdane kode gradi zaupanje med ekipami v organizaciji, kot so sistemska podpora, tehnična podpora in prodaja.

Pogosto izdajanje lahko še zmanjša potrebo po natančnih ocenah težavnosti uporabniških zgodb in poenostavi določanje prioritete novih zahtev, kar dodatno razbremeni ekipo.

Ekipa se drži omejitve WIP tako, da ko je ta v nekem stolpcu dosežena, ne prevzame nove delovne naloge, ampak pomaga preostalim v ekipi, da hitreje končajo naloge, ki so že v delu, ali pa skušajo prepoznati zastoje oz. ozka grla (angl. bottleneck) in najti načine

za njihovo odpravo. Zastoj pomeni, da je nek stolpec poln, njegovo naslednje stanje pa je prazno. Z nastavitvijo omejitve WIP želimo doseči čim manj zastojev.

Ekipe lahko uporabijo svoje ikone za omejevanje obsega dela na ravni posameznika, kar je uporabno za tiste člane, ki se radi vključujejo v vsako delo, ali za tiste projekte, kjer je dobro, da neko delovno nalogo spremlja ista oseba od prevzema do izdaje. S tem, ko vsakemu članu dodelimo določeno število ikon, ki jih lahko pritrdi na tablo, preprečimo, da bi si kdo naložil preveč dela. Taka omejitev se lahko določi tudi s posebnimi vrsticami na tabli (angl. swim lane), kjer je vsakemu posamezniku dodeljena lastna t. i. plavalna proga, omejitev pa je postavljena na nivoju vrstice.

Omejitev WIP je lahko izražena tudi kot seštevek velikosti vseh delovnih nalog v stolpcu. V tem primeru se lahko prevzame nova delovna naloga le, če njeno število točk skupaj s številom točk vseh nalog v stolpcu ne presegajo omejitve WIP stolpca.

2.3.4 Upravljanje delovnega toka

Ko smo analizirali svoj delovni tok in vzpostavili njegovo vizualizacijo, ga lahko začnemo izboljševati. Vizualizacija pomaga prepoznati zastoje in jih odpraviti, delovni tok se da vedno še izboljšati.

2.3.4.1 Kako postavimo omejitev WIP

Delovne naloge bodo bolj gladko prehajale skozi tok s pravilno optimizirano omejitvijo WIP. Nizka omejitev WIP skrajša potrebni čas, naloge prehajajo skozi tok hitreje, kar pomeni, da smo kot ekipa bolj učinkoviti. Prenizka omejitev WIP pa lahko pripelje do slabše izrabe virov (angl. idle people), nizke produktivnosti ali vodi do rezerv virov (angl. slack). Prevelike rezerve so nezaželene, saj večina podjetij ne želi plačevati zaposlenih, ki čakajo na delo. Pri omejitvi 1 lahko do čakanja pride že zaradi predajanja dela ali posameznikov, ki niso v službi in ne morajo prevzeti dela, zato taka omejitev ni ustrezna za vsako razvojno ekipo. Po drugi strani pa zaradi previsoke omejitve WIP delovne naloge stojijo (angl. idle tasks), kar slabša potrebni čas.

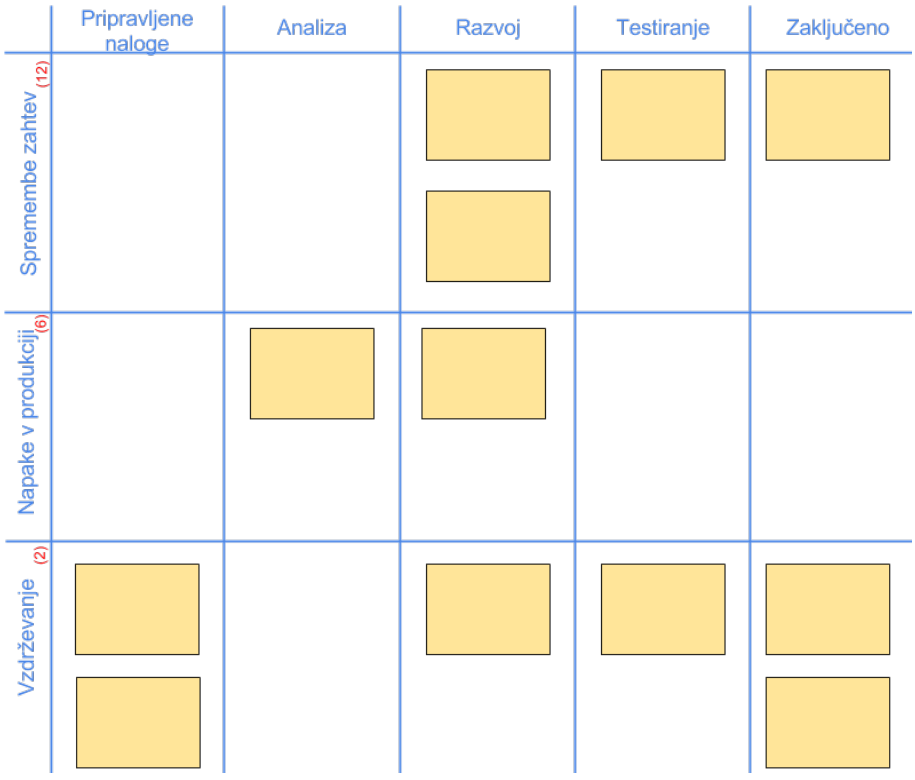
Omejitev WIP je torej treba uravnotežiti tako, da najdemo pravo razmerje med hitrim prehajanjem nalog skozi delovni tok in dobro izrabo virov.

Pri omejevanju obsega dela Kanban ne predpisuje pravil, spodbuja pa diskusijo. Omejitev WIP se postavlja izkustveno, na podlagi že doseženih rezultatov. Če se pokaže, da ekipa pogosto prekrši omejitve in v naslednjih stanjih nastajajo zastoji, je morda čas, da se omejitev WIP zviša in preveri, kako to vpliva na prehajanje nalog skozi delovni tok. Po drugi strani pa je dobro omejitev znižati, če je ekipa ne dosega, in v sistemu ni zastojev, kar bo pripeljalo do rezerv virov in s tem priložnosti za diskusijo in izboljšanje. Brez rezerve si člani ekipe praviloma ne morajo privoščiti časa za razmislek o opravljanju svojega dela

in načinih, kako bi ga lahko opravili bolje. Rezerve so potrebne za izboljšanje odzivnosti za nujne primere, izboljšanje procesa, kar vodi v stalne izboljšave. So tudi odlična priložnost za preverjanje, ali ima ekipa za uspešno zapolnjevanje vrzeli dovolj strokovnjakov z različnih področij. Vendar morajo biti tudi rezerve na neki način omejene in ko to omejitev dosežemo, je treba razmisliti, kako lahko v prihodnje proces izvajamo s čim manj izkoriščenimi rezervami [9].

Kanban ne predpisuje omejitve WIP za vsako stanje. Omejitev WIP razdelimo med tiste stolpce, za katere menimo, da bodo vplivali na lepše prehajanje nalog skozi delovni tok. Omejitev je odvisna od velikosti razvojne ekipe in njene razpoložljivosti, velikosti delovnih nalog ter želje po nenehnem izboljšanju organizacije.

Opustitev omejevanja obsega dela ni zaželena, saj tak način dela največkrat pripelje do preobremenitve celotne ekipe ter odstrani njeno željo po izboljšanju dela. Ekipe, ki so vpeljale omejitev na začetku svojega procesa, so poročale o pospešeni rasti zmogljivosti in organizacijske zrelosti, boljših poslovnih rezultatih ter pogosti predvidljivi dobavi zelo kakovostne programske opreme. Na drugi strani pa so imele ekipe, ki so odlašale z vzpostavitvijo omejitve WIP, večinoma težave in so dosegle slabše izboljšanje [1].



Slika 2.4: Razdelitev kapacitete dela po tipu delovne naloge.

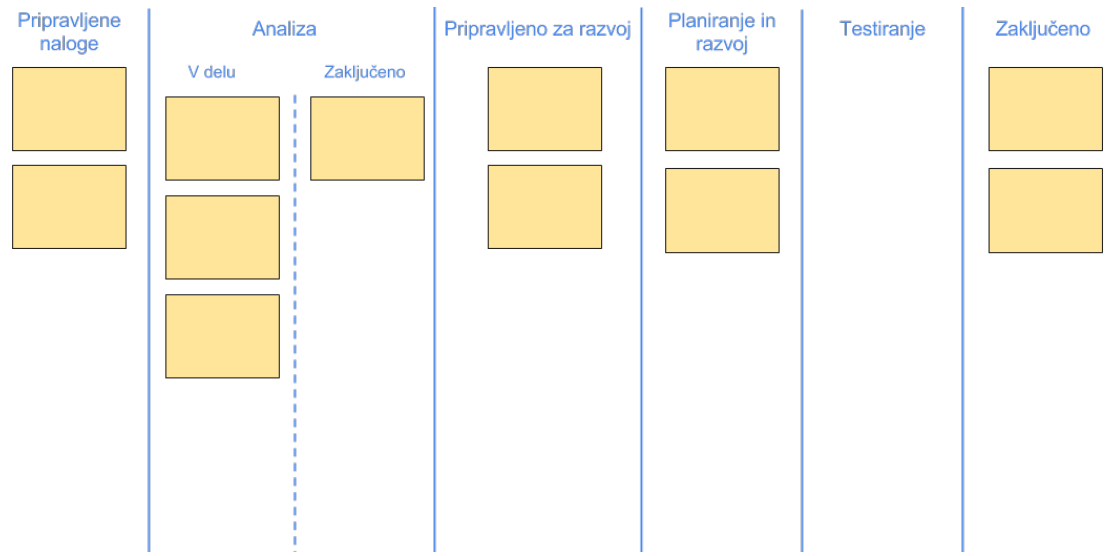
Za začetek se omejitev WIP lahko nastavi na število, ki predstavlja velikost ekipe. Lahko se jo nastavi na dvakratno vrednost velikosti ekipe in ustrezno zmanjšuje vsakih

nekaj tednov, lahko se uporabi tehniko, ki jo je predstavil Don Reinertsen na Lean Kanban konferenci 2012, ki povprečno število delovnih nalog, na katerih ekipa dela v neomejenem sistemu, podvoji in potem zmanjšuje vsakih nekaj tednov za od 20 % do 30 %, dokler se ne pokažejo težave, kot so dolge čakalne vrste in rezerve [74]. Ni tako pomembno, da najdemo najbolj optimalno omejitev v prvem poskusu, bolj pomembno je, da izberemo neko število in to število spreminjamo z učenjem. Iščemo tako omejitev, ki nas žene h končanju dela in izboljšanju procesa.

Po vzpostavitvi omejitve WIP v našem sistemu lahko dodatno razdelimo kapaciteto dela po tipu ali razredu delovne naloge. Kapaciteto porazdelimo med vse tipe in jih prikažemo s plavalnimi progami, na primer 60 % za spremembe zahtev, 10 % za vzdrževanje ter 30 % za napake pri produkciji, kar omogoča, da se lahko posvetimo vsem tipom zahtev, ki vstopijo v sistem kanban. Primer take uporabe je prikazan na sliki 2.4.

2.3.4.2 Čakalne vrste in izravnalniki

V delovni tok lahko vključimo dodatne stolpce, ki služijo kot čakalne vrste (angl. queue), ki izboljšajo predajo delovnih nalog v naslednje stanje. Primer uporabe čakalnih vrst je predstavljen na sliki 2.5. Vrste dodajamo pred tista stanja, ki imajo omejene vire, ali tam, kjer si želimo vizualizirati razliko med čakanjem in delom. Lahko se uporabljajo tudi kot signal, da je neko delo pripravljeno in se lahko prevzame v naslednje stanje. Primeri takih stolpcev so »pripravljene naloge« – prvi stolpec na tabli, »pripravljeno za razvoj« – signalizira, da je analiza končana in da je naloga pripravljena za razvoj, »razvoj zaključen« – signalizira, da je naloga razvita in pripravljena za testiranje.



Slika 2.5: Čakalne vrste in izravnalniki

Poleg čakalnih vrst lahko dodamo v delovni tok še stolpce, ki pomagajo izravnavati tok

(angl. buffer). S tem ko jih dodamo pred zastoje nam omogočajo, da lahko nadaljujemo delo. Vrste in izravnalniki pripomorejo k bolj gladkemu toku dela in bolj predvidljivemu potrebnemu času.

Čakalne vrste lahko dodamo na tablo, kot samostojne stolpce z lastno omejitvijo WIP ali pa razdelimo stolpec na dva podstolpca, za katerega velja skupna omejitev WIP.

Vrste in izravnalniki večajo omejitev WIP v sistemu in podaljšujejo potrebni čas, zato je dobro, da so omejitve teh stolpcev čim manjše.

2.3.4.3 Nujne delovne naloge

Nujna delovna naloga (angl. expedite ali silver bullet) je tista, za katero je smiselno prenehati delo v izvajanju, in se posvetiti njenemu čimprejšnjemu končanju. Lahko je to odprava napake na produkcijskem okolju ali sprememba pogodbe, ki jo je treba vgraditi v programski produkt, poslovna priložnost, ki se je ne sme izpustiti itd. Kakor koli že, posebnih nujnih primerov se ne sme zanemarjati.

Delovne naloge, ki so nujne, lahko obravnavamo v posebni plavalni progi (angl. expedite swim lane). Zahteve, ki so nujne, vstopajo v delovni tok prioritetno. Obravnava takih zahtev upočasni drugo delo, zato je treba sprejeti nekaj pravil, da ne bo prihajalo do zlorab, npr. dodatnega dela, ki prihaja od vsepovsod. Taka pravila so lahko, da se sme obravnavati le en nujni primer naenkrat, da se lahko obdela največ dva nujna primera na teden, da ne prištevamo dela iz plavalne proge v omejitev WIP, da nujni primer ne sme prekiniti dela v izvajanju itd.

Ko v sistem vstopi nujni primer, se mu lahko posveti celotna ekipa z razlogom, da ga čim prej odpravi (angl. swarming). Swarming je koristen tudi za odpravo blokad, okvar, reševanje prevelikih kompleksnih nalog ali v primeru, ko dosežemo omejitev WIP. Ko je nujni problem odpravljen, lahko ekipa izvede retrospektivo, kjer analizira obravnavo tega primera, vzroke zanj in kako se mu lahko v prihodnosti izogne.

2.3.4.4 Napake

Napake lahko obravnavamo na dva načina. V obeh primerih pomeni odprava napak dodatno delo, zato se mora delo prištevati omejitvi WIP.

Prva možnost je, da kartico z delovno nalogo, ki jo je potrebno dodelati, prestavimo nazaj v stolpec »razvoj«, kjer je obravnavana na enak način kot preostale. Tak pristop je bolj mehak, saj ne obravnava napake urgentno in je tako manj moteč. V primeru, da je omejitev WIP v stolpcu »razvoj« že dosežena, moramo dodelavo prestaviti v čakalno vrsto, kjer mora počakati, da pride na vrsto, kar je lahko slabost.

Druga možnost je, da kartico z delovno nalogo, ki jo je potrebno dodelati, prestavimo na posebno mesto za dodelave (angl. rework station) in jo obravnavamo kot nujno delovno

nalogo. To je bolj trd pristop, saj obravnavamo napake nemudoma, ko jih odkrijemo.

2.3.4.5 Drugi načini za izboljševanje toka

Da bo delo skozi delovni proces teklo bolj gladko in ne bo povzročalo morebitnih blokad je priporočeno, da se zahtevne delovne naloge razdelijo na enako velike manjše naloge. Z enako velikimi nalogami dosežemo bolj enakomeren tok in boljšo predvidljivost, kar pomaga graditi zaupanje in zmanjšuje potrebo po posebnih nujnih primerih.

Tok lahko dodatno izboljšamo:

- s planiranjem;
- z vnaprej pripravljenimi dodatnimi viri, ki lahko v primeru, da je potreba, vskočijo in pomagajo ekipi;
- z ustrezno načrtovanim procesom, ki omogoča medsebojno sodelovanje;
- z odstranjevanjem blokad in tako, da prosimo druge za pomoč;
- s skupinskim reševanjem problemov, t. i. swarmingom;
- z uporabo analize temeljnih vzrokov;
- z izogibanjem dodelav, kar lahko dosežemo s tehnikami programiranja v paru, testno vodenim razvojem, sprotno integracijo ter avtomatizirani testi;
- z razvojem le tistih rešitev, ki jih naročnik potrebuje;
- z ustrezno sestavljeno razvojno ekipo strokovnjakov z vseh področij, ki lahko prevzame svojo funkcionalnost in jo spremlja do izdaje (angl. feature team);
- s tehniko časovnega okvirja;
- z odstranjevanjem zastojev z modelom teorije omejitev;
- z določenimi jasnimi pravili, na kakšen način prevzemamo nove naloge;
- s soočanjem z dnevnimi problemi na dnevnih sestankih.

2.3.4.6 Časovni okvir

Določanje časovnega okvirja (angl. timeboxing) je osnovni princip, ki zagotavlja, da je projekt dokončan v časovnih rokih in v okviru proračuna. Je tehnika, ki se nam pomaga osredotočati na pomembne naloge. Kanban ne predpisuje časovnega okvirja, uporaba le-tega pa je možna s postavljanjem rokov za dokončanje, različnimi razredi kartic ali z dogovorom o ravni storitve (angl. Service Level Agreement – SLA).

2.3.4.7 Dnevni sestanki (angl. daily standups)

To so hitri stoječi sestanki pred kanban tablo, ki potekajo ob isti uri, kjer se ekipa koordinira, izmenjuje informacije in se uči. Skupaj se posvetijo delu in dogajanju na tabli in prepustijo razprave, ki se ne tičejo celotne ekipe, za drug sestanek. Delovnim nalogam na tabli se posveča po načelu vlečenja, kar pomeni, da se sprehajamo in pogovarjamo o delovnih nalogah od desne proti levi, torej začnemo pri tistih delovnih nalogah, ki so bližje končnemu stanju.

Vprašanja, ki si jih lahko zastavimo, so:

- Kaj lahko naredimo, da delovno nalogo prestavimo bližje končnemu stanju?
- Kdo bo prevzel to nalogo?
- Kdo od članov ekipe dela na delovni nalogi, ki še ni predstavljena na tabli?
- Ali dela ekipa na najbolj pomembnih nalogah?
- Ali je kaj takšnega, za kar mora vedeti celotna ekipa?
- Ali je vse razumljivo oz. ali so na tabli kake nejasnosti?

2.3.4.8 Retrospektiva

Retrospektive je sestanek, katerega cilja je izboljšanje procesa s proučevanjem le-tega in učenjem iz njega. Na vsakem sestanku želimo poiskati nekaj sprememb, ki jih lahko vpeljemo pred naslednjo retrospektivo. Ostati želimo le pri toliko spremembah naenkrat, kot jih lahko prenesemo. Tako omejujemo WIP in se izboljšujemo s pogostimi manjšimi spremembami. Dnevni red sestanka je vzpostavljanje pravega razpoloženja za sestanek, zbiranje podatkov in dogodkov za analizo v obdobju po zadnji retrospektivi, analiziranje teh podatkov in ugotovitev sklepov, sprejetje sprememb za izboljšanje pred naslednjo retrospektivo ter za konec hiter povzetek retrospektive in izmenjava mnenj o poteku sestanka.

2.3.5 Jasna pravila v procesu

Ljudje radi organiziramo svoje delo na podlagi rutin, nenapisanih načel in predvidevanj. Ker imamo vsi svoja pričakovanja in si po svoje razlagamo stvari, lahko večkrat prihaja do nesporazumov [20]. Da bi lahko začeli diskusijo o izboljšanju dela, reševanju nedoslednosti in konfliktov, moramo imeti v procesu definirana nedvoumna načela. Taka diskusija se dela na podlagi objektivnih podatkov oz. rezultatov, ki nam jih pokaže vizualizacija delovnega toka, in ne na podlagi lastnega razmišljanja, čustev ali nepreverjenih podatkov. Načela niso pravila, ki jih je treba slepo upoštevati, so priporočila, ki se lahko po ustrezni diskusiji tudi prekršijo.

Ekipa ima lahko težave, če se njeni člani v nekem trenutku zaradi pomanjkanja informacij odzovejo popolnoma drugače. Da bi se izognili nesporazumom, je smiselno definirati izstopna in vstopna merila za prehajanje delovnih nalog skozi stanja na tabli. Merila lahko zapišemo na tablo in vsakič, ko želimo delovno nalogo premakniti, preverimo, ali jim ustreza. Na dnevnih sestankih in retrospektivah merila preverjamo in po potrebi prilagajamo. Taki merili sta npr., da delovne naloge ne smejo preiti v stolpec za testiranje, dokler jih niso pregledali drugi člani ekipe oz. so bile predstavljene celotni ekipi, ali da je stabilnost produkcije bolj pomembna od vzdrževanja, vzdrževanje pa je bolj pomembno od razvoja novih funkcionalnosti.

Kombinacija jasnih načel, transparentnosti in vizualizacije dela pooblasti člane, da sprejemajo lastne odločitve in sami upravljajo tveganje (angl. risk management). Vodstvo bolj zaupa ekipi in procesu, ko spozna, da ta temelji na načelih, ki pomagajo zmanjševati tveganje in uresničevati pričakovanja naročnikov [1].

2.3.6 Planiranje, določanje prioritete in ocenjevanje težavnosti

2.3.6.1 Planiranje (angl. planning)

Planiranje je aktivnost, ki se izvaja vnaprej. Zaradi pogostega spreminjanja zahtev, planiranja ni dobro izvajati prezgodaj, da ga ne bi bilo treba ponovno ponoviti. Ker nosi določene stroške, ga tudi ne želimo izvajati prepozno ali prepogosto. Planirati želimo ravno ob pravem času. Če izdajamo verzije redno, potem je smiselno, da tudi planiramo redno. Dobro je vedeti, kaj bomo delali, vendar pa preveč planiranja veča omejitev WIP.

Na sestanku za planiranje izdaje (angl. release planning meeting) pripravimo naslednje delovne naloge za razvoj. Čim več imamo delovnih nalog v sistemu, tem večjo omejitev WIP imamo, naloge počasneje prehajajo skozi tok, kar pomeni, da potrebujemo več časa, da jih končamo in smo manj agilni. Ohraniti želimo nizke stroške planiranja in pripraviti minimalno dolg seznam vnaprej pripravljenih delovnih nalog za razvoj.

V ta namen nam lahko koristi vpeljava planiranja, ki je vodeno dogodkovno (angl. event driven planning), kar pomeni, da nam bodo dogodki v procesu dali signal, kdaj je traba začeti neko aktivnost. Tak sistem za signalizacijo se lahko vizualno prikaže na tabli, npr. s črto med karticami v stolpcu »pripravljene naloge«, ki označuje mesto (angl. order point), kjer še lahko prevzamemo zadnjo delovno nalogo, preden skličemo sestanek za planiranje.

2.3.6.2 Ritem (angl. cadence)

Ritem zagotavlja predvidljivost. Z rednim ritmom lahko merimo in napovedujemo naš napredek k ciljem. Medtem ko ima Scrum, kjer delo poteka v iteracijah, le en sam ritem, Kanban uporabe ritmov nima definirane. Glavni poudarek je na prehajanju delovnih nalog

skozi tok, ki je predstavljen s tablo.

Pri Kanbanu se lahko uporablja več ritmov, kar pomeni, da ima lahko vsaka aktivnost, komuniciranje in sodelovanje svoj ritem. Planiranje se lahko izvaja v rednih intervalih ali pa je vodeno dogodkovno. Tudi retrospektive, prezentacije ter proslavljanje uspehov imajo lahko svoj ritem.

S krajšanjem dolžine sprinta se Scrum približuje Kanbanu. Metodi, kjer se Kanban prilagodi Scrumu, rečemo Scrumban.

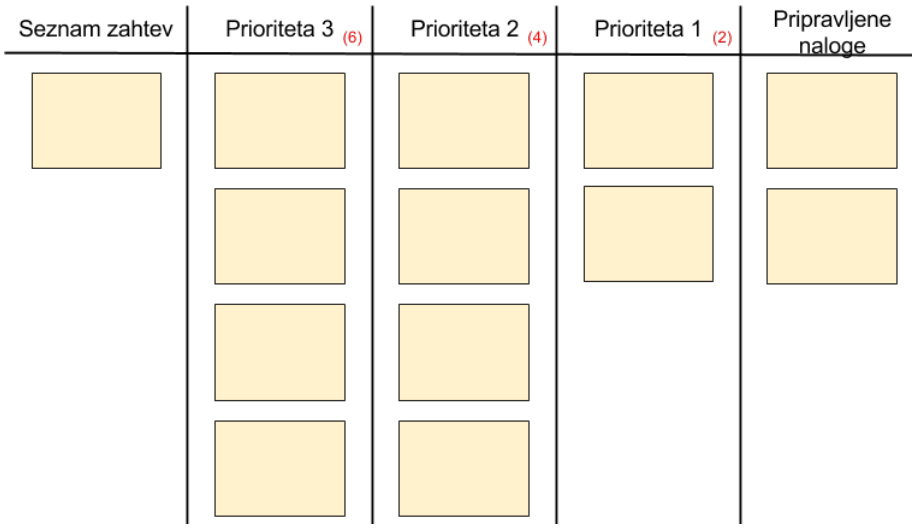
2.3.6.3 Določanje prioritet (angl. backlog prioritization)

Določanje prioritet delovnim nalogam je zelo pomembno, saj tako vizualiziramo pomembno delo in razvijamo po ustreznem vrstnem redu. Ta aktivnost se odvija na sestanku za določanje prioritet (angl. queue replenishment meeting). Ritem tega sestanka vpliva na velikost seznama zahtev in s tem tudi na potrebni čas sistema, zato je priporočeno, da so sestanki čim bolj pogosti. Nekatere ekipe se raje odločajo za take sestanke, ki so sklicani glede na povpraševanje [1].

Ker je določanje prioritet pomembno opravilo, se veliko organizacij temu vedno bolj posveča in išče nove tehnike, s katerimi bo delo potekalo bolj učinkovito. Corey Ladas je avtor vizualne tehnike Priority Filter, predstavljene na sliki 2.6. To je dodatna tabla, razdeljena na več stolpcev, ki imajo lastno omejitev WIP. Stolpci v tabli so »seznam zahtev« ter stolpci za »prioriteto 3, 2 in 1«, kamor razvrstimo delovne naloge v neurejenem vrstnem redu. Ko se sprosti kapaciteta za prevzem novega dela, se vprašamo, katero je naslednje najbolj pomembno delo iz stolpca s »prioriteto 1«, in ga prestavimo na kanban tablo v stanje »pripravljene naloge«. Na enak način prestavimo delovno nalogo iz stolpca s »prioriteto 2« v »prioriteto 1« ter enako za stolpec s »prioriteto 3« in stolpec »seznam zahtev«. Zahteva iz seznama zahtev se spremni v delovno nalogo šele, ko je umeščena v stolpec s »prioriteto 3« [9].

Če zmoremo planirati dovolj hitro in dobro ter nam določanje prioritet več zahtevam hkrati ne prinaša nobenih prednosti, potem lahko uporabimo seznam zahtev velikosti ena. V trenutku, ko delo prevzame razvojna ekipa, je načrtovalni ekipi poslan signal, da ta izbere naslednjo nalogo. Če je načrtovalna ekipa dovolj hitra, razvojni ekipi ni treba čakati. Kočno stanje h kateremu stremimo, je planiranje po potrebi (angl. prioritization-on-demand). Enako logiko lahko vpeljemo v ritem izdajanja verzij. Ko najdemo optimalno velikost serij zahtev za izdajanje, jo skušamo še izboljšati in rezultat našega truda je lahko na koncu izdajanje zahtev po potrebi (angl. features-on-demand) [9].

Neke vrsto tolažbo za naročnika oz. deležnike projekta lahko daje prikaz pričakovanega čakalnega časa delovne naloge v seznamu zahtev (angl. Disney wait times). Do pričakovanega časa pridemo po vzoru čakalnih časov v zabaviščnih parkih. Zanesljiv izračun



Slika 2.6: Tehnika Priority Filter

pričakovanega časa veča predvidljivost ekipe in dostavljanja izdaj, kar posledično poveča zaupanje naročnika v ekipo [20]. Pričakovani čas lahko določimo iz izmerjenega potrebnega časa za manjše, srednje ter velike naloge. Dodatno lahko v odstotkih spremljamo, koliko delovnih nalog uspešno končamo v izmerjenih potrebnih časih in prilagodimo pričakovani čas.

2.3.6.4 Ocenjevanje (angl. estimation)

Ocenjevanje težavnosti uporabniških zgodb nam pomaga oceniti, kdaj bo neka funkcionalnost končana oz. koliko truda moramo vložiti, da jo končamo. Ocenjevanje je velikokrat napačno, saj je težko napovedati prihodnost. Kljub temu se izkaže uporabno pri upravljanju tveganj, prepoznavanju negotovosti, pri zmotnih prepričanjih, nedoslednosti ter drugih zadevah, ki jih je treba bolj podrobno preveriti. Dobro je, da razumemo delo, ki ga moramo opraviti, vendar pa je tudi pomembno, da se z ocenjevanjem ne ukvarjamo predolgo in preveč natančno.

Tako kot pri Scrumu, se lahko uporablja za ocenjevanje težavnosti uporabniških zgodb relativna ocena – število točk, ki predstavlja primerjavo velikosti različnih zgodb in ne točne ocene, izražene v številu dni ali ur. Točke so lahko izražene npr. s števili iz Fibonaccijevega zaporedja. Težavnost lahko ocenimo tudi brez številk, npr. z oznako konfekcijske velikosti. Obstaja kar nekaj tehnik, ki se uporabljajo za ocenjevanje težavnosti uporabniških zgodb, vse pa v ekipi spodbujajo vprašanja, diskusijo in sodelovanje.

V Kanbanu pretirano ocenjevanje predstavlja odpadke, zato nas ta metoda spodbuja, da uporabimo alternative. Namesto, da govorimo v dnevih, raje govorimo o verjetnosti. To pomeni, da se ne osredotočamo na spremljanje zaključenih uporabniških zgodb, ampak

raje na dostavljanje zanesljivih stabilnih verzij z neko verjetnostjo, v izbranem obsegu (angl. scope) z viri, ki so nam na voljo [17]. Hitrost ekipe in tudi potreben čas nam dajeta informacije le o preteklih dogodkih, zato se napovedovanja (angl. forecasting) lotimo empirično, za kar so nam v pomoč znanstvene tehnike, kot sta npr. zakonitosti Littlovega pravila iz teorije čakalnih vrst ali simulacijsko metodo Monte Carlo. Na spletu se je razvilo gibanje The NoEstimates Movement, ki zagovarja tak način napovedovanja [20].

2.3.7 Stalne izboljšave in spoštovanje

Uvedba Kanbana sama po sebi ne zagotavlja stalnih izboljšav in medsebojnega spoštovanja, razkrije pa priložnosti za izboljšanje na teh dveh področjih. Stalne izboljšave je skupni izraz za več metod za razvoj in uvajanje t. i. dobrih sprememb in najboljših praks za stalno izboljševanje izdelkov in procesov z izrabo znanja, izkušenj in veščin vseh zaposlenih v podjetju. Možni načini za spodbujanje t. i. kaizen kulture so redne retrospektive, spontani kaizen sestanki (angl. the after meeting), kjer se člani ekipe po dnevnih sestankih samoiniciativno pogovarjajo o izboljšanju procesa, analiza temeljnih vzrokov, ki pomaga poiskati, od kod izvirajo nastali problemi, ter kanban kata, ki z vnaprej definiranimi vedno enakimi vprašanji trenira ekipo, da sama izboljša probleme v procesu.

Kanban z razkrivanjem problemov, osredotočanjem na njihovo reševanje ter preprečevanjem ponavljanja olajša nastanek zaupanja vredne, kompetentne, medsebojno sodelujoče ter nenehno napredujoče organizacije.

Zaposleni v kaizen kulturi so pogumni posamezniki, ki znajo prevzeti pobudo in delajo dobronamerno. Počutijo se varni, ni jih strah odkrito govoriti o problemih, ponosni so na svojo strokovnost in dosežke ter želijo postati še boljši. Uspeh ekipe jim je bolj pomemben od lastnega. Spoštujejo drug drugega ne glede na svoj položaj v organizaciji. Vzpostavitev Kanbana praviloma pripelje do takih kulturnih sprememb in organizacijam pomaga dozoreti [1].

2.3.8 Vpeljava zank s povratnimi informacijami

Povratne informacije so pomembne za uspešno upravljanje delovnega toka. Če želimo tekoče prehajanje nalog, moramo imeti vzpostavljene zanke s povratnimi informacijami. Primeri takih zank so: programiranje v paru, vzpostavljen ritem analiziranja metrik, dnevnih sestankov, retrospektiv, spontanih kaizen sestankov, sestankov za določanje prioritete, sestankov za negovanje seznama zahtev izdelka (angl. backlog grooming ali backlog triage).

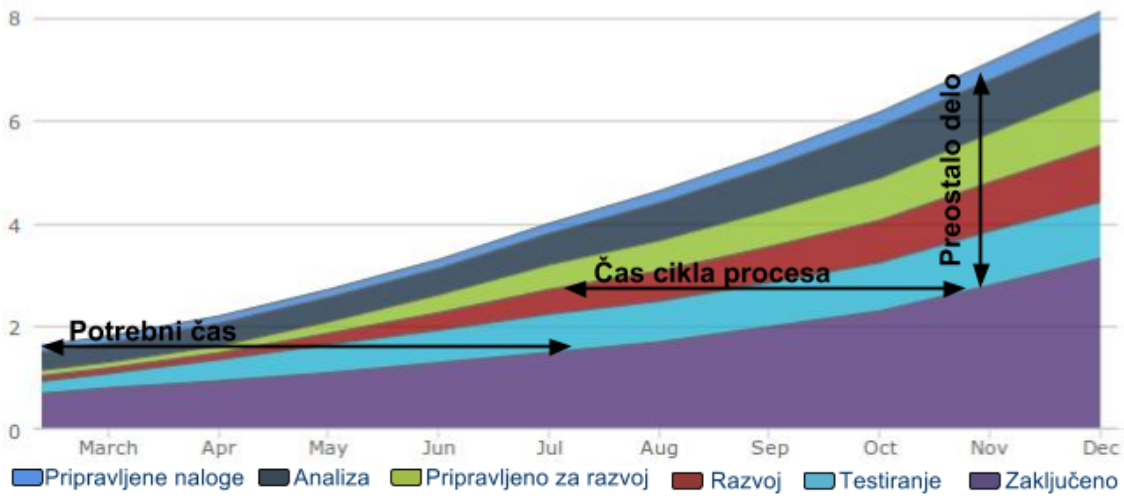
2.3.9 Izboljšanje sodelovanja, spodbujanje eksperimentiranja z znanstvenimi metodami in modeli

Ta princip spodbuja ekipe, da se izobražujejo in iščejo načine za izboljševanje svojega procesa z modeli, kot so vitko razmišljanje (angl. Lean Thinking), sistemsko razmišljanje iz teorije sistemov (angl. Systems Thinking), o katerih pišeta Russel L. Ackoff v svoji knjigi Differences That Make a Difference in Peter Senge v knjigi The Fifth Discipline, teorija omejitev (angl. Theory of Constraints – TOC), o kateri piše Eliyahu Goldratt v svoji knjigi The Goal ali teorija čakalnih vrst (angl. Queuing Theory), o kateri piše Donald G. Reinertsen v svoji knjigi The Principles of Product Development FLOW.

2.3.10 Metrike za spremljanje napredka

Kanban ne predpisuje metrik za spremljanje delovnega toka, jih pa tudi ne prepoveduje. V Kanbanu nas manj zanima spremljanje, ali projekt poteka po načrtu oz. ali bo pravočasno končan. Bolj pomembno je, da nam meritve pokažejo, da je naš sistem predvidljiv, da dela, kot je bil načrtovan, da je naša organizacija agilna, se odziva na spremembe, osredotoča na tok ter stremi k stalnim izboljšavam.

Ekipa se mora sama odločiti, katero metriko želi uporabljati za spremljanje svojih rezultatov. Metrike prikažemo na vidnem mestu, da spodbujajo razmišljanje ter vprašanja. Uporabimo lahko metrike za merjenje potrebnega časa, čas cikla procesa, števila opravljenih delovnih nalog v nekem obdobju, števila okvar in blokad, števila opravljenih nalog glede na dogovorjeni rok za dokončanje itd.



Slika 2.7: Kumulativni diagram delovnega toka

Prikažemo jih lahko z različnimi tipi diagramov, pogosta je uporaba bolj naprednega

diagrama – kumulativnega diagrama delovnega toka (angl. Cumulative Flow Diagram – CFD), ki spremlja za vsak dan posebej število nalog v posameznih stolpcih. Iz diagrama, primer na sliki 2.7, se da razbrati povprečne potrebne čase, čase cikla procesa, velikost seznama zahtev in spremljati omejitev WIP oz. število nalog v izvajanju ter druge podrobnosti, ki pomagajo prepoznati področja za izboljšanje procesa.

2.3.11 Skalabilnost

Zaradi narave dela v sistemu vlečenja dela, kjer manjše spremembe pogosto izdajamo naročniku, je Kanban idealen za dela, ki spadajo k vzdrževanju programske opreme. Sistem s karticami in hitrimi naročili novih zahtev je prav tako ustrezen za službo za podporo uporabnikom in operativo.

Kanban pa lahko prilagodimo tudi razvoju večjih projektov, kjer se držimo enakih načel. Vzpostavimo omejitev WIP, uporabljamo načela vlečenja dela in vizualiziramo delo. Upoštevamo druga načela vitkosti in agilnosti ter vrednote in začnemo pri obstoječem delovnem toku in procesu. Želimo izdajati verzije pogosto, po določeni prioriteti, upravljati tveganje, napredovati kljub nepopolnim informacijam, zgraditi kulturo z visoko stopnjo zavesti in se hitro odzivati na spremembe [1].

V knjigi Cory Ladasa je zapisano, da je optimalna velikost kanban ekipe 25 članov. Za večje projekte, kjer delamo z ekipami, ki imajo več kot 50 članov, pa je mogoče delo organizirati vzporedno v manjših celicah, a je treba poskrbeti za enotnost med njimi [9].

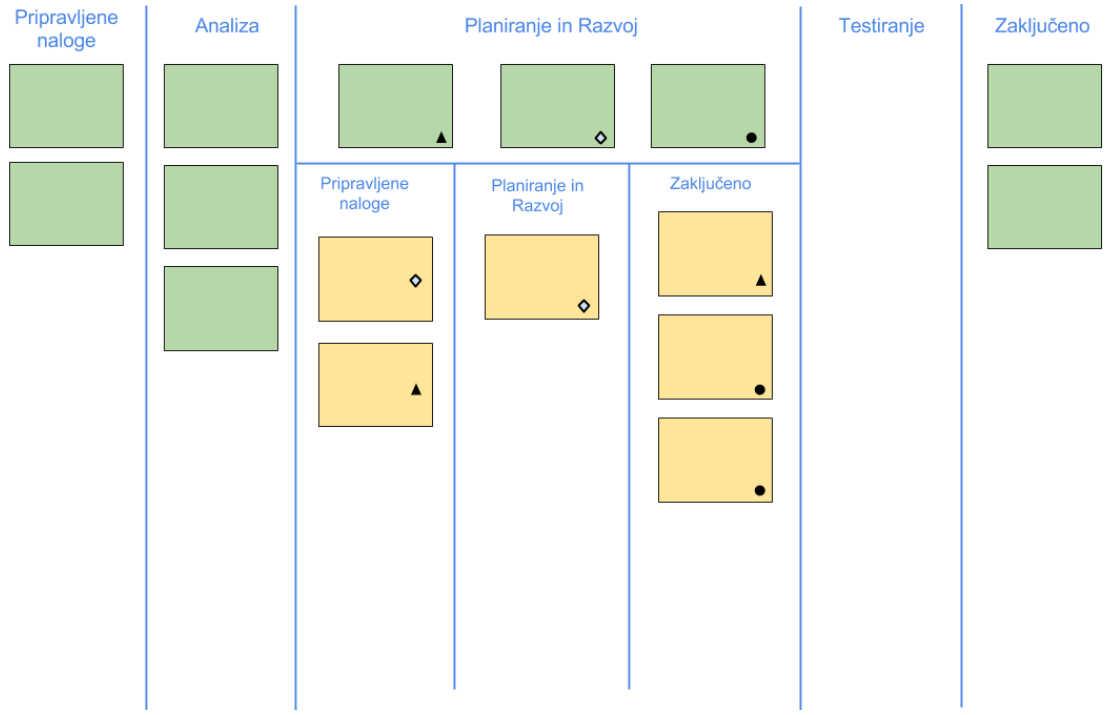
Zahtevano delo se lahko razdeli na različne tipe delovnih nalog. Če t. i. epske zgodbe razdelimo na manjše, enako velike uporabniške zgodbe in te naprej na še manjša opravila (angl. task), imamo trinivojsko hierarhijo in tri tipe delovnih nalog. Kanban fizična tabla se praviloma uporablja za prikaz prvih dveh nivojev. Če je definiran tretji nivo, se ta običajno ne beleži na sami tabli, ampak na pomožni lokaciji, saj se ne osredotoča na trud in delovanje in nima posebne vrednosti za naročnika.

Na višjem nivoju spremljamo število večjih zgodb, ki jih lahko izdamo in tržimo, na nižjem nivoju pa spremljamo tok ter prepustnost in predvidljivost. Delo omejimo na obeh nivojih. Tako delo vizualiziramo na dvonivojski strukturi kanban table brez plavalnih prog ali z njihovo pomočjo. Vsaki večji zgodbi dodelimo svojo ekipo, ki razdeli zgodbo na manjše funkcionalnosti in jih sama prepelje skozi tok na nižjem nivoju table. Ko zgodbo zaključi, prevzame novo.

2.3.11.1 Dvonivojska struktura kanban table (angl. two-tiered card walls)

Večje uporabniške zahteve ali epske zgodbe so prikazane na sliki 2.8 z zelenimi karticami. Prehajajo od leve proti desni od stanja »pripravljene naloge«, »analiza«, »planiranje in razvoj«, »testiranje« do »zaključeno«. Epske zgodbe, ki so trenutno v planiranju ali razvoju,

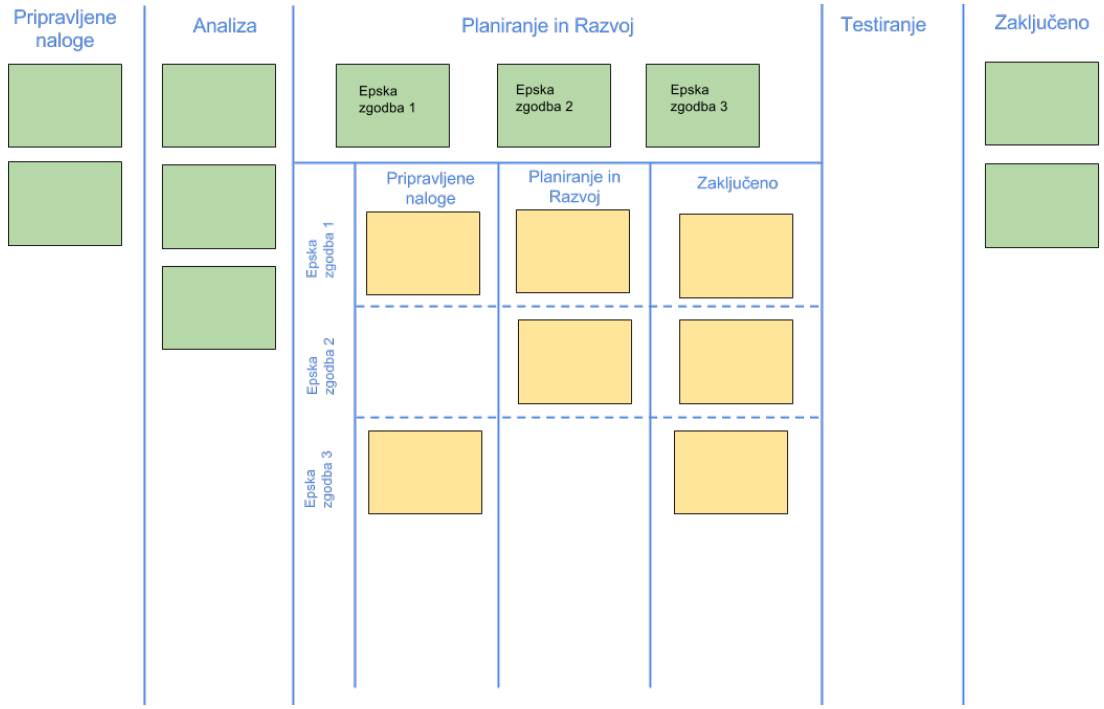
so razdeljene na manjše funkcionalnosti ali uporabniške zgodbe. Vsaki manjši funkcionalnosti dodelimo svojo kartico in jo z oznako povežemo s pripadajočo epsko zgodbo. Na sliki 2.8 so prikazane z rumeno barvo. Manjše funkcionalnosti prehajajo skozi stanja »analiza«, »planiranje in razvoj«, »testiranje« in »zaključeno«. Stanja na nižjem nivoju so lahko enaka tistim na višjem, ni pa nujno.



Slika 2.8: Dvonivojska struktura

2.3.11.2 Dvonivojska struktura kanban table s plavalnimi progami (angl. two-tiered board with horizontal swim lanes)

Uporabniške zahteve ali epske zgodbe na višjem nivoju so prikazane na sliki 2.9 z zelenimi karticami. Stanja, skozi katera prehajajo, so »pripravljene naloge«, »analiza«, »planiranje in razvoj«, »testiranje« in »zaključeno«. Epske zgodbe, ki so trenutno v planiranju ali razvoju, so razvrščene navpično, vsaka v svojo plavalno progo, in razdeljene na manjše funkcionalnosti ali uporabniške zgodbe. Plavalna proga ima svoja stanja, skozi katera morajo prehajati manjše funkcionalnosti. Stanja so lahko enaka kot na višjem nivoju. Kartice z nižjega nivoja so rumene barve. Število plavalnih prog predstavlja omejitev WIP višjega nivoja, omejitev WIP nižjega nivoja pa je glede na želje ekipe določena za vsako plavalno progo posebej.



Slika 2.9: Dvonivojska struktura s plavalnimi progami

2.4 Primerjava metod Scrum in Kanban

Scrum	Kanban
Več omejitev.	Manj omejitev.
Obseg dela je omejen posredno na nivoju iteracije.	Obseg dela je omejen neposredno za vsako stanje posebej.
Samoorganizirana razvojna ekipa.	Razvojna ekipa pooblaščena, da sama sprejema odločitve.
Enako dolge iteracije.	Različni ritmi, ki so lahko dogodkovno vodeni.
Predpisano je ocenjevanje težavnosti uporabniških zgodb.	Ocenjevanje ni obvezno, vendar pa uporabno.
Razvojna ekipa zajema strokovnjake z vseh področij.	Lahko zajema specialiste za posamezna področja.
Nova struktura ekipe in nove 3 vloge.	Ni predpisanih novih struktur in vlog.
Osnovna metrika za izboljševanje procesa je hitrost ekipe.	Osnovna metrika za izboljševanje procesa je potrebni čas.

Scrum	Kanban
Nove naloge ne smemo sprejeti sredi iteracije.	Novo opravilo lahko sprejmemo takoj, ko imamo prosto kapaciteto.
Uporablja lastno izrazoslovje.	Uporablja se obstoječe izrazoslovje.
Delo se razdeli na manjše dele, ki jih je moč opraviti v sprintu. Velikost zgodbe je omejena z dolžino sprinta.	Razdelitev dela ni potrebna, vendar pa zaželena.
Tabla se briše na začetku vsakega sprinta.	Tabla je trajna.
Seznam zahtev sprinta pripada eni sami razvojni ekipi.	Kanban tablo lahko uporablja več ekip.
Zgodbe v seznamu zahtev izdelka morajo biti razvrščene po prioriteti.	Določanje prioritete ni obvezno.
Razvojna ekipa se obveže za obseg dela, ki ga bo opravila v iteraciji.	Obveza ni predpisana.
Predpisana je uporaba padajočega diagrama.	Nobena metrika ni posebej predpisana.
Predpisani so dnevni sestanki.	Sestanki niso predpisani, vendar pa so uporabni.

Tabela 2.1: Primerjava metod Scrum in Kanban [10, 13, 18].

Obe metodi sta vitki in agilni, uporabljata tehnike vlečenja dela. Razvojna ekipa odloča, kdaj in koliko dela bo prevzela; sta transparentni, stremita k izboljševanju razvojnega procesa na podlagi doseženih rezultatov (angl. inspect & adapt pri Scrumu oz. angl. kaizen pri Kanbanu) in omogočata odzivanje na spremembe namesto sledenja planu.

Scrum je ogrodje za iterativni in inkrementalni razvoj procesa, daje poudarek na sodelovanju z uporabniki in na zgodnjem in pogostem dostavljanju delujoče programske opreme. Predpisuje način, kako začeti proces s pooblaščenim produktnim vodjo, ki odloča, kaj bomo delali, ter samoorganizirano razvojno ekipo, ki odloča, kako bomo delali in izvaja delo.

Kanban pa ponuja zelo dobro vidljivost delovnega toka, začne na obstoječem procesu in ga postopoma izboljšuje z omejevanjem obsega dela, odstranjevanjem odpadkov in zastojev z namenom, da skrajša potrebni čas, poveča prepustnost ter omogoči nenehno izboljševanje procesa.

Zaradi narave dela je Scrum dober za razvoj novih projektov, kjer ekipa potrebuje več mesecev, da inkrementalno ustvari nekaj velikega z rednim ritmom izdaj. Za delo s ponavljajočimi aktivnostmi, ki lahko vsebujejo nepredvidljive urgentne spremembe ali pa

potrebujejo neregularen ritem izdaje, je boljša izbira Kanban [6].

Ker je Scrum bolj strukturiran, se je izkazal kot zelo primeren za ekipe, ki se še učijo, kako sodelovati. Člani se z upoštevanjem pravil laže naučijo sodelovati, in ko jim sčasoma način dela postane rutina, začnejo proces izboljševati in vanj vnašati tudi svoje principe agilnosti. Kanban je po drugi strani manj strukturiran in zato bolj primeren za ekipe, ki že delujejo celostno, se ne osredotočajo na proces, ampak raje na delo. Da lahko ekipa izkoristi vse prednosti Kanbana, potrebuje kar nekaj obzirnosti, zrelosti in odgovornosti za vzpostavitev mehanizmov s povratnimi informacijami ter za upravljanje in izboljševanje delovnega toka, kar je lahko izziv za začetniške ekipe [6].

Scrum in Kanban nista izključujoči si metodi, ki bi medsebojno tekmovali, ampak se odlično dopolnjujeta in s tem pomagata rešiti več problemov s še bolj učinkovitim razvojem [10].

2.5 Scrumban

Omenili smo že, da je metoda Scrum skupaj z njenimi hibridi najbolj priljubljena agilna metoda. Ken Schwaber, eden od avtorjev metode Scrum, ocenjuje, da 75 % organizacij, ki uporabljajo Scrum, s to metodo ne dosega želenih koristi. Scrum razkrije vse pomanjkljivosti in nepravilnosti znotraj produktnega razvoja v organizaciji. Namera Scruma je, da jih naredi transparentne, zato da se jih lahko odpravi. Preveč organizacij pa namesto, da bi reševale prikazane pomanjkljivosti in nepravilnosti, metodo Scrum raje spreminja, zato da jim ta bolj ustreza [70].

Scrum je bil zasnovan kot ogrodje, v katerem se lahko uporabi dobre prakse za izboljšanje kakovosti in produktivnosti, kot so npr. prakse ekstremnega programiranja. Z razumevanjem vitkega razmišljanja metodo lahko razumemo še bolje in se naučimo optimizirati tiste procese, o katerih metoda ne govori. Scrum ima po mnenju Kena Schwaberja namenoma nekaj vrzeli, kjer moramo uporabiti najboljše prakse, kot je npr. upravljanje tveganja, metoda pa nam s svojo transparentnostjo pokaže uspešnost izbranega pristopa z namenom, da lahko ta pristop še izboljšamo.

En od takih pristopov je Scrumban, ki je kombinacija lastnosti Scruma in Kanbana. Scrumban je pragmatična uporaba principov in praks Kanbana v obstoječem scrum okolju z uporabo znanstvenih teorij o menedžmentu in upravljanju znanja. Spodbuja dozorevanje ekip, da bolje izkoristijo prednosti, ki jih nudi Scrum in ponuja alternativno pot do agilnosti v primerih, kjer sama metoda Scrum ni rešitev. Uporabljeni principi Kanbana lahko izboljšajo agilnost procesa Scrum, pomagajo upravljati tveganje, izboljšati dostavo storitev in spodbujajo stalne izboljšave [17].

Uspešnost Scrumbana se je pokazala z rezultati Siemensonve študije [76], objavljene

februarja 2014. Pred vpeljavo Scrumbana se je podjetje pri upravljanju razvoja po metodi Scrum soočalo z mnogimi težavami. Ekipa je kljub trdemu delu, nadurami in hudim pritiskom imela težave z dostavljanjem izdaj v obljubljenih rokih, imeli so težave pri planiranju in dokončevanju zgodb v sprintu, kar je predvsem v zadnjih tednih pred koncem sprinta pripeljalo do hitenja za doseganjem čim večjega števila točk in posledično nepremišljenega ter naglega testiranja. Ekipa je prevzemala preveč novih nalog, ki jim jih ni uspelo zaključiti pred izdajo in ni znala prepoznati zastojev. Metrike, ki so jih uporabljali za spremljanje hitrosti ekipe, niso odražale pravega stanja, ekipni padajoči diagrami niso pokazali transparentnosti, ki so jo potrebovali za upravljanje razvoja takega obsega in kompleksnosti. Študija poroča o 33-odstotnem povečanju produktivnosti in drastičnem zmanjšanju okvar pri 500-članski ekipi po implementaciji Scrumbana.

Ekipe, ki se odločijo za Scrumban, pogosto obdržijo scrum izdelke in karakteristike, kot so vloge, časovni okvir, dnevni sestanki, ritem izdajanja, retrospektive itd., in s tem spoštujejo princip »začni v kateri koli fazi projekta«, kar je prvi korak apliciranja Kanbana v scrum okolje. Naslednja dva koraka sta vizualizacija dela in vzpostavitev jasnih pravil v procesu [9].

Scrumban prikaže stare uveljavljene ritme različnih dogodkov z novo vizualizacijo. Tok je definiran v sprintu. Scrum približamo vitkemu delovnemu toku s skrajšanjem dolžine iteracije, vzpostavljanjem tehnik vlečenja dela ter tako, da počakamo z dodeljevanjem novih zahtev do zadnjega smiselnega trenutka. Med »seznam nalog sprinta« in stanjem »razvoj« dodamo stolpec »pripravljene naloge«, ki vsebuje zahteve iz seznama nalog sprinta z najvišjo prioriteto. To nam razdvoji proces dodeljevanja dela in proces določanja prioritete zahtevam ter olajša dodeljevanje.

Na tej točki lahko začne ekipa razmišljati o omejitvi dela, ki ustreza njeni zmogljivosti in sprejme pravila, kot so npr., da lahko kdor koli in kadar koli dodaja zahteve v seznam nalog sprinta, vedno raje zaključujemo delo, ki je že v razvoju, pred prevzemanjem novega, delamo le na eni nalogi hkrati, razen v primeru blokade, takrat pa lahko prevzamemo novo nalogo, vendar ne več kot eno, itd. Še vedno se držimo časovnega okvirja, upoštevamo iteracije in načrtovalne cikle kot pri Scrumu.

Sčasoma, ko naš proces postane bolj zrel, učinkovit ter odziven, lahko začnemo omejevati velikost seznama nalog sprinta. Namesto, da za vsako iteracijo načrtujemo celoten obseg dela, omejimo velikost seznama nalog sprinta na povprečno število zahtev, ki jih izdamo v iteraciji, kar zmanjša stroške in aktivnosti načrtovanja in nam omogoča, da čas, ki smo ga privarčevali, vložimo v boljši nadzor kakovosti tistih nalog, ki so napredovale v stolpec »pripravljene naloge«.

Naslednji korak, ki razdvoji planiranje in izdajanje verzije, nas že popelje prek metode Scrum. S tem, ko vzpostavimo ločene ritme, ne planiramo več izdaj, ampak planiramo

takrat, ko je to za nas priročno, in izdajamo, ko je to potrebno. Od tukaj naprej proces upravljamo po metodi Kanban.

Poglavje 3

Primerjava elektronskih kanban orodij

3.1 Merila za primerjavo

Za podporo razvoja programske opreme po metodi Kanban imamo na voljo veliko število spletnih orodij.

Orodja za primerjavo smo izbrali iz seznama spletnih kanban orodij, predstavljenih na spletni strani kanban skupnosti Limited Wip Society [61]. Orodja smo razdelili v štiri skupine.

V prvi skupini so izključno kanban orodja, ki so bila izdelana posebej za za vizualizacijo delovnega toka in so v skupnosti kanban najbolj cenjena in priporočena [45]. Glavni predstavniki orodij iz te skupine so SwiftKanban [65], Leankit [60], Targetprocess [46], Kanbanery [56], AgileZen [64], Kanbanize [58] ter KanbanTool [55].

V naslednjo skupino spadajo orodja bolj znanih večjih podjetij, ki imajo največjo bazo uporabnikov, kot so Rally enterprise tool [53], Atlassian Jira Agile [51] ter VersionOne [52]. Po mnenju kanban skupnosti [54, 66] pa ne dosegajo enake kakovosti kot orodja, omenjena v prvi skupini.

Tretjo skupino predstavljajo orodja, ki niso bila izdelana neposredno za podporo sistemov kanban, vendar jih je mogoče uporabiti tudi v ta namen. Orodja, kot so Trello [68], Asana [47] ali Lino [62], omogočajo prikaz enostavnih tabel z opravili in so koristna za tiste skupine, ki jim ustreza uporaba fizične table, iz nekega razloga pa potrebujejo elektronsko tablo brez avtomatskih izvajanj analiz in prikazov metrik. V to skupino sem uvrstila tudi spletno aplikacijo Google Drawings, ki je na voljo vsem uporabnikom storitve Google Drive [50]. Aplikacija ponuja enostaven način ustvarjanja diagramov in grafikonov, kar lahko izkoristimo za simulacijo fizične table na oddaljenih lokacijah [49].

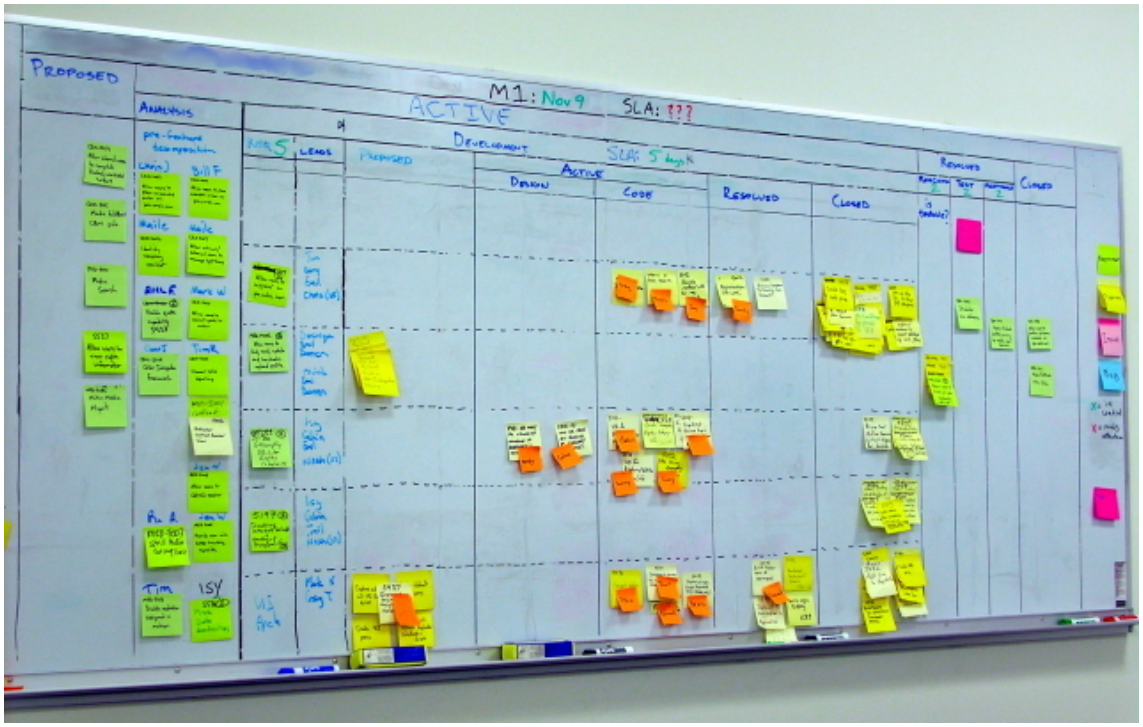
V četrto skupino pa smo uvrstili odprtokodna orodja, kot so orodja Kanbanik [57],

Kanboard [59], VisualWIP [71] in Personal Kanban [63]. Odprtokodna orodja so načeloma manj sofisticirana, vendar še vedno uporabna.

V diplomskem delu smo iz vsake skupine izbrali po eno orodje in naredili primerjavo. Izbrali smo orodja LeanKit iz prve skupine, Jira Agile iz druge, Trello iz tretje in Kanboard iz četrte skupine. Za primerjavo zapišemo med rezultate tudi značilnosti fizične kanban table, katere primer je predstavljen na sliki 3.1.

V diplomskem delu nas je zanimalo, ali izbrana elektronska orodja predstavljajo sistem kanban na enako enostaven in pregleden način kot fizična tabla in ali so uporabnejša od fizične table. Cilj je bil poiskati tako orodje, ki podpira naslednje značilnosti kanban table:

- kreiranje table s poljubnimi stolpci;
- kreiranje nove table s kopiranjem strukture obstoječe table;
- omejevanje obsega dela v izvajanju na ravni stolpca ali na ravni posameznika;
- razdelitev posameznih stolpcev na podstolpce, možnost dodajanja več ravni podstolpcev in omejevanje obsega dela na podstolpcih;
- uporabo plavalnih prog in omejevanje obsega dela na ravni plavalne proge;
- obravnavanje nujnih nalog;
- razvrščanje nalog po prioriteti v stolpcu »pripravljene naloge«;
- prikaz meril oz. pravil za prehajanje delovnih nalog skozi delovni tok;
- prikaz točke order-point;
- prikaz pričakovanega čakalnega časa delovne naloge;
- kreiranje kartice z različnimi informacijami (opis, šifra, pod katero se delovna naloga beleži v elektronskem sistemu za vodenje in sledenje delovnim nalogam, dogovorjeni rok za dokončanje, velikost naloge, ikona oz. ime lastnika kartice, tip, razred, indikator napredka, časovni žigi vstopov v stanja, čustva povezana z delom);
- prilagoditev prikaza informacij in barve kartice;
- prikaz blokiranih kartic na tabli;
- razdelitev večjih delovnih nalog na manjše, možnosti povezave in prikaza v dvonivojski strukturi;
- izvajanje različnih analiz in prikazovanje metrik za spremljanje napredka;
- zahtevnost učenja za uporabnika.



Slika 3.1: Primer fizične kanban table [72]

3.2 Pregled obstoječih orodij

3.2.1 LeanKit

Orodje LeanKit podpira vizualizacijo dela po vitkem in agilnem razvoju. Orodje je preizkušeno z uspešnimi kanban implementacijami s 500 uporabniki, poroča se tudi o primerih implementacije do 1000 uporabnikov [69]. Orodje je od leta 2009 na voljo kot spletna aplikacija, do katere dostopamo prek brskalnika ali mobilnega telefona oz. tablice na operacijskih sistemih iOS ali Android. Podpira integracijo z več kot 300 drugimi sistemi, na voljo pa je tudi API (angl. Application Programming Interface), ki omogoča integracijo po meri uporabnika.

Orodje, testirano decembra 2014, je na voljo v treh različicah: osnovni, ekipni in portfolijo. Portfolijo različica je na voljo brezplačno, v preizkusni izdaji za prvih 30 dni uporabe. LeanKit ponuja tudi dodatne storitve, kot so najem zasebnega oblaka, izvoz naprednejših poročil v ločeno podatkovno bazo ter dodatno varnost. Orodje se s svojim izgledom skuša približati klasični fizični tabli. Elektronska tabla v LeanKitu je bele barve, kartice na njej pa predstavljajo samolepilne lističe, ki jih pomikamo skozi delovni tok s potegom miške ali prek akcij v meniju.

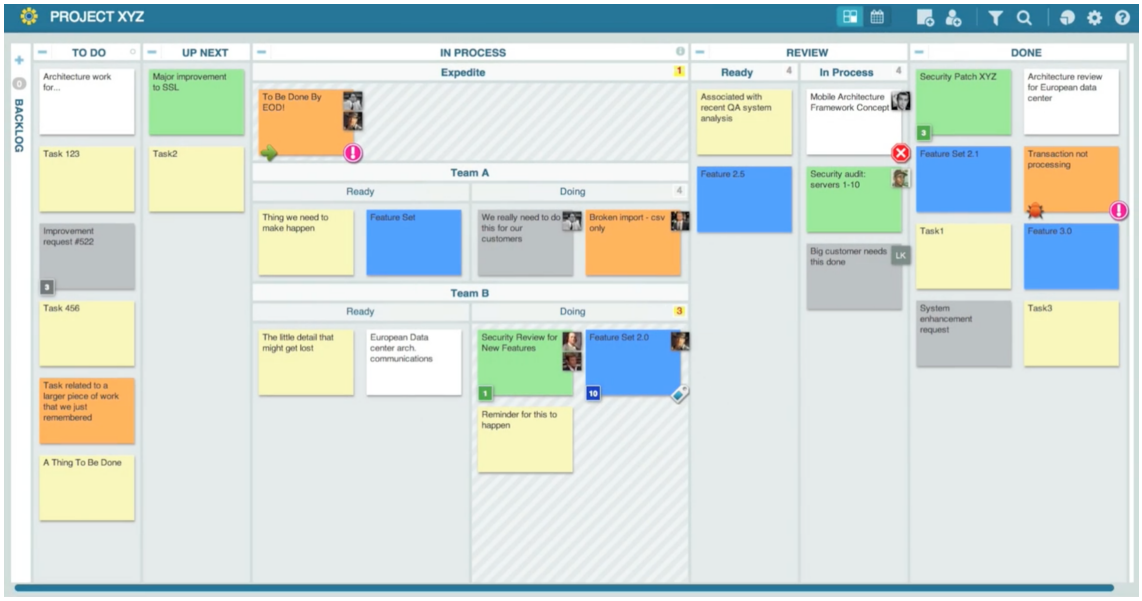
Novo tablo lahko ustvarimo na več načinov. Zgradimo jo lahko sami z vstavljanjem poljubnih stolpcev, ali pa jo ustvarimo iz vzorčnih predlog, ki so nam na voljo. Orodje

dopušča kreiranje nove table tudi s kopiranjem strukture obstoječe table.

Kreiramo lahko poljubno število stolpcev. Na novo kreirani tabli se privzeto prikažeta stolpca »seznam zahtev« in »arhivirano oz. zaključeno«, kjer zbiramo celotno delo, ki ga želimo opraviti, ter delo, ki je končano. Primer table je prikazan na sliki 3.2. Stolpce imamo možnost preimenovati, jim določiti širino ter zamenjati vrstni red. Vsakemu stolpcu lahko določimo omejitev WIP. Lahko ga razdelimo navpično na več podstolpcev, ki jih uporabimo kot čakalne vrste ali izravnalnike. Podstolpcu lahko določimo lastno omejitev WIP ali nastavimo skupno omejitev WIP na višjem nivoju, ki velja za vse podstolpce. Stolpce lahko razdelimo na neomejeno število nivojev. Če določimo omejitev WIP na več nivojih, program vedno upošteva omejitev, ki je nastavljena na najvišjem nivoju.

Stolpce lahko razdelimo tudi vodoravno na plavalne proge, ki jim je možno določiti omejitev WIP, kar je uporabno za omejevanje dela na ravni posameznika ali na ravni ekipe. Uporabno je tudi za prikaz dela po aktivnostih, ki se morajo odvijati istočasno, ali za razdeljevanje kapacitete dela po tipu ali razredu naloge.

S kreiranjem podstolpcev in plavalnih prog lahko brez težav ustvarimo dvonivojsko strukturo table. Za lažje delo nam orodje v stolpcu ponuja kloniranje obstoječe strukture, bodisi plavalne proge ali podstolpcev. Stolpec lahko izbrišemo, če v njem ni kartic. Imamo možnost dodajanja opisa oz. zapisati vanj pravila in merila, ki jih želimo upoštevati pri prehajanju kartic skozi delovni tok. Če stolpec vsebuje opis, se ta ne prikaže samodejno, ampak šele, ko z miško kliknemo na gumb za dodatne informacije na vrhu stolpca. Stolpce lahko po želji skrijemo ali razkrijemo, kar omogoča boljšo preglednost.



Slika 3.2: Kanban tabla v orodju Leankit [60]

Pri dodajanju kartic nas orodje opozori, če je omejitev WIP stolpca že dosežena. Kar-

tico lahko kljub temu dodamo, vendar orodje zahteva pojasnilo in obarva celoten stolpec rdeče, s čimer vizualno signalizira prekoračitev omejitve WIP.

Z uporabo plavalne proge lahko dodamo vrstico za nujne delovne naloge. Če določimo plavalni progi omejitev WIP 1, vzpostavimo pravilo, da se sme obravnavati le eno nujno nalogo hkrati. Omejitev WIP lahko določimo za vsakega uporabnika posebej. Pri dodeljevanju novih nalog uporabniku nas orodje z opozorilom spomni, da je njegova lastna omejitev že dosežena, in zahteva pojasnilo v primeru, da jo želimo prekršiti.

Tudi nove kartice lahko dodamo na več različnih načinov. Lahko jih uvozimo, prestavimo iz druge tabele ali pa s kopiranjem obstoječe kartice.

Kartici lahko določimo naslov, opis, tip in razred naloge, začetni datum ter dogovorjeni rok za dokončanje, prioriteto, informacijo o blokadi skupaj z razlogom zanjo, povezavo do elektronskega sistema za vodenje in sledenje delovnih nalog, velikost naloge ter lastnika kartice. Možen je še vnos komentarjev in priponk ter pregled zgodovine z vsemi časovnimi žigi vstopov v stanja in drugimi spremembami. Orodje kartici samodejno dodeli identifikacijsko številko, na prikazu na tabli pa ponuja možnost, da namesto te, prikaže ročno vpisano oznako iz drugega elektronskega sistema za vodenje in sledenje delovnim nalogam.

Orodje ima svoj nabor ikon, s katerimi prikazuje določene informacije na kartici. Prikaza ikon ne moremo prilagoditi. Tudi velikost in položaj ikon se določi avtomatsko.

Za tip in razred naloge je na voljo prednastavljeni nabor vrednosti, ki se po želji lahko prilagodi. V nastavitvah lahko dodajamo in brišemo nove tipe in razrede ter jim določimo barve oz. ikone za prikaz.

V primeru, da smo nastavili dogovorjeni rok za dokončanje, se prikazani ikoni na kartici spreminja barva, ko se približujemo vpisanemu datumu. Ko se nam čas do roka izteka, nam orodje lahko pošlje tudi e-mail z obvestilom.

Za prioriteto je prav tako na voljo prednastavljeni nabor vrednosti, ki ga ni mogoče spreminjati in žal tudi ne vpliva na razvrščanje nalog po prioriteti v stolpcu »seznam zahtev« ali »pripravljene naloge«. Naloge različnih pomembnosti se sicer prikažejo z različnimi ikonami na kartici, a menim, da bi nam vrstni red nalog po prioriteti še dodatno olajšal izbiranje nove naloge. Program tudi ne omogoča dogodkovno vodenega planiranja s prikazom točke order-point.

Velikost naloge je polje za vnos številke. Numerično vrednost, ki jo vpišemo, lahko predstavlja število točk, število iz Fibonaccijevega zaporedja, konfekcijsko velikost, ocenjeno potrebno delo v dnevih, število posameznikov, ki delajo na nalogi itd. Če določimo velikost naloge, lahko v nastavitvah orodja spremenimo, da se omejitev WIP izračuna na podlagi seštevka velikosti vseh nalog in ne števila kartic v stolpcu. Vpisana vrednost ne vpliva na razvrščanje nalog v stolpcu.

Blokada na kartici je prikazana s posebno ikono, orodje pa jo kljub temu pusti, da se

pomika skozi delovni tok. Lastnik kartice se prikaže s svojimi inicialkami ali ikono, ki jo v orodje naloži sam. Kartici lahko dodelimo več lastnikov.

Funkcija dodajanja komentarjev in priponk omogoča lažje sodelovanje in zelo dobro nadomesti vso dodatno komunikacijo prek drugih sistemov, npr. pošiljanje navodil in specifikacij prek e-mailov ter uporabo drugih elektronskih orodij za vodenje in sledenje delovnim nalogam.

Orodje ne dopušča spreminjanja barve kartice; barva je odvisna od izbranega tipa kartice. Ker pa lahko kreiramo poljubno število tipov kartic in s tem določimo barvo, lahko tako vplivamo na izbiranje barve kartice. Čustva, povezana z delom ter druga opažanja si lahko izmenjujemo med komentarji kartice. Če želimo spremljati komentarje, se lahko nanje naročimo prek e-maila ali RSS.

Orodje za kartice v stolpcu »pripravljene naloge« ne prikazuje pričakovanega čakalnega časa delovne naloge, le-ta se lahko oceni na podlagi metrik za spremljanje napredka.

Indikatorji napredka niso prikazani neposredno, lahko jih razberemo iz zgodovine kartice.

LeanKit omogoča še dodajanje manjših opravil (angl. task) na kartice in s tem podpira trinivojsko hierarhijo (angl. Taskboards). Tretji nivo se prikaže v nastavitvah kartice na posebni tabli s prevzetimi tremi stolpci »pripravljene naloge«, »v delu« in »zaključeno«.

Opravilom lahko določimo enake značilnosti kot karticam. Kartica, ki je razdeljena na več opravil, prikazuje dodaten indikator napredka, ki beleži število dokončanih opravil.

Tablo lahko delimo z drugimi notranjimi ali zunanjimi uporabniki. Dostopamo lahko do celotne zgodovine sprememb, ki jih je mogoče filtrirati po uporabniku ali intervalu, kar je zelo uporabno pri izmenjavi informacij na dnevnih in drugih sestankih. Kadar kartica zamenja stanje, nam orodje samodejno v realnem času prikazuje obvestila, s katerimi nas opozori na spremembe na tabli. Na to opozorilo se lahko tudi naročimo prek e-maila ali RSS.

Vse metrike za spremljanje napredka se računajo na podlagi podatkov, zapisanih v skupni zgodovini table.

Orodje omogoča še nastavitev pravic uporabnikom in na podlagi teh prilagaja vsebino, do katere lahko uporabniki dostopajo.

Orodje omogoča kreiranje povezav »starš – otrok« (angl. Drill-Throughs) med karticami različnih tabel, kar olajša spremljanje nalog, ki so razdeljene med več ekip. Kartice lahko predstavljajo projekte, nove funkcionalnosti ali druge delovne naloge in zato se delegiranje nalog v taki hierarhiji lahko uporabi tudi za kreiranje portfelja, ki nam pomaga pri upravljanju več kanban tabel. Orodje zajema še statistiko za upravljanje projektnega portfelja (angl. Project Portfolio Management – PPM) in tako daje nosilcem projekta dodatno preglednost dela.

Ena od funkcionalnosti orodja je tudi filtriranje podatkov, ki jih vidimo na tabli. Filter omogoča pogled le nad tistimi delovnimi nalogami, ki ustrezajo našim merilom in tako prikaže delo, ki je relevantno za posameznika.

Orodje za prikaz seznama nalog za prijavljenega uporabnika omogoča poleg vizualizacije dela na tabli še prikaz njegovih delovnih nalog na koledarju.

Analize in metrike za spremljanje napredka, ki jih LeanKit podpira, so:

- CFD, ki za vsak dan spremlja število delovnih nalog v posameznih stolpcih. Po potrebi lahko vklopimo dodatno funkcijo spremljanje napredka na grafu dokončanega dela (angl. Burn-up), ki se izriše na CFD in omogoča boljši dnevni pregled nad številom novih in zaključenih delovnih nalog.

- Diagram povprečnega cikla obdelave (angl. Cycle Time Diagram), ki za neko časovno obdobje prikaže povprečni čas, v katerem smo obdelali nalogo.

- Diagram porazdeljenosti nalog (angl. Card Distribution Diagram), ki na krožnem grafikonu beleži porazdeljenost dela na tabli glede na uporabnika, stolpec, tip, razred ali prioriteto delovne naloge.

- Diagram učinkovitosti (angl. Efficiency Diagram), ki spremlja, koliko časa delovna naloga stoji v stanjih tipa »pripravljeno«, »v delu« in »zaključeno«. Vsak stolpec na tabli, ki ga želimo spremljati, mora imeti v nastavitvah določeno eno od teh treh tipov stanj.

- Diagram odstopanja od povprečnega cikla obdelave (angl. Process Control Diagram), ki za vsako delovno nalogo v procesu spremlja odstopanje od povprečnega časa izdelave.

LeanKit pa omogoča še uvoz podatkov v orodje KanbanSim podjetja Focused Objective, ki zna s simulacijsko metodo Monte Carlo empirično napovedovati verjetnost dokončanja naloge v določenem času.

Pri izračunu poprečnega časa in časa cikla procesa lahko sami izberemo začetno in končno stanje, ki ga orodje upošteva pri izračunu. Tudi pri prikazu metrik lahko izbiramo, katere stolpce želimo vključiti v analizo oz. katerih ne želimo.

Analize lahko dodatno filtriramo po uporabniku, tipu, prioriteti, oznaki ter velikosti delovne naloge. Ekipe, ki v nastavitvah kartice vpišejo število točk uporabniške zgodbe, imajo možnost določiti, da se število točk pri izračunu analize upošteva kot dodaten parameter.

Analize se lahko izvajajo za eno tabelo ali na ravni portfolia za več tabel hkrati.

Orodje lahko uporabljajo tudi ekipe, ki razvijajo programsko opremo po metodi Scrum, tako da si kreirajo bolj preprosto scrum tablo. Žal pa LeanKit ne omogoča planiranja

izdaje, pregleda hierarhije uporabniških zgodb ter grafa hitrosti ekipe, kar vse ponuja konkurenčno orodje SwiftKanban.

Podjetje LeanKit pripravlja novo verzijo orodja, v katerem nameravajo spremeniti algoritme za merjenje in prikazovanje metrik, dodati posebne table dostopne naročnikom, narediti spremembe na uporabniškem vmesniku za lažje dodajanje kartice in urejanje podatkov ter povezav med tablami, dodati podporo za metode SAFe, Scrum in TFS, razširiti API za upravljanje z uporabniki, podpreti izvoz grafov v PDF in Excel datoteke, dodati možnost spreminjanja ikon na kartici ter še mnogo drugih sprememb.

LeanKit je zelo napredno kanban orodje in je za uporabo kar zahtevno. Podjetje sicer nudi izredno dobro uporabniško podporo, jasna navodila z video predstavitvami, uporabne tedenske nasvete ter novice prek e-maila ter tudi brezplačna spletna izobraževanja o metodi Kanban ter orodju LeanKit. Orodje uporabniku omogoča komentiranje svojih opažanj, opozarjanje na pomanjkljivosti, podajanje kritik in predlogov za izboljšavo, kar razvijalci LeanKita upoštevajo pri razvoju novih različic, mogoče pa je tudi vključevanje uporabnikov v začetno testiranje.

3.2.2 Atlassian Jira Agile

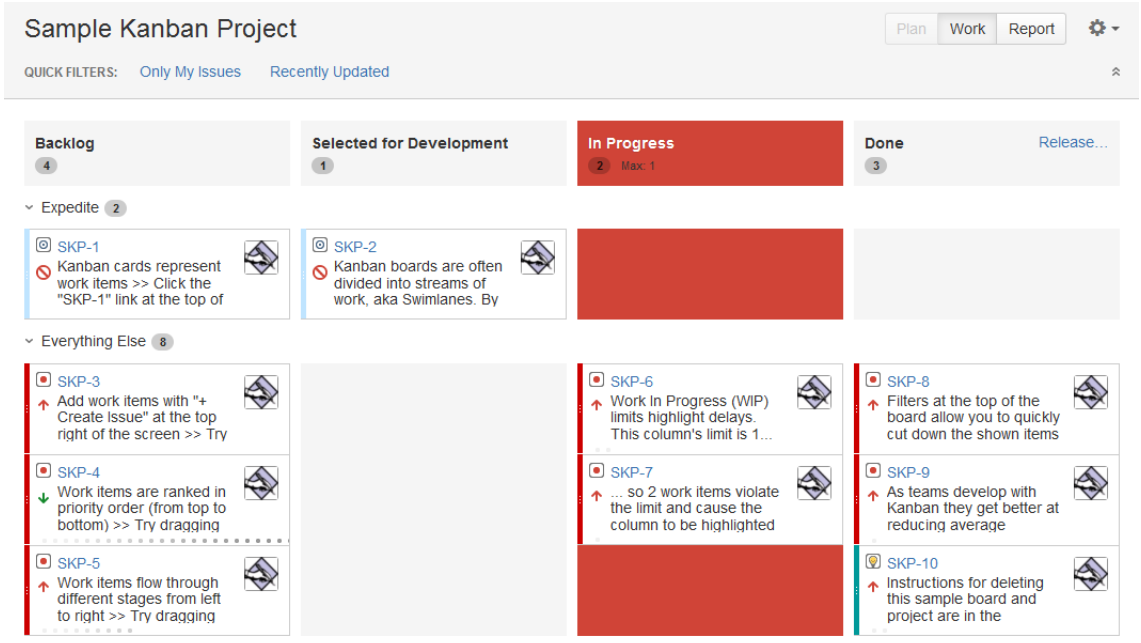
Jira Agile je vtičnik za podporo agilnega razvoja programske opreme po metodah Scrum in Kanban. Je nadgradnja programskega orodja Jira, ki je med bolj razširjenimi in naprednimi elektronskimi orodji, ki so namenjeni podpori projektnim aktivnostim. Jiro uporabljamo za pregled napak, izboljšav in zahtev ter vodenje in slednje priponk, sprememb, komponent in različic. Druge lastnosti so tudi zmogljivo filtriranje, prilagodljive statistike, sistem pravic uporabnikov, enostavna integracija z drugimi sistemi ter prilagodljiv sistem obveščanja. Jira Agile vtičnik je bil testiran v mesecu maju 2015. Leta 2009 je Atlassian odkupil orodje Greenhopper, ki je bilo leta 2013 preimenovano v Jira Agile. Jira Agile je na voljo skupaj z orodjem Jira brezplačno, v preizkusni izdaji za 7 dni. Plačljiva verzija je na voljo kot storitev v oblaku ali kot namestitev na lastni strežnik. Jira je spletna storitev, do katere dostopamo prek brskalnika in je prilagojena tudi za mobilne naprave in tablice. Cena za vtičnik in orodje skupaj zavisi od števila uporabnikov, ki bodo orodje uporabljali. Poleg Jira Agile vtičnika so na voljo tudi drugi plačljivi vtičniki, kot so npr. Jira Service Desk za lažjo podporo uporabnikom, Jira Capture za lažji prikaz napak z zaslonskimi slikami, Confluence za lažje medekipno sodelovanje in kreiranje baze znanja, FishEye za lažje upravljanje razvojne kode ter izvajanje pregleda kode, Bamboo za avtomatizacijo testov ter sprotno integracijo in še več kot 1700 drugih vtičnikov, ki so na voljo na spletni strani Atlassian Marketplace [48]. Vsi nastavki za Jira storitev v oblaku žal niso na voljo. Tudi Jira ima tako kot LeanKit možnost uporabe API za integracijo po meri uporabnika.

Jira Agile deli svoje funkcionalnosti glede na izbrano metodo razvoja programske

opreme. Za scrum ekipe omogoča pregled pretekle zmogljivosti za lažje planiranje sprintov, upravljanje seznama zahtev po prioriteti, možnost komentiranja posameznih zahtev v seznamu, kar spodbuja medekipno sodelovanje, ocenjevanje težavnosti uporabniških zgodb ter prikaz zahtev na scrum tabli. Podprta je tudi sledljivost za vsak zahtevek ter vsako potrditev (angl. commit) izvorne kode na repozitorij. Svoj napredek lahko ekipa spremlja na padajočem diagramu spremljanja napredka, kar pomaga pri retrospektivi.

Jira Agile priporoča uporabo kanban table za tiste ekipe, ki sledijo nekemu delovnemu toku, kot so služba za podporo uporabnikom ter operativa. Vtičnik ponuja vizualizacijo želenega delovnega toka, uporabo plavalnih prog za lažje upravljanje dogovorov o ravni storitve ter enostavno posodabljanje Jirinih zahtevkov, kar na kanban tabli.

Kartice na tabli so v resnici Jirini zahtevki. Tako na tabli ne ustvarjamo novih kartic, ampak prikažemo obstoječe zahtevke z uporabo filtrov. Filtre, ki omogočajo zelo natančno izbiro meril za prikaz kartic nastavimo s posebnim jezikom za kreiranje poizvedb JQL. Primer table je prikazan na sliki 3.3.



Slika 3.3: Kanban tabla v orodju Jira [51]

Nov zahtevek lahko kreiramo ročno, prek CSV-datoteke ali e-maila. Informacije na kartici se na zahtevo dodajajo kot polja. Tako ji lahko nastavimo tip zahtevka ter prioriteto, ki imata prednastavljen nabor vrednosti, opis ter lastnika zahtevka. Nastavimo lahko le enega lastnika. Dodamo lahko še povezavo do epske zgodbe in sprinta ter priložimo različne priponke in zaslonske slike. Orodje nam daje možnost sledenja sprememb na zahtevi, dodajanja komentarjev in kreiranja podopravil. V nastavitvah je na voljo tudi obsežen nabor možnosti novih polj, ki jih lahko dodamo na zahtevo. Kreiramo pa lahko tudi polja

po meri uporabnika. Primeri polj, ki nam bi koristila na kanban tabli in na zahtevku niso privzeta, so velikost naloge, razred naloge ter dogovorjeni rok za dokončanje. Čustva, povezana z delom, lahko uporabniki zapišejo med komentarje.

Orodje Jira beleži še status zahtevka, ki je tudi prednastavljen nabor vrednosti z možnostjo urejanja oz. dodajanja.

Nabor vrednosti za tip zahtevka lahko uredimo v nastavitvah orodja. Obstoječim tipom lahko spremenimo ime ter prikazano ikono, dodamo pa lahko tudi nov tip. Enako velja za nabor vrednosti prioritete zahtevka, dodamo lahko novo ali uredimo obstoječo prioriteto, ji spremenimo naziv ali zamenjamo barvo ikone.

Večina informacij na kartici tako izhaja iz zahtevka, nekaj značilnosti kartice pa nastavljamo tudi v nastavitvah kanban table. Kartici lahko določimo barvo, ki je odvisna od izbranega tipa zahtevka, prioritete, lastnika kartice ali drugega pogoja, ki ga nastavimo s pomočjo JQL-filtra. Primer takega pogoja je lahko filter za vse kartice z dogovorjenim rokom za dokončanje v naslednjih sedmih dneh.

V nastavitvah kanban table lahko tudi označimo, da se prikaz kartic v vseh stolpcih razvrsti po prioriteti zahtevka. Na kartici se privzeto izpišejo podatki o identifikacijski številki zahtevka, kratek opis, ikona lastnika kartice ter ikona za tip zahtevka. Po želji lahko dodamo še tri poljubna obstoječa polja iz zahtevka ter nastavimo kartici indikator napredka, ki prikaže, koliko časa je bila kartica v posameznem stolpcu.

Ko ustvarimo novo kanban tablo, se ta izriše s tremi privzetimi stolpci, in sicer »pripravljene naloge«, »v delu« in »zaključeno«. Tablo lahko kreiramo tako, da jo zgradimo sami, jo uvozimo iz obstoječega shranjenega JQL-filtra ali pa kopiramo obstoječo tablo, s katero se prenesejo tudi vse strukture. V nastavitvah table lahko nastavimo pravice za dostopanje do nje ter ji določimo JQL-filter za prikazovanje zahtev. Tabla omogoča še dodatno t. i. hitro filtriranje kartic na tabli glede na prednastavljena merila, kot so npr. »moje kartice«, »nazadnje posodobljene kartice« oz. lastna izbira.

Stolpce na tabli lahko dodajamo, odstranimo in jim zamenjamo vrstni red. Vsakemu stolpcu lahko nastavimo ime, lastno omejitev WIP, ki lahko izključuje ali vključuje podopravila ter mu dodelimo status zahtevka, ki se bo nastavil vsaki kartici, ki bo vstopila v ta stolpec. Stolpec nima možnosti dodajanja opisa. Orodje ne omogoča spreminjanja širine stolpca, saj je le-ta odvisna od števila stolpcev v tabeli. Skrivanje oz. razkrivanje stolpcev za lažjo preglednost ni omogočeno. Stolpce lahko izbrišemo tudi, če so v njih kartice, saj te niso neposredno vezane na stolpce, ampak na status zahtevka, ki je dodeljen stolpcu.

Vrstice, ki jih lahko dodamo na tablo se imenujejo plavalne proge in so namenjene posebnim primerom. Kartice ne moremo ročno prenesti v plavalno progo, vse nujne primere, ki jih želimo prikazovati v plavalni progi, pa moramo definirati z uporabo JQL- filtra. Če želimo implementirati blokade, lahko to storimo tako, da dodamo nov tip prioritete, npr.

»ovira«, ali pa dodamo novo polje na zahtevo v njenih nastavitvah. Tip prioritete ali novo dodano polje predstavljata zastavico, ki jo uporabimo pri kreiranju JQL-filtra za kreiranje plavalne proge. Kartica z blokado se prestavi v plavalno progo, še vedno pa se lahko premika po stanjih. Podobno lahko prikažemo tudi kartice z dogovorjenim rokom dokončanja. Tudi v Jira Agile orodju nisem našla podpore oz. implementacije dogodkovno vodenega planiranja in točke order-point.

Stolpcev ne moremo razdeliti navpično na podstolpce ali vodoravno na plavalne proge. Orodje tako ne omogoča uporabe dvonivojske strukture.

Pri prvem kreiranju kanban table se zahtevek ter tudi njegova podopravila, ki ustrezajo izbranemu filtru, prikažejo kot kartice v stolpcu »pripravljene naloge«. Tudi vsi novi zahtevki se vedno prikažejo v tem stolpcu, četudi imamo s stolpci po meri implementirano drugačno vizualizacijo. V nastavitvah orodja Jira imamo možnost definirati novi delovni tok, ki bo poskrbel, da se novo kreirani zahtevki prikažejo v želenem stolpcu. Kartice tako kot v Leankit orodju prestavljamo v druge stolpce s potegom miške. V primeru, da je WIP omejitev stolpca že dosežena, se stolpec obarva rdeče.

Za sedaj orodje ne omogoča WIP omejitev na ravni uporabnika, ampak omogoča pa izračun WIP omejitve na podlagi seštevka velikosti vseh nalog v stolpcu. Uporabniki orodja Jira imajo na spletni strani podjetja Atlassian dostop do Jira projekta, kjer lahko predlagajo izboljšave.

Ko ekipa izda novo verzijo, s klikom na poseben gumb »izdaja« (angl. release) odstrani vse kartice iz stanja »zaključeno«. Če ne želimo uporabljati omenjene funkcionalnosti, lahko nastavimo filter prikaza zahtev tako, da se kartice pri izpolnjenih pogojih v stanja »zaključeno« ne prikazujejo.

Tabla je na voljo vsem uporabnikom, ki lahko dostopajo do filtra, na katerem je osnovana tabla. Uporabniki potrebujejo tudi ustrezne pravice za branje table. Če želimo deliti tablo z drugimi Jirinimi uporabniki, moramo ustvariti oz. popraviti filter. Če želimo tablo deliti z zunanjimi uporabniki, jim moramo dodeliti tudi Jirino uporabniško ime.

Zgodovina sprememb na tabli ni dostopna na enem mestu, orodje ponuja le pregled zgodovine posameznega zahtevka. Prek e-maila ali RSS pa lahko naročimo rezultate shranjenega filtra za prikaz zahtev, ki jih želimo spremljati.

Tako kot Leankit tudi Jira Agile ne prikazuje pričakovanega čakalnega časa delovne naloge na kartici, ko ta vstopi v stolpec »pripravljene naloge«. Na kanban tabli sta na razpolago dve metriki za spremljanje napredka, CFD ter diagram odstopanja od povprečnega cikla obdelave (angl. Control Chart). Pri prikazu metrik orodje ponuja, enako kot Leankit, izbiro stolpcev, ki jih upošteva pri izračunu. Rezultate lahko omejimo tudi glede na izbrane plavalne proge ali s filtrom. Vtičnik nudi dodatne metrike za scrum ekipe, ponuja pa še možnost izvoza podatkov v obliki različnih grafov ter krožnih grafikonov. Na

ta način lahko izvozimo graf za povprečno število dni v nekem obdobju, ko zahtevki niso zaključeni (angl. Average Age Chart), graf novo kreiranih ter zaključenih zahtev, v nekem obdobju (angl. Created vs. Resolved Chart), krožni diagram za porazdeljenost zahtevkov glede na neko polje iz zahtevka (angl. Pie Chart), graf za povprečno število dni v nekem obdobju, ko so bili zaključeni zahtevki še v delu (angl. Resolution Time) itd.

V primeru, da nam navedene lastnosti orodja ne zadoščajo, lahko za bolj pregledno vizualizacijo delovnega toka namestimo poseben vtičnik Artezio Kanban Board for Jira. Namestimo lahko tudi vtičnik Jira Wallboards, ki omogoča vizualizacijo na veliki televiziji, npr. pritrjeni na steno v pisarni. Jira Agile ne podpira projektnega portfelja upravljanja, kot ga Leankit, za ta namen spodbuja uporabo epskih zgodb, prilagoditev delovnega toka, natančnih opisov polj ter namestitev dodatnih vtičnikov [73]. Če ekipa meni, da so funkcionalnosti Jira Agile vtičnika povsem nesprejemljive ali pa ga iz nekega drugega razloga ne želijo uporabljati, lahko to orodje integrirajo z orodjem SwiftKanban prek vtičnika SwiftSync. Jira skrbi za upravljanje zahtevkov, SwiftKanaban skrbi za vizualizacijo delovnega toka na kanban tabli, SwiftSnyc pa skrbi za sinhronizacijo med obema orodjema.

Podjetje Atlassian ima na svoji spletni strani kar nekaj dokumentacije ter uporabniških navodil za uporabo vtičnika Jira Agile. Poleg dokumentacije je za podporo naročnikom na voljo še plačljiva storitev Jira Agile Support, odgovori na pogosto zastavljena vprašanja, Jira spletni treningi ter različne informacije o agilnem razvoju. Potrebovala sem kar nekaj energije in časa, da sem uspešno vzpostavila želeno vizualizacijo, ker razpoložljiva dokumentacija dostikrat ni bila ustrezna. Odgovore sem morala poiskati s pomočjo Googla. Preizkusna različica traja le 7 dni, kar je zelo malo časa, da preučimo vse zmogljivosti in pomanjkljivosti programa. Storitev v oblaku, ki sem jo testirala, je bila takrat zelo počasna. Orodje za bolj napredno uporabo potrebuje dodatne vtičnike, kar zviša njegovo ceno. Uporaba različnih nastavkov pa lahko pri posodabljanju orodja Jira pripelje tudi do težav s kompatibilnostjo različic.

3.2.3 Trello

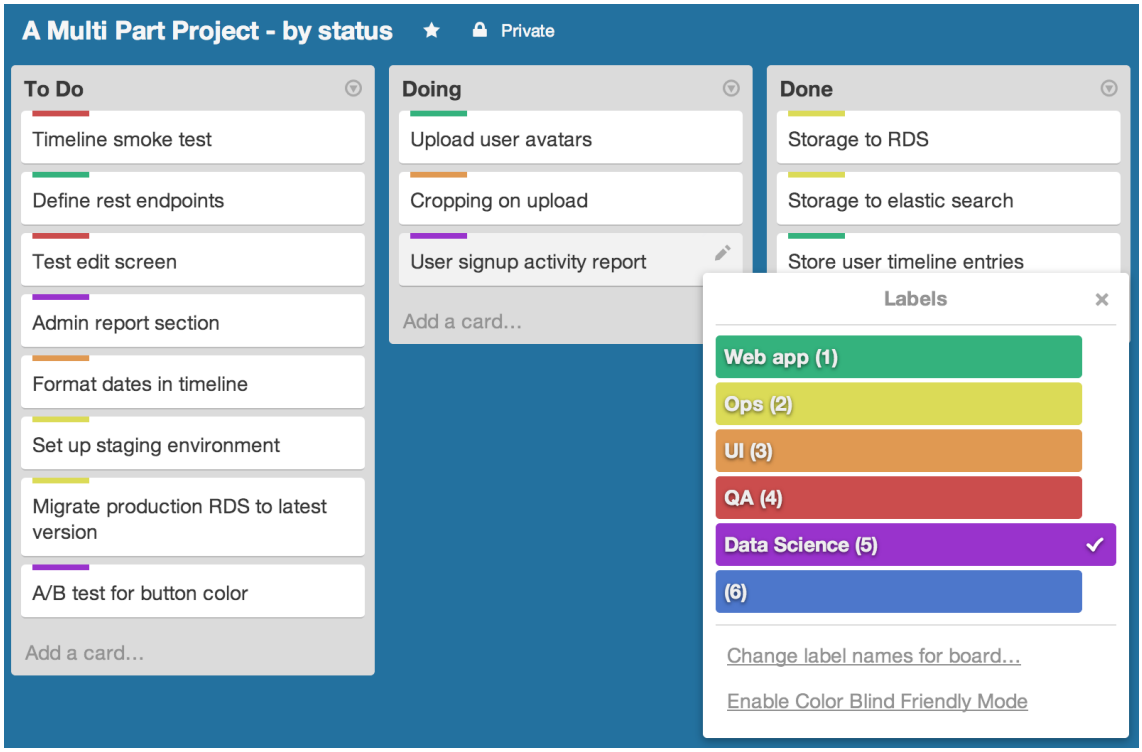
Orodje Trello, testirano junija 2015, je preprosta spletna aplikacija, ki prikaže sezname delovnih nalog razporejene po stolpcih. Naloge predstavljajo kartice, ki jih lahko s potegom miške prestavljamo v druge sezname in jih v seznamu tudi razvrščamo. V prvi vrsti je orodje namenjeno upravljanju seznama nalog, lahko pa ga uporabljamo za vizualizacijo delovnega toka s sistemom kanban. Videti je kot fizična tabla, kartice na njej pa kot samolepilni lističi. Je zelo enostavno in preprosto orodje, namenjeno posameznikom in tudi velikim organizacijam. Orodje je bilo izdano leta 2011, septembra 2014 pa so dosegli že 5 milijonov uporabnikov. Orodje je na voljo kot spletna aplikacija, do katere lahko dostopamo prek vseh naprav in brskalnikov. Na svoje mobilne naprave ali tablice z operacijskimi

sistemi iOS, Android ter Windows lahko namestimo tudi mobilno verzijo aplikacije. Poleg brezplačne različice je na voljo še Trello Gold, ki je bolj napredna različica in ponuja menjavo ozadja, dodatne sličice oz. nalepke (angl. stickers), ki jih lahko uporabimo na tabli ter več prostora za priponke. Ponujajo pa še različici Tello Business Class in Enterprise za ekipe, ki potrebujejo večjo varnost, integracijo z Google aplikacijami, bolj natančno upravljanje pravic uporabnikov, naprednejši izvoz podatkov ter telefonsko uporabniško podporo in osebnega svetovalca za račun.

Trello ima na voljo API, ki omogoča povezovanje z drugimi orodji. API lahko uporabimo tudi za izvoz podatkov za varnostno kopijo ali celo za pretvorbo podatkov v druge formate.

V brezplačni verziji lahko uporabnik kreira poljubno število tabel. Ustvarimo jih ročno prek gumba ali pa tako, da kopiramo obstoječo tablo. Na tablo lahko dodamo poljubno število stolpcev, ki se v programu imenujejo seznami (angl. lists). Primer table je prikazan na sliki 3.4. Uredimo jim lahko imena, jim spremenimo vrstni red, ne moremo pa jim določati širine. Največja razlika med orodjem Trello in drugimi orodji za podporo kanban razvoju je ta, da Trello nima privzetega omejevanja obsega dela. Omenjeno prakso Kanbana lahko implementiramo na različne načine. Kot na fizični tabli lahko tudi tukaj zapišemo omejitev WIP, ki je samo opisna, v naslovu stolpca, a v primeru prekoračitve nas orodje na to ne opozori. Druga možnost je namestitev dodatka za brskalnik, kot je npr. Kanban WIP for Trello ali CardCounter for Trello. Trello pa ne podpira kreiranja plavalnih prog.

Posamezne kartice dodajamo na tablo ročno ali po e-mailu. Kartice lahko tudi prestavljamo iz ene tabele v drugo. Kartici lahko dodamo opis, slike, priloge, dogovorjeni rok za dokončanje, barvne oznake, lastnika ali lastnike kartice ter zapiske diskusij vseh udeleženih uporabnikov. Program ne nudi polja za velikost kartice in tudi tukaj moramo biti malo kreativni. Lahko zapišemo velikost v naslov kartice, kartici lahko dodamo toliko nalepk, kot smo ji dodelili točk, ustvarimo lahko posebne lastnike, ki predstavljajo velikost kartice in jih dodelimo karticam, ali pa namestimo katerega od dodatkov za brskalnike, kot je npr. Scrum for Trello, ki doda omenjeno funkcionalnost. Kartica vsebuje še seznam opravil s potrditvenimi polji (angl. todo checklist), kar omogoča, da ostaja razumne velikosti, medtem ko v njej še vedno beležimo podrobnosti. Sezname opravil lahko kopiramo iz drugih kartic, opravila na seznamu pa lahko tudi pretvorimo v kartice. Kartice vsebujejo tudi polje za povezavo, ki se lahko uporabi za vnos povezave do mesta v izvorni kodi na GitHub računu. Čustva povezana z delom lahko zapišemo v komentarje na kartici ali v opis, kjer lahko uporabimo tudi smeške. Opis na kartici podpira Markup sintakso. Za boljšo prepoznavnost lahko kartici spremenimo še naslovno stran, kamor naložimo želeno sliko. Karticam v stolpcu lahko s potegom miške urejamo vrstni red in tako označimo prioriteto



Slika 3.4: Kanban tabla v orodju Trello [68]

delovnih nalog ali pa uporabimo funkcijo glasovanja (angl. Voting on cards). Na voljo pa nam je tudi indikator za prikaz starosti kartice (angl. Card Aging), ki spreminja barvo tistim karticam, ki jih dolgo nismo urejali.

Drugi dodatki za brskalnik Chrome so tudi Export for Trello, ki izvozi podatke v Excel, SpeakUp, ki je dodatek za spodbujanje medekipnega sodelovanja ter ustvarjanja novih idej. Povežemo ga lahko tudi z orodjem Toggl [38], ki skrbi za spremljanje porabljenega časa na delovni nalogi, kar je koristno za izstavljanje računov naročniku, in z orodjem Zapier [40] za integracijo s storitvijo GitHub za hranjenje repozitorijev.

Orodje nima privzetih metrik in poročil, lahko jih namestimo z dodatki za brskalnik Reports for Trello ali z nakupom storitev drugih podjetij, kot sta plačljiv Chartbreeze [24] ali brezplačna Trello Cumulative Flow [39] oz. Burndown for Trello [21].

Na voljo je iskanje kartic s posebnimi ukazi ter osnovno filtriranje glede na izbrano oznako, lastnika kartice, dogovorjeni rok za dokončanje in ključne besede.

Tablo lahko delimo z drugimi Trellovimi uporabniki in se naročimo na vse spremembe, ki se zgodijo na tabli.

Poleg vizualizacije delovnega toka na tabli ponuja orodje še pregled kartice na koledarju glede na dogovorjeni rok za dokončanje.

Orodje je zaradi svoje preprostosti ter videza, ki spominja na pravo fizično tablo, zelo enostavno za uporabo. Na spletni strani je na voljo izčrpna dokumentacija in tudi na

YouTubu ali Googlu lahko najdemo zelo veliko člankov z opisi različnih načinov uporabe orodja Trello. Tako je za začetnike manj »zastrašujoče« kot kakšno drugo orodje, hkrati pa je primerno tudi za poznavalce, ki ga lahko uporabljajo z bližnjicami na tipkovnici. Deluje na vseh napravah, ponuja enostavno upravljanje s pravicami uporabnikov in zato zelo poenostavi izmenjavo informacij z naročnikom. Uporabnik si lahko ustvari Trello račun na podlagi Googlovega računa, kar še olajša dostopnost.

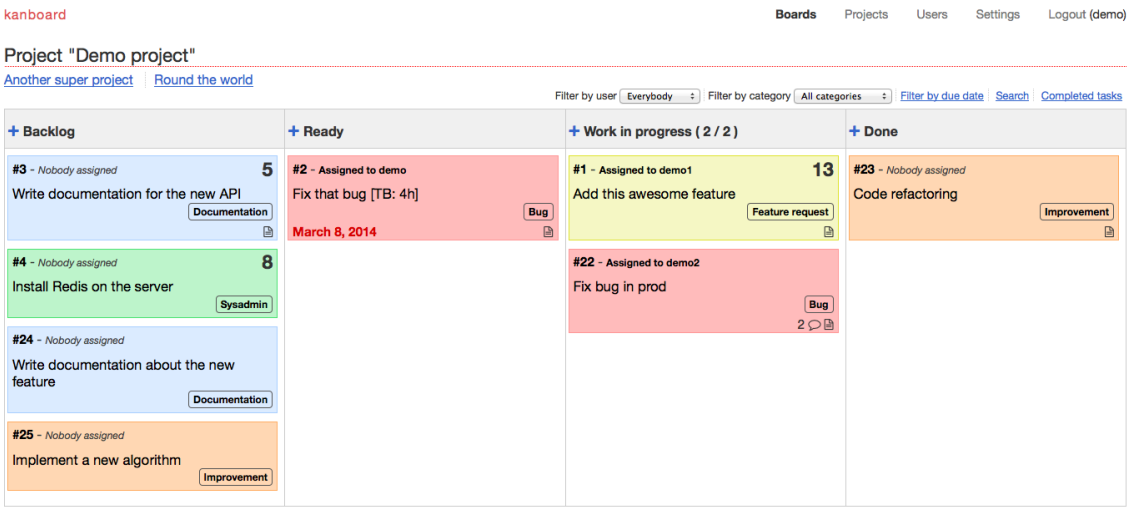
3.2.4 Kanboard

Kanboard je brezplačno odprtokodno orodje, namenjeno podpori razvoja programske opreme po metodi Kanban. Testirano je bilo junija 2015. Avtor orodja je Frédéric Guillot, orodje je bilo izdano leta 2014. Za delovanje potrebujemo PHP 5.3.7 ali novejšo verzijo na Apache ali Nginx strežnika. Kanboard privzeto shranjuje podatke v podatkovno bazo Sqlite, podpira pa tudi uporabo podatkovnih baz Mysql ali Postgresql. Za njegovo namestitev prilepimo izvorno kodo v ustrezno datoteko. Natančna in jasna navodila za namestitev orodja so na voljo na spletni strani orodja. Preden pa se odločimo za namestitev, lahko preizkusimo spletno demo različico. Kanboard je preveden v 18 jezikov in testiran v brskalnikih Mozilla Firefox, Safari, Google Chrome in Internet Explorer 11 ter iOS in Android tablicah. Možna je integracija s spletnimi repozitoriji Github, Bitbucket ter Gitlab, ki omogočajo avtomatske spremembe na tabli, in integracija z Hipchat, Slack ter Jabber. Ta omogoča pošiljanje obvestil za spremembe na tabli v omenjena orodja za ekipno komunikacijo (angl. private group chat). Prijavljanje v orodje je možno tudi prek sistemov LDAP, ActiveDirectory, Google in Github. Na voljo je API za kreiranje nove table, delovnih nalog, uporabnikov ter dostopanje do vseh funkcionalnosti, ki jih imamo v uporabniškem vmesniku. Kanboard lahko upravljamo tudi prek ukazne vrstice na strežniku. Skupnost je aktivna in praktično vsak mesec izda novo različico orodja ter objavi novičko z opisom sprememb. Orodje lahko uporabljamo tudi prek bližnjic na tipkovnici.

Na prvi strani se uporabniku prikažejo vsi projekti, pri katerih dela, njegove delovne naloge (angl. tasks) ter manjša opravila (angl. subtasks). Pokaže se tudi koledarski pregled kartic v delu in seznam zgodovine sprememb v sistemu. Videz table je mogoče popolnoma spremeniti s slogovnim jezikom CSS, navodila za to pa najdemo v dokumentaciji pod naslovom Contribution. Nekatere podatke na tabli, kartice, manjša opravila, zgodovino sprememb na kartici ter dnevni povzetek projekta lahko izvozimo v CSV-datoteko.

Vsaka nova tabla predstavlja nov projekt. Delovne naloge so predstavljene s karticami na tabli in jih premikamo s potegom miške. Stolpcem lahko določimo naziv, spremenimo jim lahko vrstni red ter jim določimo omejitev WIP. Primer table je prikazan na sliki 3.5. Če presežemo omejitev WIP, se celotni stolpec obrava rdeče. Orodje dopušča kreiranje plavalnih prog, ki jih po želji lahko skrijemo oz. razkrijemo. Omejitev WIP na stolpcu

je ločena za vsako plavalno progo posebej. Omogočeno je kopiranje table, kjer lahko označimo, katere entitete želimo kopirati. Entitete, ki so na voljo za kopiranje, so kategorije, akcije, plavalne proge ter kartice.



Slika 3.5: Kanban tabla v orodju Kanboard [59]

Karticam lahko določimo naziv, opis, kategorijo, lastnika, plavalno progo, stolpec, barvo, dogovorjeni rok za dokončanje, velikost naloge, izraženo s številko, in tudi ocenjeno težavnost, ki predstavlja število dni. Opis na kartici podpira Markdown sintakso. V njenih nastavitvah lahko dodatno kreiramo manjša opravila, povezave, komentarje, dokumente in zaslonske slike. Nastavimo lahko ponovljivost kartice, jo kopiramo ali prestavimo na drugo tablo. Lahko jo zapremo ali izbrišemo. Zaprte kartice se na tabli ne prikažejo več, prikažejo se v seznamu zaključenih nalog. Po želji lahko vklopimo beleženje časa. V nastavitvah kartice je omogočen pregled zgodovine sprememb (angl. task transitions). Kartice lahko ustvarimo tudi prek e-maila. Kartica na tabli prikaže posebne ikone za indikator napredka, ki kaže, koliko časa je bila kartica v stolpcu, njeno starost, kategorijo, lastnika, dogovorjeni rok za dokončanje, odstotek opravljenih manjših opravil, opis in velikost naloge. Prikaza kartice ni mogoče spreminjati.

Uporabnikom lahko določimo urno postavko in razpored dela ter ga naročimo na obveščanje o spremembah na tabli prek e-maila ali RSS.

Na voljo so tudi avtomatske akcije oz. pravila, ki se izvedejo po nekem dogodku. Primeri so avtomatska sprememba barve kartice, kadar jo prevzame določen uporabnik, nastavitev blokade kartice, avtomatska zapora kartice, ko se ta prestavi v stolpec »zaključeno«, avtomatsko določanje lastnika kartice v trenutno prijavljenega uporabnika, ko ta prestavi kartico v stolpec »v delu« itd. Vse akcije so opisane v dokumentaciji orodja.

Vse spremembe na tabli se beležijo v zgodovini projekta, beležijo se tudi spremembe

na ravni uporabnika.

Tablo lahko filtriramo glede na lastnika, dogovorjeni rok za dokončanje ali kategorijo.

Metrike, ki jih orodje podpira, so CFD, padajoči diagram spremljanja napredka, krožni diagram za porazdeljenost kartic po lastniku kartice (angl. User repartition) ali po stolpcih (angl. Task distribution). Vse grafe lahko izvozimo v CSV-datoteko.

Tablo lahko samo za branje delimo z zunanjimi uporabniki, ki se jim ni treba prijaviti v orodje.

Na spletni strani projekta je na voljo dokumentacija, ki je še v nastajanju, a kljub temu nisem potrebovala veliko časa, da sem namestila orodje in se ga naučila uporabljati. Vse predloge za izboljšavo orodja, morebitne napake ali svoje prispevke lahko oddamo na GitHub repozitoriju projekta [27].

3.3 Primerjava

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Na voljo brezplačno	prvih 30 dni	prvih 7 dni	osnovna verzija	da	da
Povezljivost z drugimi programi	LeanKit API	Jira API	Trello API	JSON-RPC API	ne
Poljubno število stolpcev	da, nastavljiva širina	da, fiksna širina	da, fiksna širina	da, fiksna širina	prilagodimo po želji
Kopiranje table	da	da	da	da	ne
Omejevanje obsega dela	na nivoju stolpca, uporabnika ali plavalne proge	na nivoju stolpca, vključuje ali izključuje podopravila	na nivoju stolpca s posebnim dodatkom za brskalnik	na nivoju stolpca in plavalne proge	zapišemo na vrh stolpca ali vrstice, nudimo le določeno število ikon ali uporabimo plavalne proge za omejevanje na nivoju uporabnika
Prekoračitev omejitve WIP	stolpec se obarva rdeče, zahteva vpis pojasnila	stolpec se obarva rdeče	ne, stolpec se obarva rdeče s posebnim dodatkom za brskalnik	stolpec se obarva rdeče	ne

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Razdelitev stolpcev navpično na podstolpce	neomejeno nivojev, omejimo delo na katrem koli nivoju	ne	ne	ne	prilagodimo po želji, omejitev določimo za posamezen stolpec ali več hkrati
Kreiranje plavalnih prog	da	da	ne	da	da
Obravnavanje nujnih delovnih nalog	z uporabo plavalne proge	z uporabo plavalne proge in JQL filtra	ne	z uporabo plavalne proge	z uporabo plavalne proge ali na posebni tabli
Kriteriji za prehajanje delovnih nalog v drugo stanje	v opis stolpca	ne	ne	ne	zapišemo poleg table
Upravljanje pravic uporabnikov	zelo napredno, določimo kdo pregleduje, ustvarja, premika kartice, itd.	zelo napredno, uporabimo vloge, nastavimo pravice za katerokoli projektno aktivnost	nastavimo pravice za komentiranje, glasovanje ter dodajanje novih članov	nastavimo dve uporabniški vlogi, eno za delo s tablo, drugo za upravljanje nastavitev	ustni dogovor

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Prilagoditev prikaza informacij na kartici	spremenimo ikono tipu in razredu naloge	spremenimo ikono tipu zahtevka in prioriteti, dodamo 3 dodatna polja iz zahtevka	naložimo sliko za naslovno stran kartice	ne	odebeljen dogovorjeni rok za dokončanje, barve in prikaz po želji
Sprememba barve kartice	odvisna od izbranega tipa naloge	odvisna od določenega JQL pogoja	barvne oznake	da	da
Več lastnikov kartice	da	ne	da	ne	da
Zajemanje informacije na kartici	opis, šifra, tip in razred naloge, dogovorjeni rok za dokončanje, prioriteta, blokada, lastnik, komentarji, priponke, zgodovina	identifikacijska številka zahtevka, opis, ikona lastnika, tip zahtevka	opis, slike, priloge, dog. rok za dokončanje, barvne oznake, lastnik, zapiski, povezave	opis, kategorija, lastnik, plavalna proga, stolpec, barva, dog. rok za dokončanje, velikost naloge, ocenjena težavnost, povezave, komentarji, dokumenti, zaslonske slike, zgodovina	prilagodimo po želji

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Prikaz blokade	posebna ikona na kartici	potrebno dodati novo polje zahtevku in nastaviti filter za prikaz blokade na plavalni progi	ne	ne	z uporabo druge barve samolepilnega lističa, magneta ali na posebni plavalni progi oz. ločeni tabli, lahko beležimo št. blokiranih dni
Prikaz čustev povezanih z delom	med komentarji	med komentarji	med komentarji	med komentarji	z uporabo smeškov
Indikator napredka	niso prikazani na kartici, razberemo iz zgodovine sprememb	prikaže št. dni, ko je kartica v nespremenjenem stanju	starost kartice	prikaže št. dni, ko je kartica v nespremenjenem stanju	narišemo pikico za vsak dan kartice v razvoju
Časovni žigi vstopov v stanja	v zgodovini sprememb	v zgodovini sprememb	v zgodovini sprememb	v zgodovini sprememb	ne
Naročanje na spremembe	email, RSS	email	email	email, RSS	ne

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Izvoz podatkov	CSV, KanbanSim	CSV, Excel, XML	Trello JSON, Excel z namestitvijo posebnega dodatka za brskalnik	CSV	ne
Razvrščanje po prioriteti	ročno	ročno ali avtomatsko glede na prioriteto zahtevka	ročno ali glede na dog. rok za dokončanje	ročno	ročno
Razdelitev delovnih nalog na manjša opravila	da, upravljamo na ločeni tabli za opravila na kartici	da, upravljamo kot kartice na isti tabli	da, upravljamo seznam opravil na kartici	da, upravljamo seznam manjših opravil na kartici	da, uporabimo samolepilne lističe različnih barv in velikosti
Dvonivojska struktura	da, z neomejenim nivojem podstolpcev in plavalnih prog	ne	ne	ne	prilagodimo po želji
Točka orderpoint	ne	ne	ne	ne	da
Pričakovan čakalni čas delovne naloge	ni neposredno prikazan, razberemo iz diag. povprečnega cikla obdelave	ni neposredno prikazan, razberemo iz diag. povprečnega cikla obdelave	ne	ne	ne

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Oddaljen dostop do table	da, potrebna prijava in ustrezne pravice	da, potrebna prijava in ustrezne pravice	da, tudi brez prijave	da, tudi brez prijave	projiciranje s kamero
Metrike za spremljanje napredka	CFD, diag. povp. cikla obdelave, diag. porazdeljenosti nalog, diag. učinkovitosti, diag. odstopanja od povp. cikla obdelave	CFD, diag. odstopanja od povp. cikla obdelave in mnogi drugi	ne, CFD s posebnim dodatkom za brskalnik	CFD, padajoči diag. spremljanja napredka, diag. porazdeljenosti nalog	ne
Filtriranje podatkov	da, glede na izbrani atribut s kartice	da, glede na prednastavljene JQL kriterije	da, glede na izbrano oznako, lastnika, dog. rok za dokončanje ali ključne besede	da, glede na lastnika, dog. rok za dokončanje ali kategorijo	ne
Koledarski prikaz	da	ne, na voljo s posebnim vtičnikom	da	da	ne

Merilo	LeanKit	Jira Agile	Trello	Kanboard	Fizična tabla
Beleženje porabljenega časa	ne, lahko razberemo datum prevzema kartice in datum zaključitve kartice iz zgodovine	da	ne, na voljo s posebnim dodatkom za brskalnik	da	ne
Scrum podpora	scrum tabla, velikost naloge, omejimo WIP na izbrano hitrost ekipe, ročno arhiviramo vsebino table po končani iteraciji, graf dokončanega dela	hitrost ekipe, upravljanje seznama zahtev po prioriteti, ocenjevanje težavnosti up. zgodb, scrum tabla, padajoči diag. spremljanja napredka	scrum tabla, velikost naloge ter padajoči diag. spremljanja napredka s posebnim dodatkom za brskalnik	scrum tabla, velikost naloge, padajoči diag. spremljanja napredka	scrum tabla
Zahtevnost učenja	učenje zahtevno, kreiranje table zelo napredno, vendar pregledno in razumljivo	učenje in kreiranje table zelo zapleteno	enostavno	več dela z namestitvijo, učenje srednje zahtevno, kreiranje table enostavno	učenje zahtevno, kreiranje table enostavno

Tabela 3.1: Primerjava orodij.

Poglavje 4

Načrt vpeljave

V tem poglavju bomo predstavili predlog vpeljave metode Kanban v realni projekt. Predstavili bomo projekt in razvojno ekipo, dosedanje vodenje razvoja programske opreme, razloge za spremembo projektne metodologije ter izdelali načrt vpeljave metode Kanban.

4.1 Opis projekta

Med letoma 2007 in 2012 je avtorica diplomskega dela kot razvijalka programske opreme sodelovala pri projektu prenove informacijskega sistema za podporo poslovnih procesov na področju premoženjskih zavarovanj ene od naših večjih zavarovalnic. Projekt je izvajalo srednje veliko slovensko IT-podjetje, ki se ukvarja z razvojem in trženjem programske opreme za poslovna okolja in ima med 100 in 150 zaposlenih sodelavcev.

Pri projektu prenove informacijskega sistema je delovala razvojna ekipa, sestavljena iz petih razvijalcev programske opreme, skupaj s projektnim vodjo, arhitektom programskih rešitev, analitikom in sodelavcem za testiranje aplikacij. Ta razvojna ekipa je v skladu z organizacijsko strukturo podjetja o svojem delu poročala še direktorju razvoja informacijskih rešitev za zavarovalnice.

Proces razvoja programske opreme je potekal tradicionalno po fazah:

- analiza zahtev, pri kateri smo identificirali in dokumentirali naročnikove zahteve;
- načrtovanje, pri katerem smo izdelali načrt programske opreme;
- kodiranje, pri katerem smo načrt programske opreme prevedli v programski jezik Java;
- testiranje, pri katerem smo preverjali kakovost izdelane programske opreme in ustreznost osnovnim zahtevam, zapisanim v specifikaciji naročnika;

- uvajanje, pri katerem smo šolali uporabnike in nameščali izdelano programsko opremo v uporabniško okolje.
- vzdrževanje, pri katerem smo programsko opremo prilagajali različnim zunanjim okoliščinam, v katerih je sistem deloval.

Razvojna ekipa je izdajala redne delujoče verzije programskega produkta, vendar je z večanjem obsega dela in s projektnimi pristopi, ki smo jih uporabljali, postalo usklajevanje in obvladovanje prioritet produkta težavno, njegova implementacija pa premalo učinkovita. Slabosti, ki so se pokazale pri našem delu, so bile predvsem počasnejši razvoj programske opreme in s tem nedoseganje rokov ter slabša kakovost programske kode, kar se je izražalo tudi z nezadovoljstvom tako naročnika kot tudi članov razvojne ekipe.

Na to so vplivali predvsem naslednji dejavniki:

- pomanjkljiva komunikacija z naročnikom;
- pomanjkliva komunikacija med člani razvojne ekipe;
- nejasno definirane prioritete;
- nejasna vizija razvoja produkta;
- nekoordinirani viri zahtev;
- nakopičeno vsebinsko znanje na ravni posameznikov in ne na ravni celotne razvojne ekipe.

Pri preučevanju literature za diplomsko delo smo se seznanili z različnimi agilnimi praksami in načeli vitkosti, za katere avtorica diplomskega dela meni, da bi lahko koristila razvojni ekipi, če bi jih uporabljala pri reševanju težav, s katerimi se je takrat soočala.

Na podlagi tako pridobljenih spoznanj smo zato v diplomsko delo vključili tudi predlog vpeljave načel vitkosti po metodi Kanban v tradicionalni proces razvoja projekta, pri katerem je avtorica diplomskega dela sodelovala in ima s takim načinom projektnega dela praktične izkušnje.

4.2 Predlog vpeljave metode Kanban

V razvojni ekipi našega projekta večina članov že vrsto let razvija programsko kodo na tradicionalen način. Razvijalci delajo po svojih, že davno ustaljenih navadah, ki jih je težko spremeniti, saj so trdno prepričani, da kar najbolje služijo svojemu namenu. Pri tem pa so tudi dovolj usposobljeni, da znajo produkt, kljub začetnim napakam, tik pred izdajo stabilizirati in ga izdati naročniku v zadovoljivi kakovosti in v zanj še sprejemljivem času po načrtovanem roku.

Prav zato je bistvenega pomena, da na prvem mestu razvojno ekipo prepričamo, da je treba spremeniti način dela, in ji pojasnimo, zakaj je taka sprememba pomembna. Ekipi damo dovolj časa in ji omogočimo dostop do baze znanja ali ji zagotovimo šolanje za uporabo metode Kanban s kanban trenerjem. Nato postopoma uvajamo majhne spremembe, ki so obvladljive in naj ne bi povzročale nejevolje in odpora članov razvojne ekipe.

Prednost metode Kanban namreč je, da ga ekipe lahko vpeljejo takoj, ko se z njo seznanijo, in postopno uvajajo izboljšave v delovni proces v kateri koli fazi projekta, ne da bi se le-ta radikalno spremenil. Delo razvojne ekipe tako poteka skoraj enako kot prej, obdržimo obstoječe vloge in terminologijo, poudarek je na kakovosti programske kode in na prepoznavanju najbolj pomembnega dela. Kritično delo, za katerega želimo, da je opravljeno najprej, prepoznavamo s pomočjo vizualizacije delovnega toka – s kanban tablo.

4.2.1 Vizualizacija delovnega toka

Za začetek bomo uporabili fizično kanban tablo, ki je bolj enostavna za vzpostavitev kot elektronska verzija, saj omogoča hitre in enostavne spremembe ter prilagoditev delovnemu procesu. Velika in dobro postavljena tabla ter premikanje samolepilnih lističev namreč povečuje medekipno sodelovanje in prevzemanje odgovornosti.

Kasneje, ko se ekipa navadi na delo po kanban sistemu, vpeljemo še elektronsko verzijo kanban table in poskrbimo za njeno dnevno sinhronizacijo. Ker razvojna ekipa že vrsto let uporablja elektronski sistem za vodenje in sledenje delovnim nalogam Atlassian Jira, je smiselna uporaba vtičnika Jira Agile, ki v orodje Jira dodaja možnost uporabe kanban table.

Posamezna stanja v delovnem procesu, ki jih uporabimo na tabli, določimo s poimenovanjem korakov našega procesa. Ti so naslednji:

- Prevzem Jira zahtevka
- Analiza
- Načrtovanje
- Razvoj
- Testiranje
- Izdaja verzije

Prvi korak prevzemanje Jira zahtevka pretvorimo v stanje »seznam nalog«, kjer hranimo vse zahtevke z opisi uporabniških zgodb. Zahtevki na naši tabli, predstavljajo kanban kartice. Dodamo stanje »pripravljene naloge«, kjer prikažemo trenutno najbolj pomembne zahtevke, ki jih dodatno razvrstimo po prioriteti.

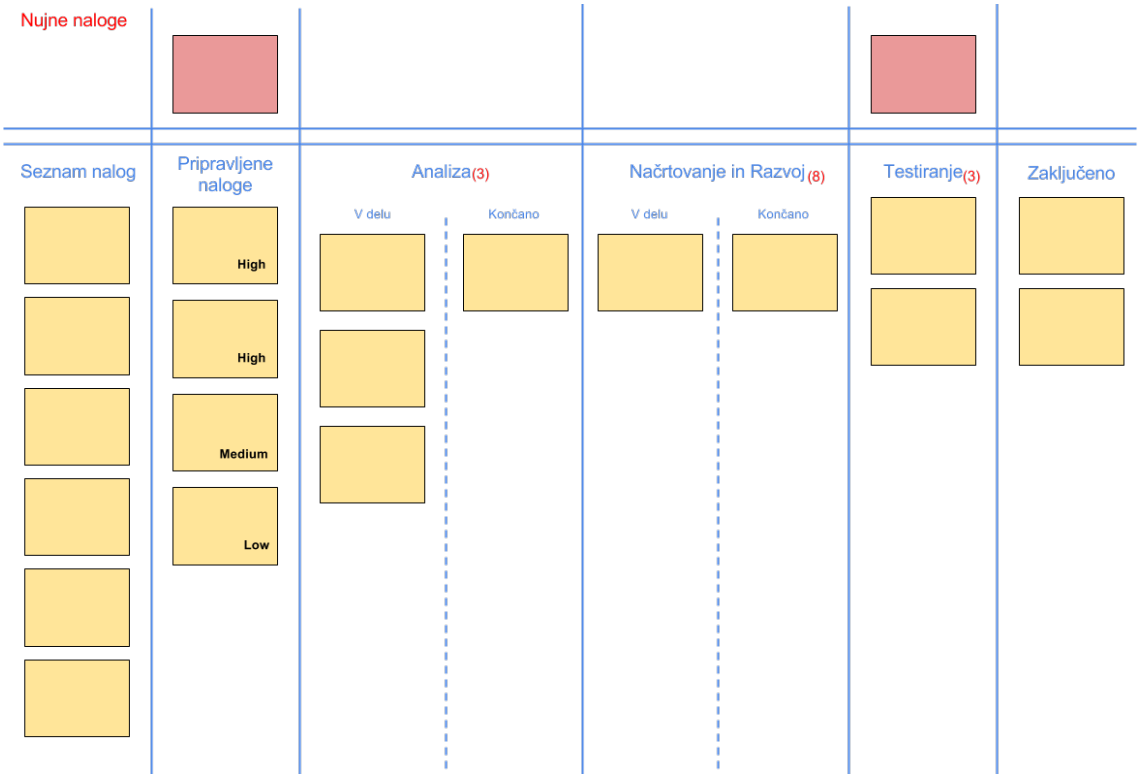
Cilj koraka »analiza« je popolno razumevanje nalog in potrebnega dela za dokončanje teh.

Koraka »načrtovanje« in »razvoj« lahko združimo v eno stanje, saj ju praviloma opravlja ista oseba. V tem koraku, razvijalec razdeli uporabniške zgodbe na enako velike manjše naloge, naredi načrt razvoja, jih razvije in tudi potrdi izvorno kodo na repozitorij.

Koraka »analiza« ter »načrtovanje in razvoj« lahko za začetek prikažemo vsakega z dvema podstolpcema. Levi podstolpec »V delu« vsebuje kartice, ki se analizirajo oz. planirajo in razvijajo, desni podstolpec »Končano« pa tiste kartice, ki so že analizirane oz. planirane in razvite.

Stolpec »testiranje« vsebuje kartice, ki se morajo preveriti. Zadnji korak, kamor postavimo kartice po preverjanju, predstavimo s stanjem »zaključeno«. Ko pripravimo izdajo nove verzije programa, odstranimo kartice iz stanja »zaključeno«.

Na tablo dodamo še plavalno progo za obravnavanje posebnih primerov. Izgled naše tabla je predstavljen na sliki 4.1.



Slika 4.1: Kanban tabla z omejenim obsegom dela

4.2.2 Omejevanje obsega dela

Omejitev obsega dela sprva nastavimo na dvakratno vrednost velikosti razvojne ekipe in jo ustrezno zmanjšujemo vsakih nekaj tednov, dokler ne naletimo na zastoje. Naša razvojna

ekipa ima 7 članov, skupna omejitev WIP je tako 14 delovnih nalog in razdelimo jo med tiste stolpce, za katere menimo, da bodo vplivali na lepše prehajanje nalog skozi delovni tok. V našem primeru omejitev WIP razdelimo med stolpce »analiza«, »načrtovanje in razvoj« ter »testiranje«. Stolpcu »načrtovanje in razvoj« za začetek omejimo WIP na število razvijalcev, ki je 5, in 50-odstotne rezerve. Omejitev WIP stolpca »načrtovanje in razvoj« je tako 8 delovnih nalog. Stolpcema »analiza« in »testiranje« pa za omejitev WIP razdelimo preostalih 6 delovnih nalog, in sicer po 3 vsakemu.

4.2.3 Vzpostavitev pravil v procesu

Za začetek vzpostavimo mehanizem za primerjanje delovnih nalog za določanje njihovih prioritet, nato se dogovorimo, da se pri prevzemu novega dela vedno izbere vrhnjo kartico s seznama »pripravljenih nalog«. Poleg tega vzpostavimo naslednja pravila:

- Preden se lotimo novih delovnih nalog, preverimo, ali lahko pomagamo končati tiste naloge, ki so že v izvajanju.
- Pogledamo, ali so morebitni zastoji oz. blokade, ki upočasnjujejo delovni tok in jih skušamo odpraviti.
- V sistem povlečemo novo delo, a le v primeru, da ne kršimo omejitve WIP.

Druga pravila, ki jih vzpostavimo, so:

- možnost preurejanja kartic v stolpcu »pripravljene naloge« glede na trenutno prioriteto nalog;
- jasna definicija končanega dela;
- obravnava le enega nujnega primera naenkrat ter obravnava napak kot preostalih nalog, torej s postavitvijo kartice nazaj v stolpec »načrtovanje in razvoj«.

Poleg tega redno uporabimo dobre inženirske prakse za programiranje in testiranje, se dnevno sestajajmo na dnevnih sestankih, po vsaki novi izdaji programskega produkta pa izvedemo retrospektivo in proslavimo uspeh.

Poglavje 5

Sklepne ugotovitve

Pri izdelavi diplomskega dela smo se spoznali z agilnimi in vitkimi metodologijami za razvoj programske opreme. Podrobno smo analizirali metodi Scrum in Kanban ter ugotovili, da se metodi ne izključujeta. Scrumban, ki je kombinacija vitkega razmišljanja in agilne prakse, sodeč po raziskavah, postaja vedno bolj priljubljena ter tudi uspešna metoda. Spoznali smo, da vizualizacija delovnega toka, omejevanje obsega dela ter načelo vlečenja dela razbremenijo ekipo ter vplivajo na produktivnost in kakovost dela. Scrumban in Kanban temeljita na sistemu kanban, ki pomaga ustvariti tako organizacijsko klimo, da se ljudje radi vključujejo v delo in so avtonomni.

V diplomskem delu smo analizirali orodja za elektronsko in fizično podporo sistema kanban. Ugotovili smo, da je na voljo veliko spletnih orodij, ki različno dobro podpirajo zmogljivosti metode Kanban in potrebujejo kar nekaj konfiguracije, da jih pravilno vzpostavimo. Neprimerno bi bilo izpostaviti neko orodje kot najboljše, pri izbiri je namreč pomembno, da dobro poznamo okoliščine, v katerih delamo, in razumemo, za kakšne namene želimo orodje uporabljati.

V nadaljnji karieri nam bodo koristile izkušnje in znanja, ki smo jih pridobili pri preučevanju metodologij in analiziranju elektronskih kanban orodij. Tradicionalno voden projekt, pri katerem je avtorica sodelovala, bi spremenili z metodo Kanban, s katero lahko začnemo delati ne da bi spreminjali obstoječi razvojni proces, kjer spoštujemo trenutne vloge in postopoma spreminjamo proces – evolucijsko, in ne revolucijsko kot pri metodi Scrum. Koraki, s katerimi začnemo uvedbo metode Kanban so vzpostavitev vizualizacije delovnega toka, omejevanje obsega dela in definiranje jasnih pravil v procesu.

Diplomsko delo prikaže uspešnost metode Kanban pri manjših projektih, zato ni presenetljivo, da se v zadnjih letih išče načine za prilagajanje vitkih in agilnih metod, da ustrezajo čim večjim projektom. Ideja za nadaljnje delo je preučitev metod za podporo velikih in porazdeljenih projektov, kot so Scaling Agile at Spotify [32], Scaled Agile Framework [32], Disciplined Agile Delivery [25], Large-Scale Scrum [28], The Scaled Profes-

sional Scrum [33] ter Enterprise Scrum [26]. Prav tako bi se bilo zanimivo poglobiti v teoretične modele za napovedovanje poteka projekta, npr. s teorijo čakalnih vrst ali s simulacijsko metodo Monte Carlo.

Seznam slik

Seznam tabel

Literatura

[1] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[2] Kent Beck, Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley; 2nd edition, 2004.

[3] Ken Schwaber, Mike Beedle. *Agile Software Development with Scrum*. Pearson, 2001.

[4] Mike Burrows. *Kanban from the Inside: Understand the Kanban Method, connect it to what you already know, introduce it with impact*. Blue Hole Press, 2014.

[5] Mike Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 2009.

[6] Johanna Rothman, Scott W. Ambler, Suzanne Robertson, Ron Jeffries, Peter Kaminski, Israel Gat, Hubert Smits, Hillel Glazer. Crossing the Agile Divide: Scrum or Kanban? *Agile Product & Project Management*, 15(2):1–24, 2014. Dostopno na `http://www.cutter.com/project/fulltext/reports/2014/02/index.html`.

[7] Henrik Kniberg. *Scrum and Xp from the Trenches*. Lulu Press, Inc., 2007.

[8] Henrik Kniberg. *Lean from the Trenches: Managing Large-Scale Projects with Kanban*. Pragmatic Bookshelf; 1 edition, 2011.

[9] Corey Ladas. *Scrumban - Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press, 2009.

[10] Vilijan Mahnič. Improving Software Development through Combination of Scrum and Kanban. *Recent Advances in Computer Engineering, Communications and Information Technology*, strani 281–288, 2014. Dostopno na `http://www.wseas.us/e-library/conferences/2014/Tenerife/INFORM/INFORM-40.pdf`.

[11] Hirotaka Takeuchi, Ikujiro Nonaka. The New New Product Development Game. *Harvard Bussines Review*, 64(1):137–146, 1986. Dostopno na `https://hbr.org/1986/01/the-new-new-product-development-game`.

[12] Andrew Pham, Phuong-Van Pham. *Scrum in Action.* Cengage Learning PTR; 1 edition, 2011.

[13] Andrew Thu Pham, David Khoi Pham. *Business-Driven IT-Wide Agile (Scrum) and Kanban (Lean) Implementation: An Action Guide for Business and IT Leaders.* CRC Press; 1 edition, 2012.

[14] Daniel H. Pink. *Drive: The Surprising Truth About What Motivates Us.* Riverhead Books, 2011.

[15] M.A. Poppendieck, M., Cusumano. Lean Software Development: A Tutorial. *Software, IEEE*, 29(5):26–32, 2012. Dostopno na `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6226341&filter%3DAND%28p_IS_Number%3A6276289%29`.

[16] Mary Poppendieck, Tom Poppendieck. *Lean Software Development: An Agile Toolkit.* Addison-Wesley Professional, 2003.

[17] Ajay Reddy. *The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban.* Addison-Wesley Professional; 1 edition, 2015.

[18] Henrik Kniberg, Mattias Skarin. *Kanban and Scrum - making the most of both.* Lulu Press, Inc., 2010.

[19] Dag I.K. Sjøberg, Anders Johnsen, Jørgen Solberg. Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software*, 29(5):47–53, 2012. Dostopno na `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6231615&filter%3DAND%28p_IS_Number%3A6276289%29`.

[20] Marcus Hammarberg, Joakim Sunden. *Kanban in Action.* Manning Publications; 1 edition, 2014.

Spletni viri

[21] Burndown for Trello. Dostopno na `https://www.burndownfortrello.com/`.

[22] The Chaos report 1994. Dostopno na `http://www.standishgroup.com/sample_research_files/chaos_report_1994.pdf`.

[23] The Chaos report 2011. Dostopno na `http://www.versionone.com/assets/img/files/ChaosManifest_2011.pdf`.

[24] Chartbreeze. Dostopno na `http://www.chartbreeze.com/`.

[25] Disciplined Agile Delivery. Dostopno na http://www.disciplinedagiledelivery.com/.

[26] Enterprise scrum. Dostopno na http://www.enterprisescrum.com/.

[27] Kanboard issue tracker. Dostopno na https://github.com/fguillot/kanboard/issues.

[28] Large-Scale Scrum. Dostopno na http://less.works/.

[29] Lead Time vs. Cycle Time. Dostopno na http://www.isixsigma.com/community/blogs/lead-time-vs-cycle-time/.

[30] Open Kanban – An Open Source, Ultra Light, Agile & Lean Method. Dostopno na http://agilelion.com/agile-kanban-cafe/open-kanban.

[31] Principi v ozadju agilnega manifesta. Dostopno na http://agilemanifesto.org/iso/sl/principles.html.

[32] Scaled agile framework. Dostopno na http://scaledagileframework.com/.

[33] Scaled Professional Scrum. Dostopno na https://www.scrum.org/Resources/What-is-Scaled-Scrum.

[34] State of Agile Development Survey 2009. Dostopno na http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf.

[35] State of Agile Development Survey 2011. Dostopno na http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf.

[36] State of Agile Development Survey 2013. Dostopno na http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf.

[37] State of Agile Development Survey 2014. Dostopno na http://info.versionone.com/state-of-agile-development-survey-ninth.html.

[38] Toggl. Dostopno na https://www.toggl.com/.

[39] Trello Cumulative Flow. Dostopno na http://trello-cfd.azurewebsites.net/.

[40] Zapier. Dostopno na https://zapier.com/zapbook/github/trello/.

[41] Media multitaskers pay mental price, Stanford study shows, 2009. Dostopno na http://news.stanford.edu/news/2009/august24/multitask-research-study-082409.html.

[42] 2013 IT Project Success Rates Survey Results, 2014. Dostopno na http://www.ambysoft.com/surveys/success2013.html.

[43] How Kanban Works, 2014. Dostopno na http://www.infoq.com/articles/how-kanban-works.

[44] Manifest agilnega razvoja programske opreme, 2014. Dostopno na http://agilemanifesto.org/iso/sl/.

[45] Why You Need Dedicated Kanban Software, 2014. Dostopno na http://edu.leankanban.com/blog/why-you-need-dedicated-kanban-software.

[46] Agile Project Management Software, 2015. Dostopno na http://www.targetprocess.com/.

[47] Asana, 2015. Dostopno na https://asana.com.

[48] Atlassian Marketplace, 2015. Dostopno na https://marketplace.atlassian.com/.

[49] GDocs Kanban, 2015. Dostopno na https://docs.google.com/drawings/d/1XBSbKYrnRjIsn6Q1YjB5-FryujyONXuPicQJumPoiFI/edit.

[50] Google Drive, 2015. Dostopno na https://drive.google.com.

[51] Jira Agile, 2015. Dostopno na https://www.atlassian.com/software/jira/agile.

[52] Kanban Board Software, 2015. Dostopno na http://www.versionone.com/product/kanban-board/.

[53] Kanban for Software Development Teams, 2015. Dostopno na http://www.rallydev.com/editions/features/kanban-software-development-teams.

[54] Kanban online tools, 2015. Dostopno na https://groups.yahoo.com/neo/groups/kanbandev/conversations/topics/18830.

[55] Kanban Tool – Online Kanban Board for Business, 2015. Dostopno na http://kanbantool.com/.

[56] Kanbanery – Visual Project Management Tool the Kanban Way, 2015. Dostopno na https://kanbanery.com.

[57] Kanbanik – Free and open source kanban board, 2015. Dostopno na https://code.google.com/p/kanbanik/.

[58] Kanbanize – Kanban software for businesses, 2015. Dostopno na https://kanbanize.com/.

[59] Kanboard – Simple and open source visual task board, 2015. Dostopno na http://kanboard.net/.

[60] LeanKit – Lean for Business, 2015. Dostopno na http://leankit.com/.

[61] The Limited WIP Society Tools, 2015. Dostopno na http://limitedwipsociety.ning.com/page/tools.

[62] Lino – Sticky and Photo Sharing for you, 2015. Dostopno na http://en.linoit.com.

[63] Portable Kanban, 2015. Dostopno na http://dmitryivanov.net/personal-kanban-app/.

[64] Project Management Software AgileZen, 2015. Dostopno na http://www.agilezen.com/.

[65] SwiftKanban – Visualize Work, Manage Flow with Kanban, 2015. Dostopno na http://www.swiftkanban.com/.

[66] Tool Roundup & Comparison, 2015. Dostopno na https://groups.yahoo.com/neo/groups/kanbandev/conversations/topics/18537.

[67] Toyota Production System, 2015. Dostopno na http://en.wikipedia.org/wiki/Toyota_Production_System.

[68] Trello, 2015. Dostopno na https://trello.com.

[69] Using the Kanban Method, 2015. Dostopno na https://groups.yahoo.com/neo/groups/kanbandev/conversations/topics/18537.

[70] Matt Callanan. Ken Schwaber on Scrum, 2010. Dostopno na http://blog.mattcallanan.net/2010/02/ken-schwaber-on-scrum.html.

[71] CodePlex. Visual WIP, 2015. Dostopno na http://visualwip.codeplex.com.

[72] Corey Ladas. Kanban bootstrap, 2007. Dostopno na http://leansoftwareengineering.com/2007/10/27/kanban-bootstrap/.

[73] Dan Radigan. Project portfolio management with Jira Agile, 2013. Dostopno na http://blogs.atlassian.com/2013/10/project-portfolio-management-with-jira-agile-12/.

[74] Don Reinertsen. The Science of WIP Constraints, 2012. Dostopno na http://www.lean-kanban.eu/sessions/reinertsen/.

[75] Ken Schwaber, Jeff Sutherland. The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game, 2013. Dostopno na http://www.scrumguides.org/.

[76] Bennet Vallet. Kanban at Scale – A Siemens Success Story, 2014. Dostopno na http://www.infoq.com/articles/kanban-siemens-health-services.