Agilne metode razvoja programske podrške

Šiško, Mirko

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

AGILNE METODE RAZVOJA PROGRAMSKE PODRŠKE

Mirko Šiško

Split, rujan 2022.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

AGILNE METODE RAZVOJA PROGRAMSKE PODRŠKE

Mirko Šiško

SAŽETAK

Obrađena tema projekta i njegovog vođenja. Sudionici vođenja projekta kao i njegovog izvršavanja. Tradicionalni načini vođenja projekta i njihove prednosti i mane. Detaljno opisan princip agilnog upravljanja razvoja programske podrške. Opisane razne agilne metode. Usporedba sa tradicionalnim metodama. Usporedba raznih alata za agilno upravljanje projektima. Primjena pokazana kroz aplikaciju vođenja projekta.

Ključne riječi: agilno upravljanje, scrum, kanban, vodopadni pristup,

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 41 stranica, 16 grafičkih prikaza, 1 tablica i 11 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: **Doc. Dr. sc. Goran Zaharija,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Ocjenjivači: **Doc. dr. sc. Monika Mladenović,** *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Doc. Dr. sc. Divna Krpan, *docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu*

Rad prihvaćen: **rujan 2022**

Basic documentation card

Thesis

University of Split
Faculty of Science
Department of Informatics
Ruđera Boškovića 33, 21000 Split, Croatia

AGILE SOFTWARE DEVELOPMENT METHODS

Mirko Šiško

ABSTRACT

Covered the topic of project and its management. Participants in project management as well as its execution. Traditional methods of project management and their advantages and disadvantages. The principle of agile management of software development is described in detail. Various agile methods are described. Comparison with traditional methods. Comparison of various tools for agile project management. Application demonstrated through project management application.

Key words: agile development, scrum, kanban, waterfall,

Thesis deposited in library of Faculty of science, University of Split

Thesis consists of: 41 pages, 16 figures, 1 table and 11 references. Original language: Croatian.

Mentor: **Doc. Dr. sc. Goran Zaharija,** *Docent of Faculty of Science, University of Split*

Reviewers: **Doc. Dr. sc. Monika Mladenović,** *Docent of Faculty of Science, University of Split*

Doc. Dr. sc. Divna Krpan, *Docent of Faculty of Science, University of Split*

Thesis accepted: **september 2022**

IZJAVA

kojom izjavljujem s punom materijalnom i moralnom odgovornošću da sam završni rad s naslovom AGILNE METODE RAZVOJA PROGRAMSKE PRODRŠKE

izradio samostalno pod voditeljstvom izv. prof. doc. dr. sc. Goranom Zaharijom. U radu sam primijenio metodologiju znanstvenoistraživačkog rada i koristio literaturu koja je navedena na kraju diplomskog rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući naveo u diplomskom radu na uobičajen, standardan način citirao sam i povezao s fusnotama s korištenim bibliografskim jedinicama. Rad je pisan u duhu hrvatskog jezika.

Student

Mirko Šiško

Sadržaj

Uvod

Razvoj programske podrške (eng. *Software development*) je postupak osmišljavanja, specificiranja, dizajniranja, programiranja, dokumentiranja, testiranja i ispravljanja programskih pogrešaka koje su uključenje u stvaranje i održavanje aplikacija, okvira ili drugih softverskih komponenti. To je postupak pisanja i održavanja izvornog koda, ali u širem smislu uključuje sve što je uključeno između koncepcije željenog softvera i konačne manifestacije softvera, ponekad u planiranom i strukturiranom procesu. Stoga, razvoj softvera može uključivati istraživanje, novi razvoj, izradu prototipa, ponovnu upotrebu, ponovni inženjering, održavanje ili bilo koje druge aktivnosti koje rezultiraju softverskim proizvodima.

Softver se može razviti u razne svrhe, a tri su najčešće za zadovoljavanje specifičnih potreba određenog klijenta / poduzeća, za zadovoljavanje uočenih potreba određenog niza potencijalnih korisnika ili za osobnu upotrebu. Razvoj ugrađenog softvera (eng. *embedded software*) kakav se koristi za kontrolu potrošačkih proizvoda, zahtijeva integriranje razvojnog procesa s razvojem kontroliranog fizičkog procesa. Sistemski softver se temelji na aplikacijama i samom procesu programiranja, a često se razvija zasebno.

Potreba za boljom kontrolom kvalitete procesa razvoja softvera stvorila je disciplinu softverskog inženjerstva kojoj je cilj primijeniti sustavni pristup prikazan u inženjerskoj paradigmi na proces razvoja softvera.

Postoje mnogi pristupi upravljanju softverskim projektima, poznati kao modeli životnog ciklusa (eng. *life cycle methods)* razvoja softvera, metodologije, procesi ili modeli. Model vodopada tradicionalna je verzija, nasuprot novim inovacijama agilnog razvoja softvera.

1. Vođenje projekta

Točnije, što je projekt? Projekt je privremeni napor poduzet za stvaranje jedinstvenog proizvoda, usluge ili rezultata. Projekt je privremen jer ima definiran početak i završetak u vremenu, a time i definirani opseg i resurse.

Projekt je jedinstven po tome što nije rutinska operacija, već određeni skup operacija osmišljenih za postizanje jedinstvenog cilja. Dakle, projektni tim često uključuje ljude koji obično ne surađuju – ponekad iz različitih organizacija i s više područja.

Upravljanje projektom (eng. *project management*) je primjena procesa, metoda, vještina, znanja i iskustava za postizanje specifičnih ciljeva projekta u okviru dogovorenih parametara. Upravljanje projektom ima konačne rezultate koji su ograničeni na vremenski okvir i proračun. Ključni čimbenik koji razlikuje upravljanje projektom od samo „upravljanja" jest taj što ima konačni rezultat i konačni vremenski raspon, za razliku od upravljanja koje je stalan proces. Zbog toga projektnom profesionalcu treba široki spektar; često tehničke vještine, a svakako vještine upravljanja ljudima i dobra poslovna svijest.

Projektima je potrebno upravljati jer je profesionalni softverski inženjering uvijek podložan budžetu i rokovima. Kriterij uspjeha se razlikuje od projekta do projekta, ali za većinu projekata važni ciljevi su:

- Isporuka projekta klijentu u dogovorenom vremenu
- Troškovi unutar budžeta
- Isporuka softvera koji ispunjava očekivanja korisnika
- Održavanje sretnog i funkcionalnog razvojnog tima

1.1. Proces vođenja projekta

Ne postoji samo jedan način vođenja projekta. Mnoge organizacije provode mnogo vremena radeći pogreške i prilagođavajući svoj pristup kako bi ga ispravile kako treba, samo da bi otkrile da ga treba ponovno prilagoditi.

Nove potrebe i ciljevi, novo ili drugačije osoblje i stručnost te razvoj ili nova tehnologija samo su od nekih razloga zbog kojih se procesi moraju prilagoditi. Zato je toliko važno imati osnovni okvir za funkcioniranje projekata u organizaciji ili timu.

Većina modela organizira aktivnosti u 3 osnovne faze:

- **Istraživanje, planiranje** – organizacija će obično provesti određenu razinu istraživanja kako bi utvrdila valjanost projekta. To može biti u obliku istraživanja tržišta, istraživanja korisnika, analize konkurencije ili drugih aktivnosti. Ovo je također faza kada se projektni tim okuplja kako bi se definirao način rada i plan izvršenja uzimajući u obzir sve vanjske čimbenike
- **Izvršenje** – izvršenje projekta može se odvijati na nekoliko načina ovisno o metodologiji koju organizacija ili tim izabere. Ova faza je prožeta suradnjom, stvaranjem, pregledom i ponavljanjem. Tim predstavlja rad, te se povratne informacije prihvaćaju što dovodi do konačnog rezultata. U ovoj fazi je voditelj projekta najaktivniji.
- **Testiranje, mjerenje i ponavljanje** – u agilnom vođenju, napraviti će se minimalni održivi proizvod (MVP) kako bi se dobile rane povratne informacije. Timovi će uzeti rezultate testiranja te izmijeniti ili nadograditi proizvod kako bi napravili nešto što je bliže određenim ciljevima.

1.2. Voditelj projekta

Što radi voditelj projekta? Bez obzira gdje rade, oni zagovaraju dobrobit ljudi koji su uključeni u njihove projekte i nastoje donijeti ili olakšati strateške odluke koje podržavaju ciljeve njihovih projekata. Ovaj posao zahtijeva ravnotežu upravljanja administrativnim detaljima projekta i njegovih ljudi.

Nemoguće je napraviti standardni opis posla za voditelja projekta. Posao se uvelike razlikuje ovisno o organizaciji i softverskom proizvodu koji se razvija. Međutim, većina voditelja preuzima odgovornost u nekoj fazi za neke ili sve sljedeće aktivnosti:

- **Planiranje projekta** – voditelji projekta odgovorni su za planiranje, procjenu i raspored razvoja projekta te dodjeljivanje zadataka zaposlenicima. Oni nadziru rad

kako bi osigurali da se provodi u skladu s potrebnim standardima i prate napredak kako bi provjerili je li razvoj unutar danog vremenskog okvira i proračuna

- **Izvještavanje** – voditelji projekta obično su odgovorni za izvješćivanje o napretku projekta klijentima i voditeljima tvrtke koja razvija softver. Moraju biti sposobni komunicirati na više razina, od detaljnih tehničkih informacija do sažetaka upravljanja. Moraju pisati sažete, koherentne dokumente koji apstrahiraju ključne informacije iz detaljnih izvješća o projektu. Moraju biti u mogućnosti prezentirati te informacije tijekom pregleda napretka.
- **Upravljanje rizikom** – voditelji projekta moraju procijeniti rizike koji mogu utjecati na projekt, pratiti te rizike i poduzeti mjere kada se pojave problemi.
- **Upravljanje ljudima** – voditelji projekta odgovorni su za upravljanjem timom ljudi. Moraju izabrati ljude za svoj tim i uspostaviti načine rada koji vode učinkovitom timskom učinku.
- **Pisanje prijedloga** – prva faza softverskog projekta može uključivati pisanje prijedloga za dobivanje ugovora za izvođenje određenog posla. Prijedlog opisuje ciljeve projekta i način na koji će se on provesti. Obično uključuje procjene troškova i rasporeda te opravdava zašto bi projektni ugovor trebao biti dodijeljen određenoj organizaciji ili timu.

1.3. Tipovi upravljanja projektom

Metode upravljanja projektom mogu se primijeniti na bilo koji projekt. Često je prilagođen određenoj vrsti projekata na temelju veličine projekta, industrije ili sektora.

Uobičajeno među svim vrstama upravljanja projektima jest da se usredotočuju na tri važna cilja: vrijeme, kvalitetu i proračun. Uspješni projekti dovršavaju se prema rasporedu, u okviru proračuna i prema prethodno dogovorenim standardima kvalitete, tj. ispunjavanju željeznog trokuta (eng. *iron triangle*) kako bi se projekti mogli smatrati uspjehom ili neuspjehom.

Za svaku vrstu upravljanja projektima, voditelji razvijaju ili koriste ponovljive predloške koji su specifični za industriju s kojom se bave. To omogućuje da projektni planovi postanu

vrlo temeljiti i ponovljivi, sa posebnom namjerom da se poveća kvaliteta, snize troškovi isporuke i skrati vrijeme za isporuku rezultata projekta.

Postoji niz pristupa organiziranju i dovršavanju projektnih aktivnosti, uključujući: fazni, vitki, iterativni i inkrementalni. Postoji i nekoliko proširenja za planiranje projekata, na primjer na temelju ishoda ili aktivnosti.

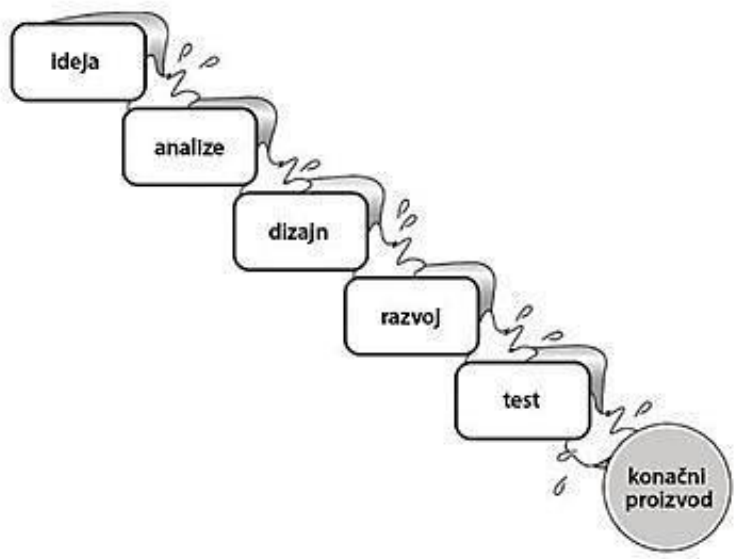
Bez obzira na primijenjenu metodologiju, mora se pažljivo razmotriti cjelokupni ciljevi projekta, vremenski raspored i trošak, kao i uloge i odgovornosti svih sudionika i dionika.

1.3.1. Fazno upravljanje projektima (vodopadni / eng. *waterfall*)

Fazni ili scenski pristup razbija i upravlja radom kroz niz različitih koraka koje treba dovršiti, a često se naziva „tradicionalnim“ ili „vodopadnim“. Naziva se vodopadnim zbog kaskade jedne faze u drugu. Model vodopada je primjer procesa vođenog planom, u principu planiranje i raspoređivanje svih aktivnih procesa se obavlja prije nego počne rad na njima. Iako se može razlikovati, obično se sastoji od pet procesnih područja:

1) **Analiza i definicija zahtjeva** – usluge sustava, ograničenja i ciljevi utvrđuju se konzultacijama s korisnicima sustava. Zatim se detaljno definiraju i služe kao specifikacije sustava.
2) **Dizajn sustava i softvera** – proces dizajna sustava dodjeljuje zahtjeve hardverskim ili softverskim sustavima uspostavljanjem ukupne arhitekture sustava. Uključuje identificiranje i opisivanje temeljnih apstrakcija softverskog sustava i njihovih odnosa.
3) **Implementacija i jedinično (eng. *unit*) testiranje** – tijekom ove faze, dizajn softvera realizira se kao skup programa ili programskih jedinica. Testiranje jedinice uključuje provjeru zadovoljava li svaka jedinica svoju specifikaciju.
4) **Integracija i testiranje sustava** – pojedinačne programske jedinice ili programi su integrirani i testirani kako bi se osiguralo ispunjavanje softverskih zahtjeva. Nakon testiranja softverski se sustav isporučuje klijentu.
5) **Rad i održavanje** – ovo je najduža faza (ne nužno) životnog ciklusa. Sustav je instaliran i pušten u upotrebu. Održavanje uključuje ispravljanje grešaka koje nisu

otkrivene u ranijim fazama životnog ciklusa, poboljšanje implementacije jedinica sustava i poboljšanje usluga sustava kako se otkrivaju novi zahtjevi.



Slika 1: Vodopadni model

U načelu, rezultat svake faze je jedan ili dva dokumenta koji su odobreni. Sljedeća faza ne bi trebala započeti dok prethodna faza ne završi. U praksi se ove faze preklapaju i međusobno se prenose informacije. Tijekom dizajniranja identificiraju se problemi sa zahtjevima. Tijekom kodiranja identificiraju se problemi sa dizajnom itd.

Zbog troškova proizvodnje i odobravanja dokumenata, iteracije mogu biti skupe i mogu zahtijevati značajne prerade. Stoga je nakon malog broja ponavljanja normalno zamrznuti dijelove razvoja, poput specifikacije i nastaviti s kasnijim fazama razvoja.

Mnoge industrije koriste varijacije ovih faza projekta i nije rijetkost da se faze preimenuju kako bi bolje odgovarale organizaciji. Iako vodopadni pristup dobro funkcionira za male, dobro definirane projekte, često rezultira izazovom ili neuspjehom na većim projektima ili onim koji su složeniji ili imaju više nejasnoća, problema i složenosti.

1.3.2. Iterativno i inkrementalno upravljanje projektima

U kritičnim studijama upravljana projektima primijećeno je da fazni pristupi nisu dobro prikladni za projekte velikih tvrtki s nedefiniranim, dvosmislenim ili zahtjevima koji se brzo i često mijenjaju ili onim sa visokim stupnjem rizika. Konus nesigurnosti objašnjava nešto od toga jer planiranje u početnoj fazi projekta pati od visokog stupnja nesigurnosti. To postaje osobito istinito jer je razvoj softvera često realizacija novog proizvoda.

Te se složenosti bolje rješavaju istraživačkim ili iterativnim i inkrementalnim pristupom. Razvilo se nekoliko modela iterativnog i inkrementalnog upravljanja projekta, uključujući agilno upravljanje, metodu razvoja dinamičkih sustava, ekstremno upravljanje projektima i inovativni inženjering.

Iterativni i inkrementalni razvoj je proces koji kombinira iterativnu metodu projektiranja s inkrementalnim modelom izgradnje. Dijeli se na dva dijela:

1) **Inkrementalni** – rastavlja proces razvoja softvera na male, upravljive dijelove poznate kao inkrementi. Svaki se inkrement nadograđuje na prethodnu verziju tako da se poboljšanja rade korak po korak
2) **Iterativni** – aktivnosti razvoja softvera se sustavno ponavljaju u ciklusima poznatim kao iteracije. Nova verzija softvera se proizvodi nakon svake iteracije dok se ne postigne optimalni proizvod

Inkrementalni razvoj osigurava da tim može napraviti promjene rano u procesu umjesto da čekaju do kraja, kada iscrpe vremenski okvir i budžet.

Prednosti iterativnog pristupa su brojne:

- Klijenti mogu rane inkremente koristiti kao prototipove i steći iskustvo koje informira njihove zahtjeve kasnije inkremente sustava
- Klijenti ne moraju čekati da cijeli sustav bude isporučen da bi dobili vrijednost od njega
- Proces dobiva prednosti inkrementalnog razvoja u smislu da bi trebalo biti relativno lagano ugraditi promjene u sustav
- Budući da se prvo isporučuju usluge najvišeg prioriteta, a potom integriraju, najvažnije usluge dobivaju najviše testiranja. To znači da je manja vjerojatnost da će se korisnici susresti s kvarovima softvera u najvažnijim dijelovima sustava

Međutim postoje i problemi s ovim pristupom:

- Većina sustava zahtijeva skup osnovnih objekata koje koriste različiti dijelovi sustava. Budući da zahtjevi nisu detaljno definirani dok se ne implementira inkrement, može biti teško identificirati zajedničke pogodnosti koje su potrebne svim inkrementima
- Iterativni razvoj također može biti težak kada se razvija zamjenski sustav. Korisnici žele sve funkcionalnosti starog sustava i često nisu volji eksperimentirati s nepotpunim novim sustavom. Stoga je teško dobiti korisne povratne informacije.
- Bit iterativnih procesa je da se specifikacija razviju usporedno s softverom. Međutim, to je u sukobu s modelom nabave mnogih organizacija, gdje je kompletna specifikacija sustava dio ugovora o razvoju sustava.

Iterativni razvoj omogućava da se poboljšanja rade na kontinuiranoj osnovi, tako da će krajnji rezultat vjerojatno biti isporučen na vrijeme i biti više kvalitete.

Iterativni pristup ima tri važne prednosti:

1) Trošak prilagođavanja promjenjivim zahtjevima klijenata je smanjen. Količina analize i dokumentacije koja se mora ponovno izraditi mnogo je manja u usporedbi s npr. Modelom vodopada.
2) Lakše je dobiti povratne informacije od klijenata o obavljenom razvoju. Klijenti mogu komentirati demonstracije softvera i vidjeti koliko je implementirano.
3) Moguća je brža isporuka i implementacija korisnog softvera klijentu, čak i ako nisu uključene sve funkcionalnosti. Klijenti mogu koristiti softver i dobiti vrijednost od njega prije samoga kraja razvoja.

Inkrementalni razvoj ima dva problema, iz perspektive upravljanja:

1) Proces nije vidljiv. Voditelji trebaju redovite rezultate kako bi mjerili napredak. Ako se sustavi brzo razvijaju, nije isplativo proizvoditi dokumente koji odražavaju svaku verziju sustava.
2) Struktura sustava ima tendenciju degradacije kako se dodaju novi inkrementi. Osim ako se vrijeme i novac ne potroše na refaktoriranje radi poboljšanja softvera, redovite promjene imaju tendenciju kvariti njegovu strukturu. Uključivanje daljnjih promjena postaje sve teže i skuplje.

2. Agilne metode

Agilno upravljanje projektima je iterativni pristup isporuci projekta tijekom njegovog životnog ciklusa. Iterativni životni ciklus sastoji se od nekoliko ponavljanja ili inkrementalnih koraka prema završetku projekta. Jedan od ciljeva agilnog pristupa je isporuka rezultata tijekom cijelog postupka, a ne samo na kraju.

Kao što samo ime govori, agilno upravljanje omogućuje timovima brzu promjenu smjera i fokusa. S 94% tvrtki koje prakticiraju agilno upravljanje u 2016. godini, postalo je industrijski standard za upravljanje projektima.

Povijest ovog pristupa počinje 1957. godine: u to vrijeme Bernie Dimsdale, John Von Neumann, Herb Jacobs i Gerald Weinberg koristili su inkrementalne razvojne tehnike, gradeći softver za IBM i Motorolu. Iako, ne znajući kako klasificirati pristup koji su prakticirali, jasno su shvatili da se on u mnogočemu razlikuje od „vodopada“.

Međutim, moderni agilni pristup službeno je uveden 2001. godine, kada se skupina od 17 stručnjaka za razvoj softvera sastala kako bi razgovarali o alternativnim metodologijama upravljanja projektima. Imajući jasnu viziju fleksibilnog, laganog (eng. *lightweight*) i timski usmjerenog (eng. *team oriented*) pristupa razvoju softvera, to su zacrtali u „Manifesto for Agile Software Development“[1].

S ciljem otkrivanja boljih načina razvoja softvera, Manifest jasno precizira temeljna načela novog pristupa:

„Kroz ovo djelo došli smo do vrijednosti:

Pojedinci i interakcije prije procesa i alata

Ispravan softver prije sveobuhvatne dokumentacije

Suradnja kupaca prije pregovora o ugovoru

Odgovor na promjenu prije držanja plana.“

Nadopunjena sa dvanaest načela agilnog softvera, filozofija je postala univerzalni i učinkoviti novi način upravljanja projektima. Softver nije razvijen kao jedan cjelina već kao

niz inkrementa, pri čemu svaki inkrement uključuje novu funkcionalnost sustava. Iako postoji mnogo pristupa brzom razvoju softvera, oni dijele neke temeljne karakteristike:

- Procesi specifikacije, dizajna i implementacije su isprepleteni. Ne postoji detaljna specifikacija sustava, a projektna dokumentacija je minimizirana ili generirana automatski od strane programskog okruženja koje se koristi za implementirani sustav. Dokument korisničkih zahtjeva definira samo najvažnije karakteristike sustava.
- Sustav je razvijen u nizu verzija. Krajnji korisnici i drugi dionici sustava uključeni su u specifikaciju i procjenu svake verzije. Oni mogu predložiti promjene softvera i nove zahtjeve koji bi se trebali implementirati u kasnijoj verziji sustava.
- Korisnička sučelja sustava često se razvijaju korištenjem interaktivnog razvojnog sustava koji omogućuju brzo kreiranje dizajna sučelja crtanjem i postavljanjem ikona na sučelje. Sustav tada može generirati web sučelje za preglednik ili sučelje za određenu platformu kao što je Microsoft Windows.

Kao i svaki drugi profesionalni proces razvoja softvera, agilnim razvojem treba upravljati tako da se vrijeme i resursi iskoriste na najbolji način. To zahtijeva drugačiji pristup upravljanju projektima koji je prilagođen inkrementalnom razvoju i posebnim prednostima agilnih metoda. Za razliku od uobičajenog linearnog modela vodopada, agilni projekti sastoje se od niza manjih ciklusa – sprintova. Svaki od njih je minijaturni projekt: ima „backlog“ i sastoji se od faza dizajniranja, implementacije i testiranja u unaprijed definiranom opsegu posla.

Slika 2: Agilni razvojni ciklus

Na kraju svakog sprinta isporučuje se potencijalni dodatak proizvodu. Stoga, sa svakom iteracijom proizvodu se dodaje nova značajka, što rezultira postupnim rastom projekta. S obzirom da su značajke provjerene pri početku razvoja, šanse za isporuku potencijalno neuspjelog proizvoda znatno su manje.

Sažmimo glavne aspekte agilnog upravljanja:

- Fleksibilnost – Opseg posla može se mijenjati prema novim zahtjevima
- Podjela posla – Projekt se sastoji od malih ciklusa
- Vrijednost timskog rada – Članovi tima blisko surađuju i imaju jasnu viziju svojih odgovornosti
- Iterativna poboljšanja – Često se vrši procjena posla obavljenog unutar ciklusa kako bi konačni proizvod bio bolji
- Kooperacija sa klijentom – Kupac je usko uključen u razvoj i može promijeniti zahtjeve ili prihvatiti prijedloge tima

Prema istraživanju, davanje prednosti fleksibilnosti o brzoj promjeni, agilni pristup pruža sljedeće koristi:

- Sposobnost upravljanja promjenjivim prioritetima
- Povećana produktivnost tima preko dodjele dnevnih zadataka
- Bolja preglednost projekta zahvaljujući jednostavnom sustavu planiranja

2.1. Agilni okviri (eng. *frameworks*)

Agilno upravljanje je pojam za široki spektar metodologija i tehnika koje dijele gore navedena načela i vrijednosti. Svaka od njih ima svoja područja upotrebe i prepoznatljive značajke. Najpopularniji okviri su Scrum, Kanban, Hybrid, Lean, Bimodal i XP.

	Tradicionalni razvoj	Agilni razvoj
Osnovna pretpostavka	Sustavi se u potpunosti mogu specificirati, predvidljivi su i razvijaju se kroz dugo i detaljno planiranje	Prilagodljivi softver visoke kvalitete razvijaju mali timovi koji koriste principe stalnog razvijanja dizajna i testiranja baziranog na brzim povratnim informacijama i promjenama
Stil managementa	Zapovijedanje - kontrola	Vodstvo - kolaboracija
Management znanja	Eksplicitan	Prešutan
Komunikacija	Formalna	Neformalna
Model razvoja	Model životnog ciklusa	*Evolutionary-delivery* model
Organizacijska struktura	Mehanička (birokratska, visoka razina formalizacije), povoljna za velike organizacije	Organska (fleksibilna i participativna, potiče socijalnu interakciju), povoljna za male i srednje organizacije
Kontrola kvalitete	Otežano planiranje i striktna kontrola. Otežano i kasno testiranje	Stalna kontrola zahtjeva, dizajna i rješenja. Stalno testiranje
Zahtjevi korisnika	Detaljizirani i definirani prije pisanja koda	Interaktivni input
Trošak ponovnog početka	Veliki	Mali
Razvojni smjer	Fiksiran	Može se lako mijenjati
Testiranje	Nakon stoje pisanje koda dovršeno	Tijekom svake iteracije
Uključenost klijenta	Mala	Velika
Dodatne sposobnosti koji programeri moraju imati	Ništa osobito	Sposobnost međuljudske komunikacije i osnovno poznavanje poslovanja
Veličina projekta	Velika	Mala i srednja
Programeri	Orijentirani na plan, imaju dovoljno sposobnosti, pristup prema vanjskom znanju	Prilagodljivi, s naprednim znanjem, na istoj lokaciji i skloni suradnji

Klijenti	Imaju pristup znanju, kooperativni, reprezentativni i ovlašteni	Posvećeni, puni znanja, kooperativni, reprezentativni i ovlašteni
Zahtjevi	Veoma stabilni, poznati u-naprijed	Nenadani, sa brzim promjenama
Arhitektura	Dizan za sadašnje i predvidljive zahtjeve	Dizajn za sadašnje zahtjeve
Ponovno modeliranje	Skupo	Nije skupo
Veličina	Veliki timovi i projekti	Mali timovi i projekti
Glavni ciljevi	Visoka sigurnost	Brza vrijednost

Tablica 1: Razlike između tradicionalnog i agilnog upravljanja projektom

2.1.1. Scrum

Scrum je dominantna agilna metodologija, fokus je na upravljanju iterativnim razvojem, a ne na specifičnim tehničkim pristupima agilnom softverskom inženjerstvu. Prvi put je spomenut 1986. Hirotaka Takeuchi i Ikujiro Nonaka u „New Product Development Game“[2]. Formuliran je 1995. godine. Ken Schwaber i Jeff Sutherland, autori knjige „The Scrum Guide“[3], predstavili su ga na konferenciji OOPSLA (eng. *Object-Oriented Programming, Systems, Languages & Applications*). Prezentacija se temeljila na znanju koje su stekli primjenom metode tijekom prethodnih godina. Iako je Scrum predstavljen prije „Agile Manifesto“, temelji se na načelima i vrijednostima navedenim u tom dokumentu.

Scrum se temelji na sustavnim interakcijama između tri glavne uloge: Scrum Master, vlasnik proizvoda i tim.

- Scrum Master – središnja osoba u projektu. Glavna uloga je uklanjanje prepreka koje bi mogle spriječiti tim da radi efikasno.
- Vlasnik proizvoda (eng. *Product owner*) – klijent ili vlasnik dionica, aktivno uključen u projekt, iznosi opću viziju proizvoda i pruža povratne informacije o obavljenom poslu nakon svakog sprinta
- Tim – skupina ljudi koja je odgovorna za implementaciju proizvoda.

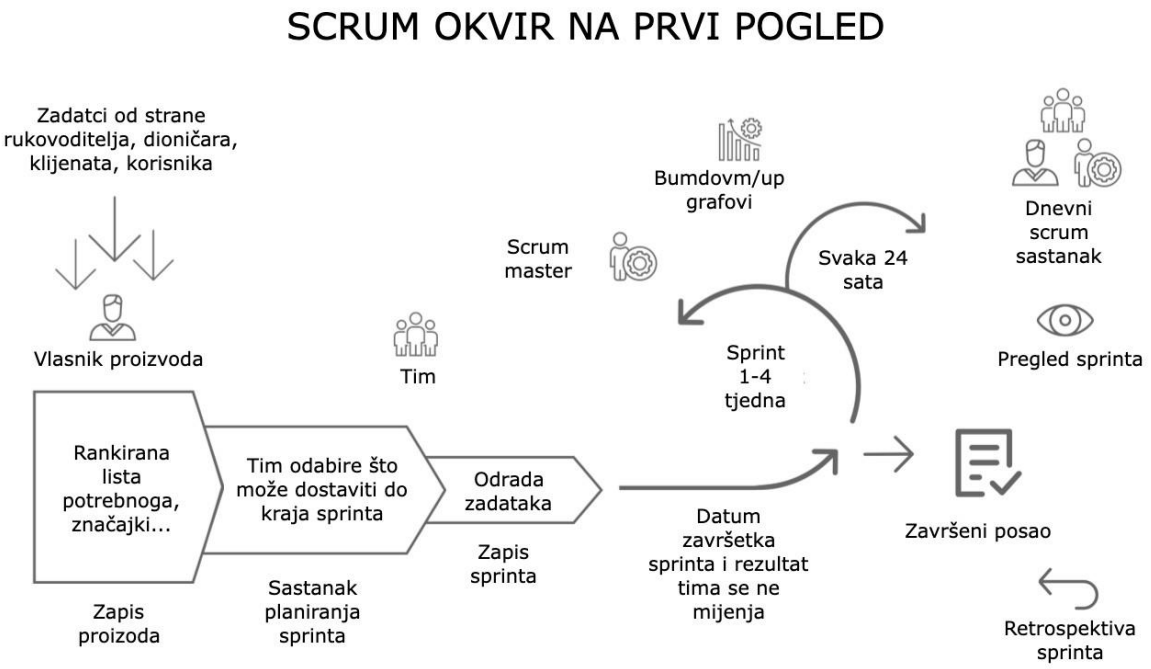
U scrumu postoje tri faze. Prva je je okvirna faza planiranja u kojoj se utvrđuje opći ciljevi projekta i dizajnira arhitektura softvera. Nakon toga slijedi niz ciklusa sprinta, gdje svaki ciklus razvija prirast sustava. Konačno, faza zatvaranja projekta gdje se projekt završava i

potrebnu dokumentaciju kao što su okviri za pomoć sustava i korisničke priručnike te procjenjuje lekcije naučene iz projekta.

2.1.1.1 Sprint i artefakti (eng. *Artifacts*)

Osnovna jedinica rada u scrumu je sprint, kratki razvojni ciklus potreban za stvaranje koraka/značajke/inkrementa proizvoda koji se može isporučiti. Ključne karakteristike ovog procesa su sljedeće:

- Sprintovi su fiksne duljine, obično 2-4 tjedna.
- Polazna točka za planiranje je zaostatak proizvoda, što je popis posla koji treba obaviti na projektu. Tijekom faze procjene sprinta, to se pregledava i dodjeljuju prioriteti i rizici. Klijent je usko uključen u ovaj proces i može uvesti nove zahtjeve ili zadatke na početku svakog sprinta.
- Faza odabira uključuje cijeli projektni tim koji radi s klijentom na odabiru značajki i funkcionalnosti koje će se razviti tijekom sprinta.
- Nakon što su oni dogovoreni, tim se organizira za razvoj softvera. Održavaju se kratki dnevni sastanci koji uključuju sve članove tima kako bi se pregledao napredak i, ako je potrebno, promijenili prioriteti rada. Tijekom ove faze tim je izoliran od klijenta i organizacije, a sva komunikacija kanalizirana je kroz Scrum master-a.
- Na kraju sprinta, obavljeni posao se pregledava i predstavlja klijentu. Tada počinje sljedeći ciklus sprinta.



Slika 3: Scrum okvir na prvi pogled

Scrum se oslanja na tri glavna artefakta koja se koriste za upravljanje zahtjevima i praćenje napretka – „Product backlog“, „Sprint backlog“ i „Sprint burndown chart“. Proces se provodi kroz niz ponavljajućih sastanaka kao što su: Daily Scrum (Standup), Sprint Planning, Review i Retrospective meetings.

Product backlog – uređena lista značajki koje bi mogle biti potrebne u konačnom proizvodu

Sprint Backlog – lista zadataka koje tim mora izvršiti da bi isporučio inkrement funkcionalnog softvera na kraju svakog sprinta. Drugim riječima, članovi tima se dogovore koje će proizvode isporučiti i definiraju plan kako to učiniti

Sprint Burndown Chart – ilustracija preostalog posla u sprintu. Pomaže timu i Scrum masteru jer prikazuje napredak iz dana u dan i može predvidjeti hoće li se cilj sprinta ispuniti prema rasporedu.

2.1.1.2 Scrum sastanci

Dnevni scrum je sastanak tijekom kojeg tim koordinira posao i zadaje plan za sljedeća 24 sata. Sastanak traje 15 minuta i trebao bi se održavati svakodnevno u isto vrijeme.

Posao koji se treba dovršiti se planira na **Sprint planiranju**. Svi koji su uključeni u sprint sudjeluju, te odgovaraju na dva ključna pitanja: koji posao se može obaviti i kako će se taj posao obaviti. Ne traje dulje od 8 sati za jednomjesečni sprint.

Na kraju svakog sprinta, tim i vlasnik proizvoda sastaju se na **pregledu sprinta**. Tijekom ovog neformalnog sastanka, tim pokazuje dovršeni posao i odgovara na pitanja vezana u proizvod. Svi sudionici surađuju na sljedećem koraku kako bi se povećala vrijednost proizvoda. Pregled sprinta je 4-satni sastanak za jednomjesečne sprintove.

Cijeli tim odlazi na **Retrospektivne sastanke** kako bi razmislili o svom radu tijekom sprinta. Sudionici diskutiraju o tome što je pošlo za rukom, a što ne, pronalaze načine za poboljšanje i planiraju kako implementirati te pozitivne promjene. Održava se nakon pregleda sprinta i prije sljedećeg sprint planiranja. Traje oko tri sata za jednomjesečne sprintove.

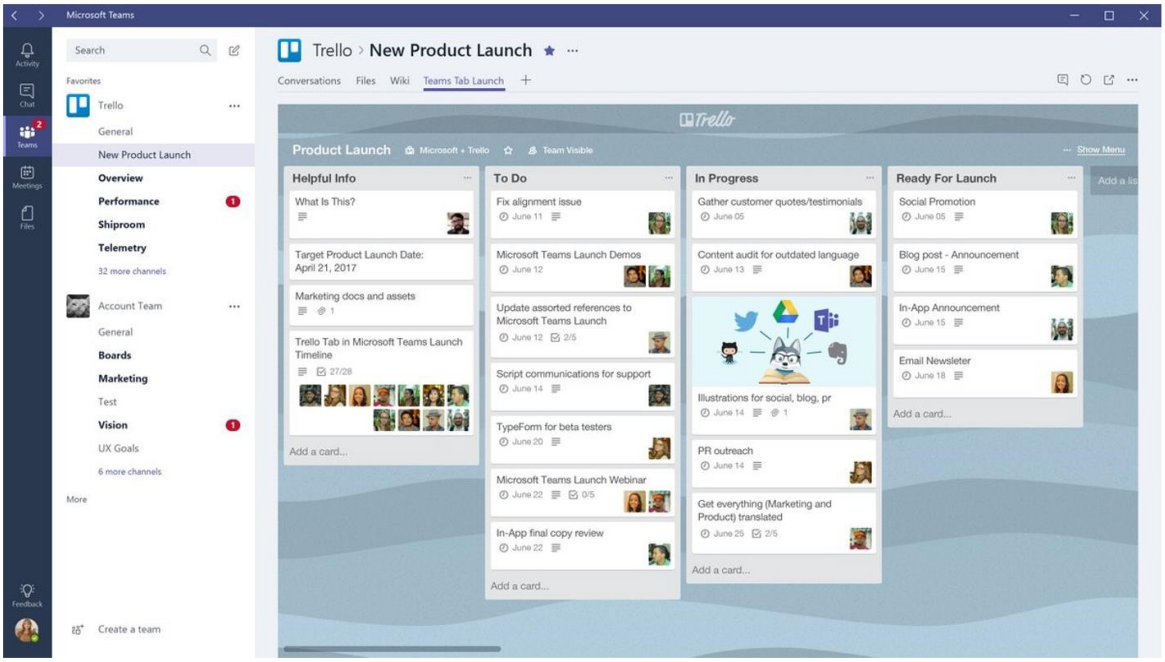
2.1.2. Kanban

Još jedan uobičajeni okvir za upravljanje projektima je kanban. Potječe iz vizualnog sistema karata koji se koristi u Toyotinoj proizvodnji kao metoda kontrole proizvodnje. Kanban je jednostavan, ali moćan pristup razvoju softverskih proizvoda.

Slika 4: Kanban ploča

Kanban se fokusira na vizualizaciju tijeka rada i daje prioritet radu u tijeku (eng. *Work in progress* / WIP) ograničavajući njegov opseg kako bi ga učinkovito prilagodio kapacitetu tima. Čim se jedan zadatak obavi, tim može uzeti sljedeću stavku. Dakle, razvojni proces

nudi veću fleksibilnost u planiranju, jasne ciljeve i transparentnost. U kanbanu nisu potrebni standardni postupci kao ni fiksne iteracije, za razliku od scruma. Razvoj projekta temelji se na vizualizaciji tijeka rada kroz Kanban ploču, obično predstavljenu ljepljivim bilješkama i pločama ili mrežnim alatima poput Trella.



Slika 5: Trello

Trello automatizira i digitalizira Kanban. Zbog tipa podataka o radnom zadatku koje sadrži kanban kartica, svi u timu znaju tko je odgovoran za taj određeni zadatak, kada bi trebao gotov… Članovi tima mogu također ostavljati komentare, priložiti snimke zaslona, dokumente ili linkove.

Timovi koji koriste kanban alate rade u suradnji. Sposobnost da prate napredak pomaže suradnicima da shvati svačiji osobni doprinos u postizanju cilja, što rezultira fokusiranjem na kvalitetan završetak određenog zadatka na vrijeme.

2.1.3. Hibrid (eng. *Hybrid*)

Hibridno upravljanje projektima steklo je značajnu popularnost posljednjih godina. Kombinirajući višestruke metodologije upravljanja projektima, hibridno upravljanje projektima uključuje prednosti drugih metodologija. I dok se pojam hibridnog upravljanja projektima odnosi na bilo koju novu metodologiju stvorenu kombinacijom dviju ili više postojećih metodologija, u praksi, hibridni pristup općenito ujedinjuje agilne metode s

vodopadnim. Stručnjaci uzimaju prednosti filozofije agilnog upravljanja, a što se tiče proračuna, planiranja, tu koriste pristup vodopada. Koristeći načela i jednog i drugog pristupa tvrtke mogu povisiti šanse realiziranja uspješnih projekata. Na primjer, planiranje se može izvršiti pomoću sprintova, testiranje se može odvijati tijekom razvoja, a povratne informacije se mogu prikupljati uobičajeno. Ostale modifikacije vodopadnog pristupa podrazumijevaju korištenje Kanban ploče i retrospektivno organiziranje. Značajke hibridnog okvira ovise o projektu.

Mnogo je čimbenika koji mogu biti povezani s povećanom potrebom za upravljanjem hibridnim projektima. Tri najvažnija su:

- **Povećana složenost projekta** – kroz gotovo svaku industriju, projekti postaju sve složeniji, zahtijevajući poboljšana načela upravljanja kako bi se učinkovito doveli do završetka
- **Poveća tržišna konkurencija** – kako se pojavljuju nove tehnologije i novi igrači ulaze na tržište, ako žele ostati konkurentni, tvrtke širom svijeta otkrivaju da moraju učiniti više kako bi predvidjele i proaktivno odgovorile na probleme koji se pojavljuju
- **Razvoj očekivanja kupaca** – povećani fokus na usredotočenost na kupca stvorio je povećana očekivanja kupaca, posebno u vezi s brzinom, upotrebljivošću i personalizacijom. Tvrtke sada moraju biti u mogućnosti odgovoriti brže nego ikad kako bi odgovorile na promjenjive potrebe svojih klijenata

Kombinacijo agilne i vodopadne metodologije, hibridno upravljanje nudi nekoliko jasnih prednosti:

- **Kompatibilnosti** – uzimajući najbolje iz oba svijeta i dopuštajući značajnu prilagodbu specifičnoj za projekt, hibridno upravljanje projektima može se jednostavno promijeniti na bilo koje veličine, u gotovo svakoj industriji. Ova poboljšanja kompatibilnost često ga čini glavnom metodologijom za organizacije koje trebaju biti u mogućnosti baviti se nizom vrsta projekata
- **Jasnoća odgovornosti** – hibridno upravljanja projektima jasno mapira čitave projekte od početka do kraja, s pojedinostima o cjelokupnom opsegu projekta i odgovornostima onih koji ga vode do kraja. Zaposlenici, menadžeri i ključni dionici mogu odmah vidjeti u kojoj je fazi projekt, koji sljedeći koraci moraju biti dovršeni da bi se pomaknuo naprijed i to je uključen u svaki korak na putu

- **Detaljno planiranje** – zajedno s utvrđivanjem odgovornosti, aspekt planiranja hibridnog upravljanja projektima omogućuje tvrtkama izradu detaljnih planova i točnih procjena troškova. Faze su povezane s određenima isporukama i jasnim postupkom pregleda, a unaprijed definirani sprintovi omogućuju brzo i predvidljivo isporučivanje novih značajki
- **Fleksibilnost** – iskorištavajući punu prednost povećane prilagodljivosti agilne metodologije, hibridno upravljanje omogućuje timovima da jednostavno ponovno procijene projekte usred razvoja, zakrećući se tamo gdje je to potrebno kako bi se bolje pozabavili problemima u nastajanju i promjenom prioriteta

Što se tiče tijeka rada, upravljanje hibridnim projektom sastoji se od pet koraka:

1. **Komponente** – komponente su specifične jedinice projekta; pojedinačni građevni blokovi koji predstavljaju zahtjeve proizvoda
2. **Broj staza** – ovisno o određenim komponentama, projekt će imati barem jednu stazu, ali može imati i nekoliko
3. **Popis zadataka (eng. *Backlog*)** – backlog detaljno opisuje vitalne zadatke koje je potrebno izvršiti u svakoj komponenti. Zadaci se određuju iz backloga
4. **Sprintevi** – hibridno upravljanje projektima posuđuje koncept sprinta iz agilne metodologije. Ti sprintevi predstavljaju stvarni rad koji ide u dovršetak zadataka, uključujući istraživanje, razvoj, testiranje i izdavanje. Svaki sprint obično traje od četiri do osam tjedana, ali može trajati više ili manje, ovisno o projektu
5. **Objavljivanja** - u završnoj fazi projekt je dovršen i izdan. Ova faza također uključuje prikupljanje i uključivanje povratnih informacija korisnika nakon objave

Hibridno upravljanje projektima preuzima dvije bitne uloge iz svojih sastavnih metodologija. Od tradicionalnih pristupa preuzima ideju voditelja projekta, a od agilnog ulogu scrum mastera.

Kako ove dvije uloge međusobno djeluju u upravljanju hibridnim projektima? Odgovornost upravljanja projektom pada na voditelja projekta. On preuzima potpuno vlasništvo nad uspjehom projekta i također je odgovoran za nadgledanje učinkovitog planiranja. Definira ciljeve za zadane vremenske okvire, razlažući glavne ciljeve na pod-ciljeve povezane s određenim komponentama.

Nakon što su postavljeni ciljevi, voditelj projekta radi na definiranju zadataka za svaku komponentu. Imenuje scrum mastera koji će upravljati svakim sprintom i pružiti pojedinosti o tome kako ih treba implementirati. Scrum master tada može izgraditi vlastite timove koji će pomoći u ispunjavanju potreba projekta.

Iako voditelj projekta preuzima ukupnu odgovornost za projekt, trebao bi blisko surađivati sa scrum masterom kako bi učinkovito raščlanio faze projekta u jasno definirane zadatke, te uspostavio razumne rokove i napravio raspored rada za projekt. Scrum master zauzima glavno poziciju za vrijeme trajanja svakog sprinta, a nakon završetka, voditelj projekta dolazi kako bi pregledao rezultate sprinta. Voditelj projekta zatim prosljeđuje svoje rezultate i preporuke natrag scrum masteru koje je odgovoran za uvođenje svih potrebnih poboljšanja u proces prije početka idućeg sprinta.

Pojednostavljeni način gledanja na odnos između voditelja projekta i scrum mastera je razmišljanje o voditelju kao primarno povezanim sa front-end zadatcima, dok scrum master određuje bitne back-end zadatke. Dakle, voditelj bi bio odgovoran za stvari poput prikupljanja i organiziranje povratnih informacija korisnika, definiranja komponenti i postavljanja zahtjeva; scrum master bi preuzeo odgovornost za rukovanje razvojnim sprintovima, upravljanje backlog zadatcima i izdavanje gotovih proizvoda.

2.1.4. Vitki (eng. *Lean*)

Jedan od 5 najčešće korištenih agilnih okvira. Predstavljen je od strane Mary i Tom Poppendick u njihovoj knjizi „Lean Software Development: An Agile Toolkit“[4]. Tu je navedeno 7 osnovnih načela:

1. **Ukloniti nepotrebno** – u kontekstu projekta, „nepotrebno“ označava sve što ne pridonosi vrijednosti projekta
2. **Naglasak na učenju i znanju** – u vitkom okviru, razvoj softvera se smatra procesom učenja. Developeri ne pišu čist kod iz prvog pokušaja. Nakon što lociraju i poprave greške, kod postaje bolji i bolji. Stoga, najbolji način da se unaprijedi razvoj softvera je da se naglasi učenje
3. **Odlučiti što kasnije moguće** – kasne odluke su informiranije jer se baziraju na činjenicama
4. **Dostaviti što prije moguće** – četvrto načelo se odnosi na prednosti brzog razvoja softvera. Kratki razvojni ciklusi omogućavaju developerima da saznaju više

dobivanjem povratnih informacija. Također omogućuju kupcu da odgodi donošenje konačne odluke dok ne dobije više informacija

5. **Osnažiti tim** – developeri bi trebali imati pravo da donose razne tehničke odluke jer znaju detalje posla više od bilo koga
6. **Graditi integritet/kvalitetu** – percepcija korisnika softvera i njegovih karakteristika mora se podudarati. Ako korisnik misli da softver ima sve potrebne značajke i jednostavan je za koristiti, taj softver ima integritet
7. **Gledati cjelokupni projekt** – inženjeri bi trebali preuzeti odgovornost za ukupnu učinkovitost sistema, umjesto da se usredotoče na svoj mali dio rada. Ako se stručnjaci drže ovog načela, mogu stvoriti sistem s integritetom

Ova načela opisuju vitku filozofiju, cilj joj je pružiti više vrijednosti za manje truda, ulaganja i vremena. Lean pristup je iterativni i inkrementalni okvir. Stoga se kao i u bilo kojem drugom agilnom pristupu proizvod dostavlja u inkrementima u ranim fazama razvoja. Daljnji razvoj ovisi o povratnim informacijama korisnika.

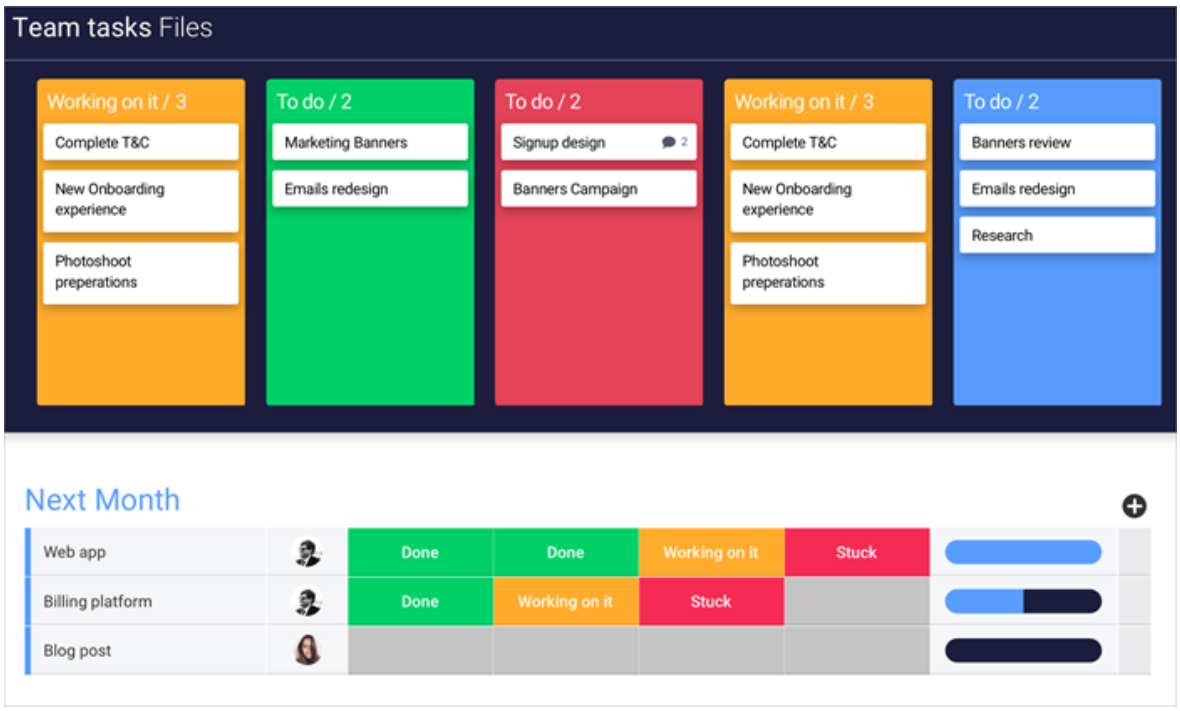
Vitko upravljanje projektima je usavršila Toyota. Dva temeljna fokusa su: isporuka vrijednosti kupcu i minimiziranje otpada, a voditelji nastoje uspješno isporučiti projekte ostajući vjerni tim vrijednostima.

Vitki agilni okvir ima 3 vrste otpada:

1. **Muda** – vrijeme i trud potrošeni na aktivnosti koje kupcu nemaju nikakvu vrijednost
2. **Mura** – neravnomjeran, nekonzistentan proces proizvodnje, što rezultira gubitkom vremena i resursa
3. **Muri** – prekomjerna upotreba strojeva ili zaposlenika

2.1.5. Alati za agilno upravljanje projektima

Monday (monday.com)



Slika 6: Monday

Značajke:

- Razvoj projekta može se pratiti putem kanbana, vremenske trake ili grafikona
- Funkcionalnosti za planiranje sprinteva, stvaranje korisničkih priča i dodjeljivanje članovima tima
- Izvještavanje

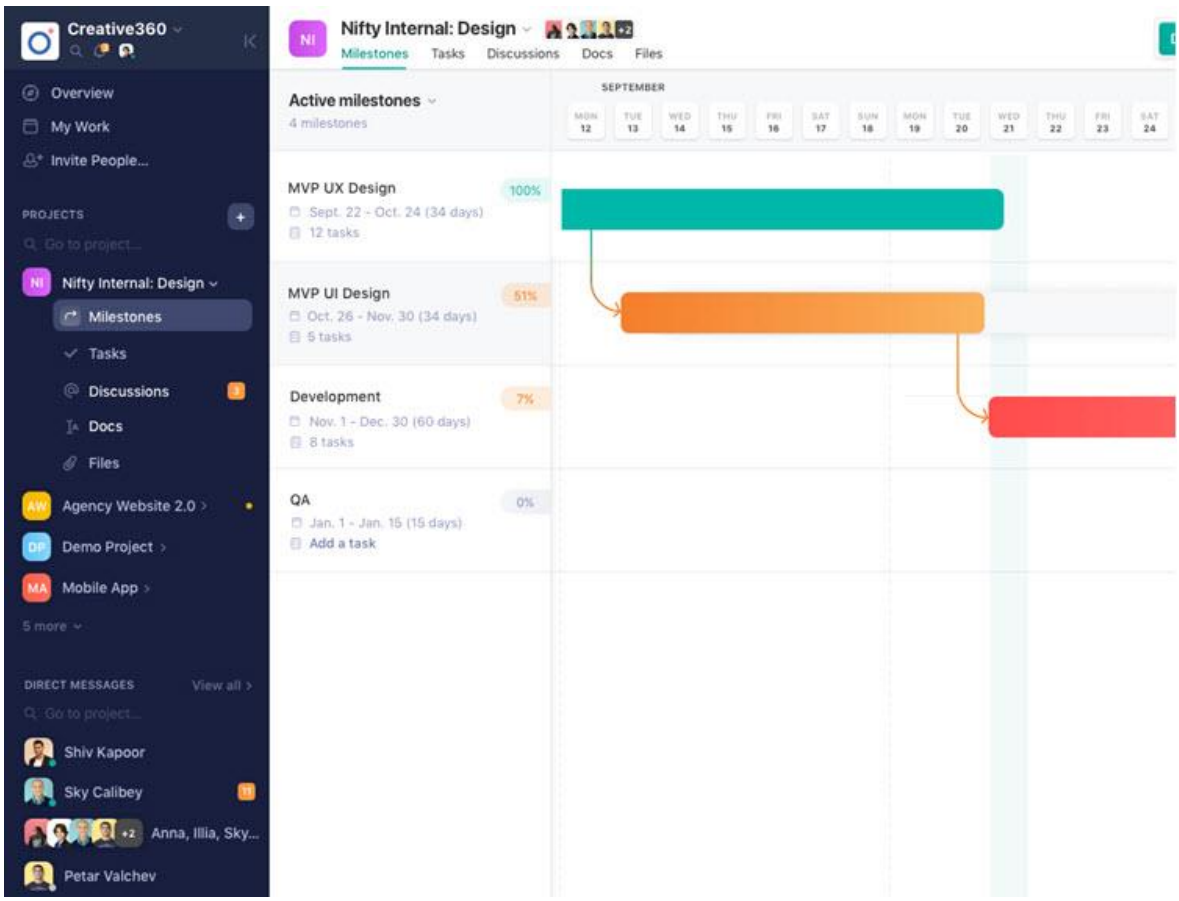
Prednosti:

- Pruža dobre značajke suradnje
- Intergracija sa drugim aplikacijama

Nedostatci:

- Cijena

Nifty (niftypm.com)



Slika 7: Nifty

Značajke:

- Projektima se može upravljati putem zadataka u stilu kanbana
- Pregled projekta se pruža iz ptičje perspektive
- Dokumenti se mogu kreirati izravno unutar svakog projekta

Prednosti:

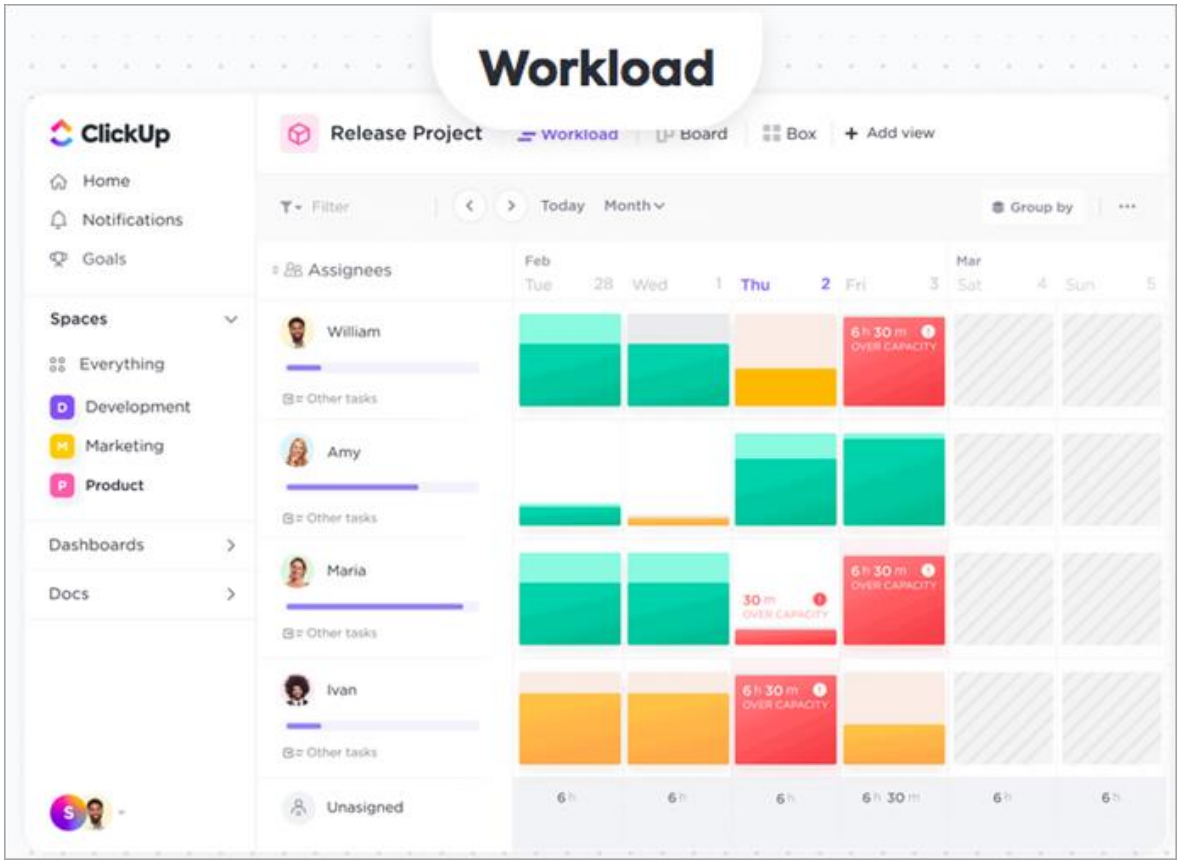
- Prekrasno sučelje

- Lakoća korištenja

Nedostatci:

- Ništa značajno inovativno

ClickUp



Slika 8: ClickUp

Značajke:

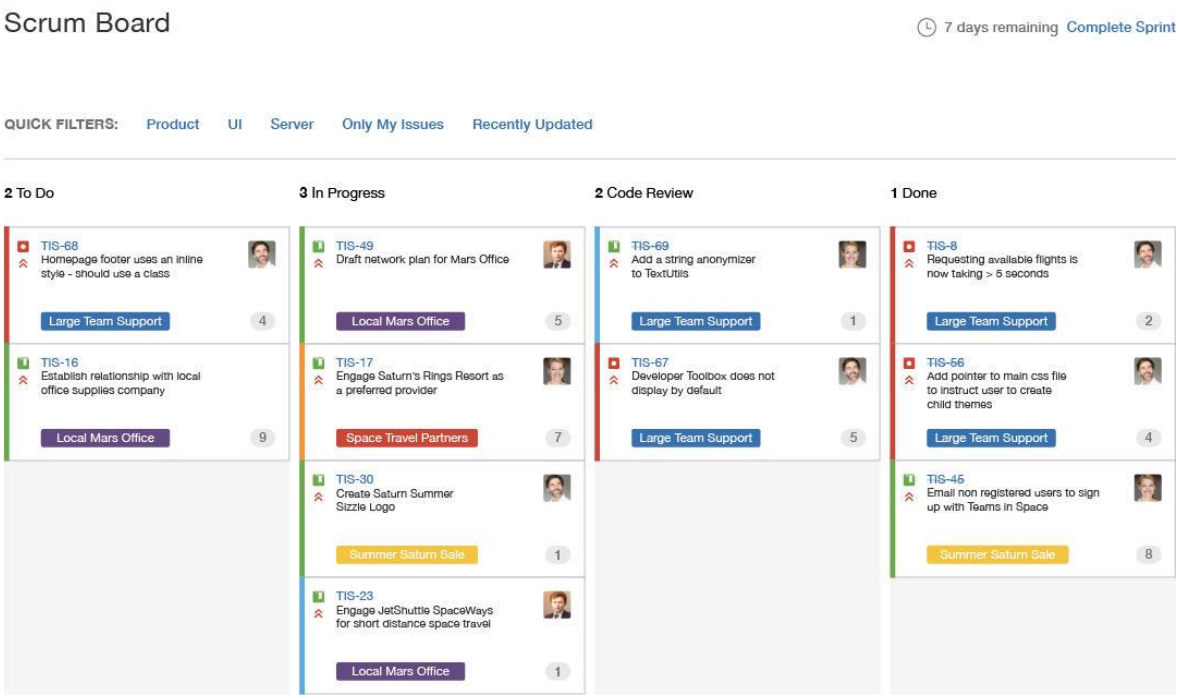
- Suradnja u stvarnom svijetu
- Nadzorna ploča
- Automatizacija

- Više prikaza uključujući prikaz radnog opterećenja

Prednosti:

- Drag and drop funkcija
- Napredni filteri, sortiranje i pretraživanje
- Predlošci
- Moguće upravljanje više projekata odjednom

JIRA



Slika 9: Jira

Atlassian Jira jedan je od najboljih alata za upravljanje projektima koje koriste agilni timovi.

Značajke:

- **Prilagodljive scrum ploče** koje se mogu prilagoditi tijeku rada vašeg tima. Ove scrum ploče koriste se za vizualizaciju cjelokupnog rada u sprintu. Svaki backlog se automatski premješta na novi sprint
- **Fleksibilne kanban ploče** za kontinuiranu isporuku maksimalnog učinka uz minimalni napor.
- **Prikaz sprinta u stvarnom vremenu** uz pomoć grafikona, kumulativnog dijagrama toka, grafikona brzine…
- Prilagođeni filtri

Itd.

Prednosti:

- Vrlo je prilagodljiv potrebama projekta
- Zreo i provjeren produkt koji koriste tisuće tvrtki diljem svijeta
- Jeftin za male timove

Nedostatci:

- Teško za postaviti i potrebno je dugo vremena da se nauči pravilno koristiti
- Mnoge osnovne značajke dostupne su samo kao plaćeni dodaci
- Previše značajki ga čini složenim za koristiti

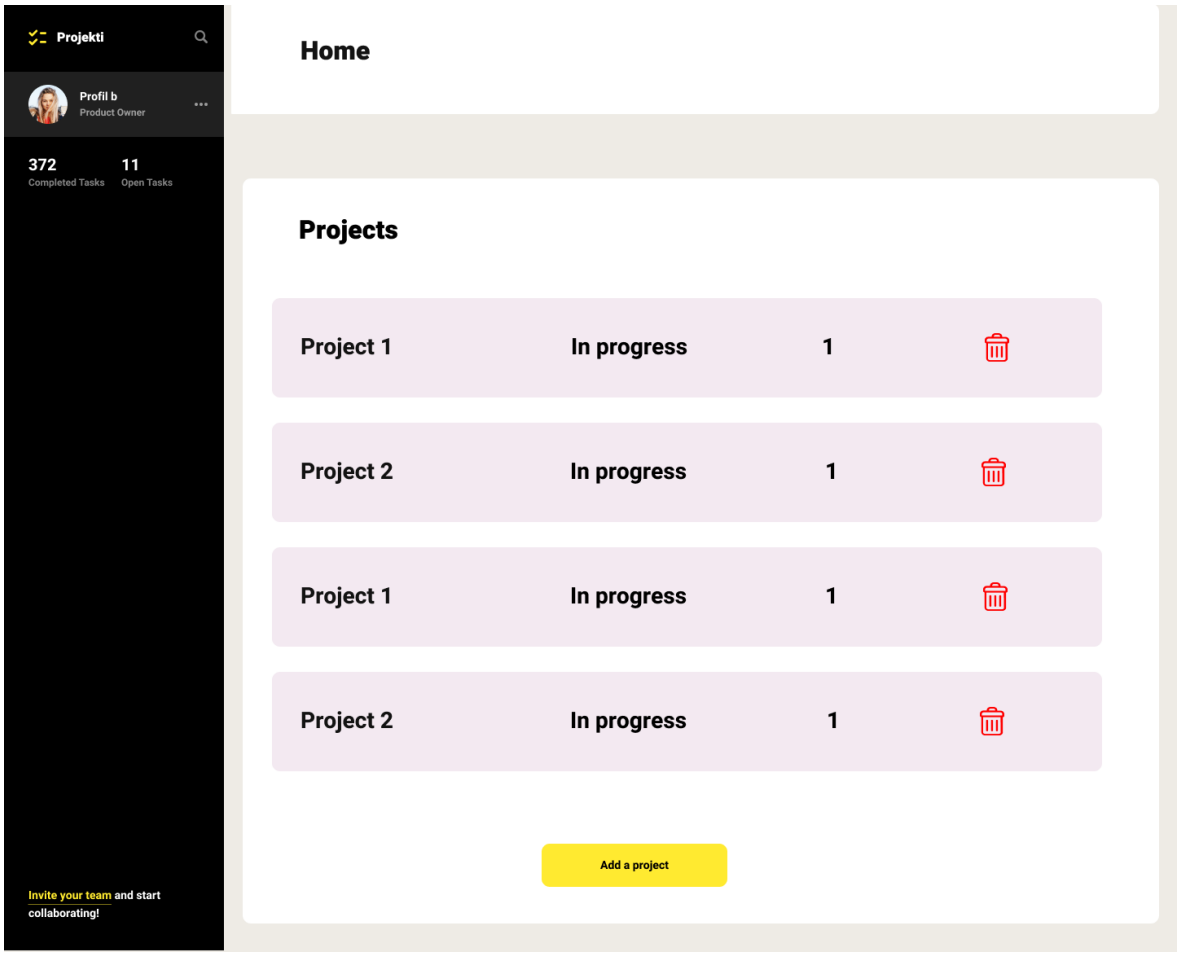
3. Praktični dio

U praktičnom dijelu rada napravljen je alat za pomoć pri agilnom vođenju projekta.

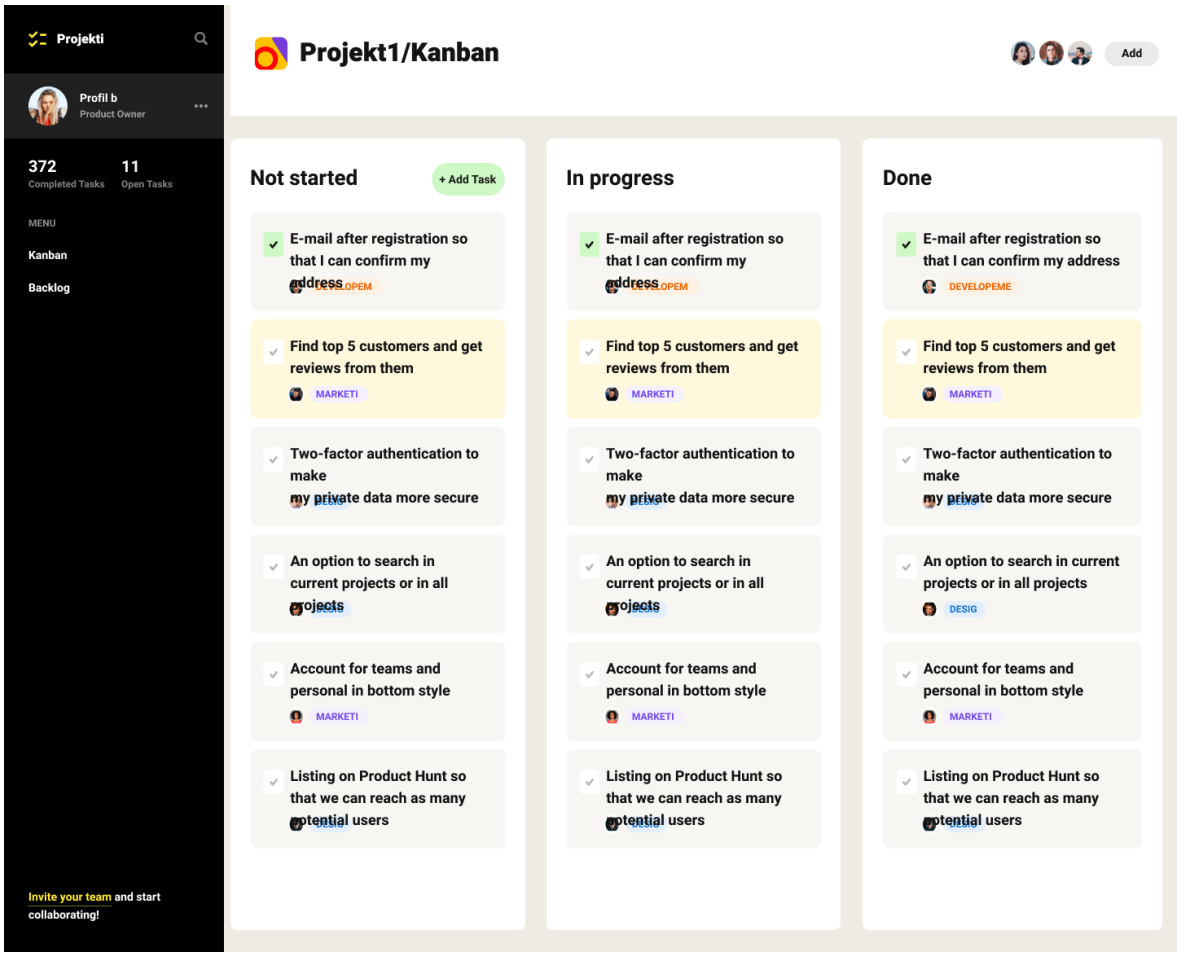
Cilj alata je demonstrirati različite mogućnosti koje smo naveli u prethodnom dijelu. Prije same implementacije bilo je potrebno definirati zahtjeve aplikacije koji su u skladu sa prethodno opisanim principima.

3.1. Dizajn

Prvi korak u realizaciji aplikacije je dizajn. Za inspiraciju su korištene razne stranice kao što su Pinterest i razni blogovi, također i već postojećim aplikacijama. Za stvaranje skice korišten je program Figma koji služi za izradu prototipa aplikacija kao i web stranica.



Slika 10: Dizajn "Projects" stranice



Slika 11: Dizajn "kanban" stranice

3.2. Zahtjevi aplikacije

Nakon što smo odlučili koje tehnologije koristiti pri izradi aplikacije došlo je vrijeme na implementaciju. Prije izrade svake aplikacije potrebno je prikupiti dovoljno informacija o zahtjevu na kojim će se temeljiti pojedina aplikacija. Svrha zahtjeva je odgovoriti na dva najvažnija pitanja “Tko će koristiti pojedini sustav” i “Koja će biti funkcija sustava”.

Zahtjevi se dijele u dvije vrste: korisnički zahtjevi i zahtjevi za razvojni tim.

Korisnički zahtjevi opisuju što bi rješenje trebalo raditi sa stajališta korisnika.

Zahtjevi softverskog sustava se često dijele i na:

1. **Funkcionalne zahtjeve** – izjave o uslugama koje bi sustav trebao pružiti, kako bi sustav trebao reagirati na određene ulaze i kako bi se sustav trebao ponašati u određenim situacijama. U nekim slučajevima, funkcionalni zahtjevi mogu također eksplicitno navesti što bi sustav trebao raditi

2. **Nefunkcionalne zahtjeve** – A

Zahtjevi:

1. Baza podataka – za čuvanje podataka o postojećim korisnicima/adminima, projektima i taskovima
2. Autentifikacija – mogućnost prijave kao korisnik i kao admin
3. Evidencija projekata – stvaranje i brisanje projekata
4. Dodavanje taskova – stvaranje novih i evidencija gotovih taskova
5. UI – intuitivno korištenje aplikacije
6. Drag and drop – premještanje taskova iz novih u trenutne/gotove

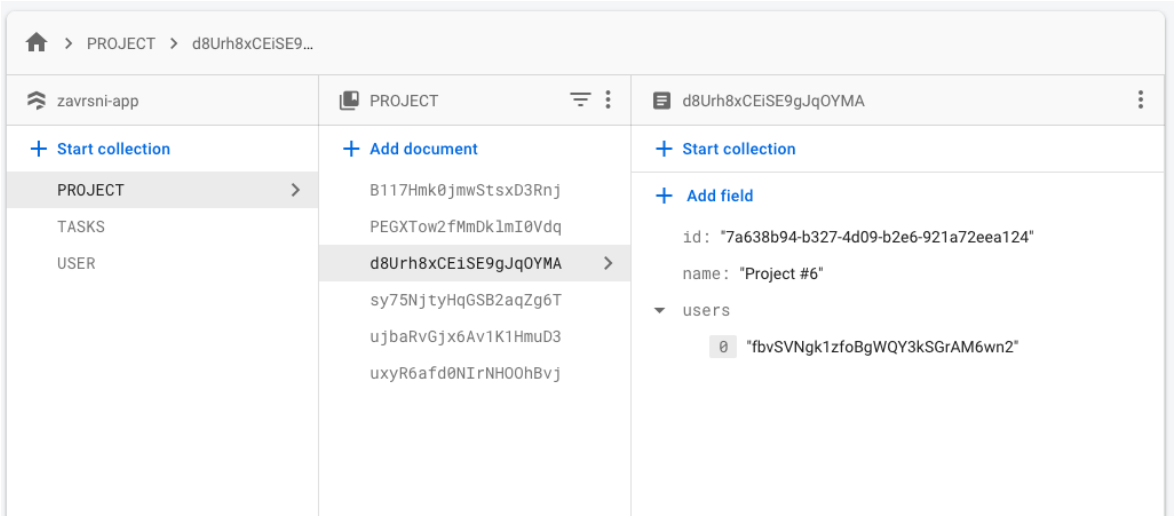
3.3. Implementacija koda

Nakon što su određeni zahtjevi aplikacije i njen dizajn, napisan je kod koji će ispuniti navedene zahtjeve.

3.3.1. Baza podataka

Prvi korak u implementaciji je postavljanje Firebase projekta i stvaranje Firestore Database (baze podataka).

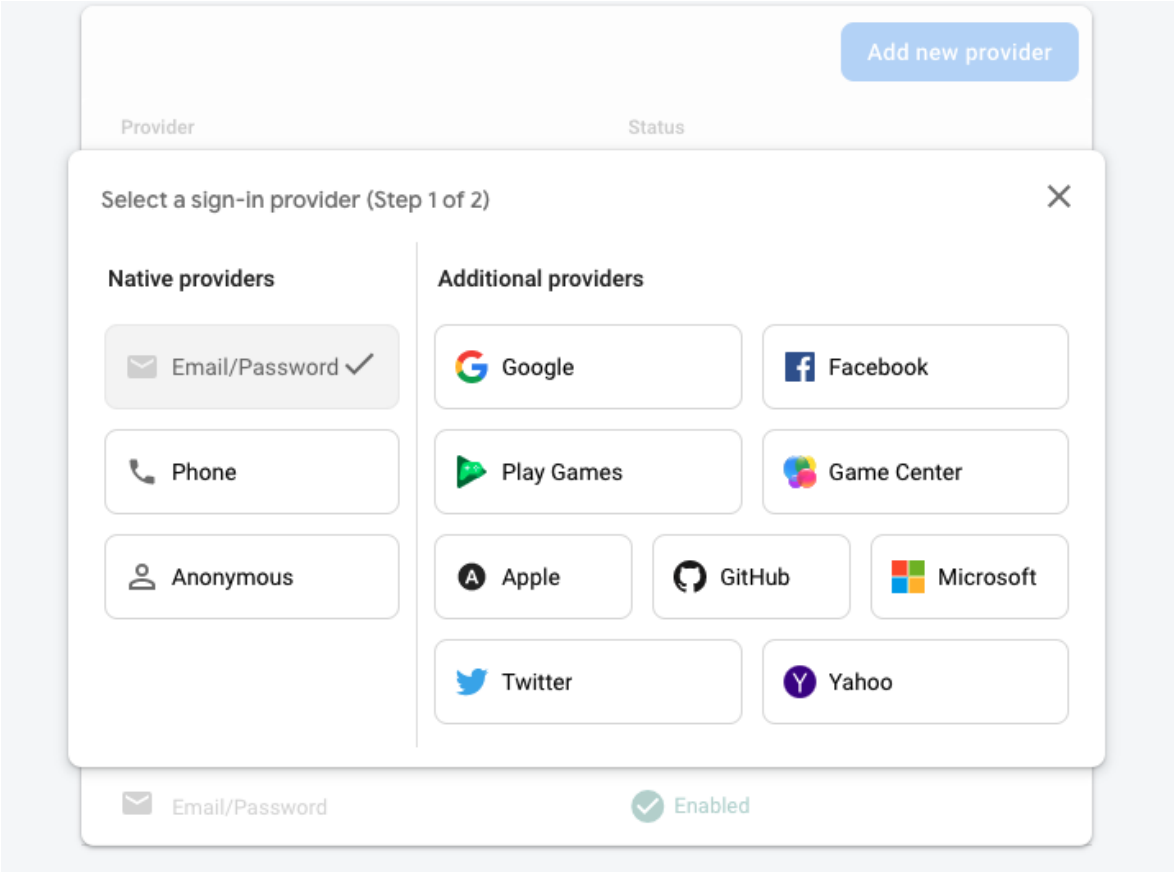
Idući korak je bio napraviti takozvane „Collections“ u Firestore-u za određene podatke. Sve se radi preko Firebase UI-a, stoga nikakav unos koda nije potreban.



Slika 12: Struktura naše baze podataka

3.3.2. Autentifikacija

Idući korak je bio napraviti autentifikaciju. To je također napravljeno preko Firebase-a pošto već ima ugrađenu autentifikaciju. Postoje razne mogućnosti prijava kao što su Google, Facebook, Apple, Github, itd. Za ovaj projekt je izabrana Email/Password prijava. Da bi je aktivirali, odemo na Authentication tab u meniju i kliknemo „Add new provider“ te odaberemo željenu opciju i odaberemo „enable“.



Slika 13: Dodavanje autentifikacije u Firebase-u

Nakon toga je napisana logika za sign up i log in, korišten je Context kako bi imali pristup trenutnom korisniku kroz cijelu aplikaciju.

U datoteci „AuthContext.js“ postavljeni su potrebni state-ovi.

```
const [currentUser, setCurrentUser] = useState();
const [isAuth, setIsAuth] = useState(false);
const [loginError, setLoginError] = useState("");
```

Nakon toga je napravljena funkcija za stvaranje novog korisnika.

```
const signup = async (email, password, username, redirectTo)
=> {
    try {
      const { user } = await createUserWithEmailAndPassword(
        auth,
```

```
        email,
        password
      );
      await setDoc(doc(db, "USER", user.uid), {
        id: user.uid,
        username,
        email,
        role: "none",
        isAdmin: false,
      });
      setCurrentUser({
        id: user.uid,
        username,
        email,
        role: "none",
        isAdmin: false,
      });
      setIsAuth(true);
      redirectTo();
    } catch (error) {
      console.log(error.message);
    }
  };
```

Firebase Authentication sprema korisnike u posebnu Authentication bazu podataka, no za naš zahtjev su potrebni i u Firestore bazi podataka sa nekim dodatnim svojstvima, stoga prilikom stvaranja novih korisnika spremamo ih i u jednu i u drugu bazu.

Nakon toga napravimo funkciju za prijavu i odjavu.

```
const login = async (email, password, redirectTo) => {
    try {
      const { user } = await signInWithEmailAndPassword(auth,
      email, password);
      setIsAuth(true);
      const tempRef = doc(db, "USER", user.uid);
      const tempSnap = await getDoc(tempRef);

      if (tempSnap.exists()) {
        setCurrentUser(tempSnap.data());
        redirectTo();
```

```
      } else {
        console.log("No user found");
      }
    } catch (error) {
      setLoginError(error.message);
    }
  };
```

```
const logout = () => {
    signOut(auth)
      .then(() => setIsAuth(false))
      .catch((err) => console.log("Error", err.message));
  };
```

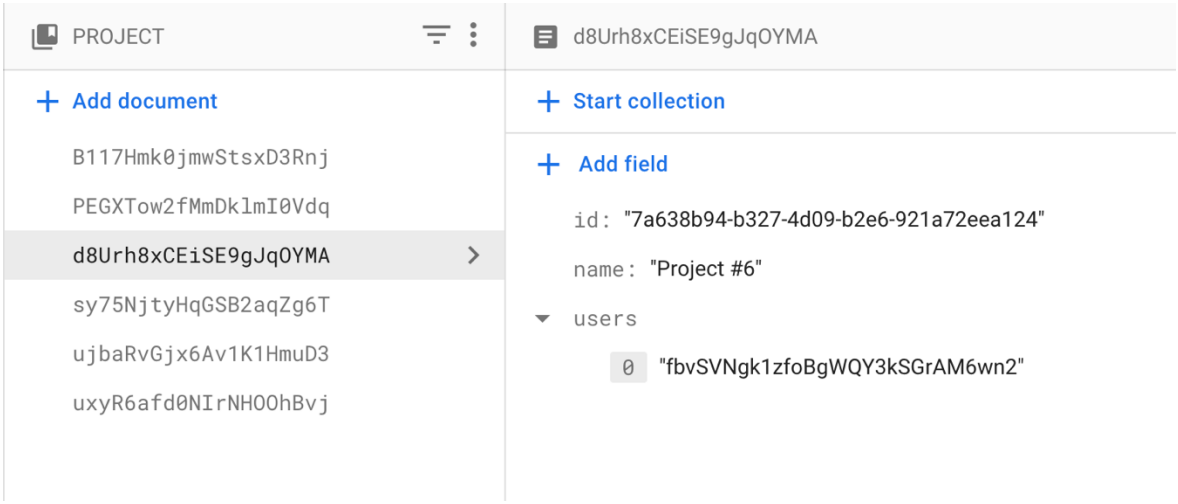
Funkcije potrebne da se ovo izvede su dohvaćene iz firebase/auth biblioteke.

```
import {
  createUserWithEmailAndPassword,
  signInWithEmailAndPassword,
  signOut,
  onAuthStateChanged,
} from "firebase/auth";
```

3.3.3. Evidencija projekata

Evidencija je također napravljena pomoći Firebase baze podataka. Napravljena je kolekcija u koju se dodaju projekti koji sadrže sljedeće podatke:

- **Id** – kako bi se razlikovali
- **Name** – svaki projekt ima svoje ime
- **Users** – listu usera koji imaju pristup tom projektu

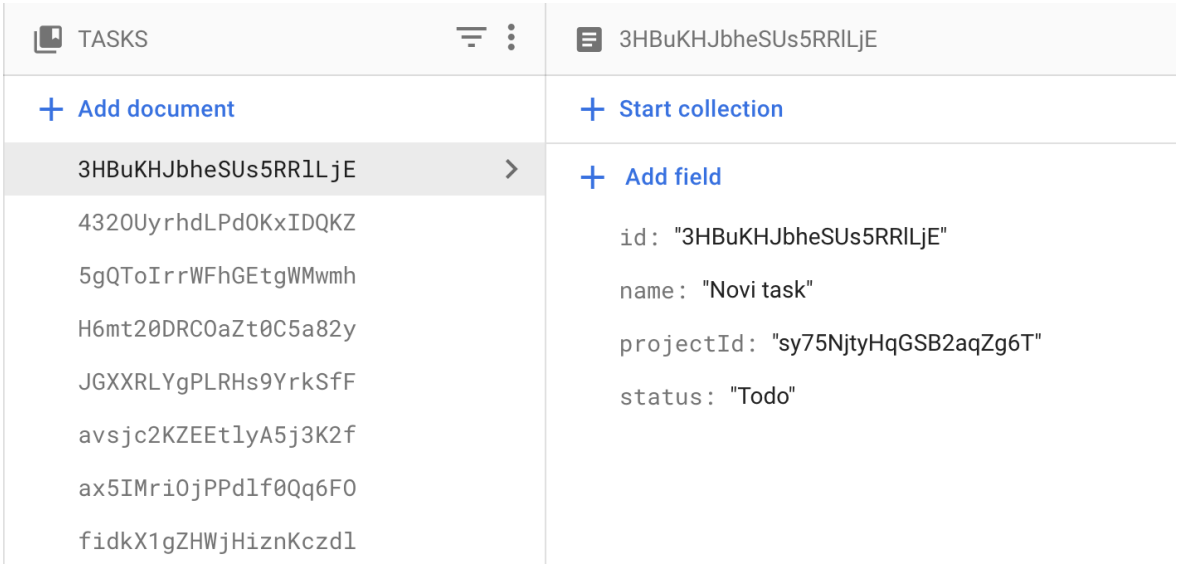


Slika 14: Projects kolekcija

3.3.4. Dodavanje i evidencija taskova

Svaki projekt ima taskove. Taskovi sadrže iduće podatke:

- **Id** – potrebno da bi se razlikovali
- **Name** – ime
- **projectId** – kako bi znali koji task prikazati u kanban stranici do koje dolazimo klikom na projekt
- **Status** – mijenja se prilikom drag and droppa u različite stupce kanban ploče

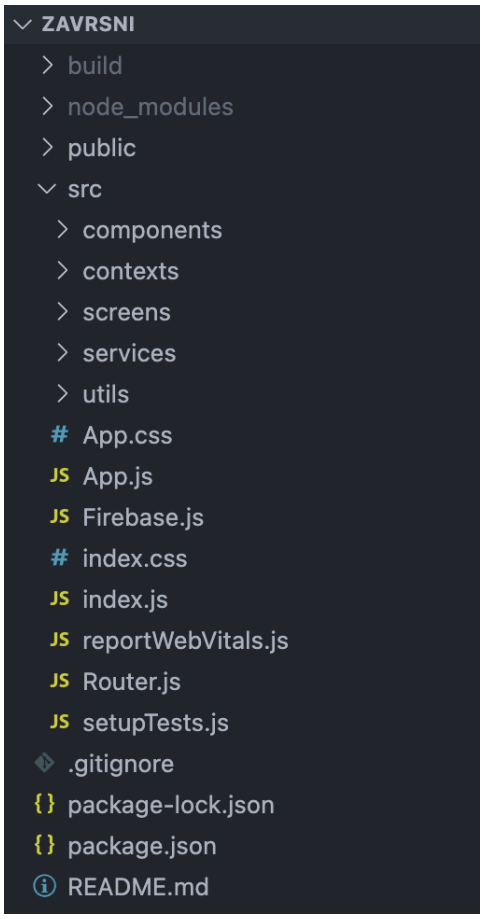


Slika 15: Task kolekcija

3.3.5. UI

Nakon toga je postavljen ReactJS projekt sa naredbom CRA (create react app) – okuženje za učenje React-a i najbolji način za početak izrade nove single-page aplikacije napravljen od strane Facebook (meta) developera.

Projekt je bio podijeljen po ovoj strukturi datoteka.



Slika 16: Folder struktura naše aplikacije

- **Components** – sadrži sve komponente korištene u projektu
- **Contexts** – sadrži sve kontekste
- **Screens** – sadrži sve stranice (eng. *page*) koji se prikazuju
- **Services** – sadrži sve API pozive
- **Utils** – sadrži sve pomoćne funkcije

Prvi korak je bio spojiti Firebase sa React aplikacijom, što se može vidjeti u datoteci pod imenom „Firebase.js“.

```
import { initializeApp } from "@firebase/app";
import { getFirestore } from "@firebase/firestore";
import { getAuth } from "firebase/auth";
```

```
// When we want our db info to be private
const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket:
process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId:
process.env.REACT_APP_FIREBASE_SENDER_ID,
  appId: process.env.REACT_APP_MESSAGING_APP_ID,
  measurementId:
process.env.REACT_APP_FIREBASE_MEAUSEREMENT_ID,
};

const app = initializeApp(firebaseConfig);

export const db = getFirestore(app);

export const auth = getAuth(app);
```

4. ZAKLJUČAK

Cilj ovog rada bio je upoznati se sa agilnim metodama projektnog menadžmenta i navesti njihove prednosti i mane.

Rad započinje definiranjem projektnog menadžmenta te projekta kao osnovne privremene jedinice koja ima svoj početak i kraj. Čitatelje se upoznaje s ulogom i kompetencijama voditelja projekta, osobe koju je organizacija koja provodi projekt imenovala radi postizanja projektnih ciljeva. Opisuje se tradicionalni model vođenja projekta na temelju sekvencijalnih ciklusa, gdje izlaz svake faze postaje ulaz u sljedeću fazu.

Tradicionalni model nije sklon i ne prihvaća promjene, a svaka promjena zahtjeva dodatan je posao procjene i reevaluacije cijelog projekta. Zbog tromosti i nesklonosti promjenama, tradicionalno vođeni projekti češće ne uspijevaju biti provedeni do kraja.

Primjenom Scruma u razvoju proizvoda u prikazanom hibridnom modelu omogućavaju se brze iteracije i preinake, što daje odgovor na brze promjene korisničkih zahtjeva te reakcije na poteze konkurenata.

Sam projektni model evoluira, ovisno o iskustvu članova tima i prilagođava se organizaciji u kojoj se koristi, stoga primarni model predstavlja samo početnu točku za daljnja unapređenja u ovisnosti o potrebama organizacije.

Tijekom rada na projektu u kojemu sudjeluje više od jedne osobe potreban je sustav za komunikaciju, praćenje i kontroliranje iste. Kako bi se komunikacija uspješno i pravovremeno odvijala potreban je određeni sustav. Za izradu takvog sustava korištene su navedene tehnologije.

Tijekom izrade rada naučio sam da je planiranje pola puta do ostvarivanja cilja jer ukoliko se pogriješi prilikom planiranja može značiti da smo planirali odmah neuspjeh

5. LITERATURA

[1] Beck, Kent, et al. "Manifesto for agile software development." (2001): 2006.

[2] Takeuchi, Hirotaka, and Ikujiro Nonaka. "The new new product development game." *Harvard business review* 64.1 (1986): 137-146.

[3] Sutherland, Jeff, and Ken Schwaber. "The scrum guide." *The definitive guide to scrum: The rules of the game. Scrum. org* 268 (2013).

[4] Poppendieck, Mary, and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley, 2003.

[5] Sommerville, Ian. "Software engineering 9th Edition." (2011).

[6] https://www.pmi.org/about/learn-about-pmi/what-is-project-management

[7] https://www.apm.org.uk/resources/what-is-project-management/

[8] https://www.teamgantt.com/guide-to-project-management/what-is-project-management

[9] https://www.apm.org.uk/resources/find-a-resource/agile-project-management/

[10] https://www.altexsoft.com/whitepapers/agile-project-management-best-practices-and-methodologies/

[11] https://www.servicenow.com/products/business-management/what-is-hybrid-project-management.html

6. SKRAĆENICE

MVP	*Minimal viable product*	minimalni održivi proizvod
OOPSLA	*Object-Oriented Programming, Systems, Languages & Applications*	objektno orijentirano programiranje, sistemi, jezici i aplikacije
WIP	*Work in progress*	rad u tijeku
CRA	*Create react app*	stvaranje početne react aplikacije