Agilne metode u razvoju informacijskih sustava

Gojani, Dominik

VELEUČILIŠTE U RIJECI

Dominik Gojani

AGILNE METODE U RAZVOJU INFORMACIJSKIH SUSTAVA

(završni rad)

Rijeka, 2021.

VELEUČILIŠTE U RIJECI

Poslovni odjel

Stručni studij Informatika

AGILNE METODE U RAZVOJU INFORMACIJSKIH SUSTAVA

(završni rad)

MENTOR

dr.sc. Ida Panev, viši predavač

STUDENT

Dominik Gojani

MBS: 2422000018/14

Rijeka, rujan 2021.

VELEUČILIŠTE U RIJECI
Poslovni odjel

Rijeka, 22.03.2021.

ZADATAK
za završni rad

Pristupniku DOMINIK GOJANI **MBS: 2422000018/14**

Studentu preddiplomskog stručnog studija Informatika izdaje se zadatak za završni rad – tema završnog rada pod nazivom:

AGILNE METODE U RAZVOJU INFORMACIJSKIH SUSTAVA

Sadržaj zadatka:
Objasniti Agilnu, Scrum, Kanban i Scrumban metodologiju. Praktično prikazati i dokumentirati Scrum metodologiju na primjeru razvoja informacijskog sustava.

Preporuka:
Rad obraditi sukladno odredbama Pravilnika o završnom radu Veleučilišta u Rijeci.

Zadano: 22.03.2021. **Predati do: 15.09.2021.**

Mentor:
Dr. sc. Ida Panev, viši predavač

Pročelnik odjela:
Dr. sc. Anita Stilin, viši predavač

Zadatak primio dana: 22.03.2021.

Dominik Gojani

Dostavlja se:
- **mentoru**
- **pristupniku**

I Z J A V A

Izjavljujem da sam završni rad pod naslovom AGILNE METODE U RAZVOJU INFORMACIJSKIH SUSTAVA izradio samostalno pod nadzorom i uz stručnu pomoć mentora dr.sc. Ide Panev.

Dominik Gojani

(potpis studenta)

Sažetak rada

Tijekom proteklih godinu dana mogli smo posvjedočiti užurbanoj i prisilnoj digitalizaciji zbog određenih svjetskih događaja. Mnogi su bili prisiljeni prebaciti svoje poslovanje u digitalno okruženje što se kod nekih odvilo i u 100% kapaciteta poslovanja. Ono što je navelo korisnike na to nije se moglo izbjeći pa je glavni cilj bio što brže koristiti softverska rješenja. Ono što se navodi kao glavna prednost agilnih metoda je brza distribucija softvera uz što lakše prikupljanje zahtjeva korisnika. Agilne metode se uzimaju kao brzo rješenje razvoja sustava kad se na tržište mora plasirati bilo kakva verzija proizvoda neovisno o dovršenosti iste. Agilni način razvoja se diči time što omogućava baš to - razvoj i distribuciju softvera koji ne mora nužno biti dovršen, ali može biti funkcionalan do neke mjere koja je prihvatljiva za krajnjeg korisnika i/ili klijenta.

Ključne riječi: *Agile, Waterfall, Scrum, Kanban, Scrumban*

Sadržaj

1. Uvod

Razvoj informacijskih sustava je razvoj koji će nakon svog završetka dovesti do krajnjeg cilja, a to je završen proizvod kojeg će krajnji korisnik moći koristiti. U razvoju informacijskih sustava postoje faze razvoja i načini na koje se može upravljati razvojem informacijskog sustava. U ovom završnom radu bit će govora o agilnim metodama u razvoju informacijskih sustava. Prije svega, navest će se glavna načela agilnog razvoja softvera. Govorit će se o *Scrum*, *Kanban* i *Scrumban* metodologijama. Između ove tri navedene metodologije naglasak će biti na *Scrumu* iz razloga što autor ovog završnog rada koristi navedenu metodologiju na radnom mjestu. *Kanban* je nešto što autor također koristi, no u manjem obimu. Na primjeru izmišljenog informacijskog sustava moći će se vidjeti pristup *Scrum* metodologije, kako se ona primjenjuje u tom informacijskom sustavu te koji je postupak od prikupljanja zahtjeva klijenta do završnog proizvoda i prikupljanja eventualnih grešaka u sustavu.

2. Što je to agilni razvoj softvera?

Agilni razvoj softvera je način na koji se softver izrađuje, nevezano za tehnološki razvoj, od početka do kraja, to jest od prikupljanja zahtjeva klijenata do završnog proizvoda.

Agilni razvoj softvera nastao je 2000. godine, kada se grupa od 17 programera našla u Oregonu kako bi održali diskusiju vezanu uz pronalaženje što bržeg načina izbacivanja novog softvera na tržište [1]. Ono što se htjelo postići su dva glavna cilja:

- Korisnicima što prije izbaciti softver kako se ne bi događale situacije u kojima on predugo stoji u fazi razvoja.
- Dobivanje povratnih informacija od strane korisnika kako bi se pogreške i zahtjevi odradili u što stvarnijem i bržem vremenu.

Ono što je odlučeno, tj. ono oko čega su se složili su takozvana načela - proglas o metodi agilnog razvoja softvera. Ono što je doneseno je proglas agilnog razvoja softvera koji se nalazi niže[1]:

- Potrebno je tražiti bolje načine razvoja softvera razvijajući softver i pomažući drugima pri njegovom razvoju.

Takvim radom se uči više cijeniti:

- Ljude i njihove međusobne odnose nego procese i alate.
- Upotrebljiv softver nego iscrpnu dokumentaciju.
- Suradnju s naručiteljem nego pregovaranje oko ugovora.
- Reagiranje na promjenu nego ustrajanje na planu.

Drugim riječima, iako se cijene vrijednosti na desnoj strani, više se vjeruje u one na lijevoj [2].

[1] U popisu literature nalazi se poveznica [2] koja vodi na izvorni oblik proglasa.

2.1. Ljudi i njihovi međusobni odnosi naspram procesa i alata

U ovom radu će biti razmotreni svaki od navedenih dijelova Agilnog proglasa. Prvo načelo je navedeno u nazivu ovog poglavlja. Ono što se želi reći ovim načelom jest to da je prilikom promatranja ostalih načina izrade softverskih rješenja uočen uzorak ponašanja koji daje naglasak na procese i alate. Pod procese se misli na sve one obaveze koje osoba mora učiniti kako bi došla do krajnjeg cilja, a koji u većini slučajeva nisu fleksibilni. Navest ćemo jedan primjer: recimo da pri izvršavanju nekog zadatka osoba treba prvo prijaviti neki obrazac u kojem se izražava o svojim obvezama vezanim za taj zadatak. Agilna metodologija se ne oslanja na dio gdje je glavni interes na procesima koji produljuju vrijeme izrade softvera, nego je na ljudima i međusobnim odnosima. To znači da procesi bez kojih se normalno može funkcionirati postaju suvišni i izbacuju se iz svakodnevnog rada. Također, bitniji je međuljudski odnos, tj. odnos između programera. To bi značilo da je bitnije da se osobe unutar jednog tima slažu, nego da ispunjavaju nepotrebne obrasce kako bi zadovoljili već ustaljenu metodologiju i način rada.

Drugi dio koji je naveden su alati. Naglasak na ljude i međuljudske odnose naspram alata se može opisati s obzirom na alate koje programeri koriste. Npr. bitnije je da se ljudi u timu slažu i zajednički dolaze do rješenja poslovnih problema, nego da bespotrebno troše vrijeme na zadovoljavanje normi alata koje je tvrtka odredila kao obvezne za korištenje. Ovdje nisu uključeni alati koji služe za komunikaciju, a oko kojih su se usuglasili svi članovi tima, nego razvojni softver. Programeru se daje slobodno pravo korištenja željenog softvera, bez zadovoljavanja nekih predodređenih pravila.

2.2. Uporabljiv softver naspram iscrpne dokumentacije.

Jedan od glavnih načina razvijanja softvera u tzv. *Waterfall* metodologiji[2] je iscrpna dokumentacija. Neka vrsta dokumentacije mora postojati u razvoju softvera, no detaljne

[2] Sekvencijalni proces u kojem napredak teče kroz nekoliko faza (analiza zahtjeva, dizajn, razvoj, ispitivanje i provedba) od vrha do dna, analogno vodopadu [3].

specifikacije u obliku beskrajnih dokumenata su nešto što agilni procesi nikako ne potiču. Navedimo jedan primjer. Klijent želi novu značajku u već postojećem softveru. Agilna metodologija je jasna po tom pitanju; dokumentiraju se neke iteracije sastanaka, dogovaraju se poslovni procesi, ali se ne vodi dokumentacija tako da se zapisuje svaka izrečena riječ. Također, umjesto korištenja alata za pisanje dokumenata, koriste se alati za upravljanje zadatcima u koje se unose što detaljnije specifikacije, tj. provodi se centralizacija izvora informacija uz nove značajke softvera, zahvaljujući kojoj su informacije dostupne svim programerima.

Također, ponekad se ide toliko daleko da se dokumentira svaka malena promjena koja se provede unutar softvera, s posebnom suglasnošću vodećih osoba u timu. To predstavlja problem zbog toga što se programeri znaju osjećati kao osobe koje vode administrativne poslove koji ne doprinose konačnom cilju - funkcionalnom softveru.

Ono što je također naglašeno u iscrpnoj dokumentaciji je i povezanost s procesima navedenim u prvom načelu. Kako bi se efikasno vodila dokumentacija, moraju postojati procesi kojima bi se definirali trenutci u kojima se neki dio razvoja softvera dokumentira. Ne samo da je stavljen naglasak na dokumentiranju, nego i na definiranju načina na koji se nešto dokumentira. To označava da se čak prije samog početka upuštanja u pisanje dokumentacije mora potrošiti dragocjeno vrijeme na definiranje načina pisanja; dragocjeno vrijeme koje se može utrošiti u definiranje specifikacija ili samu izradu softvera.

2.3. Suradnja s naručiteljem naspram pregovaranja oko ugovora

Ovo načelo isprva možda zvuči nejasno jer bi se osoba mogla osjećati nesigurno ukoliko je veći naglasak na suradnji s klijentom od precizne definicije ugovora. Ono na što se zapravo želi staviti naglasak u ovom načelu jest fleksibilnost. Fleksibilnost je nešto što se očekuje s obje strane. Na primjer, razvojni tim zaposlen od strane nekog klijenta. U ugovoru su definirana pravila razvoja,

rok predaje, te okviri specifikacija. Tijekom procesa razvoja dogode se neke neočekivane promjene. Klijent je uvidio nedostatak određenih značajki. Zbog toga razvojni tim zaključuje da je za razvoj potrebno više vremena nego što je procijenjeno. Zbog fiksno određenih vremenskih razdoblja u ugovoru, zbog novonastale situacije dolazi do nepotrebnih tenzija i nesuglasice. Iz toga se jasno vidi da je navedeni primjer nepoželjan uzimajući u obzir načelo ovog potpoglavlja.

Ono na što agilni proces stavlja naglasak jest fleksibilnost kao što to već navodi prvi odlomak ovog potpoglavlja. Fleksibilnost bi značila jasno razumijevanje činjenice da se poslovni procesi i poslovna logika ponekad ne slažu sa programerskog logikom. Također, obje strane trebaju shvatiti da se prepreke mogu javiti na bilo kojem stupnju razvoja softvera te da to ne treba značiti ništa loše. Umjesto definiranja striktnog roka u obliku datuma, može se definirati okvirni rok dostave gotovog proizvoda. Umjesto definiranja cijelog proizvoda od početka do kraja, softver se može dostavljati u smislenim cjelinama.

2.4. Reagiranje na promjenu naspram ustrajanja na planu.

Za ovo načelo se može reći da objedinjuje sva ostala načela. Za lakše razumijevanje ovog načela i pravilno objašnjenje načina na koji objedinjuje ostala načela, navodi se sljedeći primjer: usred razvojnog procesa dogodi se nenadana promjena jer klijent zahtijeva vizualnu promjenu jednog dijela softvera. U *Waterfall* metodologiji to je jedna od najgorih situacija koja se može dogoditi. Prvo se revidiraju specifikacije kako bi se utvrdilo jesu li u skladu s ugovorom. Tada se ponovo uređuje dokumentacija jer ona više ne odgovara izvornim zahtjevima klijenta. U to situaciji najviše su oštećeni tim programera i menadžment koji moraju sve stavke odraditi prema unaprijed određenoj fiksnoj metodologiji. S druge strane, u agilnoj metodologiji potiče se na promjene za koje se smatra da su dobrodošle. Razlog tome je što se sve promjene mogu obaviti u relativno kratkom roku jer razvojni tim ne usporavaju stavke poput ugovora, dokumentacije i slično. Na

primjer, *Scrum* metodologija koja je agilna nema mogućnost mijenjanja specifikacija unutar zadanog okvira *Sprinta,* no to ne znači da se *Product Backlog* ne može mijenjati bez ograničenja.[3]

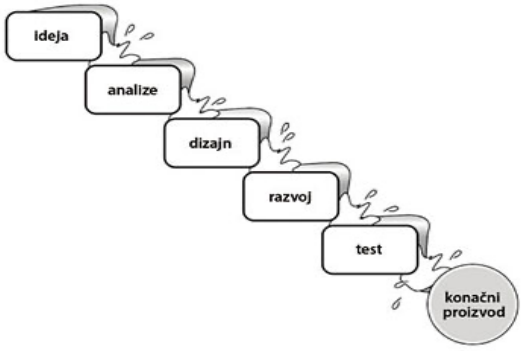
3. Usporedba *Waterfall* i *Agile* metodologije

Da bi se uvidjela razlika između *Waterfall* i *Agile* metodologije, navest će se nekoliko značajki obje metodologije. U prvom potpoglavlju će biti objašnjen *Waterfall* način rada, dok će u drugom potpoglavlju biti objašnjen Agilni način rada.

3.1. *Waterfall* (tradicionalni pristup)

Waterfall metodologija, koja se također naziva i tradicionalna metodologija zasniva se na doslovnom obliku slapa. Slika 1 pobliže objašnjava način na koji se izrađuje softver u toj metodologiji.

Slika 1: Tradicionalni pristup razvoju softvera



Izvor: http://www.infotrend.hr/clanak/2013/8/agilni-razvoj-softvera,77,1013.html (20.08.2021.)

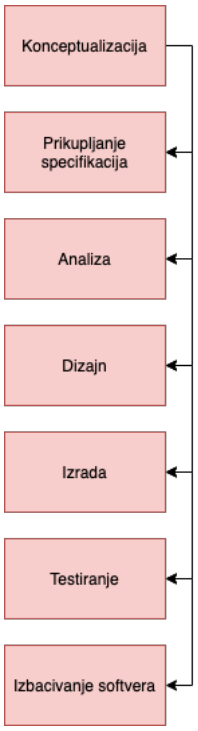
[3] Više o nazivljima koji se koriste u *Scrum* metodologiji može se pronaći u poglavlju *Scrum* metoda.

Na slici 1 može se vidjeti način na koji tradicionalni način razvoja softvera funkcionira. Sastoji se od nekoliko faza: ideja, analiza, dizajn, razvoj i test, nakon kojih se dobiva konačni proizvod. Ono po čemu je tradicionalni način specifičan je to što se svaka faza odvija u svoje vrijeme i nema preskakanja niti jedne od navedenih faza. On je također znakovit po tome što je procesno orijentiran, što znači da se procesi odvijaju sekvencijalno, bez mogućnosti fleksibilnosti.

3.2. Agilna metodologija

Agilna metodologija usporedno s tradicionalnom metodologijom zasniva na poticanju promjena unutar softvera tijekom razvoja i nakon izlaska softvera. Također, preskakanje ili micanje iz jedne faze u drugu je nešto u potpunosti normalno za agilnu metodologiju.

Slika 2: Agilna metodologija



Izvor: Autor

Na slici 2 jasno se može uočiti princip na kojem se zasniva agilna metodologija. Primjerice, razvojni tim može preskočiti fazu dizajna, te odmah krenuti u izradu softverskog rješenja. Ono na čemu je najveći naglasak u agilnom okruženju je mogućnost izvođenja svih faza razvoja softvera u isto vrijeme ili preskakanje nekih od faza. Navest će se jedan primjer: u okruženju gdje se razvija softver mobilne aplikacije, faza izrade može početi prije nego faza dizajna krene i/ili završi. Računalni programer može bez dizajna korisničkog sučelja pripremiti i krenuti sa izradom osnovnog softverskog kostura koji ne ovisi o dizajnu, no uvelike utječe na performanse neke aplikacije.

Postoji još jedan dio agilne metodologije koji se navodi kao prednost naspram tradicionalne metodologije, a to je suradnja svih članova razvojnog tima i sudjelovanje svih članova tima u svim fazama projekta. Na primjer, dizajner razvojnog tima može imati i programerske predispozicije te uvelike pridonijeti razvojnom procesu s direktnim povratnim informacijama koje se tiču same implementacije programerskog koda. Tim za testiranje razvojnih rješenja može doprinijeti svojim povratnim informacijama jer je posljednja linija između izlaska softvera i krajnjeg korisnika te je upoznat sa svim dijelovima softvera na način koji se u nekoj situaciji možda i ne tiče programera.

Sljedeće poglavlje će se detaljnije dotaknuti agilne metode *Scrum* koju autor ovog rada svakodnevno koristi.

4. *Scrum* metoda

Scrum metoda je agilna metoda. Unatoč tome, ima neka pravila, uloge, alate i sastanke kojih se *Scrum* timovi drže i koje cijene.

Gledajući usporedbu *Scruma* i agilne metodologije, *Scrum* je metoda, dok je agilna metodologija način razmišljanja. Ono što neki timovi misle jest to da mogu biti agilni tim preko noći. Samo zato što tim počne koristiti *Scrum* metodu to ne znači da su automatski agilni tim. Ono što *Scrum* nudi razvojnim timovima je mogućnost prilagodbe nastalim uvjetima i proizvodnim specifikacijama. U *Scrumu* naglasak je na promjenama specifikacija, prilagodbi na te promjene,

davanju prednosti značajkama naspram ostalih već poznatih značajki te kratkim razmacima izlazaka softvera, ali u manjim inkrementima.

4.1. Artefakti u *Scrum metodi*

Scrum artefakti se mogu podijeliti u tri cjeline i označavaju glavne dijelove *Scrum* metode:

1. *Product Backlog* - ovo je glavna lista svih zadataka i svih predmeta koji se moraju izvršiti. Ova listu održavaju menadžeri proizvoda ili vlasnici proizvoda.[4] Ono što se u njemu nalazi su specifikacije, potpune ili nepotpune, želje vlasnika proizvoda i korisničke priče koje pomažu razumijevanju načina na koji se mora izraditi softversko rješenje.
2. *Sprint Backlog* - ovo je glavna lista svih korisničkih priča, zadataka i popravaka grešaka u sustavu koji se moraju napraviti u određenom vremenskom razdoblju. Razvojni tim održava sastanke sa vlasnikom proizvoda u kojima se govori o onom što se treba ili želi učiniti u sljedećem vremenskom razdoblju.
3. Inkrement - označava iskoristiv proizvod ili značajku koja je spremna za izlazak nakon predodređenog vremenskog perioda.

5. Uloge i značajke *Scrum* metode

Sad kad su jasno definirani artefakti *Scrum* metode, ono što će ovo poglavlje objasniti su uloge u *Scrum* metodi. *Scrum* metoda ima jasno definirane uloge koje pomažu u što bržem i efikasnijem načinu dovođenja običnog zahtjeva klijenta do funkcionalnog proizvoda ili značajke.

[4] Više o ulogama u *Scrumu* u poglavlju Uloge u *Scrum* metodi.

5.1. Uloga *Scrum Mastera* u odnosu na razvojni tim

Scrum Master je uloga koja je u sredini razvojnog procesa. To je uloga koja se nalazi između *Product Ownera* (u nastavku teksta: Vlasnik Proizvoda) i razvojnog tima. Ona je poveznica između pružatelja specifikacija značajki i onih osoba koje će izrađivati značajke.

Scrum Master ima nekoliko odgovornosti. Ono što je bitno za primijetiti je da, prilikom prelaska na agilnu metodu u razvoju, *Scrum Master* je osoba na koju se razvojni tim može pouzdati za pravilnu implementaciju *Scrum* agilne metode. Ovo su neke od glavnih odgovornosti *Scrum Mastera* [4]:

1. Dnevni sastanci - dnevni sastanci su sastanci u *Scrum* metodi na kojima se raspravlja o tekućim zadacima koje razvojni tim obavlja te o poteškoćama na koje nailaze pri razvijanju softvera uz naglasak na rješavanje tih poteškoća. Uvriježena praksa su tri pitanja koja se postavljaju na tim sastancima: što je rađeno jučer, što će se raditi danas i postoje li ikakve poteškoće u razvoju. Valja napomenuti da se gore navedena pitanja više ne smatraju obvezna u *Scrum* metodologiji. Cilj toga je bilo da se *Scrum* ne ograničava na propisana pravila te da se razvojni tim može opustiti i bez obaveze odgovaranja na navedena pitanja ukoliko se unutar jednog dana nije ništa promijenilo, ali da opet imaju želju podijeliti svoj napredak sa ostalim članovima tima.
2. Iteracije i planiranje *Sprintova* - iteracije se obavljaju s Vlasnicima Proizvoda gdje se odlučuje na kojim zadacima će se raditi u sljedećem predodređenom vremenskom razdoblju[5]. Ono za što je zadužen *Scrum Master* je zaštita razvojnog tima od obvezivanja na količinu zadataka koje nisu u stanju izvršiti.
3. Pregled *Sprinta* i retrospektiva - nakon što je *Sprint* obavljen, *Scrum Master* uz Vlasnike Proizvoda sagledava *Sprint* i njegovu učinkovitost te predlaže i dogovara poboljšanja *Sprinta.*

[5] Vremenska razdoblja u kontekstu *Scruma* se nazivaju *Sprintovi.* Više o njima u poglavlju *Sprint.*

4. Održavanja *Scrum* ploče - *Scrum* ploča je virtualna ploča koja sadržava takozvane kartice i zadatke te korisničke priče o značajkama koje razvojni tim izrađuje. *Scrum Master* je zadužen za ploču, njene ažurne informacije te da se softver za održavanje *Scrum* metode redovno održava[6].
5. Jedan na jedan sastanci - ukoliko članovi iz razvojnog tima nisu s nečime zadovoljni, *Scrum Master* uloga obuhvaća uklanjanje ili pokušaj uklanjanja svih elemenata koji blokiraju programera da adekvatno funkcionira u krugu *Scruma.*
6. Izvještavanje - Vlasnici Proizvoda žele imati uvid u efikasnost razvojnog tima. *Scrum Master* ima mogućnost, obvezu i alate za postizanje redovnog izvještavanja u kojem god obliku odgovara Vlasnicima Proizvoda.

5.2. *Sprint*

Sprint kao pojam označava vremensko razdoblje u kojem se razvojni tim obvezuje da će obaviti zadani i dogovoreni posao. *Sprint se* u smislu cjeline sastoji od nekih osnovnih elemenata. Vlasnik Proizvoda zajedno sa razvojnim timom razgovara o sljedećim koracima koji se moraju poduzeti u sljedećem *Sprintu* kako bi se neka značajka dostavila i bila spremna za izlazak. Da bi svaki član bio ažurno obaviješten, održavaju se dnevni sastanci te se na njima utvrđuju elementi koji koče daljnji razvoj softvera. Nakon određenog vremenskog razdoblja[7] *Sprinta* razvojni tim ima priliku pokazati što je uspio izraditi u prošlom *Sprintu.* Na posljetku dolazi do sastanka na kojem se razvojni tim osvrće na prošli *Sprint* uz razuman razgovor o poljima razvoja koja se mogu poboljšati.

[6] Održavanje tehničkih značajki poput boja, pravila, automatizacije, korisnika softvera i ostalog.

[7] Sprintovi su većinom u trajanju od jedan do dva tjedna.

5.3. Korisničke priče

Korisničke priče su pojam u agilnom razvoju softvera koji označava odnos između krajnjeg korisnika i korisničkog sučelja [5]. Glavna značajka je opis neke značajke te kako će ona koristiti krajnjem korisniku. U agilnom razvoju softvera naglasak je na ljudima, a korisnička priča stavlja krajnjeg korisnika na prvo mjesto. Korisničke priče nemaju tehničke elemente nego pružaju razlog i odgovor na pitanje zašto se neki dio softvera ili značajke mora izraditi, izmijeniti ili popraviti.

Korisničke priče označavaju najmanju jedinicu obujma posla u agilnom razvoju softvera. Korisničke priče označavaju krajnji cilj te ne označavaju značajku, gledajući sa strane krajnjeg korisnika softvera. Korisničke priče se sastoje od tri glavna elementa koji se mogu vidjeti u sljedećem primjeru:

Kao **kupac**, želim moći **povećati količinu proizvoda** direktno iz košarice, kako bih **što brže došao do plaćanja**. Iz ovog primjera, lako se može zapaziti koja su to tri elementa:

1. Tko - ovaj dio označava tko je akter ili osoba koja zahtijeva neki krajnji cilj. Ona može biti krajnji korisnik, no može biti član razvojnog tima jer, na primjer, jedan od članova tima želi unaprijediti kôd u nekoj značajki radi poboljšanja performansi.
2. Što - ovaj dio označava ono što akter želi postići kao krajnji cilj. Ovdje se ne opisuje dio korisničkog sučelja poput boja, veličine teksta i sl., nego samo ono što se želi postići na kraju radnje.
3. Zašto - svaki zahtjev mora proizaći iz nekog razloga. U ovom dijelu se navodi razlog zbog kojeg postoji korisnička priča. Jednim dijelom se odgovaranjem na pitanje "zašto" odgovara i na krajnji cilj, jer su drugi i treći dio korisničke priče usko povezani.

Korisničke priče u suštini opisuju ono što krajnji korisnik želi postići u obliku koji je jasno razumljiv razvojnom timu. Unatoč tome što korisnička priča izgleda vrlo kratko, to ne znači da se u zadacima ne smije navesti adekvatan opis korisničke priče, jer korisnička priča sama po sebi ne znači da će sadržavati sve moguće odgovore na moguća pitanja razvojnog tima.

5.3.1. Bodovanje korisničkih priča

Bodovanje korisničkih priča je metrika koja se koristi u agilnim metodama kako bi se procijenilo koliko je vremena potrebno da bi se dovršila neka korisnička priča. Procjena vremena se obavlja u razvojnom timu i preporučljivo je da se ona odvija sa članovima koji će razvijati korisničke priče. Tradicionalna metoda koristi dane, tjedne ili mjesece kako bi procijenila koliko vremena je potrebno za razvoj. Razlozi zašto agilne metode koriste bodovanje korisničkih priča su sljedeći [17]:

1. datumi nisu u mogućnosti prikazati rad koji nije nužno povezan za razvojem softvera poput elektroničke pošte i sastanaka
2. datumi imaju emocionalnu povezanost između značajke i programera, dok bodovanje korisničke priče miče tu povezanost
3. svaki razvojni tim će na drukčiji način bodovati trajanje svog rada što znači da će njihova brzina razvoja biti drukčija te u konačnici onemogućava korištenje bodovanja korisničke priče na način da bi se na krivi način pokazala efikasnost razvojnog tima
4. nakon što razvojni tim procijeni određeni broj korisničkih priča, naredne priče će biti sve lakše procijeniti zbog uvježbanosti
5. bodovanje korisničkih priča nagrađuje članove tima na temelju rješavanja problema po težini, ne po vremenu koje je utrošeno da se korisnička priča dovrši

5.4. *Backlog* proizvoda

Backlog proizvoda označava popis koji sadrži sav dio posla, zadatke, korisničke priče i greške u sustavu koje nisu trenutno u statusu razvoja ili održavanja. Uobičajeno je da se sve stavke slažu po pitanju prioriteta i hitnosti.

Backlog se sastoji od različitih elemenata. Sastoji se od korisničkih priča, greški u sustavu, zadataka, želja, značajki, istraživanja i jedne bitne stvari vezane za lakši menadžment razvoja softvera, a to su takozvani *Epici.*

Epic je cjelina softvera za koju je potreban razvoj dulji od tri ili četiri dana. *Epic* je zapravo ono što obuhvaća sve korisničke priče, zadatke i podzadatke [6]. U nastavku se navodi primjer jednog *Epica* i njegove povezanosti sa ostalim gore navedenim elementima:

Epic nazvan Košarica se sastoji od više korisničkih priča. Jedna od tih priča je navedena u prošlom poglavlju Korisničke priče, no razvojni tim je shvatio da je *Epic* previše generaliziran te su im potrebni daljnji detalji. Ono što proizlazi iz *Epica* je to da se on mora "raspisati", to jest mora se razlomiti u manje dijelove koji će razvojnom timu biti jasniji. Košarica sama po sebi ne znači mnogo, ali objedinjuje sve korisničke priče u jednu cjelinu na koju se razvojni tim može obvezati.

Važno je spomenuti da osoba koja je zadužena za održavanje *Backloga je* Vlasnik Proizvoda. Njegova zadaća je biti jasan u definiranju korisničkih priča, raspoznati situaciju u kojoj je neka korisnička priča ili zadatak previše kompleksan i razlomiti je u manje dijelove, biti jasan i dosljedan u opisivanju korisničkih priča te prepoznati loše definiranu priču te je na vrijeme urediti na programeru razumljiv način.

6. *Kanban* kao agilna metoda razvoja informacijskih sustava

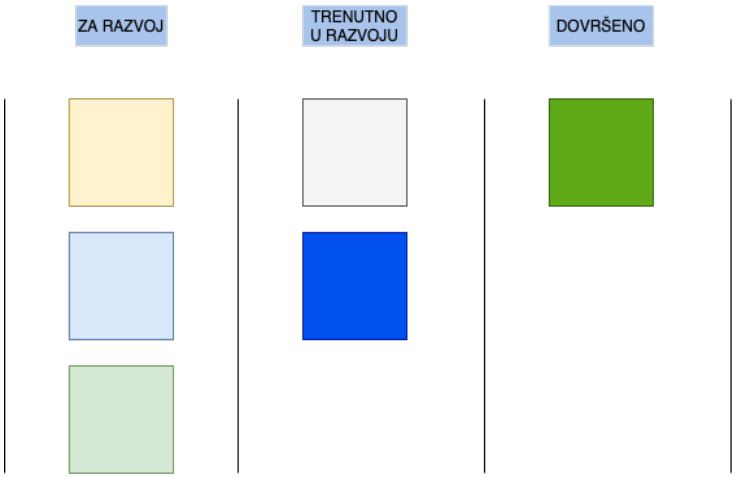
Kanban je jedna od metoda agilne metodologije koja se u nekoliko značajki razlikuje od *Scruma.* Jedna od glavnih različitosti je nepostojanje *Sprintova.* U *Kanban* metodi su zadržani svi elementi agilnog razvoja softvera, ali se taj razvoj ne ograničava na vremenska razdoblja, nego *Kanban* koristi nešto što se zove kontinuiran razvoj.

Kontinuiran razvoj označava razvoj softvera u kojem se bez prestanka kontinuirano izrađuje softverska rješenja. Razvojni tim nema vremensko razdoblje u kojem se obvezuje da će obaviti zadani dio posla, nego se razvoj zasniva na kontinuiranoj diskusiji i prioritetima. Ovdje je važno naglasiti da se Vlasnik Proizvoda na bilo kojem stupnju razvoja softvera može odlučiti da je neka značajka postala nebitna i u tom trenutku zaustaviti razvoj iste, nakon čega razvojni tim može odmah krenuti s nečim što je u zadanom trenutku bitnije. Također, *Kanban* omogućava pauziranje razvoja značajke ukoliko joj se tijekom razvoja promijenio prioritet.

6.1. Kanban ploča

Kanban ploča je glavni fokus *Kanban* razvojnog tima. Putem nje Vlasnik Proizvoda zna status svih korisničkih priča koje su trenutno u razvoju, status značajki ili popravaka.

Slika 3: Kanban ploča



Izvor: Autor

Na Slici 3 može se vidjeti struktura *Kanban* ploče. *Kanban* ploče se mogu razlomiti u pet komponenti [7]:

1. Vizualni signali - vizualni signali u *Kanban* ploči su zapravo kartice. Kartice označavaju zadatke, korisničke priče ili greške u sustavu. Kartice se koriste kako bi Vlasnik Proizvoda i svi članovi tima bili upućeni u trenutni status svih korisničkih priča, grešaka u sustavu ili zadataka koji su u razvoju.
2. Stupci - na Slici 3 jasno se vide tri stupca: za razvoj, trenutno u razvoju i dovršeno. One pomažu kako bi se znalo na kojem stupnju protoka rada se nalazi neka korisnička priča.
3. Ograničavanje trenutnog radnog opterećenja - unutar jednog stupca može se, prema dogovoru između Vlasnika Proizvoda i razvojnog tima ograničiti koliko korisničkih priča može biti na kojem stupnju razvoja. Takav način praćenja omogućuje razvojnom timu da pokuša dovršiti neku karticu i prebaciti ju u sljedeći stadij prije nego dođe do ograničenja stupca.

4. *Backlog* - u *Kanbanu* ima istu ulogu kao i u *Scrumu*: popis koji sadrži sav dio posla, zadatke, korisničke priče i greške u sustavu koje nisu trenutno u statusu razvoja ili održavanja.
5. Dovršetak - dovršetak označava kraj razvojnog ciklusa *Kanban* kartice. Većinom se označava kao zadnji stupac u *Kanban* ploči. Cilj razvojnog tima sa strane menadžmenta je dovesti korisničke priče iz *Backloga* do posljednjeg stupca *Kanban* ploče.

7. Usporedba *Kanbana* i *Scruma*

Gledajući *Kanban* i *Scrum* obje metode su vrlo slične uz neke razlike koje su jače izražene naspram ostalih [14]. *Kanban* se zasniva na ograničenju trenutnog radnog opterećenja, održavanju, korištenju kartica i *Kanban* ploče. *Scrum* se zasniva na *Sprintovima* i koristi dnevne sastanke za održavanje koraka sa svim članovima razvojnog tima. Obje metode koriste *Backlog* uz iznimku *Kanbana* koji ne nalaže nužno postojanje *Backloga* izvan prvog stupca *Kanban* ploče.

Kako bi se jasnije vidjele glavne značajke, ali i razlike između te dvije metode, slijedi tablica 1 koja omogućava jasno vizualno razlikovanje:

Tablica 1: Vizualni prikaz razlika Scruma i Kanbana

	Scrum	Kanban
Način rada	*Sprintevi* (jedan ili dva tjedna)	Neprekidni tijek rada
Način izlaska softvera	Na kraju svakog *Sprinta*	Stalni neograničeni izlasci
Uloge	Vlasnik Proizvoda, *Scrum Master*, razvojni tim	Uloge nisu unaprijed definirane

Metrike	Brzina razvoja (eng. *Velocity*[8])	ograničenje radnog opterećenja, količina vremena provedena u razvoju kartice naspram zadržavanja kartice u stupcima
Promjene u razvoju softvera	Promjene nisu dobrodošle unutar *Sprinta*	Promjene su uvijek dobrodošle

Izvor: https://www.atlassian.com/agile/kanban/kanban-vs-scrum (30.08.2021.)

8. *Scrumban* kao agilna metoda razvoja informacijskih sustava

Scrumban je agilna metoda koja je hibrid *Scrum* i *Kanban* metode. U *Scrumbanu* postoji pet pravila koja pomažu u uspješnoj implementaciji *Scrumban* metode unutar razvojnog tima [8]:

1. Izrada *Scrumban* ploče - *Scrumban* ploča je vrlo slična *Kanban* ploči. U nju se može dodati onoliko stupaca koliko razvojni tim u suradnji s Vlasnikom Proizvoda smatra da je potrebno. Ono na što se mora paziti jest da se ne koristi prevelika količina stupaca koja bi mogla dovesti do nepreglednosti i manjka snalaženja.
2. Određivanje trenutnog radnog opterećenja - kao i u *Kanban* metodi, potrebno je odrediti koliko kartica može ući u jedan *Sprint.*

[8] Količina rada kojeg razvojni tim može obaviti u zadanom *Sprintu.*

3. Određivanje prioriteta na ploči - jedna od glavnih razlika između *Scrum* i *Kanban* metoda jest to što prioritete i dodjeljivanje zadataka u *Scrum* metodi obavlja *Scrum Master*, dok u *Scrumban* metodi dodjeljivanje zadataka vrši razvojni tim, a *Scrum Master* određuje prioritet zadataka.
4. Nepostojanje procjenjivanja - u *Scrum* metodi se zadatci mogu procjenjivati na vremenskoj bazi ili na neki drugi način koji najviše odgovara razvojnom timu. *Scrumban* uzima neprekidni tijek rada iz *Kanbana* i zasniva se na određivanju prioriteta svih zadataka ili kartica na ploči.

8.1. Kada koristiti *Scrumban?*

Scrumban je metoda koja nije prikladna za sve moguće situacije u kojima se razvija neka vrsta softvera. Njegovo korištenje se preporuča u sljedećim scenarijima [8]:

1. Održavanje postojećih projekata - situacije u kojima razvojni tim održava već napravljen projekt u većini, ali se nadodaju preinake ili značajke od malo značaja.
2. Tim koji ima probleme sa *Scrumom* - postoje situacije u kojima se razvojni tim ne osjeća ugodno u korištenju *Scrum* metode zbog njezinih ograničenja ili zbog toga što klijent nema dovoljno resursa za potpunu adaptaciju *Scrum* metode.
3. Fleksibilnost naspram *Scrum* metode - naspram *Scruma, Scrumban* omogućava razvojnom timu da sam odlučuje na koji način alocirati resurse. Također, premještaj na nove projekte je puno lakši za članove *Scrumban* tima naspram *Scrum* tima.

9. Upravljanje razvojem informacijskog sustava metodom *Scrum*

Glavni cilj ovog poglavlja je predočiti način na koji *Scrum* metoda funkcionira unutar razvojnog tima. Bit će prikazana interakcija između Vlasnika Proizvoda, *Scrum Mastera* i

razvojnog tima. Alat koji će biti prikazan je Jira[9]. Ono što će se prikazati je izmišljen sustav koji će za potrebe ovog završnog rada biti pojednostavljen. Vlasnik proizvoda će zahtijevati implementaciju jednostavne i statične web stranice koja će sadržavati nekoliko elemenata. Projekt će se sastojati od glavne početne stranice i stranice koja govori nešto više o tvrtki čiji je naziv Tvrtka1. Protok informacija će teći od Vlasnika Proizvoda do *Scrum Mastera* i razvojnog tima. Ono što će biti već definirano su specifikacije i zahtjevi koje je Vlasnik Proizvoda dobio od klijenta pa za razvojni tim već ima spremne korisničke priče. Količina korisničkih priča za potrebe ovog rada neće prelaziti deset priča jer je naglasak nije na pričama i detaljnosti specifikacija, nego uvriježenom procesu to jest načinu upravljanja razvojem informacijskog sustava.

Projekt u alatu Jira biti će nazvan Tvrtka1, a označavat će web stranicu tvrtke Tvrtka1. Imenovanje projekta u Jiri nema nikakvih pravila te se može koristiti bilo koji naziv, a za potrebe ovog rada, budući da se radi o fiktivnom projektu, naziv će biti kao što je gore navedeno.

10. Korisničko sučelje alata Jira

Glavni cilj ovog poglavlja je prikazati korisničko sučelje alata Jira. Nakon što će biti prikazani svi dijelovi korisničkog sučelja, slijedit će poglavlje u kojem će se simulirati tok zadataka kroz opisano sučelje.

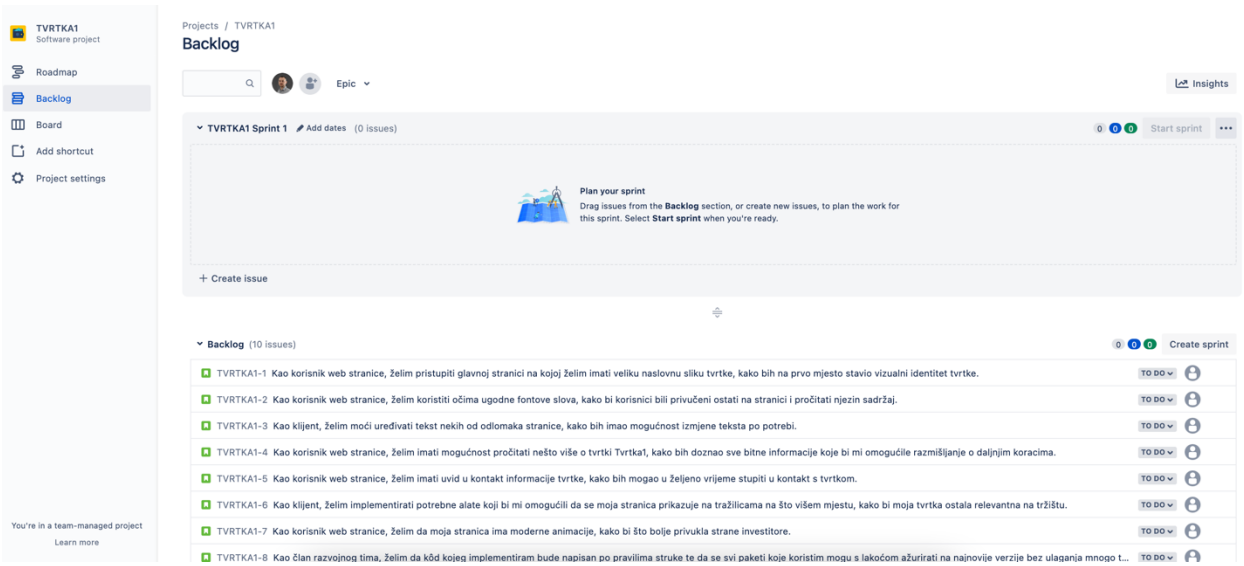
10.1. Osnovno sučelje *Jire*

Jira je alat koji je u svojoj izvornoj verziji bio namijenjen praćenju grešaka u sustavima. Trenutni opseg značajki koje Jira nudi su upravljanje projektima, upravljanje razvojem informacijskih sustava i praćenje grešaka u sustavima.

[9] JIRA™ je alat kojeg autor ovog teksta koristi pri upravljanju razvojem informacijskih sustava [15].

Jira kao alat nudi mogućnost korištenja dvije vrste projekta: projekt kojim upravlja tim ljudi (u nastavku teksta: Jira projekt) ili projekt kojim upravlja organizacija. U ovom radu će se koristiti projekt kojim upravlja tim ljudi jer mnoge postavke unutar takvog projekta su predodređene i nije ih potrebno mijenjati. Također, u ovom projektu naglasak je da se razvojni tim sastoji od tri programera, stoga projekt kojim upravlja organizacija nije potreban.

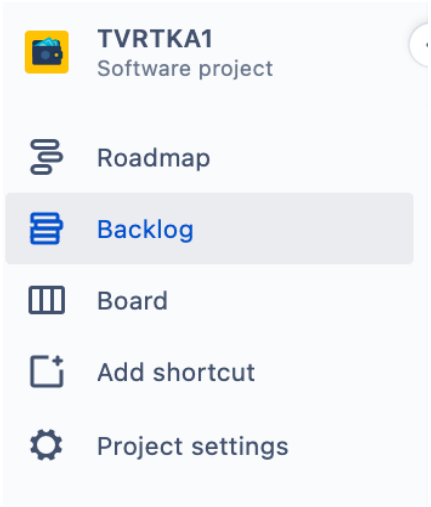
Slika 4: Osnovno sučelje *Jire*



Izvor: Autor

Na slici 4 može se uočiti nekoliko elemenata Jira projekta. Svaki od elemenata će biti objašnjen na što jasniji način kako bi se lakše mogao predočiti tijek zadataka i zahtjeva klijenta.

Slika 5: Glavne postavke Jira projekta



Izvor: Autor

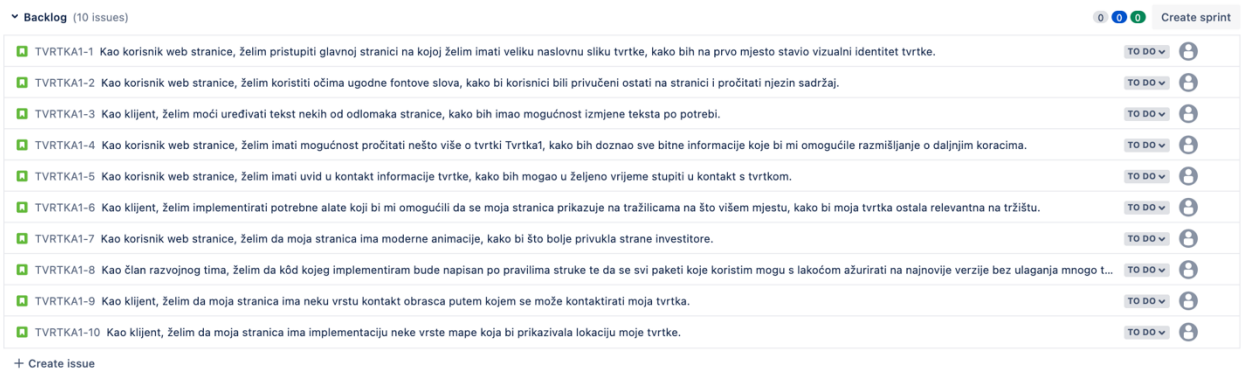
Slika 5 prikazuje glavne postavke Jira projekta. One se sastoje od *Roadmapa*[10]*, Backloga,* ploče (eng. Board) i Postavki projekta (eng. Project settings). U Postavkama projekta se mogu namještati svojstva poput pristupa projektu, količine podataka koji se mogu prikupljati unutar korisničkih priča, automatizacije poput pravila za tok zadataka i ostalo.

10.2. Sučelje *Backloga*

Nakon što je Vlasnik Proizvoda unio korisničke priče u sučelje alata Jira, pogled koji ima može se vidjeti na slici 6.

[10] *Roadmap* je osnovica za postavljanje *Epica.* Putem *Roadmapa* se s lakoćom može pratiti tijek svih *Epica* i korisničkih priča unutar *Epica,* uzimajući u obzir da *Epic* ne mora završiti u jednom *Sprintu.*

Slika 6: Sučelje *Backloga*

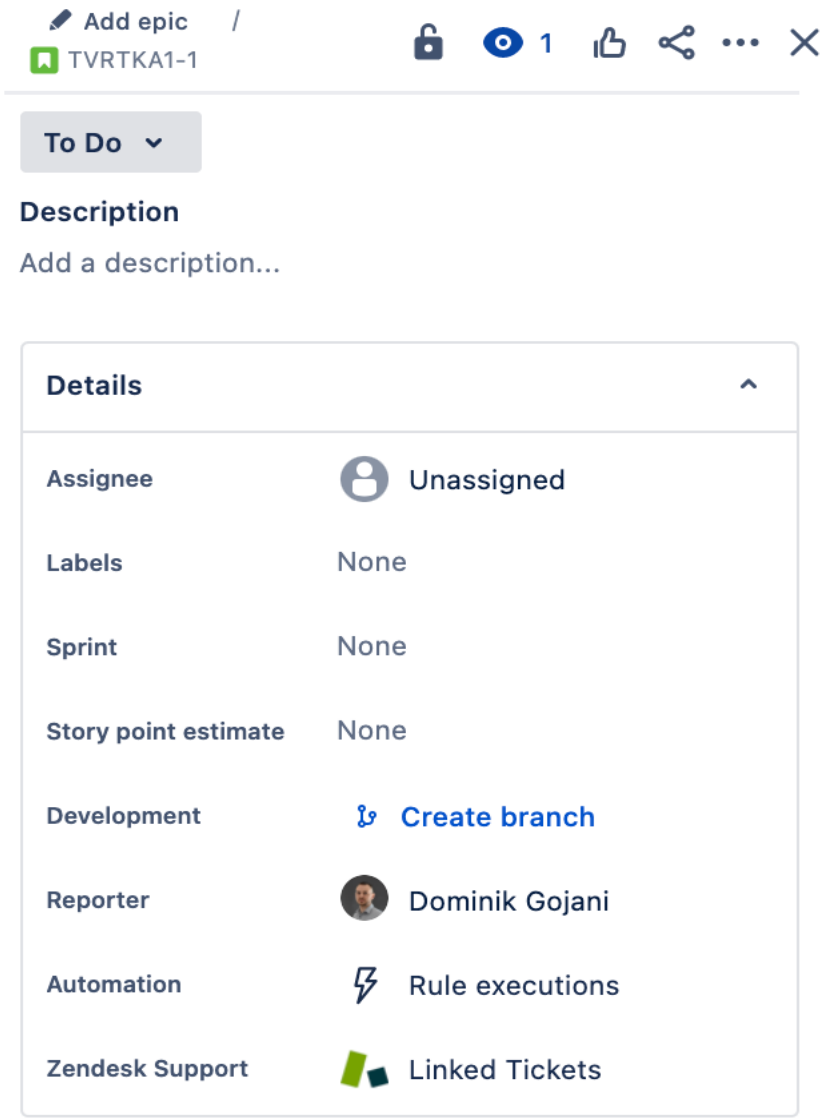


Izvor: Autor

Gledajući sliku 6 uočava se da Vlasnik Proizvoda može u ovom sučelju namještati stadij u kojem se korisnička priča trenutno nalazi i vidjeti kome je dodijeljena određena korisnička priča. Također, svaka od priča je automatski numerirana prikladnim identifikatorom koji se sastoji od imena projekta i broja koji se povećava pri svakom kreiranju priče. Vlasnik Proizvoda ne mora u naslov korisničke priče unijeti cijelu korisničku priču, nego samo razvojnom timu razumljiv kratki naslov, dok opis korisničke priče može biti one duljine koja je potrebna da korisnička priča bude jasna.

10.3. Sučelje korisničke priče

Slika 7: Sučelje korisničke priče



Izvor: Autor

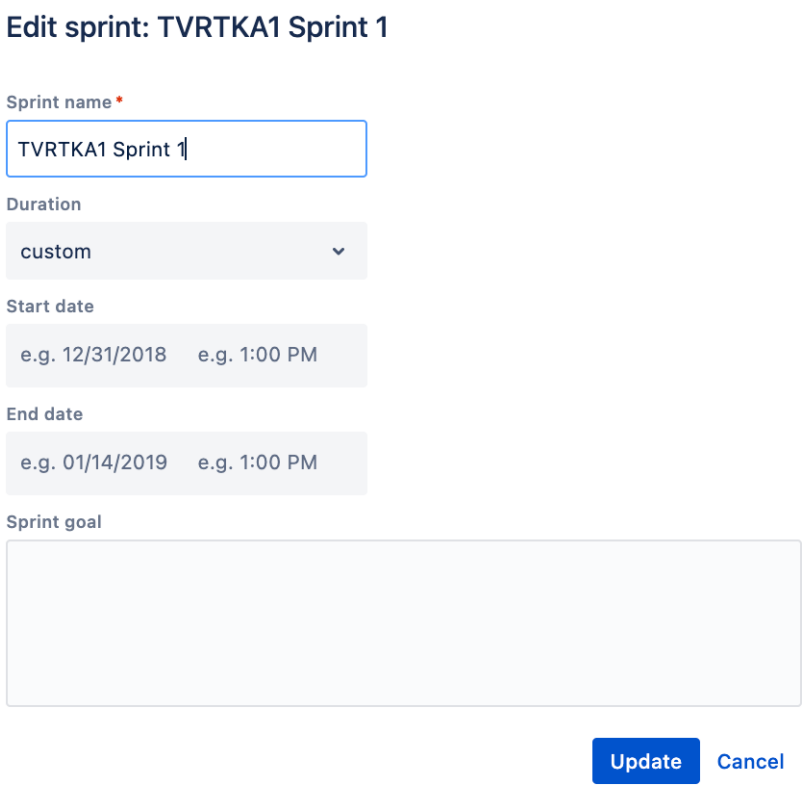
Uzimajući u obzir da se osnovne postavke ovog Jira projekta nisu mijenjale, na Slici 7 se može primijetiti nekoliko postavki:

1. Dodijeljena osoba (eng. Assignee) - osoba kojoj je dodijeljena korisnička priča.
2. Oznake (eng. Labels) - korisnička priča u sebi može sadržavati jednu ili više oznaka. Oznake služe za lakšu navigaciju po korisničkim pričama u sučelju te za bolji način filtriranja korisničkih priča. Oznake mogu označavati projekt[11] na koji se referencira priča, na primjer Nova Web Stranica ili Baza Podataka.
3. *Sprint* - korisnička priča se stavlja u *Sprint* kad se na to odluči Vlasnik Proizvoda.
4. Procjena vremena u bodovima korisničkih priča (eng. Story point estimate) - razvojni tim može i ne mora koristiti bodove korisničkih priča kako bi procijenio koliko vremena je potrebno da se jedna korisnička priča dovrši. Zbog jednostavnosti ovog Jira projekta, bodovanje korisničkih priča neće biti korišteno.
5. Izvjestitelj (eng. Reporter) - osoba koja je kreirala korisničku priču ili zadatak.
6. Automatizacija (eng. Automation) - koristeći ovu značajku, može se, na primjer, postići da se korisnička priča dodijeli određenoj osobi nakon što se pomakne u određeni stupanj razvoja.
7. Opis (eng. Description) - ovo polje se koristi za unos opširnijih objašnjenja korisničke priče. Razlozi, načini implementacije, želje, detaljne specifikacije; sve to se može unijeti u opis korisničke priče kako bi razvojnom timu bilo što jasnije što se od njih zahtijeva.
8. Komentari (eng. Comments) - služe za primarnu komunikaciju između svih dijelova *Scrum* metode. Vlasnik Proizvoda može odgovarati na pitanja razvojnog tima, predlagati nova rješenja i davati svoje mišljenje o trenutnoj implementaciji korisničke priče dok se ona nalazi u bilo kojem stupcu.

[11] U ovom kontekstu projekt označava doslovni repozitorij i projekt kojeg taj repozitorij sadrži.

10.4. Sučelje postavki *Sprinta*

Slika 8: Sučelje postavki *Sprinta*



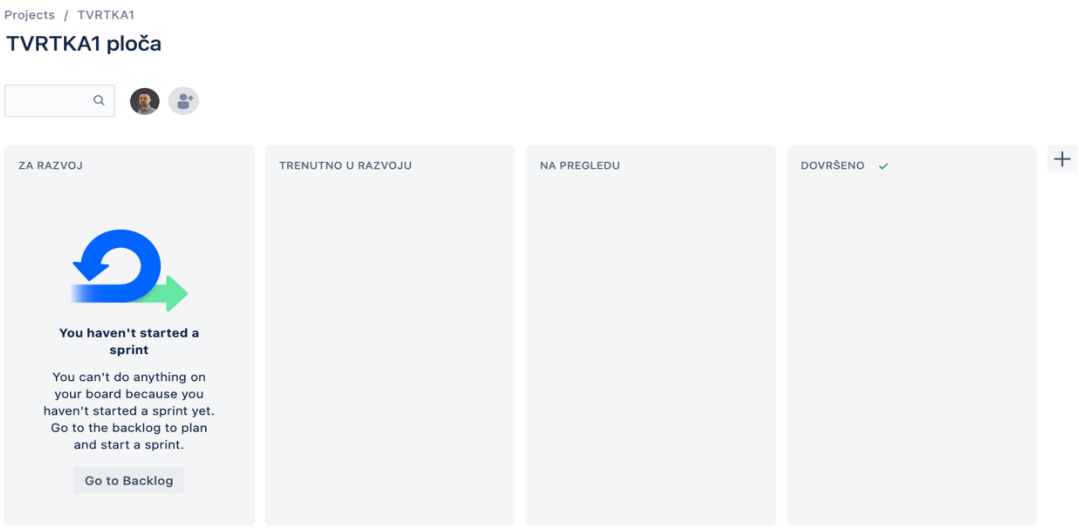
Izvor: Autor

Na slici 8 se mogu vidjeti osnovne postavke *Sprinta:*

1. Naziv *Sprinta* (eng. Sprint name) - naziv *Sprinta* u ovom Jira projektu će biti Sprint 1. Jira automatski namješta ime svakog novog *Sprinta* kao uvećan broj, to jest sljedeći *Sprint* bi bio nazvan Sprint 2.
2. Trajanje *Sprinta* - uvriježeno trajanje *Sprintova* je jedan ili dva tjedna.
3. Početak i završetak *Sprinta* - označava datum početka i datum završetka *Sprinta*.
4. Cilj *Sprinta* - koristi se za opisivanje onog što se želi postići u trenutnom *Sprintu*. U ovom Jira projektu će cilj *Sprinta* biti razvijanje i izlazak web stranice klijenta Tvrtka1.

10.5. Sučelje *Scrum* ploče

Slika 9: Sučelje *Scrum* ploče



Izvor: Autor

Na slici 9 mogu se vidjeti stupci od kojih se sastoji *Scrum* ploča. Za potrebe ovog jednostavnog projekta, ploča se sastoji od četiri stupca:

1. Za razvoj - sve korisničke priče za koje je odlučeno da će ući u *Sprint* biti će vidljive u ovom stupcu. On je početni stupac iz kojeg se kreće u razvoj te se iz njega vuku sve korisničke priče u stupce s njegove desne strane.
2. Trenutno u razvoju - sve korisničke priče koje su trenutno u razvoju se nalaze u ovom stupcu.
3. Na pregledu - nakon što programer završi s radom na korisničkoj priči, prebacuje ju u ovaj stupac. Ovaj stupac služi da bi Vlasnik Proizvoda znao kad je korisnička priča spremna za njegov pregled. U većini slučajeva Vlasnik Proizvoda ima pristup nekoj vrsti testnog sučelja u kojem može vidjeti način funkcioniranja neke značajke.
4. Dovršeno - nakon što korisnička priča, to jest značajka, bude puštena van, ona se premješta u ovaj stupac.

10.6. Korak 1: Vlasnik Proizvoda stvara korisničke priče

Gledajući s udaljenog stajališta, alat Jira se sastoji od *Backloga, Scrum* ploče i korisničkih priča. Vlasnik proizvoda koristi *Backlog* na ispravan način te umeće sve korisničke priče vezane za razvoj web stranice u *Backlog* proizvoda. Priče se sastoje od zahtjeva koje je Vlasnik Proizvoda dobio od klijenta najčešće preko direktne komunikacije i pretvorio u korisničke priče lako razumljive razvojnom timu. Priče koje je Vlasnik Proizvoda napisao su sljedeće:

1. Kao korisnik web stranice, želim pristupiti glavnoj stranici na kojoj želim imati veliku naslovnu sliku tvrtke, kako bih na prvo mjesto stavio vizualni identitet tvrtke.
2. Kao korisnik web stranice, želim koristiti očima ugodne fontove slova, kako bi korisnici bili privučeni ostati na stranici i pročitati njezin sadržaj.
3. Kao klijent, želim moći uređivati tekst nekih od odlomaka stranice, kako bih imao mogućnost izmjene teksta po potrebi.
4. Kao korisnik web stranice, želim imati mogućnost pročitati nešto više o tvrtki Tvrtka1, kako bih doznao sve bitne informacije koje bi mi omogućile razmišljanje o daljnjim koracima.
5. Kao korisnik web stranice, želim imati uvid u kontakt informacije tvrtke, kako bih mogao u željeno vrijeme stupiti u kontakt s tvrtkom.
6. Kao klijent, želim implementirati potrebne alate koji bi mi omogućili da se moja stranica prikazuje na tražilicama na što višem mjestu, kako bi moja tvrtka ostala relevantna na tržištu.
7. Kao korisnik web stranice, želim da moja stranica ima moderne animacije, kako bi što bolje privukla strane investitore.
8. Kao član razvojnog tima, želim da kôd kojeg implementiram bude napisan po pravilima struke te da se svi paketi koje koristim mogu s lakoćom ažurirati na najnovije verzije bez ulaganja mnogo truda.
9. Kao klijent, želim da moja stranica ima neku vrstu kontakt obrasca putem kojem se može kontaktirati moja tvrtka.
10. Kao klijent, želim da moja stranica ima implementaciju neke vrste mape koja bi prikazivala lokaciju moje tvrtke.

10.7. Korak 2: Vlasnik Proizvoda dogovara plan s razvojnim timom i *Scrum Masterom*

Naglasak agilnih metoda je uvijek na redovnoj i konstruktivnoj komunikaciji. U primjeru bi Vlasnik Proizvoda stoga odlučio imati tjedne sastanke sa razvojnim timom i *Scrum Masterom* na kojima bi se razgovaralo o sljedećim pitanjima:

1. Jesu li korisničke priče dovoljno jasne?
2. Mogu li vam kako pomoći u razvoju softvera?
3. Jesu li vam potrebne dodatne specifikacije?
4. Hoćemo li stići napraviti sve u zadanom roku *Sprinta*?

Pošto je razvojni tim uvidio da nema problema s opisom korisničkih priča i uvjeren je da će stići izvršiti sve zadatke u zadanom roku, Vlasnik Proizvoda ne vidi problem te u dogovoru sa *Scrum Masterom* dogovara datum početka i završetka *Sprinta*.

10.8. Korak 3: *Scrum Master* započinje *Sprint*

Prije nego *Sprint* može započeti, sve korisničke priče moraju biti dodijeljene članovima razvojnog tima. Članovi razvojnog tima imaju potpuno pravo predlagati, odabrati i međusobno premještati korisničke priče. Štoviše, poželjno je premjestiti korisničku priču na nekog drugog programera ukoliko jedan programer uvidi da nije u mogućnosti dovršiti svoju korisničku priču na vrijeme.

Scrum Master u dogovoru s Vlasnikom Proizvoda namješta *Sprint* na dva tjedna te *Sprint* može započeti.

10.8.1. Dnevni sastanci unutar *Sprinta*

Scrum Master je namjestio automatizirane sastanke svaki dan. Na njima se raspravlja o trenutnom napretku razvojnog tima vezanom za razvoj web stranice[12]. Komunikacija na tim sastancima je opuštena i ne zahtijeva predodređeni oblik.

10.9. Korak 4: Korištenje *Scrum* ploče

Gledajući *Scrum* ploču sa strane razvojnog tima i Vlasnika Proizvoda, prilikom uzimanja to jest dodjeljivanja korisničke priče u fazu razvoja, programer je dovlači do stupca TRENUTNO U RAZVOJU. Nakon što je razvoj dovršen, programer stavlja korisničku priču na neku testnu verziju softvera koja je dostupna Vlasniku Proizvoda te stavlja korisničku priču u stupac NA PREGLEDU. Potrebu za ovim stupcem je uvidio Vlasnik Proizvoda jer bi htio provjeriti odgovara li implementacija korisničke priče onome što je napisano u njoj.

Vlasnik Proizvoda je odlučio da je najbolji način dostavljanja povratnih informacija vezanih za određenu priču korištenje komentara unutar korisničke priče. Ukoliko neka od korisničkih priča ne odgovara implementaciji, ta korisnička priča se može vratiti u TRENUTNO U RAZVOJU stupac gdje se vrše izmjene sukladno zahtjevima Vlasnika Proizvoda.

Ukoliko korisnička priča odgovara implementaciji, Vlasnik Proizvoda može odlučiti da je ona spremna biti puštena van te o tome obavještava razvojni tim. U slučajevima u kojima korisničke priče ovise jedna o drugoj, one neće izaći jedna prije druge, to jest neće biti pomaknute u stupac DOVRŠENO. Stupac DOVRŠENO označava da su svi stupnjevi razvoja gotovi te je korisnička priča dostupna krajnjim korisnicima.

---

[12] *Scrum* vodič je u 2021. godini naveo novi način održavanja dnevnih sastanaka u kojem se navodi da naglasak više nije na tri uvriježena pitanja (što je odrađeno jučer, što će biti odrađeno danas, ima li kakvih problema), nego na slobodnoj komunikaciji koja nije restriktivna poput gore navedenih pitanja [16].

Gledano sa strane upravljanja projektom, nakon što sve korisničke priče završe u stupcu DOVRŠENO, značajka ili projekt se može smatrati dovršenim. Ono što slijedi nakon gore navedenih koraka su, u većini slučajeva, ispravci grešaka u sustavu ili dodavanje novih značajki. Postupak za oba slučaja je isti, to jest dodaju se nove korisničke priče u *Backlog*.

10.10. Korak 5: Prijavljivanje grešaka u sustavu

Oblikovanje prijave grešaka u sustavu može biti u potpunosti prilagođeno svakom razvojnom timu jer svaki razvojni tim može na drugačiji način zadati oblik informacija koje su potrebne kako bi se što brže došlo do rješenja problema.

Slika 10: Sučelje izvješća greške

Izvor: Autor

Na slici 10 može se vidjeti način prijavljivanja izvješća greške (eng. Bug report). Oblik je unaprijed određen sukladno dogovoru između razvojnog tima i Vlasnika Proizvoda. U ovom projektu dogovorene su sljedeće točke obrasca:

1. Opis greške - u opisu greške se može detaljno objasniti na kakvu grešku je korisnik naišao pri korištenju značajke.
2. Koraci za ponavljanje greške - koraci služe kako bi razvojni tim mogao što bolje uočiti grešku radeći iste korake koje je radio korisnik.
3. Internetski preglednik - uzimajući u obzir da se u ovom projektu radi o web stranici, internetski preglednik je nešto što može biti vrlo korisna informacija jer postoji mogućnost da se implementacija nekih od razvojnih elemenata razlikuje između više internetskih preglednika.
4. Snimka (slika) zaslona - ukoliko je to moguće, poželjno je snimiti zaslon kako bi razvojni tim imao bolju predodžbu načina na koji se greška pojavljuje.

Nakon što su sve korisničke priče dovršene i greške ispravljene, projekt se i službeno može završiti. Na *Scrum Masteru* i Vlasniku Proizvoda ostaje odluka žele li zadržati projekt u Jiri ili ga arhivirati te za potrebe održavanja otvoriti novi projekt.

Svrha ovakvog upravljanja razvojem softvera je lakše nadziranje razvojnog tima, mogućnost uvida u svaku od faza korisničke priče i razdvajanje uloga od osobe koja stvara korisničke priče, osobe koja upravlja dodjeljivanjem korisničkih priča do osoba koje su zadužene za implementaciju. Ovakav pristup također nudi transparentnost, što je jedna od odlika agilnog razvoja informacijskih sustava.

11. Zaključak

U ovom radu je prikazana Agilna metodologija, njeno porijeklo te glavni principi Agilne metodologije. Naglasak cijelog rada je bio na *Scrum* metodi, no u radu su također objašnjene metodologije *Kanban* i *Scrumban.* Također, napravljena je usporedba između *Kanban* i *Scrum* metoda gdje su vidljive jasne razlike, ali i doticajne točke tih dviju metoda. Za potrebe lakšeg razumijevanja toka zadataka, načina dostavljanja specifikacija, korištenja korisničkih priča i komunikacije između svih elemenata *Scrum* metode korišten je alat Jira čije je sučelje objašnjeno na jednostavan i koncizan način. Uz pomoć Jire je napravljen tijek korisničke priče od njenog začetka do izlaska prema krajnjim korisnicima.

Popis literature

1. https://www.planview.com/resources/guide/agile-methodologies-a-beginners-guide/history-of-agile/ (01.08.2021)
2. https://agilemanifesto.org/iso/hr/manifesto.html (15.08.2021.)
3. https://hr.strephonsays.com/agile-and-vs-traditional-software-development-methodology-11497 (05.09.2021.)
4. https://www.atlassian.com/agile/scrum/scrum-master (28.08.2021.)
5. https://www.atlassian.com/agile/project-management/user-stories (28.08.2021.)
6. https://www.atlassian.com/agile/project-management/epics (30.08.2021.)
7. https://www.atlassian.com/agile/kanban/boards (30.08.2021.)
8. https://www.productplan.com/glossary/scrumban/ (05.09.2021.)
9. https://www.pivotaltracker.com/blog/getting-started-with-agile-customer-collaboration-over-contract-negotiation(17.07.2021.)
10. https://platinumedge.com/blog/agile-manifesto-responding-to-change (18.07.2021.)
11. http://www.infotrend.hr/clanak/2013/8/agilni-razvoj-softvera,77,1013.html (20.08.2021.)
12. https://adamcogan.com/2021/05/18/8-important-changes-to-the-scrum-guide-for-2021/ (01.09.2021.)
13. https://www.atlassian.com/agile/scrum/backlogs (30.08.2021.)
14. https://www.atlassian.com/agile/kanban/kanban-vs-scrum (30.08.2021.)
15. https://www.atlassian.com/software/jira (06.09.2021.)
16. https://adamcogan.com/2021/05/18/8-important-changes-to-the-scrum-guide-for-2021/ (08.09.2021.)
17. https://www.atlassian.com/agile/project-management/estimation (08.09.2021.)

Popis slika i tablica