Analiza procesa razvoja i osiguranja održivosti informacijskih sustava

Vučković, Lovre

SVEUČILIŠTE U SPLITU

EKONOMSKI FAKULTET

DIPLOMSKI RAD

ANALIZA PROCESA RAZVOJA I OSIGURANJA ODRŽIVOSTI INFORMACIJSKIH SUSTAVA

Mentor:

izv.prof. dr. sc. Marko Hell

Student:

univ.bacc.oec. Lovre Vučković

Split, svibanj 2024.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, _____Lovre Vučković_____ ,
(ime i prezime)

izjavljujem i svojim potpisom potvrđujem da je navedeni rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja na objavljenu literaturu, što pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio navedenog rada nije napisan na nedozvoljeni način te da nijedan dio rada ne krši autorska prava. Izjavljujem, također, da nijedan dio rada nije korišten za bilo koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Split, _28.05.2024._ godine

Vlastoručni potpis : Lovre V.

SADRŽAJ:

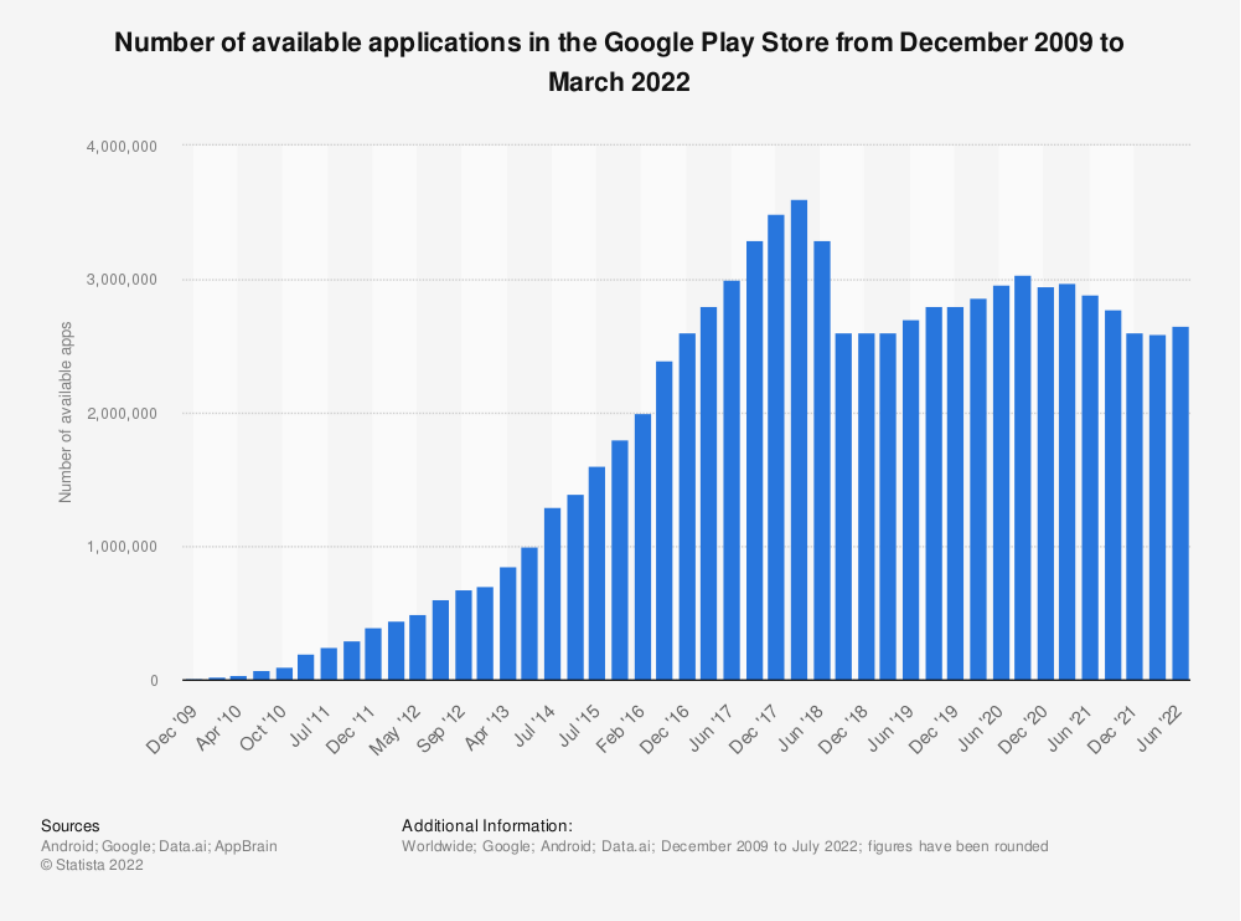
1. UVOD

1.1. Problem istraživanja

Tko je prosječan korisnik za kojeg se zapravo kreiraju aplikacije te koliko je profitabilna ideja segmentiranja tog dijela korisnika za upustiti se u kreiranje aplikacije za rješavanje njihove potrebe ili problema je prvo i glavno pitanje od kojeg kreće razvoj nekog softverskog alata, čija je funkcija na kraju pomoći korisnicima da obave određene zadatke učinkovitije i sa manje grešaka nego što bi to bilo kad bi se oni obavljali ručno (Baa, 2022). Svaka poslovna organizacija danas treba informacijski sustav za praćenje i podupiranje aktivnosti te su se tako ti sustavi razvili na svim poslovnim tržištima i industrijama (Nowduri & DBA, 2012). Suvremena računala danas i jesu nastala zahvaljujući inicijaciji i financiranju poslovnog svijeta i države još 1950-ih godina kao alat za pojednostavljivanje knjigovodstva i modernizaciju bankarstva radi smanjenja rizika poslovanja, u prvom redu ljudske greške (Barclays, 2019). Tako je samo u zadnjih 60 godina bankarstvo i ono što danas nazivamo računovodstvo revolucionarizirano do razine da govorimo o svijetu čiji je monetarni i financijski sustav baziran na digitalnom novcu i čija funkcionalnost kao temelju vrijednosti ovisi o povezanosti različitih informacijskih sustava i programa svih zemalja svijeta i povjerenja u takve sustave kako se novac bez pokrića ne bi kreirao tamo gdje ne smije (Simplified Science Publishing, 2022). Informacijski sustavi su tako kreirani idejama i potrebama vlada i velikih korporacija, no razvila se na takav način da se aplikacije kreiraju ne samo za velike korisnike sa specijalnim zahtjevima, već i za srednja i mala poduzeća i to u svakoj mogućoj industriji u kojima ona posluju, jer se sve jedan zadatak danas može optimizirati na bolji i brži način nego što je to manualno. Svako to poduzeće se sastoji od zasebnih jedinica i odjela, te se za svaki odjel unutar poduzeća u prosjeku koristi između 40 do 60 različitih aplikacija, dok se u cijeloj tvrtki u prosjeku koristi više od 200 aplikacija u istraživanju o analizi korištenja 30.000 različitih aplikacija između 190 tvrtki (Torres, 2019). Samo te brojke iz relativno malog uzorka u obziru na širinu industrija ukazuju na nebrojenu količinu trenutno dostupnih aplikacija te potrebe za razvijanjem novih aplikacija, ali jednako tako i osiguranja održivosti njihovog rada kako bi korisnici koji ih trenutno koriste bili osigurani od rizika gubitka podataka ili funkcionalnosti sustava što može dovesti do ogromnih gubitaka. Sve je više zahtjeva koji dodatno opterećuju resurse rastežući operativnu tehnologiju ostavljajući takve posljedice da jedna od pet organizacija danas doživi ozbiljan ili teški prekid rada godišnje što dovodi to značajnih financijskih gubitaka, narušavanja ugleda a i mogućeg gubitka života (Uptime Institute, 2022). Tako je u Americi 2019. godini 39% kvarova na infrastrukturi uzrokovalo financijske gubitke od 100.000 USD, dok je 2022.

taj postotak porastao na 60%, a trendovi sugeriraju da će svake godine biti najmanje 20 ozbiljnih IT ispada visokog profila diljem svijeta sa katastrofalnim troškovima (Uptime Institute, 2022). Koliko god sustav bio kvalitetan, situacija je danas takva da će u cijelom tom procesu sustav biti zagušen ili nekom infrastrukturom ili bilo kojom drugom poveznicom koja ga može ograničiti ili kompletno srušiti svojim učinkom u bilo kojem trenutku. E-visitor kao državni informacijski sustav za obveznu prijavu i odjavu turista koji funkcionalno povezuje sve turističke zajednice u Republici Hrvatskoj (eVisitor, 2022) ovdje se nameće kao dobar primjer sa svojim čestim rušenjima sustava po nekoliko sati gdje sve hotelijerske aplikacije i sustavi koji su povezani na E-visitor u potpunosti izgube mogućnosti prijavljivanja i odjavljivanja gostiju kao i nekih drugih funkcija kao dio programerskog koda aplikacije u koji su ugrađene. Sve aplikacije se tako kreiraju kroz druge aplikacije, te se za različite potrebe i kroz različite načine ili konektore spajaju na neke treće aplikacije i postaju ovisne o radu jedni drugih te se kroz tu sve raznolikiju i mnogobrojniju povezanost nameće problem upravljanja održivim razvojem i mogućnošću ažuriranja i održavanja aplikacija.

Osim poslovnih korisnika, u igri potrebe za informacijskim sustavima tu je još i sve jedan građanin planeta Zemlje život kojeg se sve više seli “u oblak” među svim ostalim podacima koji su mu potrebi za novo normalno odvijanje života. Čisti dokaz količini potreba koje aplikacije zadovoljavaju je činjenica da je broj aplikacija dostupnih samo na Google Play Store-u sredinom 2022. godine iznosio 2.6 milijuna u odnosu na samo deset godina ranije kad je ta brojka bila 0.5 milijuna te da je porast višerostruk u toliko relativno kratkom razdoblju unatoč vidljivom smanjenju za 1.1 milijuna aplikacija u razdoblju od pola godine u 2018. kad je Google odlučio obrisati trećinu svih aplikacija sa platforme radi kršenja pravila privatnosti Google Play Store-a (Ahmed, 2022) što se također može smatrati kao jedan od koraka pri kreiranju održivog informacijskog sustava cjelokupne zajednice. Slika ispod tako prikazuje kretanje ukupnog broja aplikacija na Play Store platformi kroz godine:



Slika 1 - Broj dostupnih aplikacija na Google Play Store-u kroz razdoblje 2009.-2022.

Izvor: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

Iz samo zadnjih nekoliko prikazanih izvora kvantitativnih podataka može se zaključiti kako je tržište informacijskih sustava brzo rastuće, te da je potreba za razvojnim timovima takvih sustava sve veća. Barijere ulaska su niske, a troškovi mogu biti i nepostojeći pošto sve što je potrebno za razviti aplikaciju danas je ideja i osobno računalo, što svatko tko ima znanje za razviti jedan takav proizvod već posjeduje. Računalna igra Stardew Valley je jedan od najboljih primjera takve prakse koja je u potpunosti razvijena od strane samo jedne osobe - Erica Baronea, čiji je trud urodio takvim plodom da je do 2022. godine prodano 20 milijuna kopija (Hagues, 2022) igre čija pojedinačna cijena kopije iznosi 15 USD.

Iako je kroz primjer Erica Baronea dokazano da je samostalno moguće razviti uspješnu aplikaciju, to je i dalje ekstrem, te je za većinu aplikacija općenito potrebno zaposliti suradnike ili biti dio tima gdje svatko obavlja dio posla za koji je specijaliziran, a posebno za slučajeve u kojima se razvijaju informacijski sustavi za poslovne korisnike. Tu se dolazi do idućeg problema rada – kako zapravo inicirati razvoj aplikacije i kako proces izrade aplikacije od njezine ideje do razvitka izgleda, kako postati i opstati kompetetivan u bilo kojem polju razvoja informacijskih sustava kad za svaku funkciju danas već postoji barem dovoljno dobra alternativa na tržištu te kako se to sve zapravo preslikava i izgleda na jednom stvarnom primjeru poduzeća.

1.2. Predmet istraživanja

Predmet istraživanja su procesi koji dovode do ostvarivanja cilja razvijanja aplikacije i održavanja odnosa sa korisnicima sustava te zadržavanja njihovog zadovoljstva i lojalnosti nastavka korištenja softvera koje poduzeće nudi. To je danas veoma bitno pošto je stanje na tržištu takvo da se sve više pružatelja softvera u svim industrijama okreće modelu naplate usluge na temelju pretplate ili model sponzoriran oglasima na sustav umjesto jednokratnog naplaćivanja kupnje usluge što može generirati više prihoda tijekom životnog ciklusa proizvoda nego vezanje na bilo koji drugi poslovni model(Huotari & Ritala, 2021). Taj prijelazak postaje toliko profitabilan i ekstreman za pružatelje usluga da se zadnjih par godina sve više raspravlja o automobilskoj industriji koja od vozila kao fizičkog proizvoda pokušava transformirati neke funkcije kao aplikacijske usluge koje može prodavati kroz mjesečni pretplatni model za funkcije poput daljinskog pokretanja auta, grijanih sjedala i slično (Levin, 2019).

Predmet je tako zapravo cijeli pregled razvoja i održavanja jednog takvog projekta sa naglaskom na direktno opisivanje stvarnog procesa koji su dio razvoja softvera, te samim time uloge članova tima, njihove međusobne dinamike te načina komunikacije između njih, alata i metodologija koji se koriste u te svrhe te definiranje odgovornosti i zadataka koja pada na pojedinačnog člana tima, načini planiranja i dokumentiranja takvog projekta, te klijenti kao krajnji korisnici te ideja o korisniku orijentiranog pristupa.

1.3. Ciljevi istraživanja

Cilj rada je istražiti metode upravljanja procesima razvoja i održavanja softvera u IT svijetu, ispitati njihovu efikasnost i usporediti ih sa stvarnim primjerom primjene u jednom poduzeću. Tako je cilj i prikazati metodologiju kojoj se prilagodilo poduzeće iz primjera, kakve ona rezultate pruža te koje potencijalne promjene bi mogle unaprijediti te procese stavljajući u odnos trošak implementacije i kompleksnost prijelaska na nove procese. Također, cilj je istražiti koje promjene procesa bi mogle postati neizbježne rastom navedenog poduzeća i promjenom stanja na tržištu, dijeljene infrastrukture, razvojem novih alata za upravljanje procesima i unaprjeđenjem postojeće tehnologije, metodologije i zakona za potrebe međusobno ovisnog rasta.

1.4. Istraživačka pitanja

Prema opisu predmeta i problema istraživanja kao i postavljenih ciljeva, istraživačka pitanja na koja će se tražiti odgovor su iduća:

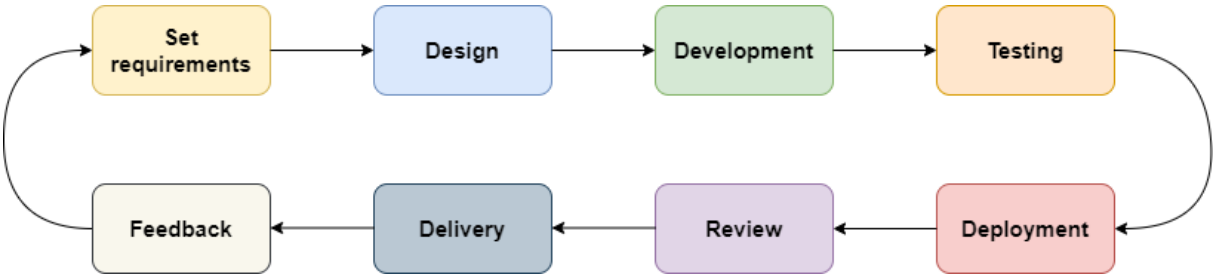
- Kako izgleda proces razvoja i ažuriranja aplikacija, tko su sudionici u tim procesima, koje su njihove funkcije, zadaci, odgovornosti i važnost u procesu te koja je važnost održavanja softvera nakon što je kreiran?
- Koje metode razvoja i ažuriranja aplikacija su najpoznatije te koje od njih pružaju najbolje rezultate?
- Koja je važnost održivosti informacijskih sustava u budućnosti, kako se implementira te iz kojih razloga se mora osigurati?
- Kako u praksi na empirijskom primjeru poduzeća izgleda, prati li neki od postojećih modela, odnosno kojem modelu je najsličniji te koliko dobro funkcionira taj model razvoja i ažuriranja softvera?

1.5. Metode istraživanja

Cilj rada će biti ostvaren kroz iduće metode istraživanja:

- **Metoda analize i sinteze** – istraživanjem različitih znanstvenih članaka i primjera IT poduzeća iz prakse izvući će se sastavni elementi predmeta i ideja obuhvaćenih njihovim istraživanjem i kompilirati u složeniju cjelinu svih glavnih zaključaka za potrebe predmeta i cilja ovog rada.
- **Metoda indukcije** – istraživanjem nekoliko pojmova ili poduzeća koji posluju u sličnim uvjetima ili koriste procese po istom principu pokušati će se dokazati ideja kako pojedini slučaj zapravo vrijedi kao općeniti zaključak, te za slučajeve koji su općenito zaključeni kao činjenica će se dokazati kako oni vrijede i za pojedine slučajeve, poduzeća ili timove.
- **Metoda deskripcije** – za potrebe opisa pojava, podataka, skraćenica i posebnih naziva u IT svijetu koristiti će se metoda deskripcije kako bi opisala njihova značenja, veze i međusobni odnosi.
- **Metoda komparacije** – kako bi rezultati dobili dodatnu dimenziju koristiti će se metoda komparacije radi boljeg uvida u razlike upravljanja procesima koje različita poduzeća ili timovi mogu imati te različite rezultate koje time mogu dobiti (Čendo Metzinger & Toth, 2020).

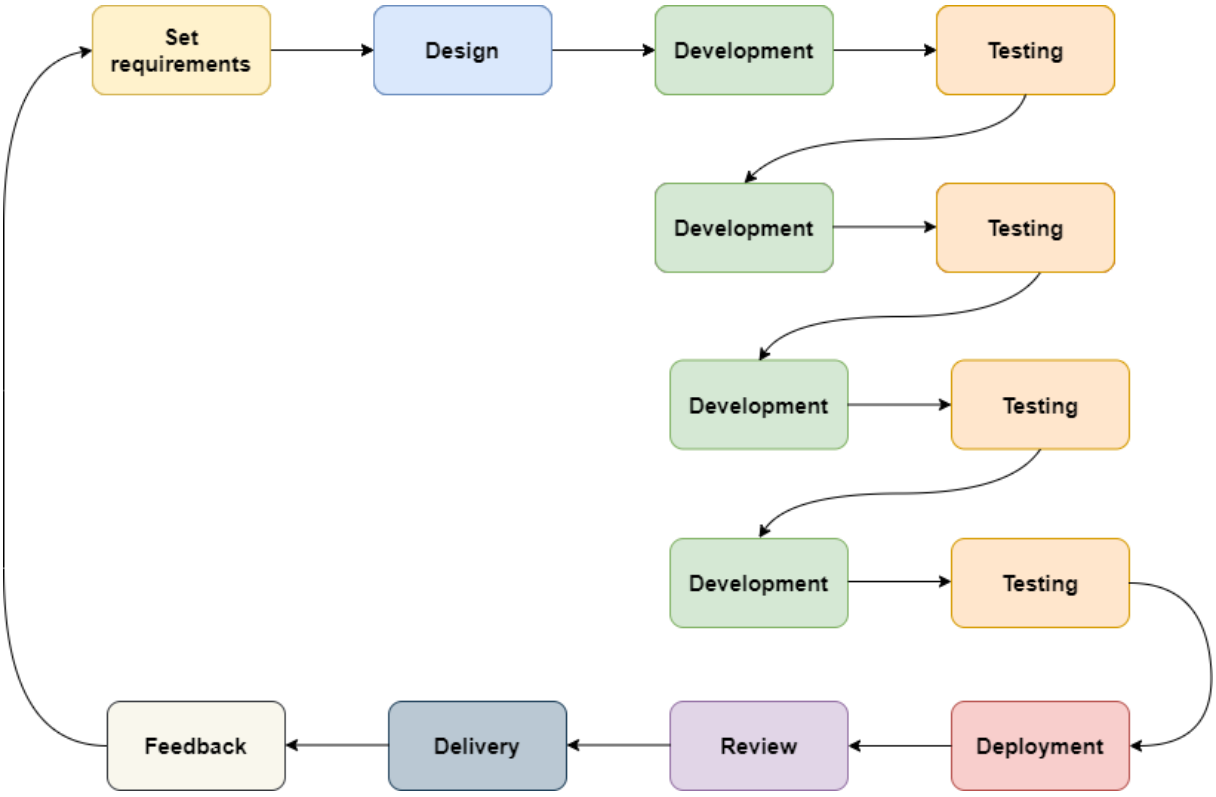
Pored navedenih metoda, koristiti će se još i metoda vizualizacije radi komuniciranja složenih informacija na jednostavne načine kako bi se objasnili procesi, usporedile promjene i uspostavili odnosi između podataka za lakše razumijevanje problema (Simplified Science Publishing, 2022) kroz grafikone i dijagrame poput onog iz slike 2 koji daje osnovni prikaz agilnog pristupa kao jednog od metodologija koji pomaže upravljanju projektima i razvoju softvera sa svojim ključnim procesima životnog ciklusa razvoja softvera navedenim na slici počevši od definiranja zahtjeva, pa prilagođavanje dizajna, razvoj softvera u skladu sa odobrenim zahtjevima, testiranje funkcionalnosti, implementacija proizvoda i pripremanje konačne verzije, pregled spremnosti proizvoda, isporuka rješenja te prikupljanje povratnih informacija (Altynpara & Rovnaya, 2022).



Slika 2 - Agilni proces razvoja softvera

Izvor: Izrada autora, prema https://www.cleveroad.com/blog/agile-software-development/

Prema osobnom iskustvu sudjelovanja u većini navedenih faza u razvoju softvera kroz različite projekte, ako se pri analizi stvarnih slučajeva uzmu u obzir samo faze razvoja i testiranja taj proces poprimi već složeniji oblik kao na slici 3, a kombiniranjem još preostalih šest faza sa dodatnim sudionicima u projektu taj prikaz postane mnogo kompliciraniji, što i predstavlja problem koji će se kroz rad na ovakav način pokušati prikazati i objasniti.



Slika 3 - Modificirani agilni proces razvoja softvera

Izvor: Izrada autora, prema https://www.cleveroad.com/blog/agile-software-development/

1.6. Doprinos istraživanja

Namjera doprinosa ovog istraživanja je u kompiliranju različitih istraživanja i praksi o razvoju informacijskih sustava kroz sve njihove faze te procese koji to omogućavaju zajedno sa pregledom arhitekture sustava, prikaz cjelokupne ideje upravljanja jednim takvim projektom te usporedba sa primjerom poduzeća iz prakse kako bi se uočile stvarne primjene te obrasci ponašanja u obavljanju procesa prilagođenih zasebnim potrebama te pregledom odnosa između članova tima zaduženih za ostvarenje projekta razvijanja te ažuriranja i održavanja aplikacija. Na taj način plan je analitički sagledati cijelu metodologiju kroz teoriju i praksu i ponuditi nove uvide u moguća poboljšanja i zamjene procesa ne samo za poduzeće iz primjera nego za općenito za poduzeća ili osobne projekte koji se bave razvojem softvera sa sličnim problemima i strukturom.

1.7. Struktura diplomskog rada

Rad će biti strukturiran na način da će se u uvodu navesti problem i predmet istraživanja, postaviti ciljevi koji se planiraju ostvariti te istraživačke hipoteze čiji će rezultat na kraju uvjetovati doprinos rada koji on donosi.

U drugom dijelu rada će se prikazati proces razvoja aplikacija, opisati najpoznatije metode za tu svrhu, objasniti značaj i funkcije svakog člana tima koji obavlja određeni zadatak u tom procesu, navesti najčešće i najznačajnije probleme prilikom odrađivanja takvog projekta te prikazati sličnosti i razlike pri upravljanju razvojem različitih vrsta informacijskih sustava.

Treći dio govoriti će o važnosti održavanja informacijskih sustava zbog međusobne ovisnosti, usporediti kad je trenutak kada neki softver prijeđe iz faze razvoja u fazu ažuriranja i održavanja te što taj pojam podrazumijeva, koja je važnost, koje su posljedice nekvalitetnog održavanja informacijskih sustava te tko ili što osigurava da se održivost ispoštuje.

Četvrti dio rada je empirijsko istraživanje gdje će se na poduzeću Hotel's Touch (STATIM d.o.o.) prikazati kako razvoj i održavanje aplikacije izgleda na pravom primjeru, koje probleme ima, koje su im posljedice te kako na njih odgovara navedeno poduzeće.

U posljednjem dijelu rada će se na osnovu teorijskog i empirijskog istraživanja donesti zaključak o spoznajama nastalim kroz ovaj rad zajedno sa sažetkom na Hrvatskom i Engleskom jeziku, uz navode literature koja je to omogućila.

2. RAZVOJ INFORMACIJSKIH SUSTAVA

2.1 Uvod

Projekt menadžment može se definirati kao niz aktivnosti i procesa koje provodi određena skupina ljudi, iz istih ili različitih područja, s ciljem stvaranja novih ili poboljšanih organizacijskih proizvoda, usluga i/ili procesa, a ideja "brže, jeftinije" je i dalje moto za postizanje organizacijskih ciljeva putem projekata (Cleland & Ireland, 2002). Tradicionalni projekti općenito su dobro definirani, s jasnim granicama, opsegom, ulogama i odgovornostima, usklađeni s ciljevima korporacije, lako se prate i upravljaju, imaju jasan raspored i rokove, te niske rizike sukoba. Međutim, tradicionalni projekti su linearni, rigidni, nefleksibilni te slabo prilagodljivi na promjene. Nasuprot tomu, informatički (IT) projekti često su složeni s nejasnim ciljevima i rješenjima koja nisu poznata na početku projekta. IT projekti danas se smatraju pokretačima promjena te olakšavateljima poslovnih procesa i usluga u organizacijama, no kako postoji snažan fokus na procjenu uspjeha, tako isto postoji fokus na neuspjehe IT projekata što je značajno iz nekoliko razloga. Najvažnije je zato što postoji mnogo neuspjeha što se može smatrati idejom da kreirani sustav ne ispunjava unaprijed definirane ciljeve ili ako projekt ne ispunjava vremenske, proračunske ili specifikacijske uvjete, ili ako su korisnici nezadovoljni novim sustavom (Jrad & Sundaram, 2015). Glavni uzroci neuspjeha u razvoju web-aplikacija u malim i srednjim poduzećima (engl. Small and medium enterprises - SME) su loš i minimalno razrađen dizajn, neorganizirani razvojni procesi i loše upravljanje razvojnim naporima. Potrebno je slijediti sustavni razvojni proces za izgradnju robusnih web-aplikacija, a izgradnja i održavanje web-aplikacija zahtijevaju podršku poput one koja je dostupna za tradicionalne softverske aplikacije kroz modele procesa životnog ciklusa. SME se kontinuirano prilagođavaju promjenjivim okruženjima i koriste kraći razvojni životni ciklus kako bi brže izbacili proizvod na tržište. Osim toga, mala poduzeća često se mijenjaju kako razvoj napreduje, tako da je često nerealno očekivati linearnu putanju do konačnog proizvoda. SME također imaju ograničene resurse za razvoj u odnosu na velika poduzeća, zajedno sa sredstvima za financiranje sporednih aspekata razvoja (poput evaluacije razvojnih procesa i upravljanja) ili istraživačkih i razvojnih (R&D) timova, te tako imaju nepotpune informacije o tržištu na kojem posluju te su tako osjetljiviji na rizike. Svi ti čimbenici mogu pridonijeti manje sveobuhvatnom modelu procesa životnog ciklusa razvoja web-aplikacija u SME, no uzimajući u obzir strukturu i ograničenja, SME bi trebali biti prvi koji će slijediti sustavni proces životnog ciklusa razvoja (Huang i ostali, 2010).

2.2 Životni ciklus razvoja softvera

Životni ciklus razvoja proizvoda (engl. System Development Life Cycle - SDLC) je ništa drugo nego opis cjelokupnog područja razvoja procesa kroz određene faze, pri čemu svaka faza opisuje svoje ciljeve i zadatke koje mora zadovoljiti kako bi prešla u drugu fazu. Tako uključuje detaljan plan kako razviti, izmijeniti i održavati softverski sustav. Životni ciklus razvoja sustava pruža skup aktivnosti koje se trebaju provesti tijekom razvoja sustava pristupom korak po korak za dovršavanje procesa razvoja (Gurung i ostali, 2020). SDLC prati šest glavnih razvojnih faza za kreiranje proizvoda koji je vrhunski u smislu kvalitete, vremenske učinkovitosti i isplativosti (Clark, 2024):

- **Planiranje i analiza** - prikupljanje poslovnih zahtjeva od klijenata i procjena izvedivosti, odnos potencijala troškova i prihoda te stvaranje razvojnog plana i ciljeva zajedno sa određivanjem vremenskih rokova.
- **Definicija zahtjeva** - prikupljanje specifikacije proizvoda kao jasne upute za razvojni tim te postavljanje i sastavljanje kvalitetne dokumentacije za taj cijeli proces.
- **Dizajn** - izvorni plan i vizija se razrađuju kao razvoj prototipa modela sustava sa izgledom i funkcijom pristupa i korištenja sustava.
- **Razvoj i testiranje** - stvarno kreiranje softvera kroz rješavanje zadanih zadataka kroz prikupljene zahtjeve. Programerski tim kreira funkcije, a testni tim potvrđuje njihovu upotrebljivost kroz testiranje performansi, funkcija i sigurnosti te po potrebi vraća neidealno napravljene zahtjeve programerskom timu za ponovne prepravke i razvoj.
- **Isporuka i pregled** - ovdje se radi o implementaciji konačnog proizvoda ili nove funkcije krajnjem korisniku, osiguravanje da sve radi kako treba te informiranje korisnika o novom proizvodu ili funkciji.
- **Održavanje** - faza u kojoj korisnici mogu pronaći bugove ili propuštene funkcionalnosti u ranijoj fazi testiranja koje moraju biti ispravljene radi dobrog korisničkog iskustva. U nekim situacijama to može dovesti vraćanje na prvi korak životnog ciklusa proizvoda (Clark, 2024).

Modeli SDLC konceptualno predstavljaju SDLC na organiziran način kako bi pomogli organizacijama da ih implementiraju, od kojih je najvažnije spomenuti četiri glavna modela:

- **Vodopadni** - linearno provođenje svih navedenih faza sekvencijalno, tako da svaka nova faza ovisi o ishodu prethodne, odnosno da teče iz jedne u drugu. Daje malo prostora za promjene nakon što se faza smatra završenom.
- **Iterativni** - sugerira da timovi započnu razvoj sa malim skupom zahtjeva te ga iterativno poboljšavaju novim verzijama dok kompletan softver ne bude spreman.
- **Spiralni** - kombinira male ponovljene cikluse iterativnog modela sa vodopadnim tako da nakon svake odrade svih faza klijent odradi evaluaciju napravljenog, te se vraća u ponovno odrađivanje cijelog tog ciklusa.
- **Agilni** - raspoređuje SDLC faze u nekoliko razvojnih ciklusa isporučujući male, inkrementalne promjene softvera po ciklusu te kontinuirano procjenjuju nove zahtjeve, prioritete, planove i rezultate kako bi mogli brzo reagirati na promjene, što ga čini iterativnim i inkrementalnim, i tako najučinkovitijim od svih modela procesa (AWS Amazon, 2024).

Definiranje agilnosti ostaje jedan od najvažnijih problema istraživanja definicije agilnog pristupa. Univerzalno razumijevanje onoga što čini agilnost nije definirano. U jednom istraživanju pregleda 482 rada (Hummel, 2014) objavljenima u najpopularnijim izvorima o agilnoj metodologiji većina radova i dalje se oslanja na Agilni manifest koji se sastoji od neprovjerenih načela i praksi praktičara koje nisu prikladne kao čvrsti teoretski okvir. Tako se agilnu metodologiju može staviti u odnos sa tradicionalnim metodologijama da se pruži generalna ideja što bi trebala predstavljati. Tradicionalne metodologije pretpostavljaju da kupci trebaju vodstvo od strane programera u vezi s zahtjevima. Najveća pretpostavka je da programeri zapravo znaju i razumiju ne samo ono što kupci trebaju na početku projekta, već i ono što će kupci trebati u budućnosti. Budući da se proces dizajniranja provodi odjednom i rezultat je gotov proizvod, programeri na kraju previše dizajniraju sustav, uključujući funkcionalnosti koje kupac često ne treba (Ionel, 2009). Agilne metodologije ne pretpostavljaju da programer zna više od kupca o njihovim potrebama. Također ne pretpostavljaju da kupac zna svoje buduće potrebe. One pretpostavljaju da niti kupci niti programeri u potpunosti ne znaju što bi zahtjevi trebali biti. Jednostavniji proizvod brzo se dovršava, a dodatna funkcionalnost dodaje se sa svakom sljedećom verzijom. Takav pristup omogućuje buduće promjene zahtjeva. Fleksibilnost ugrađena u agilne metodologije može upravljati promjenama u troškovima, opsegu i kvaliteti softvera na temelju potreba kupaca kako trenutno, tako i sa mijenjanjem postojećih i otkrivanjem novih potreba (Trumbach i ostali,

2022). Tehnike upravljanja projektima trebale bi povećati šanse za uspjeh projekta. Agilna metodologija upravljanja projektima je metoda koja je usmjerena na suradnju i učestalu komunikaciju koja je baš razvijena iz potrebe za smanjenjem šanse neuspjeha projekta kroz koncept intenzivne komunikacije između kupca/korisnika i projektnog tima (Trumbach i ostali, 2022). Tako u tradicionalnim okruženjima razvoja softvera programeri žele detaljne specifikacije, dok u agilnim metodologijama kupci i softverski inženjeri uče zajedno o zahtjevima sustava kako proces razvoja napreduje. Dok se tradicionalne metodologije oslanjaju na velik broj dokumenata, agilne metodologije koriste minimalnu dokumentaciju, dovoljnu za kvalitetno vođenje projekta. Tako najvažniji alat dokumentacije u SCRUM metodologiji, kao trenutno najpopularnijoj agilnoj metodi članovi tima prikazuju korisničke zahtjeve koji definiraju kako bi se neka funkcija u sustavu trebala ponašati da zadovolji korisnikove želje uz najmanju kompleksnost. Tako svaki zahtjev obično prolazi kroz općeniti životni ciklus - zahtjev je zapisan, prioritiziran, procijenjen, dodijeljen iteraciji ili sprintu, implementiran te prihvaćen kao završen (Ionel, 2009), što će u empirijskom dijelu rada biti točno prikazano kako se taj proces zapravo odvija između članova tima u pravom primjeru poduzeća.

Jedna provedena anketa (Jeremiah, 2024) o 601 poduzeću koji se bave razvojem softvera otkriva kako 91% ispitanika koriste agilnu ili hibridnu verziju agilnog pristupa razvoju softvera kao glavne razloge navodeći povećanu suradnju između timova koji inače ne rade zajedno, povećanu kvalitetu softvera i zadovoljstvo kupaca sa isporučenim proizvodom. Iz svih ovih razloga vidljivo je kako su tradicionalne metode zastarjele i irelevantne, te će se tako daljnji sadržaj rada usredotočiti na opisivanje agilne metodologije sa okvirima popularnih načina upravljanja razvoja informatičkim projektima unutar nje sa naglaskom na manja i srednja poduzeća sa razvojem web aplikacija i mobilnih aplikacija, dio kojih je poduzeće iz primjera u empirijskom dijelu rada. Iz istog razloga najveći će fokus biti na SCRUM metodi koju poduzeće iz primjera koristi kao hibridnu verziju metodologije za vlastite potrebe razvoja.

2.3. SCRUM

SCRUM je proces razvoja softvera temeljen na uspješnim praksama iterativnog, objektno temeljenog ciklusa razvoja. Iterativni proces je razvoj kroz ponovljene cikluse koji timovi koriste za rad na projektu u fazama, neprestano ga poboljšavajući dok ne budu zadovoljni konačnim rezultatom (Martins, 2024). SCRUM se općenito sastoji od tri faze - planiranje obuhvata rješenja, razvoj sprinta i zaključak. Prva faza uključuje zapisivanje i planiranje zadataka, te ako se radi o novom sustavu pripremu arhitekture sustava. Druga faza je faza samog izvršenja tih zadataka, odnosno razvoj i postavljanje tih funkcija na servere za testiranje i ispravljanje, a treća faza uključuje distribuciju tih promjena u sustav. Tri glavne uloge u tom cijelom procesu su:

- **Razvojni tim** - ljudi koji obavljaju rad, odnosno zadane zadatke unutar sprinta. To ne moraju samo biti programeri, već to mogu biti dizajneri, testeri, pisci sadržaja i slično.
- **Product owner** - vlasnik proizvoda je onaj koji razumije zahtjeve korisnika i poslovne mogućnosti razvojnog tima i poduzeća te tako uravnotežuje potrebe svih dionika u projektu i organizaciji. Njihova je odgovornost isporučiti najveću vrijednost tako da govori razvojnom timu što je važno isporučiti.
- **Scrum master** - to bi bila osoba koja je vlasnik cijelog procesa, odnosno osoba koja je odgovorna za povezivanje svega i osiguravanje da se Scrum provodi kako treba. Tako vlasniku proizvoda pomaže definirati vrijednost te razvojnom timu isporučiti vrijednost. U većim organizacijama služi i kao prijenos informacije što je to Scrum te kako stvara vrijednost za poduzeće (West, 2024).

Spomenuto je kako je SCRUM iterativna metoda, te je tako važno spomenuti kako se to provodi te kako zapravo razvijanje softvera kroz tu metodu izgleda. Iteracije u sustavu se zovu sprintovi koji nemaju definirano trajanje, ali je opća preporuka da to traje od 2 do 4 tjedna. Glavna funkcija sprinta je backlog u kojeg se upisuju korisničke priče, odnosno zadaci koji se prioritiziraju unutar backloga koji navodi zahtjeve za razvoj u svakom sprintu. Osim backloga osnovne komponente Scruma su planiranje sprinta, sprint backlog, dnevni SCRUM sastanak, pregled sprinta i retrospektiva sprinta. Proces funkcionira tako da se pruže funkcionalnosti putem korisničkih priča, nakon čega se planira sprint prema prioritetnim zahtjevima za razvoj tijekom sastanka za planiranje sprinta te se tako stvori sprint backlog, odnosno lista zahtjeva koji se trebaju razviti unutar tog sprinta. Ovi sastanci osiguravaju vidljivost i omogućavaju članovima tima planiranje nakon što dobiju popis zadataka. Dnevni Scrum sastanak je općenito 15-

minutni dnevni sastanak projektnog tima kojim upravlja SCRUM master kroz jednstavno izlaganje tko radi na kojem zadatku, ima li kakvih problema sa kojim su se susreli i slično. Pregled sprinta je sastanak koji se održava na kraju svakog sprinta gdje se sve nove funkcije prikazuju svim razvojnim timovima. Retrospektiva sprinta se održava kako bi se otkrili problemi u prethodnom sprintu i razvila rješenja za buduće sprintove (Çetin & Durdu, 2018). Svaka uloga, pravilo i vremenski okvir u Scrum-u dizajnirani su da osiguraju željene koristi i riješe predvidljive ponavljajuće probleme. Tako Scrum kao jedna od najčešće korištenih metodologija je često implementirana na razne načine koji odstupaju od originalnog modela što rezultira metodama razvoja zvanim “ScrumBut”. Neki tvrde da takve nepotpune implementacije krše temeljne prakse i filozofiju Scruma, dok drugi predlažu da se sve metodologije trebaju prilagoditi jedinstvenoj kulturi organizacije i integrirati s njihovim najboljim praksama. ScrumBut su tako razjašnjenja koje timovi izlažu kao razlog zbog kojeg u potpunosti ne koriste SCRUM kao metodologiju sa danim pravilima za najbolje rezultate. ScrumBut se doslovno prevodi kao “SCRUM, ali”, tako ukazujući na razloge koje timovi daju poput “Koristimo SCRUM, ali biti dio SCRUM sastanka svaki dan je previše opterećenja tako da imamo samo jedan tjedno" (SCRUM, 2024).

Rezultati jednog istraživanja (Schlauderer & Overhage, 2012) o iskustvima stručnjaka s SCRUM-om i tradicionalnim metodologijama ukazuju na nekoliko ključnih stvari. Programeri percipiraju brži izlazak na tržište u Scrum projektima, što poboljšava njihovu fleksibilnost i omogućava im brži razvoj onoga što je stvarno potrebno. To rezultira bržom isporukom korisnih softverskih proizvoda, što smatraju velikom prednošću. Također, istraživanje pokazuje da programeri smatraju kako Scrum projekti bolje ispunjavaju zahtjeve kupaca, što im omogućava osjećaj da ne rade uzalud kao što je to često bio slučaj s tradicionalnim metodologijama. Što se tiče učenja, programeri primjećuju poboljšane učinke u Scrum projektima jer mogu učiti jedni od drugih i od kupaca, što smatraju prednošću i motivacijom za prihvaćanje Scruma. Programeri također navode povećanu razinu zadovoljstva u SCRUM projektima u usporedbi s tradicionalnim metodama, zahvaljujući održivijem tempu rada i manjoj razini stresa. SCRUM projekti nude veću transparentnost u razvoju, što omogućuje programerima brže rješavanje problema. Osim toga, suradnja među programerima je poboljšana te tako vodi do učinkovitije komunikacije i razvoja projekata (Schlauderer & Overhage, 2012).

2.4. Životni ciklus razvoja web-baziranog softvera

Trenutno na svijetu postoji preko milijarde web stranica (Siteefy, 2024) i preko pet milijardi globalne populacije mobilnih uređaja čiji promet sačinjava oko 60% ukupnog internet prometa u svijetu (STATISTA, 2024), te je zato bitno sagledati razvoj kako web stranica, tako i mobilnih aplikacija da bi se osigurala kvalitetna responzivnost sustava na malim ekranima bilo to preko preglednika ili aplikacije. Internet i web se tako nastavljaju razvijati brzim tempom što je uzrokovalo promjenu informatičkih sustava sa stolnih računala na umrežene sustave te je zbog toga razvoj takvih informacijskih sustava morao prijeći sa tradicionalne metode razvoja putem životnog ciklusa razvoja softvera (SDLC) na životni ciklus razvoja web-baziranog softvera (engl. Web Based Development Life Cycle - WDLC) (Abdul-Aziz, Koronios, Gao, & Suhaizan, 2012). U nastavku će se opisati obje metode razvoja te njihova primjena.

Metodologije SDLC razvoja softvera nemaju sve potrebne značajke potrebne za sustavni razvoj web-baziranih informacijskih sustava, te je tako za razvoj web-baziranih informacijskih sustava tijekom vremena je uvedena novi paradigma koja se naziva web-bazirani razvojni životni ciklus (WDLC). Stoga postoji potreba prilagoditi ili izmijeniti konvencionalne SDLC metodologije kako bi ih se uskladilo s razvojem i održavanjem web-baziranih informacijskih sustava. Već je spomenuto kako je u SDLC u prvoj fazi planiranja sustav prioritiziran i organiziran prema zahtjevima korisnika. U fazi analize ti zahtjevi se strukturiraju kako bi se proizvela rješenja. Faza dizajna zahtjeve transformira u rješenja kao logičke i fizičke specifikacije sustava, što dovodi do faze implementacije gdje se sustav zapravo kreira kako bi ispunio specificirane zahtjeve. Faza održavanje ima za zadatak poboljšati i osigurati mogućnost budućeg korištenja sustava (Abdul-Aziz, Koronios, Gao, & Sulong, 2012). Razvoj web-baziranih informacijskih sustava može se definirati kao "sustavna metodologija koja uključuje pojednostavljen višestupanjski proces razvoja web-baziranih aplikacija prema specifikacijama i web standardima što učinkovitije moguće” (Abdul-Aziz, Koronios, Gao, & Sulong, 2012).

WDLC tako uzima kostur SDLC metodologije te ju proširuje na idući način:

- **Planiranje** - uključuje skupljanje informacija o zahtjevima kroz izravnu komunikaciju ili povezane dokumente te vizualizaciju ideje izgleda i funkcije web stranice nakon čega se izrađuje razvojni raspored i osigurava isporuka na vrijeme.
- **Analiza** - ispituje zahtjeve kako bi proizvela potpunu specifikaciju privlačnosti dizajna i informacija web aplikacije u odnosu na trenutnu situaciju na tržištu i trendove u korištenju i

izgledu web aplikacija. Ako se radi o poduzeću koje može priuštiti investiciju u neke ključne informacije o tržištu na koje se web aplikacija plasira, onda se u ovu fazu uključuje i istraživanje tržišta kako bi razvoj suzio na targetiranje ciljnog tržišta.

- **Dizajn** - priprema se wireframe, odnosno osnovni nacrt web stranice kroz prikaz bitnih komponenti i značajki poput izgleda ekrana, navigacijskih traka, komponenata UX i UI dizajna te interaktivnih elemenata (Figma, 2024).
- **Implementacija** - razvoj arhitekture web stranice na temelju trenutnih tehnoloških jezika i framework-ova, kodiranje predložaka web stranica i integriranje cloud based baza podataka i razvoj API-ja za povezivanje na druge web servise i stranice što je danas veoma važna funkcija web aplikacija. Uz to bitno je i izgraditi poslužiteljsku infrastrukturu, osigurati domenu web stranice, sigurnost i pohranu podataka te povezivanje korisničkog sučelja sa back-end platformom (Bower, 2023).
- **Promocija** - dodatna faza WDLC kojoj je cilj da oglašava dovršene web-bazirane sustave potencijalnim web korisnicima. Još jedna bitna stavka zbog koje ova faza postoji u web baziranim aplikacijama je optimizacija za tražilice (engl. Search engine optimisation - SEO), odnosno proces optimiziranja relevantnosti sadržaja na web stranici kako bi se ona mogla lakše pronaći prema korisničkim upitima pretraživanja, te kao posljedica toga bila bolje rangirana u tražilicama kao najpopularniji odabir za traženi pojam (Mailchimp, 2024).
- **Održavanje** - ima za zadatak pružiti neprekidnu pouzdanost kroz dinamičko ažuriranje sadržaja na web aplikaciji uz konstantno prilagođavanje promjenama i novim potrebama (Abdul-Aziz, Koronios, Gao, & Sulong, 2012). Za web stranice to predstavlja nekoliko glavnih točaka održavanja. Prvo su sigurnosna ažuriranja kako bi se riješile ranjivosti u sustavu od hakera. Ažuriranje plugina i tema za proširivanje funkionalnosti i ažuriranje sadržaja su bitni zbog relevantnosti i točnosti podataka i brisanja zastarjelog sadržaja radi održavanja dobrog SEO-a i kvalitete pruženih informacija korisnicima i potencijalnim kupcima. Sigurnosno kopiranje i oporavak od katastrofe su također bitni kroz održavanje sigurnosne kopije u slučaju da nešto pođe po zlu i da se web stranica vrati u nedavno sačuvanu verziju, dok praćenje i optimizacija performansi pomaže prepoznati potencijalne probleme i poduzeti korake za poboljšanje vremena učitavanje stranice i optimizaciju koda koji to drži u pozadini (Mailchimp, 2024).

2.5. Životni ciklus razvoja mobilnih aplikacija

Razvoj mobilnih aplikacija zahtijeva stalna poboljšanja i prilagodbe kako bi se zadovoljile nove tehnološke potrebe i promjene poput dizajna korisničkih sučelja za različite veličine ekrana mobilnih uređaja, korisničko iskustvo povezano s mogućnostima mobilnih uređaja, metode korisničke interakcije koje pružaju nove mobilne platforme, arhitekture i slično (Ghandi i ostali, 2017). Tako bi razvoj mobilnih aplikacija za efikasan proces izrade trebao koristiti neku specifično sastavljenu metodologiju za tu funkciju razvoja softvera kao i kod razvoja web stranica, no rezultati jednog meta istraživanja (Jabangwe i ostali, 2018) sugeriraju da iako trenutno postoji 20 modela razvoja, ti modeli su još uvijek u nezreloj fazi, iako je količina razvoja mobilnih aplikacija masivna. Svi modeli iz tih radova usvajaju pristupe zasnovane na ideji agilne metode, međutim većina radova koji predlažu modele ima vrlo nisku procjenu rigoroznosti i relevantnosti zbog nedostatka informacija o subjektima, kontekstu i razmjerima. Ovaj nedostatak relevantnosti znači da je vrijednost radova za industriju niska, no i dalje valja spomenuti ideju ove metode baš zbog potrebe za dodatnom razradom sa obzirom na veličinu tržišta kojeg je dio (Jabangwe i ostali, 2018).

Tako se agilnu metodologiju već spomenutog SDLC i WDLC može proširiti na model razvoja životnog ciklusa mobilnih aplikacija (engl. Mobile Application Development Life Cycle - MADLC) kao predložak za sustavni pristup u razvoju:

- **Planiranje i analiza** - ova faza je ista kao i kod prethodno dvije spomenute metodologije gdje je glavni cilj faze iznijeti novu ideju ili poboljšanje postojeće aplikacije uz prikupljanje zahtjeva i sastavljanje dokumentacije za generiranu ideju uz donošenje odluke hoće li razvijena aplikacija biti besplatna verzija sa ograničenim značajkama i pretplatnim modelom ili samo kao premium verzija.
- **Faza dizajna** - treba se utvrditi izvedivost na obje mobilne platforme (Android i iOS), ili alternativno, identificira se specifična ciljna mobilna platforma. Definiraju se funkcionalni zahtjevi, izrađuje arhitektura te se stvaraju storyboard-ovi, odnosno priče koja opisuju tok aplikacije kako bi njezino korištenje trebalo izgledati i kroz što jednostavnije korisničko iskustvo pružila željenu funkciju. Izrađuju se i “mockups”, odnosno makete kao vizualne reprezentacije mobilne aplikacije koje prikazuju izgled i raspored korisničkog sučelja bez funkcionalnosti (KMS Solutions, 2022).

- **Faza razvoja** - jednako kao i kod životnog ciklusa razvoja softvera, ovo je faza u kojoj programeri pišu programski kod aplikacije, no specifična je po tome što se kreiraju različiti prototipi koji se šalju korisniku za povratne informacije, te nakon što je svaki novi prototip spreman, on se integrira sa prijašnjim, testira i šalje korisniku. Razvoj korisničkog sučelja kao razvoj dizajna je također bitan kako bi se ujednačio i podržao rad na oba glavna mobilna operativna sustava (iOS i ANDROID) kako aplikacija ne bi imala različit izgled i iskustvo korištenja na različitoj platformi.
- **Faza testiranja** - specifična je po tome što se prvo obavlja na emulatoru/simulatoru, a potom na stvarnom uređaju. Tako se testiranje na stvarnom uređaju treba provoditi na više verzija operativnog sustava sa više modela mobitela i različitim veličinama zaslona kako bi se osigurala kvalitetna responzivnost aplikacije i njezina ujednačenost na različitim mobitelima.
- **Faza implementacije** - na mobilnim uređajima implementacija se odrađuje tako da se aplikacija prenosi u odgovarajuću trgovinu za korisničku potrošnju, što može biti veoma zahtjevan proces za timove neupoznate sa tom fazom, pošto iOS App Store i Google Play Store imaju poprilično rigorozne mjere, pravila i propise za implementaciju aplikacije u njihov sustav.
- **Faza održavanja** - kontinuirani proces prikupljanja povratnih informacija od korisnika te povratak u recikliranje cijelog ovog procesa razvoja kako bi se provele potrebne izmjene u obliku ispravaka grešaka ili poboljšanja, te dodatnog marketinga aplikacije (Vithani & Anandkumar, 2014), no malo više o ovom procesu u idućem poglavlju.

3. ODRŽAVANJE I ODRŽIVOST INFORMACIJSKIH SUSTAVA

3.1. Održavanje

Osnovni cilj održavanja softvera je efikasna modifikacija postojećeg sustava uz očuvanje njegove cjelokupne stabilnosti. Ono održava funkcionalnost softverskog proizvoda tokom vremena, prilagođavajući se promjenjivim potrebama korisnika i ispravljajući greške koje se otkriju nakon početnog razvoja i implementacije. Kroz održavanje, softver ostaje relevantan, pouzdan i optimiziran, s ključnim ciljem da se produži korisni vijek trajanja softvera i odgodi potreba za zamjenom što je duže moguće. Sustavi nikada nisu dovršeni već se kontinuirano mijenjaju kao odgovor na nove korisničke potrebe i promjene u okolini. Ispravljaju se nedostaci koji nisu otkriveni prije puštanja u upotrebu. Tokom implementacije, softverski sistem može pokazati nepredvidljivost, što može rezultirati pojavom grešaka koje zahtijevaju popravak. Također, održavanje ima za cilj unaprijediti kvalitetu softvera, uključujući performanse, korisničko iskustvo, pouzdanost, dostupnost, prilagodljivost i sigurnost. Osim toga, održavanje prilagođava softver novim zahtjevima, tehnologijama i propisima, te ispravlja tehničke nedostatke koji mogu ometati fleksibilnost sistema (IEEE Computer Society, 2024). Ovi različiti razlozi potrebe održavanja tako sastavljaju četiri glavne kategorije aktivnosti održavanja (ISO, 2006):

- **Savršeno održavanje** - Ovo uključuje poboljšanje funkcionalnosti softvera kao odgovor na promjene koje definira korisnik.
- **Prilagodljivo održavanje** - Ovaj proces uključuje izmjene softvera do kojih dolazi zbog promjena unutar softverskog okruženja.
- **Korektivno održavanje** - Ovaj proces uključuje ispravljanje grešaka koje su identificirane unutar softvera.
- **Preventivno održavanje** - Ovo uključuje ažuriranje softvera kako bi se poboljšala njegova mogućnost održavanja u budućnosti bez mijenjanja njegove trenutne funkcionalnosti (ISO, 2006).

Treba uzeti u obzir kako je područje razvoja softvera široko i raste svakodnevno s novim i budućim standardima i framework-ovima. Novi programski jezici i verzije pojavljuju se gotovo svakog mjeseca, donoseći nove značajke i tehnologije koje olakšavaju i ponekad mijenjaju projekt na svim razinama. Zbog toga voditelj projekta mora pratiti buduće verzije programskih jezika kako bi zadovoljio zahtjeve vlasnika projekta i uskladio se s projektnim timom (Saeed i ostali, 2019). Ključni izazov je pronaći načine za razvoj softvera tako da se softver može lakše i pouzdanije održavati i mijenjati u kasnijim fazama. Ako trenutna

arhitektura ne podržava planiranu promjenu, tada se arhitektura mora restrukturirati te napraviti promjene u arhitekturi sustava kako bi se moglo napraviti te željene promjene. Tako je programerski posao osim razvoja novih funkcija kao dio faze održavanja također samo održavanje cijele arhitekture, koda i dokumentacije kroz:

- Migraciju sa zastarjelih na moderne programske jezike.
- Migraciju sa zastarjelih na moderne baze podataka.
- Restrukturiranje koda i podataka radi uklanjanja nepotrebne složenosti (posebno složenosti koja je uvedena intenzivnim održavanjem).
- Ažuriranje alata za dokumentaciju radi upravljanja komentarima u kodu.
- Ažuriranje softvera bez potrebe za zaustavljanjem rada aplikacije (Bennett & Rajlich, 2000).

Čak i najjednostavnije aplikacije moraju redovito biti održavane kako bi nastavile funkcionirati na uređajima kroz koje im se pristupa. Tako na primjer, Google Play Store ima mnogo protokola kojih se treba pridržavati i koje treba redovito održavati nakon što je aplikacija postavljena na njihovu platformu. Nije dovoljno samo napraviti aplikaciju i omogućiti ju za preuzimanje, već uz redovita nova izdanja Android sustava aplikacije moraju postavljati ažuriranja kako bi se osigurala kompatibilnost rada postojećih funkcija na novim verzijama operativnog sustava te sigurnosna ažuriranja koja su kritična točka po kojoj Google dozvoljava da aplikacija ostane dostupna na njihovoj Google Play Store platformi (Google, 2023). Tako slika ispod prikazuje Google pravila za krajnji datum za koji se mora ciljati određena razina ažuriranja za svaku od recentnih verzija androida za nove aplikacije te za one koje već postoje na njihovoj platformi kako bi aplikacija i dalje ostala dostupna za preuzimanje na njihovoj platformi:

Android OS version (API level)	When are new app and app update submissions required to target this API level?	
	New apps	App updates
Android 13 (API level 33)*	August 31, 2023	August 31, 2023
Android 12 (API level 31)	August 31, 2022	November 1, 2022

Slika 4 - Krajnji rok ciljne razine ažuriranja aplikacija na Google Play Store platformi

Izvor: https://support.google.com/googleplay/android-developer/answer/11926878

Tako je spomenut još jedan važan aspekt održavanja sustava - održavanje sigurnosnih ažuriranja aplikacije, osobito kad su u pitanju web aplikacije sa sučeljem za login i osjetljivim privatnim podacima unutar računa. Većina aplikacija je postavljena u cloud okruženje kako bi im se pristupilo putem interneta. Stoga hakeri uvijek pokušavaju pristupiti, izmijeniti ili oštetiti njihove podatke tražeći ranjivosti u aplikaciji kako bi preuzeli kontrolu nad dijelom aplikacije ili cijelom aplikacijom. Istraživanje (Subedi i ostali, 2016) ističe da je 86% od 30.000 testiranih web stranica imalo barem jednu ozbiljnu ranjivost, a od njih više od 75% imalo je ranjivosti zbog slabosti u procesu razvoja web aplikacije. Tako se okvir agilnog pristupa razvoja može iskoristiti za poboljšanje sigurnosti sustava kao proširenje agilne metodologije za rješavanje sigurnosnih rizika. Ti rizici mogu biti tehnički rizik poput zagušenja servera, poslovni rizik gubljenja poslovnih podataka te rizik privatnosti, odnosno rizik curenja privatnih podataka (Subedi i ostali, 2016) koji su danas veoma važno područje uskladivosti sa državnim i međunarodnim zakonima i pravilima. Pravilna strategija za zaštitu privatnosti može potaknuti poslovni rast i izgraditi povjerenje s korisnicima. Tvrtke koje ne usklađuju svoje prakse s propisima o privatnosti suočavaju se s rizikom od ozbiljnih posljedica. Svjetski propisi o zaštiti podataka, kao što su npr. europski GDPR i brazilski LGPD, postaju sve rigorozniji, a tvrtke moraju pratiti promjene i prilagođavati se. Da bi se učinkovito upravljalo privatnošću, tvrtke trebaju procijeniti svoje podatke, uspostaviti programe zaštite privatnosti od početka razvoja proizvoda, te pratiti i prilagođavati se novim zahtjevima (IBM, 2024). Tako održavanje sigurnosti sustava kroz upravljanje procesom razvoja iterativno prolazi kroz faze agilnog razvoja sustava (Subedi i ostali, 2016):

- **Analiza trenutnog stanja i planiranje** - shvaćanje potrebe za promjenom dolazi kroz identificiranje mogućih napada, eksplotacijskih i autentifikacijskih rizika.
- **Dizajn** - kao na primjer dizajn izgleda i funkcije autentifikacije za login u aplikaciju, poput dvofaktorske autentifikacije.
- **Implementacija** - izrada enkripcije podataka i zadataka vezanih za proces povećanja sigurnosti ili tehničke nove vrste autentifikacije.
- **Testiranje** - testiranje novoizgrađene autentifikacije i priprema za autentificiranje svih postojećih korisnika na novi sustav sigurnosti.
- **Validacija** - provjeravanje zadovoljenja svih ciljeva nove sigurnosti podataka i uspješne tranzicije korisnika implementiranog procesa.

Zahtjev za održavanje tako zapravo prolazi kroz životni ciklus sličan razvoju softvera. Zahtjev se analizira, utvrđuje njegov utjecaj na sustav, sve potrebne izmjene se dizajniraju, kodiraju, testiraju i na kraju implementiraju. Obuka i svakodnevna podrška također su ključne komponente faze održavanja softvera što će biti opisano kroz praktičan primjer u empirijskom poglavlju.

Osim održavanja same aplikacije, još jedan aspekt održavanja sustava koji pruža aplikacije u poduzeću je i održavanje dokumentacije te tako baze znanja. Agilna metodologija u većim timovima može zahtijevati opširniju dokumentaciju zbog boljeg korištenja upravljanja znanjem (engl. Knowledge Management - KM) kao nematerijalnom vrijednošću poduzeća. Upravljanje znanjem bi predstavljalo sustavan proces unutar organizacije koji uključuje identificiranje, organiziranje, pohranjivanje i širenje informacija sa ciljem poboljšanja učinkovitosti i efikasnosti poslovanja (IBM, 2024). Tako kolektivna stručnost tima sa lako dostupnom bazom znanja dovodi do bolje distribucije važnih informacija u cilju boljeg donošenja odluka, poboljšane suradnje i uštede vremena i resursa što bi se u suprotnom potrošilo na traženje informacija (IBM, 2024). Istraživanja (Aurum i ostali, 2008) su pokazala kako programeri općenito teško objašnjavaju što rade, kako nešto rade, kako rješavaju određene probleme koristeći svoje znanje te tako teško prenose važne informacije ostalim članovima tima, što čini velike probleme u većim timovima i između timova za poboljšanje ponovljivog procesa za upravljanjem znanjem i stručnosti koji jednom iskoriste i nigdje ne dokumentiraju. I u manjim timovima upravljanje znanjem predstavlja važan resurs za poduzeće, direktno putem pohranjivanja ponovo iskoristivih podataka poput samog koda softvera, gdje se baza ponovo može iskoristiti na drugom projektu. Tako ušteđeno vrijeme na ponovo iskorištenom znanju štedi resurse kompaniji, ali i potencijalno utrošenu energiju na ponovno sastavljanje takvog znanja, što dovodi većoj održivosti, više o čemu će biti riječi u nastavku.

3.2. Održivost

Najcjelovitija definicija održivosti je da je to „*zadovoljenje potreba sadašnjosti bez ugrožavanja sposobnosti budućih generacija da zadovolje svoje potrebe*" (International Institute for Sustainable Development, 2024). Nedavno, održivost se dotiče i razvoja softvera karakterizirano načinom na koji se softver razvija, tako da se negativni i pozitivni učinci koji proizlaze iz softverskog proizvoda prate, dokumentiraju i optimiziraju tijekom cijelog životnog ciklusa proizvoda. Postoje razne definicije karakteristika održivog softvera, od kojih bi najvažnije bilo spomenuti kako su to uobičajeni kriteriji kvalitete koji proizlaze iz dobro poznatih i standardiziranih aspekata kvalitete poput učinkovitosti, ponovljivosti, izmjenjivosti i upotrebljivosti. Održivost je tako dio kvalitete proizvoda sa tri karakteristike - **potrošnja energije, optimizacija resursa i dugotrajnost**, odnosno održiv sustav bi bio onaj koji ima atribute produženog životnog vijeka, učinkovitu upotrebu resursa za postizanje ciljeva, koji održava svoju funkcionalnost unatoč kvarovima te koji je prilagodljiv promjenama (Amri & Bellamine Ben Saoud, 2014). Od održivog softverskog proizvoda se tako očekuje i održavanje proizvoda u odnosu na promjene u tehnologiji i zahtjevima (Amri & Bellamine Ben Saoud, 2014). Tako su održavanje i održivost u direktnom skladu jedno sa drugim, te je smisleno sustavno održavanje sustava važan dio postizanja održivosti nekog softvera.

Vlade i javni zagovornici sve više naglašavaju važnost ekološke održivosti, pa tako i sve više softverskih industrija poput Della i IBM-a implementira zelene strategije (Anthony i ostali, 2017). Softverske organizacije moraju razviti pristupe održivosti koji balansiraju ekonomske koristi, društvene i ekološke ciljeve. Održiva praksa u softverskim procesima može podrazumijevati fizičke resurse poput razvoja, korištenja i odlaganja računala i povezanih komponenti, ali i nefizičke resurse usvajanja održivih praksi u softverskim procesima kako bi osigurali učinkovitost usluga, smanjili troškove i poboljšali kontrolu nad sustavnom infrastrukturom (Anthony i ostali, 2017). Održiva praksa softvera odnosi se na upotrebu, implementaciju i primjenu softverskih sustava koji doprinose održivim procesima. Održivost u procesima softvera je praksa razvoja, upotrebe i odlaganja računala, poslužitelja i povezanih podsustava poput monitora, pisača, uređaja za pohranu, mrežnih komunikacijskih sustava na učinkovit i efikasan način s minimalnim ili nikakvim utjecajem na okoliš. Održive prakse u procesu razvoja softvera bi osigurale učinkovitost usluga, smanjene troškove i bolju kontrolu nad infrastrukturom sustava. Stoga, održivost ne bi trebala biti shvaćena kao trošak poslovanja, već ona pruža prilike za poduzeća da poboljšaju produktivnost usluga te tako smanjenje troškova i povećanje profitabilnosti (Anthony i ostali, 2017).

Kako je energija glavni potrošni resurs održivosti, tako postoje i neke metrike koje sagledavaju koliko energije neki proces u razvoju softvera koristi, te se mogu koristiti kao neka točka za pregled potrošnje energije i pronalaska rješenja za smanjenje tog trošenja. To su, na primjer, metrike energetskih troškova povezanih sa CPU obradom, pristupom memoriji i operacijama neke aplikacije kroz učinkovitost koda od kojih je sastavljena, metrike učinka pohrane i dohvaćanja podataka kao odnos broja obrađenih bajtova po kilovatsatu električne energije i slične metrike učinkovitosti procesa u potrošnji energije u odnosu na pruženu uslugu u nekom poduzeću (Anthony i ostali, 2017).

Tako se mogu nabrojati neke varijable usvajanja održive prakse na temelju pet glavnih procesa implementacije održivog softvera - razvoj, distribucija, operacija, odlaganje i deaktivacija. Svaka od njih može pridonijeti boljoj održivosti kroz implementaciju zelenih strategija:

- **Softverska tehnička infrastruktura** - treba korisititi efikasno kodiranje i ponovo iskoristiti postojeće module i znanje, odnosno koristiti prethodno spomenute alate baze znanja KM. Treba pratiti i kontrolirati životni ciklus spremljenih podataka za osiguravanje od redundantnosti podataka i pravovremeno održavati dokumentaciju projekta ili stanja poduzeća.
- **Mrežna kritična fizička infrastruktura** - implementirati telekonferencije, video konferencije, rad od kuće te općenito sve moguće vrste online infrastrukture za suradnju kako bi se trošilo manje resursa na dolaske na fizičke sastanke. Energetski učinkovitija svjetla i sustavi sa stand-by napajanjem, jednako kao i energetski učinkoviti komunikacijski i serveri baze podataka su još jedna funkcija održivosti, dok bi neiskorišteni serveri komunikacija i baza podataka trebali biti uklonjeni.
- **Energetska učinkovitost softvera** - primjena upravljanja energijom osobnih računala kroz korištenje održivih planova korištenja unutar postavki operativnog sustava, implementacija nadzora potrošnje energije svih primjenjivih serverskih sustava, učinkovitiji ventilatori, hlađenje i pumpe, smanjenje koraka pretvorbe energije radi smanjenja gubitka energije i slično.
- **Monitoring zelenog softvera** - praćenje otiska CO2 koje poduzeće stvara, revizija energetske učinkovitosti postojećih softverskih sustava i razvijanje portfelja zelenih softverskih usluga i proizvoda (Anthony i ostali, 2017).

World Economic Forum prognozira kako bi digitalne tehnologije mogle smanjiti globalne emisije CO2 za 20% do 2050. godine u tri sektora sa najvećim emisijama - energija, materijali i mobilnost. Ti sektori predstavljaju najveći potencijal za smanjenje emisija kroz razvoj visokoučinkovitih digitalnih tehnologija poput tehnologije donošenja odluka koje povećavaju ljudsku inteligenciju i tehnologije upravljanja koje prikupljaju podatke i mijenjaju fizičke procese kako bi bili održiviji (World Economic Forum, 2022).

- **Energetski sektor** - može iskoristiti implementaciju razvoja informacijskih sustava upravljanjem obnovljivom energijom pomoću umjetne inteligencije koju pokreće računalo u cloudu umreženi 5G tehnologijom.
- **Materijali** - mogu poboljšati rudarenje proizvodnih materijala i njihovu daljnju proizvodnju i preradu kroz analitiku velikih podataka.
- **Mobilnost** - kroz korištenje i razvoj senzorskih tehnologija za prikupljanje podataka može se u stvarnom vremenu poticati bolje donošenje odluka i time poboljšanja optimizacije ruta i smanjenje emisija u željezničkom i cestovnom prometu (World Economic Forum, 2022).

Postoje mnoga istraživanja i recentni modeli zadnjih 10 godina o održavanju i održivosti razvoja sustava, no većina njih kao zaključak postavi stajalište kako su daljnja istraživanja o tim temama potrebna kako bi se moglo doći do nekog sigurnijeg zaključka o rješenjima koja su dostupna zajedno sa empirijskim rezultatima poduzeća koja ih prakticiraju, tako da se ova tematika još može smatrati relativno novim, ali veoma važnim područjem istraživanja.

4. REZULTATI EMPIRIJSKOG DIJELA

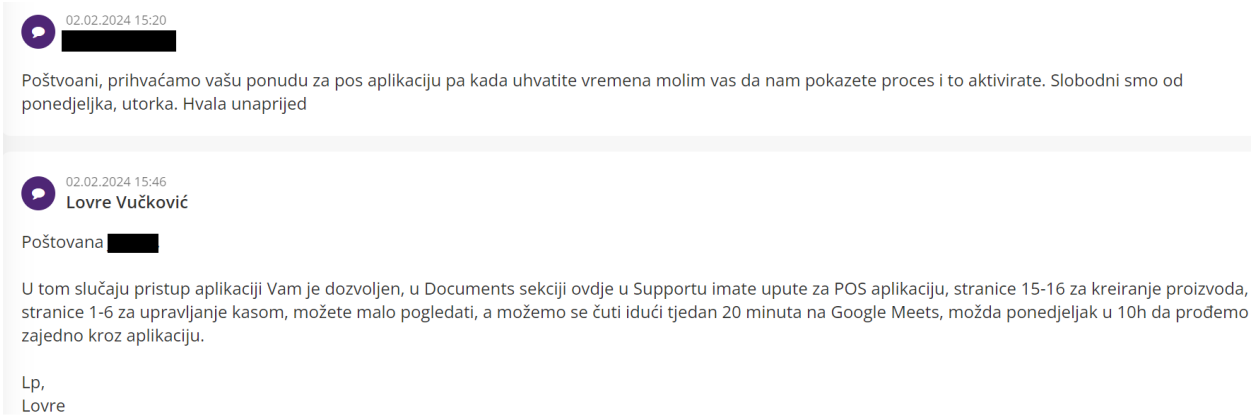
Kako bi se ostvarili postavljeni ciljevi ovog diplomskog rada, odabrana je metoda studije slučaja na konkretnom primjeru poduzeća. Studija slučaja predstavlja empirijsko ispitivanje koje istražuje fenomen u kontekstu stvarnog života, te tako nije metoda prikupljanja podataka, već metoda proučavanja društvene jedinice (Priya, 2020). Važno je napomenuti kako ova metoda ima svoje nedostatke od kojih je bitno u kontekstu ovog rada spomenuti dva; prvi nedostatak je da nije moguće generalizirati otkrića iz intenzivnog proučavanja jednog slučaja, a drugi da nije moguće da istraživač potpuno iz prikaza slučaja izbaci vlastite predrasude i pristranosti (Priya, 2020). Zato je bitno još naglasiti kako je autor teksta zaposlenik u poduzeću iz primjera te da zbog te činjenice postoji mogućnost postojanja određene razine pristranosti u interpretaciji i prezentaciji podataka.

Hotel's Touch (STATIM d.o.o.) je poduzeće iz Splita koje se bavi pružanjem usluga najma informacijskog sustava za hotelsko i restoransko poslovanje kroz sve procese obavljanja zadataka u svakodnevnom upravljanju smještajnim objektom, od apartmana sa tek nekoliko smještajnih jedinica, do hotela sa 150 soba, hostela i bungalova, ali i improviziranih smještaja poput kampova te turističkih agencija koji upravljaju sa nekoliko desetaka nepovezanih privatnih smještaja. Pruža kompletno rješenje za klijente zbog velikog broja aplikacija koje osim za upravljanje svakodnevnim hotelskim poslovanjem ima aplikacije i za evidencije radnog vremena zaposlenika i kreiranje automatskih izvještaja potrebne za računovodstva, državne službe, partnere i sve ostale povezane subjekte smanjujući vrijeme utrošeno na administraciju i vođenje raznih evidencija, bilo zakonodavnih ili nastalih iz poslovnih potreba za generiranjem korisnih informacija iz dostupnih podataka i upravljanjem tim informacijama za povećanje efikasnosti i smanjenju troškova. Tako se Hotel's Touch, osim smještajnim objektima, sa fleksibilnošću aplikacija koje ima fokusira na pružanje usluga svim vrstama i veličini poduzeća, bez obzira na prirodu poslovanja koje obavljaju, te za klijente ima i proizvodna poduzeća poput Optika Anda, Galeb, Tromont, državne službe poput Studentskog centra Split te Čistoće Sinj, no i manje prodavaonice, obrte i sve ostale vrste gospodarskih subjekata. Upravo iz tog razloga je pri razvoju aplikacije potrebno razmišljati pokušava li se fokusirati na neku nišu te aplikaciju napraviti na jednostavan način za specifičnu funkciju ili pronaći način za aplikaciju napraviti generalno korisnom različitim potrebama te tako pokriti veće tržište, ali sa rizikom kompliciranja samog razvoja te aplikacije, smanjenjem preglednosti prikaza i jednostavnosti korištenja novim korisnicima što nekim potencijalnim klijentima u besplatnom probnom periodu može presuditi kao faktor zbog kojeg neće uzeti sustav. Hotel's Touch proizvodi i svoj hardware, točnije fizičke uređaje za prijavu/odjavu na radno mjesto, kao i uređaj za izradu, reprodukciju i kontrolu

sadržaja na ekranima upravljanjem platformom kojoj se pristupa putem web preglednika. Veliki broj aplikacija u jednom sustavu koje se nudi kao zaseban paket nudi veću mogućnost da će aplikaciju uzeti korisnik koji stvarno traži funkcije koje mu ta aplikacija nudi, te da će u budućnosti samostalno odlučiti uzeti još neki paket aplikacija što poduzeću stvara dodatni *upsell*, odnosno poticanje kupaca da kupe veći paket ili verziju proizvoda od one koju su prvotno namjeravali kupiti bez dodatnog truda poduzeća za poticanje te kupnje (Salesforce, 2024). Tako poduzeće stvara i svoj tehnološki ekosustav u kojem korisnici (u ovom slučaju poslovni korisnici) kroz jedan sustav imaju mogućnost obavljati više poslovnih zadataka za koje bi im u suprotnom slučaju trebalo nekoliko različitih sustava. Prednost za korisnika je u tome što osim što se sve nalazi na jednom mjestu te što se podaci dijele između svih aplikacija što pri uzimanju nove aplikacije iz sustava razdoblje prilagodbe je jednostavno, pošto sve aplikacije dijele sličan način funkcioniranja, dizajn i ostale značajke. Prednost takvog *upsella* je i za poduzeće koje prodaje te aplikacije u tome što uštedi resurse na nekoliko načina:

- **Odjel prodaje** - uštedi vrijeme na skupljanje baze klijenta za tu drugu aplikaciju, organiziranje i održavanje sastanka
- **Odjel implementacije** - ima manje posla pošto je postavljanje nove aplikacije kod već postojećeg klijenta znatno jednostavnije od otvaranja novog klijenta zbog skupljanja inicijalnih podataka i zahtjeva koji su u zajednički sustav već upisani
- **Odjel edukacije** - će u manje vremena odraditi kvalitetniju edukaciju pošto je klijent već upoznat sa principima na koje druga aplikacija u paketu radi, a sve aplikacije dijele slične karakteristike i dizajn sa drugačijim funkcijama.

Primjer ovdje je klijent koji već godinu dana koristi aplikaciju za upravljanje recepcijskim sustavom, te se samoinicijativno odluči da će u paket uzeti i POS blagajnu za restoran. Inače, za implementaciju takvog korisnika “od nule” treba dva sata zajedno još sa dva sata edukacije, dok za postojećeg korisnika koji samo nadogradi paket aplikacija ima sve postavljeno te u 20 minuta edukacije samostalno riješi sve potrebno kako bi mogli početi sa korištenjem novog sustava. Unutar interne aplikacije Support, čiji je razvoj tema ovog rada, slika ispod prikazuje komunikaciju sa tim klijentom:



Slika 5 - Komunikacija sa klijentom

Izvor: izrada autora, prema podacima iz https://www.hotelstouch.com/

Pored aplikacija za komercijalnu upotrebu Hotel's Touch razvija i aplikacije koje ne prodaje, već koristi kao alat za:

- **Besplatnu upotrebu klijentima (Projects)** - aplikacija koja predstavlja veoma jednostavan način zapisivanja i organiziranja informacija na jednom mjestu kroz objave na kreiranim projektnim pločama te dijeljenje sa određenim korisnicima u sustavu. Aplikacija nije ništa bolja od drugih besplatnih alata koje korisnik može pronaći na internetu za upravljanjem informacijama i dijeljenje sa timom, ali prednost koju ima je da se klijentu sve nalazi na jednom mjestu te sve korisnike u sustavu ima već kreirane za druge aplikacije. Tako kao dodatna besplatna aplikacija puno znači kao dodatan razlog za usidravanje u cijeli ekosustav aplikacija kako sve više svakodnevnih poslova koje su neki do tad radili ručno sad imaju na jednom mjestu unutar svih aplikacija koje koriste u Hotel's Touch informacijskom sustavu.
- **Samostalnu upotrebu (Client365)** - razvijanje aplikacije od početka iz temelja je skup posao te uobičajeno mora imati nekakvo ekonomsko opravdanje ili kao aplikacija koja će se prodavati ili poput već spomenute aplikacije Projects kao besplatan alat za pružanje dodatne vrijednosti klijentu. Tako je razvijanje aplikacije samo za vlastitu upotrebu neprofitabilno rješenje kako se ljudske resurse koji sudjeluju u procesu razvoja aplikacije može postaviti na projekte koji, uzimajući samo troškove i dobit u obzir, imaju ekonomskog smisla u odnosu na nekomercijalnu aplikaciju za koju postoje već postojeća rješenja na tržištu sa boljom preglednošću i više funkcija. Glavni razlog zbog kojeg se u ovom slučaju kreira osobno rješenje je baš iz one povezanosti što kako puno znači klijentu da ima sve informacije u povezanim Hotel's Touch aplikacijama, tako i Hotel's Touch menadžmentu veoma puno znači poveznica svih klijenata u sustavu i svih

podataka zapisanim o njima te automatizam ažuriranja svih tih informacija kroz povezanu bazu podataka. Tako se prilikom kreiranja novog klijenta u bilo kojoj aplikaciji kroz interno povezani CRM sustav (koji je također samostalno razvijeno rješenje) svi podaci dijele u Dashboard aplikaciji te odjel prodaje, menadžmenta i tehničke podrške svi mogu koristiti aplikaciju za međusobno zapisivanje informacija o klijentu, poput informacije da li je to potencijalni klijent, postojeći, kad ga se kontaktiralo za ponudu, da li je ponuda prihvaćena, kada treba nazvati ponovo, u kojem je stanju implementacije, tko je zadužen za tog klijenta, da li je plaćanje na mjesečnoj ili godišnjoj bazi, koji računi su plaćeni a koji nisu te slične ostale informacije.

- **Upotrebu i klijentima i samostalno (Support)** - još jedna vrsta nekomercijalne aplikacije koju Hotel's Touch razvija je aplikacija koja ima funkciju i poduzeću i klijentu. To je aplikacija za prijavu problema u sustavu pod nazivom "Support" čiji je razvoj tema empirijskog dijela ovog rada.

4.1. Uvod

Ne postoji aplikacija koja je savršena, da funkcionira bez ijednog problema ili *buga*, da je idealno responzivna na sve jednom mobilnom uređaju, računalu te na svim operacijskim sustavima. Postoje aplikacije koje su napravljene kako bi se masovno prodavale ciljajući bilo koga tko za tom aplikacijom ima potrebe, a postoje aplikacije koje se kreiraju specifično za jednog klijenta sa precizno definiranim zahtjevima te konstantnom linijom komunikacije razvojnog tima sa klijentom u cijelom procesu tog razvoja. U oba slučaja razvijanje aplikacije je zadatak sa finalnim ciljem planirane verzije proizvoda kojeg se želi postići, no prvotno zacrtani cilj nikad nije krajnje odredište zato što se okružje i alati u kojem se ti sustavi razvijaju stalno mijenjaju. Te promjene uzrokuju nestabilnosti u radu aplikacije koje zahtijevaju napor razvojnog tima za stalno unaprjeđenje i evoluciju aplikacije kako bi se prilagodila novim zahtjevima i tehnološkim trendovima. Stoga, kontinuirana podrška i održavanje aplikacije postaju ključni faktori za osiguranje njezine funkcionalnosti i konkurentnosti na tržištu. To bi predstavljao podršku (*support*) u funkcionalnom smislu koji podrazumijeva tim programera zajedno sa projektnim menadžerom, dizajnerima i testerima aplikacije koji osiguravaju održavajuću funkcionalnost aplikacije. Još jedna uloga koja je bitna u tom timu je sama podrška prema krajnjem klijentu koja predstavlja podršku u tehničkom smislu, kao direktna pomoć pri postavljanju aplikacije ili upravljanju aplikacijom, no njihova funkcija je također i funkcionalna, pošto su oni ti koji iz prve ruke dobiju informacije o načinu

na koji klijent koristi aplikaciju te probleme na koje se susreće u radu sa aplikacijom. Tako bi tehnička podrška trebala biti most informacija između klijenta i razvoja, odnosno podatak o tome gdje bi se budućnost razvoja aplikacije trebala kretati poznavajući u potpunosti način na koji aplikacija radi, sve funkcionalnosti koje postoje te način na koji ih klijenti koriste. Tehnička podrška može se pružiti kroz nekoliko različitih kanala, od kojih svaki ima svoje prednosti i mane:

- **Uživo** - ovo je većini klijenata najdraži način korištenja tehničke podrške zbog same prirode opuštenosti kada se za dogovoreni sastanak uspiju pripremiti sa listom pitanja o načinu funkcioniranja aplikacije, njezinim mogućnostima i zahtjevima koje kroz svoj način korištenja imaju. Zbog samog problema logistike ovo je tip podrške koji se ne prakticira često, no slučaj u kojem se koristi je za edukaciju novih korisnika sustava koji su u blizini sjedišta kompanije, ili ako se radi o većem klijentu sa više korisnika zbog vrijednosti koju može pružiti kao dugoročni klijent dogovori se višednevni *onboarding* proces direktno u prostorima gdje posluje klijent.
- **Online sastanci** - iako ne može zamijeniti direktni kontakt, i dalje predstavlja dobar supstitut, pošto se u dogovoreno vrijeme putem jednostavnih alata za video sastanke (najčešće Google Meet zbog jednostavnosti pristupa video pozivu bez instalacija programa) može odraditi sve što se može i sastankom uživo. Tako osim štedeći vrijeme i putne troškove što poduzeću može osloboditi mnogo resursa, online sastanci doprinose ekološkoj održivosti kroz reduciranje korištenja goriva i drugih oblika energije u tom procesu. Čest je slučaj da se nakon edukacije uživo nakon što klijent zapravo počne koristiti aplikaciju tjedan kasnije vrati sa puno pitanja, tako da se odmah dogovori online sastanak da se korisnik pripremi sa svim pitanjima koje ima za korištenje, tako da nakon edukacije i sastanaka bude što manje potrebe za dodatnom pomoći. To zapravo i je bitna stavka pri pružanju što bolje i kvalitetnije podrške pri početku korištenja sustava, kako bi proces kasnije bio jednostavniji i klijentu (da ne treba stalno tražiti pomoć za svako pitanje ili zahtjev) te pružatelju informacijskog sustava (da oslobodi resurse za podršku u budućnosti), no ponekad to nije moguće ili zbog neisplativosti ulaganja u nekoliko sati ili dana edukacije nekoliko radnika poduzeću, ili sa klijentske strane jer možda ne može izdvojiti toliko vremena ili kao u nekim slučajevima, jednostavno nije zainteresiran u tom trenutku za sam proces integracije u novi sustav, iako im taj sustav može biti i glavni alat kojeg koriste za kompletno upravljanje poslom koji odrađuju.
- **Telefonski** – korisnici najčešće preferiraju ako služba za podršku ima direktnu liniju bez čekanja u redu za poziv. Još ako se radi o podršci koju mogu zatražiti 24/7, većina klijenata će to zagovarati kao najvrjedniju funkciju koju im informacijski sustav pruža. Neki će čak riskirati odabir softvera

sa manje dostupnih funkcija što im predstavlja više manualnog rada samo kako bi dobili uvijek dostupnu podršku, što dokazuje koliko je dobar sustav podrške zapravo bitan. Dodatna stvar kao dobru podršku korisnici smatraju i kad je agent za podršku uvijek ista osoba, tako da profesionalni odnos stvori prijateljsko ozračje u kojem je korisniku uvijek lakše opisati problem agentu koji već poznaje način na koji je tom korisniku postavljen sustav te funkcionalnosti koje koristi za brzi pronalazak rješenja upita. Poduzeću je ovaj oblik podrške nije idealan iz nekoliko razloga:

  - Nema pisani trag problema pa se ne može pratiti da li je to neki specifičan slučaj koji se dogodio samo jednom ili se radi o češćem slučaju da više korisnika ima problema sa istom funkcijom da se poduzmu neke korektivne akcije u smislu poboljšavanja te funkcije u aplikaciji. Još jedan veliki problem sa nepostojanjem pisanog traga je to što ako nakon pružene pomoći prođe neko vrijeme te se klijent požali na nešto što mu je podrška napravila u sustavu, nema dokaza da je klijent to zapravo tražio. Tako, na primjer, klijent u jednom slučaju nije znao upisati novog zaposlenika u sustav, a uz zaposlenika se veže OIB koji se upisuje prilikom kreiranja računa u softveru koji se šalje prema FINA-i kako bi se fiskalizirao račun u ime korisnika koji je kreirao račun. Klijent je inzistirao da podrška telefonski napravi račun za tog korisnika, ali podrška je tražila da se podatak o OIB-u dostavi na mail, što je klijent na kraju i dostavio. Nakon par mjeseci klijent se požalio na dolazak porezne inspekcije koja im je izrekla kaznu zbog krivog OIB-a sa kojim je djelatnik izdavao račune, no podrška je imala dokaz da im je krivi OIB dostavljen u mailu u prvom redu od strane klijenta, udaljavajući se od krivnje za nastali trošak zbog greške. Ovakav primjer se dogodi često na sličnu varijantu, te je iz tog razloga veoma važno imati pisani trag o svakom zahtjevu.
  - Teško je pratiti koliko je agent imao poziva, koliko su pozivi trajali uz dodatne statistike koje mogu poslužiti kao informacije isplati li se takvu podršku pružati, treba li zaposliti dodatni kadar, koliko je kvalitetna pružena pomoć i slično. Kako bi se to implementiralo, trebao bi se u potpunosti redizajnirati i personalizirati poslovni proces telefonske komunikacije, što priliči većim firmama sa podrškom poput telekoma, no za manja poduzeća sa nekoliko zaposlenih u podršci to ne predstavlja idealno rješenje.
- **Sustavi za daljinsku podršku** - ovo je način pružanja tehničke pomoći koji podiže telefonski način komunikacije na najkvalitetniju razinu. Ponekad pisani način jednostavno nije dovoljno dobar za opisati problem te može uzeti previše vremena čekanja na međusobni odgovor između klijenta i

podrške, te je u tom slučaju daljinska podrška najbolji alat. Danas su alati poput Anydeska ili Teamviewera veoma jednostavni za instalirati na računalo, te je čak i korisnicima sa gotovo nikakvim tehničkim znanjima vrlo jednostavno objasniti kako to napraviti jednom, te im onda zauvijek lakše pomoći nego objašnjavajući telefonski gdje što kliknuti u aplikaciji kako bi riješili problem. U zadnje dvije godine trend korištenja ovih aplikacija u Hotel's Touchu je toliko narastao da klijenti čim se jave telefonski samovoljno izjave pristupne podatke za daljinsku podršku zbog jednostavnosti opisa problema i međusobnog pronalaska problema kako obje strane vide isti ekran te mogu klikati bilo gdje na ekranu kako bi zajedno riješili problem.

- **Mail** - najuobičajeniji lanac komunikacije i traženja pomoći kako svi na uređajima imaju email aplikaciju preko koje jednostavno mogu opisati problem i priložiti sliku, tablicu ili neki drugi medij koji je nužan pri opisu problema. Najveća prednost maila je što se u *CC* može staviti nekog aktera koji ne koristi aplikaciju, ali su povezani sa problematikom maila o kojem se raspravlja. To su najčešće računovodstva, viši voditelji poduzeća klijenta, kupci usluga koje klijent prodaje ili zaposlenici drugih firmi sa kojima je klijent također povezan za neku drugu funkciju kao npr. kanali prodaje rezervacija koji su zainteresirani za spajanje na program i slično. Baš ta međusobna povezanost direktno nepovezanih aktera je moguća jedino kroz mail te je tako jedina mogućnost za neke specijalne slučajeve traženja pomoći da svi imaju pristup informacijama koje se dijele po tom problemu. Za sve ostalo, email je loša alternativa drugim oblicima regularne podrške jer nigdje u sustavu ne ostaje zapis tko je zatražio pomoć i kad, za koju aplikaciju, na koji način je podrška uspjela ili nije uspjela pomoći, odnosno podaci stoje u jednom sandučiću te nisu centralizirani u nekoj bazi podataka za cijelo poduzeće. Također, u neradno vrijeme zaposlenika ili na godišnjem odmoru klijent mora tražiti mail adresu druge osobe u podršci te nema konzistentnosti gdje, kada i kod koga može zatražiti pomoć te koliko će trebati da dobije povratnu informaciju o svom zahtjevu. Sve to otežava posao i podršci kad se zahtjev prebacuje sa niže razine podrške na višu za neke kompleksnije probleme, jer je jedini način za to napraviti proslijediti kompletni sadržaj maila koji može biti neorganiziran sa nepotrebnim informacijama i različitim dizajnima koristeći različite mail aplikacije smanjujući preglednost i gubeći priloge koje neke mail aplikacije nemaju tehničke mogućnosti proslijediti.
- **Ticketing sustav** (engl. Ticketing system)- pružanje podrške kroz tickete je najorganiziraniji sustav za upravljanje upitima, kako dopušta da svaki problem bude pravilno zapisan i dokumentiran zauvijek u vlastitu bazu podataka, kategoriziran po aplikaciji, korisniku i agentu koji pruži podršku, te omogućuje praćenje od inicijalnog zahtjeva pa do rješenja. Tako kvalitetan

ticketing sustav omogućava postavljanje prioriteta i upravljanje s više ticketa istovremeno te sprječava da neki problem ne bude u potpunosti riješen (kao propušteni poziv ili zaboravljeni mail) i tako osigurava i pripisuje odgovornost na pojedinog agenta za podršku. To omogućuje svim sudionicima uvid u status upita te praćenje napretka kao informaciju za optimizaciju radnih procesa i poboljšanje korisničkog iskustva. Ticketing sustav ima i svoje mane, u prvom redu zbog vremenskog kašnjenja između svakog odgovora *supporta* te povratno klijenta. Pa čak i u slučaju kad jedan i drugi odgovaraju jedan drugome unutar minute, to i dalje nije chat model u kojem se poruke automatski osvježe te se može izgubiti znatno vrijeme do pronalaska jednostavnog rješenja. Drugi je problem samo postavljanje ticketing sustava, koji je u primjeru Hotel's Toucha bio puno jednostavniji nego što bi to bio u nekom drugom poduzeću zbog biranja pravog vremena za prijelazak. Još jedan problem je prebacivanje običaja postojećim klijentima sa starih načina zatraživanja pomoći na ticketing sustav, no više i o tom u nastavku rada.

- **Chat podrška** - sve popularniji način pružanja podrške je kroz chat, gdje zbog samog tehničkog formata responzivnosti poruka bez osvježavanja stranice ili odlaskom na drugu web stranicu koja je dizajnirana za pružanje pomoći chat omogući interakciju u stvarnom vremenu između klijenta i podrške. Ovo je klijentima najdraži pisani način traženja pomoći, gdje najčešće putem Whatsapp aplikacije koju svi imaju šalju zahtjeve koje imaju, što poduzeću opet nije idealno jer nema nikakav uvid u problematiku koja se rješavala putem takvog načina pružanja pomoći kroz neku vanjsku aplikaciju. Brzina pomoći koju chat podrška pruža je najbolji alat za jednostavne upite koji ionako predstavljaju većinu ukupnog broja upita, dok je za kompleksnije upite bolji ticketing ili telefonski sustav. Najveća mana chat podrške je to što može biti dostupna samo tijekom radnih sati, dok je ticketing sustav dostupan stalno što daje podršci mogućnost da odgovore čim postanu dostupni. Idealno rješenje je ono koje dozvoljava poduzeću da ima chat sustav unutar aplikacije koji automatski kreira ticket te tako koristi neki hibridni oblik između chata i ticketinga.
- **Forum** - još jedan zanimljiv način pružanja podrške koji treba spomenuti koji koriste neka veća konkurentska poduzeća su otvoreni forumi gdje korisnici ne pitaju pitanja samo podršku, već i sve druge korisnike koji imaju korisnički račun u toj aplikaciji. Ovo predstavlja veoma kvalitetan alat poduzeću iz razloga što korisnici mogu samostalno pronaći rješenje tražeći pomoć od drugih korisnika te tražeći već postojeće teme za slične probleme drugih korisnika u prošlosti i samim time oslobađanjem resursa poduzeću. Korisnici su otvoreniji pri opisu svojih problema i potencijalnih frustracija u programu što omogućuje dodatan uvid u moguća poboljšanja.

Problemi sa forumima su što poduzeće mora imati popriličnu bazu korisnika kako bi neka aktivnost foruma bila održavana te potrebna moderacija komentara protiv zloupotrebe jednako kao i potencijalni rizik od negativnih komentara i stvaranja loše slike o poduzeću ako se aplikacijama i sadržajem ne upravlja kvalitetno.

Biranje načina pružanja vrste podrške ovisi o mnogim faktorima poput veličine poduzeća, količine klijenata i broja upita koje imaju te raspoloživim resursima odnosno vremenom koje podrška može priuštiti, no najvećim dijelom do složenosti problema u prosjeku te personalizacijom koliko se želi i može vremena posvetiti nekom klijentu, što dugoročno dovodi do zadovoljnog klijenta kao najvećeg resursa. Tako i u slučaju Hotel's Toucha se prvih godina postojanja koristio ekskluzivno telefonski i mail način pružanja podrške što je bila dovoljno dobra opcija pošto je jedna osoba mogla odrađivati cijeli posao za svoju aplikaciju, te tako za svaku aplikaciju zasebno po jedna osoba. Pri inicijalnoj izradi aplikacije Front desk za upravljanje recepcijskim sustavom koristila se podrška uživo zbog vlastite potrebe razvitka aplikacije, gdje su projekt menadžer i programer kod prvog klijenta sjedili na recepciji promatrajući iz prve ruke kako bi aplikacija trebala funkcionirati, kakve probleme rješavati te koje su funkcije najpotrebnije i najkorištenije tako da se njima doda najveća važnost pri izradi da budu što jednostavnije pri korištenju. Kako se aplikacija razvijala, razvijale su se i druge aplikacije u domeni Hotel's Toucha, ukupna baza klijenata je eksponencijalno rasla te je dijeljenje podataka između članova tima postajala sve kompliciranija. Pošto se radi o velikom broju aplikacija i klijenata, a svi podaci su decentralizirani potreba za sustavom koji to sve objedinjava je rasla sve više.

4.2. Potreba za promjenom - planiranje

Prije samog kreiranja nekog novog softverskog proizvoda dolazi pitanje što treba napraviti i zašto. To je pitanje potrebe za kreiranjem nekog novog sustava te se inače sagledava kroz kreiranje novog, implementaciju ili nadograđivanje postojećeg te se događa prirodno kako poduzeće traži kapital sa kojim će pokriti trošak postojećih resursa ljudskog potencijala koje ima na raspolaganju. U tom slučaju stvar o kojoj se mora razmišljati je koliko će se vremena zaposlenog radnika na tom projektu uložiti i da li se isplati ići u taj pothvat. No u slučaju kad poduzeće kreira neki softver za vlastite potrebe poput sustava za podršku tu dolazi u pitanje potrebe za promjenom na prvom mjestu, gdje prvo treba biti svjestan da neki proces nije dovoljno efikasan te da treba krenuti u planiranje izgradnje novog sustava po tom pitanju. U slučaju Support aplikacije to je baš promjena za centralizacijom informacija i platforme za zatraživanje pomoći. Kad postoji više različitih aplikacija koje koriste različiti profili kupaca potreba za centralizacijom podataka o tome sa čim korisnici imaju problema je sam po sebi dovoljan razlog da se ustanovi kako u sustavu treba neka promjena kako bi se te informacije koristile na strateški način. Tako se može doći do saznanja o potrebi za promjenama u aplikacijama po pitanju dizajna, funkcionalnosti, implementacije ili bilo kojeg načina korištenja sustava. Bilo koji konzultant, programer ili projekt menadžer koji su dio stvaranja i održavanja tih aplikacija mogu smatrati da znaju način na koji se aplikacija koristi, ali su korisnici ti koji na kraju pružaju informaciju valja li sustav te što treba promijeniti. Dogode se slučajevi u kojima korisnik je svjestan koliko puno funkcionalnosti aplikacija ima te koliko su vrijedne, ali naglasi kako su veoma kompliciranе za pristupiti i intuitivno koristiti tu funkciju izjavljujući kako primijete da su to radili programeri koji nemaju iskustva u tipu posla koji korisnik obavlja. Poslovni korisnik je onaj koji tu aplikaciju koristi po osam sati dnevno te je kroz praktično iskustvo jedini koji može pružiti informaciju što su mu problemi sa postojećim funkcijama ili koja je dodatna funkcija koju im treba napraviti te je baš iz tog razloga važno imati sustav u kojem će svi članovi tima razvoja aplikacije imati pristup na jednom mjestu svim informacijama koliko često se korisnici žale na neke funkcije pronalazeći tako potrebu za poboljšanjem tog dijela aplikacije. Tako sama potreba za kreiranjem aplikacije za centralni pristup informacijama upita podršci stvara i potrebu za poboljšanjem u svim aplikacijama dajući uvid u najčešće probleme koji bi se trebali popraviti u budućnosti ili trenutnim bugovima koje treba riješiti što hitnije. Još jedan važan faktor je da poduzeće ima sve podatke na jednom mjestu umjesto na nekoliko različitih nepovezanih mail sandučića i chat poruka u bazi podataka te tako pristup povijesnim podacima te vizualizacijama i statistikama koje može dobiti iz tih podataka. Također, bitno je spomenuti i potrebu za poboljšanjem efikasnosti pružanja usluge podrške te upravljanja ljudskim

resursima koji tu podršku pružaju. Kako poduzeće raste zajedno sa brojem novih klijenata i njihovih korisnika, tako raste i potreba za boljim upravljanjem upita podršci, te je nemoguće očekivati da se sa postojećim procesom neorganiziranog pružanja podrške kroz veliki broj nepovezanih kanala (mail, chat i pozivi od svakog agenta podrške zasebno) može očekivati da će budućim rastom i zapošljavanjem dodatnog kadra taj problem biti jednostavniji za riješiti, pošto u pitanju nije samo proces kreiranja aplikacije za podršku, već i razdoblje privikavanja kako zaposlenih tako u još većem redu privikavanja klijenata. Sama ideja pristupa dijeljenoj aplikaciji za isti zadatak pružanja podrške koji obavlja više zaposlenih je i mnogo bolja opcija za te zaposlene, pošto svi mogu priskočiti u pomoć ili prvi uzeti kreirani ticket te tako ponuditi i klijentu bolju podršku u odnosu kad klijent pošalje mail jednoj osobi iz podrške koja u tom trenutku može nedostupan za pružanje pomoći. Baš ta sigurnost i dijeljenje informacija pruža zaposlenima u podršci veću sigurnost i fleksibilnost da će upit klijenta biti riješena u brzom roku i kad oni osobno nisu dostupni, zato što bilo tko drugi može preuzeti ticket u dovoljno brzom roku da klijent bude zadovoljan. To zapravo stvara važnu konkurentsku moć, jer po riječima samih klijenata za aplikacije koje Hotel's Touch nudi dvije glavne stvari koje spominju kao glavne prednosti koje dobivaju su veliki broj funkcija koje aplikacije imaju te brzina i kvaliteta podrške, što su dvije točke koje moraju biti u simbiozi jedno sa drugim, pošto mnogo funkcija u sustavu ga čini kompliciranijim za korištenje te je prirodno da će ne samo novi, već i dugoročni klijent imati pitanja kako koristiti aplikaciju, te tako povećati ukupan broj upita podršci. Sve su to razlozi koji su stvorili važnu potrebu za promjenom u poslovnom procesu pružanja podrške.

4.3. Postavljanje zahtjeva - analiza

Nakon što je kolektivno utvrđeno da postoji potreba za kreiranjem sustava za pružanje podrške, zadatak je postavljen timu da kreira opću ideju kakva bi to aplikacija trebala biti i što bi trebala sadržavati. Za ostale do tada već kreirane aplikacije to je predstavljalo kompliciraniji proces pošto se radilo o prikupljanju zahtjeva kako se obavlja neka vrsta posla (restoranski, hotelski, evidencija radnika) te po tome osmisliti ideju kako bi aplikacija riješila poslovne probleme drugim poduzećima, dok je za Support aplikaciju situacija puno jednostavnija pošto se radi o jednostavnoj funkciji pružanja podrške na jednom mjestu, bez kompleksnih funkcija ili prilagođavanja tipu poslovanja za čije korištenje pruža uslugu korištenja aplikacije. Tako je u prvom redu glavno pitanje bilo koje su to funkcije koje Support aplikacija mora imati i kako ih implementirati. Na sreću, takva aplikacija je temelj svake veće organizacije pa se jednostavnim pretraživanjem nekoliko pružatelja podrške kroz ticketing sustav uspjelo doći do jednostavnog rješenja kako su glavne funkcije koje aplikacija mora imati kreiranje ticketa, odgovaranje na tickete te obavijesti korisnicima i podršci. Kako se radi o manjem poduzeću sa ukupno 10-15 zaposlenih gdje su svi u osnovnoj mjeri upoznati sa svim aplikacijama, bez obzira koju aplikaciju samostalno razvijaju nije bilo potrebe za intervjuima ili grupama za raspravu o detaljnošću kakva bi aplikacija trebala biti. Proces postavljanja zahtjeva nije bio toliko dugotrajan kako se radi o timu koji je u potpunosti prilagodljiv te se vraća u bilo koju fazu kreiranja novog sustava kad god nastane potreba za tim. Zato se sa jednostavnom idejom sustava gdje korisnik može doći i kreirati ticket, a podrška odgovoriti na njega odmah krenulo u izradu tog sustava uz svijest da će se to morati obogatiti još nekim funkcijama naknadno, pošto samo ažuriranje nove funkcije u postojećem sustavu je puno jednostavniji posao nego kreiranje temelja aplikacije. Tako se definiranjem glavne funkcije krenulo u raspravljanje kako bi cijeli proces trebao izgledati kad bi aplikacija postojala - klijent uoči problem, na najjednostavniji mogući način ju prijavi kroz Support ticket, svaki pružatelj podrške za tu aplikaciju vidi ticket, jedna osoba preuzme ticket, po potrebi ga prebaci na drugog supportera i dodjeli joj prioritet rješavanja, odvija se komunikacija između podrške i korisnika te nakon što je problem riješen, ticket se može zatvoriti ili od strane korisnika ili od strane podrške.

Tako su ustanovljene varijable:

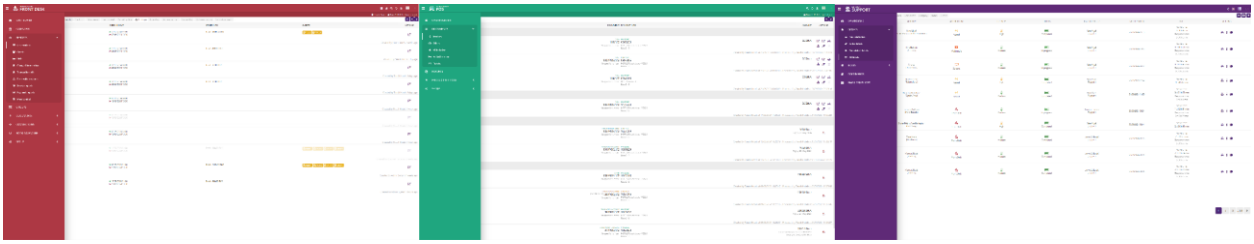
- Mora postojati hijerarhija korisnika, odnosno netko mora biti podrška, a netko mora biti korisnik te se treba smisliti gdje će se postaviti tko ima koja prava.
- Ticketi imaju status jesu li dovršeni ili nisu te tako mora postojati informacija vidljiva klijentu i podršci da znaju je li ticket preuzet ili nije, da li je u procesu rješavanja i na kraju da li je gotov i što se događa sa ticketom nakon što je dovršen.
- Ticketi moraju imati mogućnost postavljanja prioriteta od strane podrške u prvom redu sebi da se zna što je bitnije za riješiti, a kao drugo klijentu ako npr. dobije status “hitno” da je svjestan kako se njegov problem pregledava i rješava što je prije moguće.
- Ticket mora prikazivati aplikaciju za koju se odnosi i korisnika koji mu je dodijeljen za rješavanje.

Tako je nakon analizirane potrebe zahtjeva koje aplikacija treba ispuniti kroz međusobnu raspravu između cijelog Hotel’s Touch tima postavljen temelj cilja aplikacije koja će biti Support, te se tako moglo krenuti dalje u planiranje drugih varijabli, najbitniji od kojih su uobičajeno ljudski resursi i rokovi. Kako se u ovom slučaju radi o aplikaciji čije nepostojanje do tog trenutka nije značilo da se treba što prije napraviti kao neka kritična točka potrebe za promjenom u tom trenutku, već sama želja za poboljšanjem trenutnog procesa za osiguranu budućnost, ta činjenica je dovela do toga da nije postavljen nikakav vremenski rok u kojem ona mora biti postavljena i u funkciji što je uvelike pomoglo prilikom izrade, pošto su članovi tima zaposleni na drugim projektima, te bi ovo trebao biti zadatak sa strane za trenutke kada na dodijeljenim važnijim projektima nema posla pa da se to slobodno vrijeme iskoristi za razvoj aplikacije za poboljšanje procesa u budućnosti. Tako je odlučeno da jedan programer samostalno razvija aplikaciju uz projekt menadžera koji je ujedno i tester, te da se ostali članovi tima pridruže po potrebi u procesu razvijanja sa testiranjem, dodatnim idejama, pomoći u pisanju koda i slično.

4.4. Dizajn sučelja

U doba kada je jednostavan i pregledan dizajn korisničkog sučelja i ugodnost samog izgleda aplikacije standard kao neko minimalno očekivanje aplikacijama namijenjenim krajnjem korisniku, ista očekivanja se prelijevaju i na aplikacije za poslovne korisnike. Iako je za poslovne potrebe najvažnija funkcionalnost i pouzdanost aplikacije koju će koristiti cijeli radni dan gdje sam izgled aplikacije ne predstavlja toliko važan faktor nakon razdoblja privikavanja na novi sustav, direktna konkurencija jednako kao i mijenjajući trendovi u nekonkurentnim informacijskim sustavima jednostavno zahtijevaju redizajn barem na neku minimalno zadovoljavajuću razinu, a nakon toga konstantnu adaptaciju zbog promijena u navikama korisnika i trendova u dizajnu. Korisnik će se često samostalno javiti kako mu je neka funkcija kompliciranа, nije dobro objašnjena u uputama ili nepristupačna, stoji na krivom mjestu, treba previše klikova da do nje dođe ili bilo koji od drugih sličnih problema koje loš dizajn stvara. Glavna dva pitanja o dizajnu kad se radi o izradu jednostavne aplikacije poput Supporta su izgled korisničkog iskustva (UX) i korisničkog sučelja (UI) ni jedno od kojih nisu stvarali preveliko razmišljanje u Hotel's Touch poduzeću pri donošenju odluke o postavkama dizajna. Naime, glavna funkcija UX dizajna u ovom slučaju je da korisnik odmah klikom na Support aplikaciju bude odveden na početnu stranicu gdje ima listu svih svojih aktivnih ticketa te klikom na ikonu desno može kreirati novi ticket, a na lijevoj strani ima postavke u kojima može kliknuti za odlazak na listu svih zatvorenih ticketa u prošlosti. Cilj UX dizajna je ovdje smanjiti količinu informacija kako bi korisnik na najjednostavniji način dobio informacije koje je zatražio otvaranjem ticketa za pomoć te da su samo aktivni neriješeni problemi u fokusu na toj stranici. Za dizajn UI sučelja također nije bilo velikih rasprava, pošto Hotel's Touch aplikacije već posjeduju standardizirani dizajn koji dijele između svih aplikacija te se samo razlikuju po boji, dok su preostale funkcije na istim mjestima. To uvelike pojednostavljuje razvoj aplikacije kako se sam dizajn takvog sučelja nalazi u takozvanim programerskim "*assetima*" poduzeća. Imovina, odnosno *asset* bi predstavljao neki softverski artefakt, odnosno skupinu knjižnica koda, podataka ili elemenata korisničkog sučelja koji je kao takav namijenjen za višekratnu uporabu tijekom nastanka, razvoja ili poboljšanja softverskog proizvoda (Zabardast i ostali, 2022). Tako predstavlja vrijednost za poduzeće kao neke vrste neopipljive imovine baš iz razloga što se posjedovanjem takvog alata može uštediti mnogo resursa prilikom izrade drugih aplikacija tako da se dijelovi koda ne moraju pisati iznova, već postoje kao neki temelj na kojem se razvoj aplikacije može bazirati. *Asseti* su tako dio već spomenutog upravljanja bazom znanja KM koji dovodi boljoj održivosti sustava, kako poduzeću uštedom vremena, tako i globalno kroz uštedu utrošene energije. Baš ti postojeći asseti su uvelike olakšali posao samog dizajna Support aplikacije pošto su se dijelovi dizajna, ali

i nekih funkcionalnosti mogli samo prekopirati uz preinake uz kodu dok je ostatak bio samo prilagođavanje postojećeg kao varijacije na temu što je postojećim klijentima već veoma poznat dizajn. Slika ispod tako prikazuje primjer glavnog menija Front Desk aplikacije lijevo, POS u sredini, te Support desno gdje se lako primijeti kako dijele isti dizajn, samo druge funkcionalnosti za svaki meni:



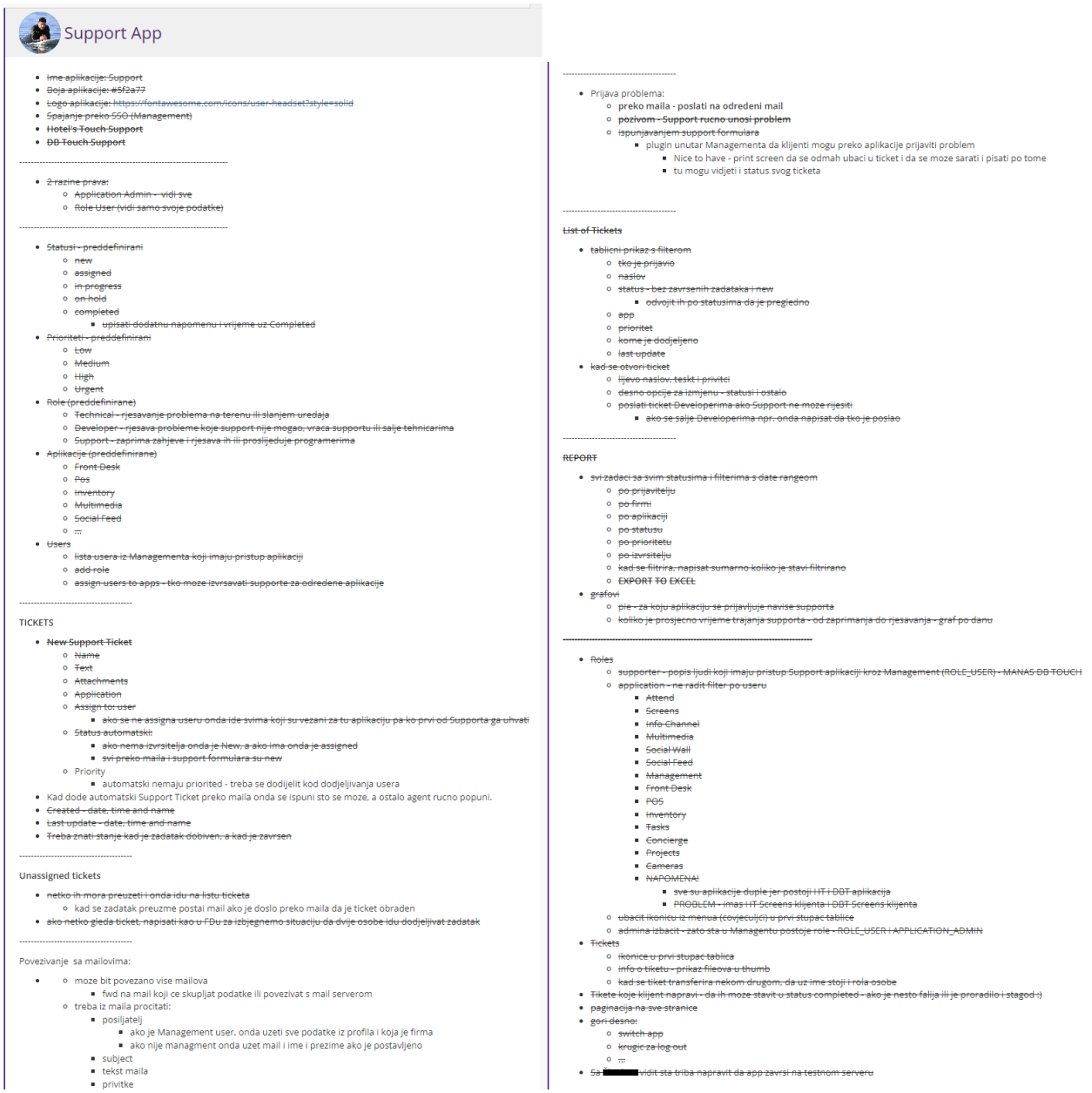
Slika 6 - Usporedba dizajna različitih Hotel's Touch aplikacija

Izvor: izrada autora

Pri recikliranju korištenja i upravljanju assetima treba biti pažljiv kako je moguće da tehnologija na kojima je baziran njihov kod ili struktura zastari te je potrebno ažuriranje dijelova asseta na svim mjestima gdje se koriste. U ovom slučaju to bi baš predstavljao UI/UX dizajn koji je po današnjim standardima zastario te će u bližoj budućnosti zahtijevati razvoj na moderniji izgled te će tako to biti nužno promijeniti na svim aplikacijama gdje se koristi kako bi se zadržala unificiranost. To jednako predstavlja projekt sam po sebi koji mora proći kroz sve faze razvoja slično kao i Support, no iako to nije funkcija koja ima vremenski rok ili hitnu potrebu za promjenom, njezin razvoj bi definitivno poboljšao neke poslovne procese poput jednostavnije edukacije i prilagođavanja novih korisnika u sustav.

4.5. Razvoj i testiranje

Najintenzivnija faza izrade novog sustava je baš dio razvoja i testiranja, prvo iz same ideje da se mora iznova kreirati neki temelj na kojem se onda nadodaju nove funkcije (ali pojednostavljen korištenjem asseta u slučaju kad su dostupni), no i zbog potrebe za povezivanjem na postojeće baze podataka gdje se prilikom razvoja može uvidjeti ako neki postojeći kod nije optimalan ili zauzima previše resursa. Baš je to bio slučaj pri izradi Supporta i spajanja na bazu podataka svih korisnika gdje se otkrio bug kad bi u pretraživanju svih korisnika u sustavu svaka pretraga prilikom upisivanja svakog slova iznova slalo zahtjev prema bazi za učitavanjem podataka te tako nepotrebno koristila ogromne hardverske resurse čineći druge aplikacije sporijima u trenucima kad se to pretraživalo. Zato prilikom razvoja i testiranja treba obratiti pozornost da ako se baza podataka dijeli sa drugim aplikacijama da ne troši resurse i zagušuje normalni tijek korištenja drugih aplikacija kako je bilo prije poveznice na novu aplikaciju. Prilikom inicijalnog razvoja aplikacije bitna je komunikacija, ali i dobro definiran plan što točno treba napraviti i gdje. U nekim slučajevima to mogu biti vizualni pregledi, gdje se dizajnira opća ideja kako bi koja funkcija trebala izgledati, no za jednostavnije funkcije dovoljne su samo tekstualne upute te u slučajevima kad programer zapne sa idejom kako bi nešto trebalo izgledati kroz međusobnu komunikaciju svih prisutnih odlučiti što se može napraviti te što je od dostupnog najbolja opcija. Tako se za vrijeme razvoja Supporta koristila interna Projects aplikacija za jednostavno pisanje postova u kojima su stajale sve potrebne informacije o ideji i zadacima što se mora napraviti prema inicijalnoj ideji, te naknadno dodavanje novih funkcija kako bi razvoj dopuštao testiranje pa iz tog nastanu nove ideje.
Tako je na slici u nastavku programeru tekstualno opisan inicijalni plan što aplikacija mora imati i na koji način funkcionirati:



Slika 7 - Lista zahtjeva funkcionalnosti aplikacije programeru

Izvor: izrada autora

Jednostavan način kvalitetno pružene dokumentacije je veoma važan faktor pri izradi ili poboljšanju sustava, što danas mnoge aplikacije za projekt menadžment pružaju na slikovitiji način kroz pristupačnije UX sučelje za sve članove u procesu. Danas se za razvoj Hotel's Touch aplikacija koristi aplikacija JIRA (više u budućem poglavlju), no i dalje važniji faktor ostaje efikasno koordiniran tim i kvalitetno predočeni zahtjevi u dokumentaciji, što je u ovom slučaju još dodatno potpomognuto činjenicom da se radi o

zaposlenicima koji su dugi niz godina u poduzeću i imaju predodžbu o svim aplikacijama i arhitekturi na kojoj su zasnovane tako da je donošenje odluka i zajednički pronalazak rješenja pri razvoju puno jednostavniji.

Testiranje je iznimno važna funkcija dobrog razvoja, te u većim razvojnim timovima i poduzećima predstavlja drugu vrstu razvoja samu po sebi, gdje se razvijaju automatizirani testovi, test case-ovi ili testni slučajevi kroz točno opisane korake što se mora testirati i koji su očekivani rezultati kako bi se test smatrao uspješnim. U slučaju Supporta potrebe za razvojem tako kompleksnog sustava testiranja nije bilo, pošto je cijeli razvojni tim upoznat sa tim kako bi aplikacija trebala funkcionirati te na koji je način povezana sa drugim aplikacijama i samim time svakom promjenom u kodu za određenu funkciju znalo se točno na što se promjena odnosi te što treba testirati. Kad je programer sam svjestan da se radi o nekoj promjeni koja bi mogla indirektno utjecati na nešto drugo u sustavu, proslijedi informaciju testeru da dobro obradi pozornost te testira sve moguće funkcije na svim povezanim mjestima kako ne bi došlo do nekog buga koji je promakao. Tako su razvoj i testeri u konstantnoj komunikaciji te zajedno traže probleme i dizajniraju nove ideje kako poboljšati inicijalni plan funkcije aplikacije nakon što su svi postojeći zadaci obavljeni. Tako se inkrementalno razvija jedna po jedna funkcija, deploya, testira, pa se lista zadataka puni sa novim idejama koje su osmišljene putem tog procesa. Zato je ovo najduža faza pošto se stalno prebacuje iz razvoja, pa u testiranje, pa nazad u razvoj, ili po potrebi nazad u dizajn pa ponovo u razvoj sve dok se ne postigne željeni rezultat. Dogode se i optimistične prognoze za neke funkcije u programu koje u toku razvoja programer shvati kao prekompleksne za izradu gdje se onda ponovo prioritizira da će se ta funkcija napraviti ili postaviti u listu “good to have” zahtjeva u budućnosti, pošto se na početku prioritizira sama izrada osnovnih funkcija aplikacije. Tako je prilikom izrade Supporta testiranje bilo u potpunosti manualno uz dijeljene informacije što i kako testirati, dok se dvije godine nakon razvoja raspravlja i o poboljšanju tog procesa automatizacijom nekih dijelova testiranja pisanjem skripti za automatsko testiranje od strane računala, razvoj kojih također predstavlja investiciju resursa u poboljšanje poslovnih procesa kroz sve iste faze ne zbog prijeke potrebe danas, već iz pojednostavljivanja procesa u budućnosti uklanjajući ljudsku grešku pri manualnom testiranju i smanjujući vrijeme potrebno za testiranje funkcija koje se ponekad svodi na uvijek isti zadatak, što može biti demotivirajući zadatak za testera.

4.6. Isporuka i pregled

Isporuka se rijetko kad u razvoju može shvatiti kao dostava finalnog softverskog proizvoda, jer i u slučaju Supporta kao i na drugim aplikacijama klijentu se dostavlja dio po dio proizvoda, baš kao odnos razvoja i testiranja. Pošto Support aplikacija nije projekt čiji je razvoj klijent zatražio, već sama potreba poduzeća za postavljanje za korištenje postojećim klijentima, onda se isporuka može gledati kao trenutak kada je proizvod postao dostupan za korištenje u okruženju svih postojećih korisnika. To uključuje postavljanje aplikacije na server i osiguravanje da sve radi kako bi trebalo prije početka stvarnog korištenja. Najvažniji dio isporuke u ovom smislu više nije sami razvoj aplikacije, već njezino promoviranje i skupljanje prvih korisnika da zapravo kroz stvarne primjere počnu koristiti aplikaciju kako bi se došlo do ideja za nove zahtjeve što odmah treba popraviti, pošto takva prva verzija proizvoda nikad neće biti savršena, već se uvidom kroz praktično korištenje dobiju zahtjevi za povratkom u prošle faze razvoja kako bi se isporučio funkcionalan proizvod. Tako nakon što je aplikacija postavljena na servere dogovoreno je kako će se svim klijentima poslati informativni mail o tome kako je u sustavu dostupna nova aplikacija, zajedno sa kratkim uputama čemu služi te kako se njome koristiti. Očekivano je kako važnu funkciju poput novog načina zatraživanja podrške postojeći korisnici neće odmah prihvatiti, pa su zato kao i pri razvoju bilo kojeg proizvoda early adopteri važna stavka za otkrivanje problema i kontinuirano poboljšanje kako bi prijelaz preostalim korisnicima bio jednostavniji i poželjniji od trenutnog načina. Osim isporuke softverskog proizvoda, češće se za isporuku govori o nadogradnji postojećeg sustava, pošto se funkcionalan proizvod isporučuje jednom, ali njegova ažuriranja se kontinuirano promjenama koda isporučuju na servere, bilo male ili velike promjene, što je temelj agilne metodologije. Faza isporuke i pregleda u tom slučaju može biti isti u smislu da se korisnicima javi kako je nova funkcija sada dostupna ili da je postojeći problem otklonjen. Ako je u pitanju manja promjena to nije poželjna praksa kako se ne bi zatrpavalo korisnike nepotrebnim porukama i podacima. To sve u početku postojanja aplikacije spada pod fazu isporuke dok se svi inicijalni problemi same funkcionalnosti ne riješe, a onda kad kroz neku subjektivnu procjenu izgleda da sve radi kako treba svi idući zahtjevi spadaju pod funkciju održavanja aplikacije kroz cijelu budućnost aplikacije koji prati isti proces kao i isporuka.

4.7. Održavanje

Neko vrijeme nakon što je aplikacija postala aktivna unatoč naporima da se počne koristiti više od dotadašnjih kanali podrške, rezultati korištenja nisu bili kao očekivani. Predstavljena je ideja da se prilikom pružanja podrške na kraju komunikacije korisnike zamoli da to odrade kroz Support i da se odradi kratki intervju zašto umjesto trenutnog načina traženja pomoći ne koriste aplikaciju, nekoliko odgovora je dalo konstruktivnu kritiku kako im je komplicirano da ako žele postaviti neku sliku *screenshota* koji često najlakše opiše problem da to moraju napraviti tako da ju postave kao zaseban prilog ticketu, umjesto mogućnosti da ju samo zalijepe u tekst kako to inače rade u mailu. Čim je problem predstavljen timu, dogovoren je i implementiran novi uređivač teksta čija izrada nije koštala puno vremena, ali je uvelike olakšala posao objašnjavanja problema u tekstu kroz slike, kako klijentima, tako i podršci koja se zapravo bazira na slikovnim uputama. Neki su se zatim požalili da su dodatna dva klika za promjenu aplikacije iz one koju koriste u Support za traženje pomoći za tu aplikaciju previše, te se tako krenulo u razvoj implementiranja kreiranja ticketa unutar svake od aplikacija. Poduzeće je imalo desetke datoteka PDF uputa za aplikacije koje nigdje nisu bile lako dostupne korisnicima već su se slale mailom po potrebi te je tako nekoliko mjeseci nakon došla na red i ideja za ubacivanjem menija “Documents” u aplikaciju za sve dostupne upute za aplikacije na koje korisnici imaju prava koja je također veoma jednostavno napravljena te pojednostavila posao podršci, a i pristup informacijama korisnicima. Sve su to primjeri održavanja aplikacije, koji ponovo izvrte cijeli krug razvoja softvera od potrebe za prilagodbom korisnika na novi sustav, prikupljanjem njihovih zahtjeva kao ideja što se treba napraviti kako bi se željena promjena dogodila, dizajn funkcije kako će na najjednostavniji način korisniku to biti omogućeno, razvoj i testiranje nove funkcije te isporuka i komuniciranje sa klijentima kako su im traženi zahtjevi sad i dostupni. Svaki zahtjev kojeg bilo koji korisnik sugerira se zapisuje u backlog listu svih zahtjeva, te se prioritizira po broju korisnika koji su zatražili sličnu stvar, po hitnosti promjene i kompleksnosti izgradnje funkcije te se zapiše tko su klijenti koji su zatražili te funkcije. Nakon što se tako funkcija izgradi, javi se korisniku koji je to i zatražio, a ako se radi o nekoj važnijoj ili većoj funkciji, ili nakon nekoliko sprintova kad se skupi više malih funkcija onda se pošalje informativni mail svim klijentima o novostima promjena u sustavu. Čest je slučaj da se klijenti povratno jave da izraze svoje zadovoljstvo da se njihove savjete sluša i po njihovim idejama gradi sustav što poboljšava partnerske odnose i tako povećavajući lojalnost sustavu dovodi do najveće prednosti za poduzeće - dugoročno zadovoljnog klijenta. Već je spomenuto kako sustav ima desetak aplikacija za različite upotrebe, te je dugoročni klijent onaj koji nakon nekog vremena korištenja jedne od aplikacija

samostalno pokrene ideju o najmu još jedne aplikacije u paket te je baš iz tog razloga najvažnija kvaliteta kojoj poduzeće može težiti dobra podrška klijentu kroz održavanje kako aplikacija, tako i odnosa sa klijentima. Postojeći klijent poduzeću ne predstavlja nikakav trošak za razliku od traženja potencijalnih novih klijenata čiji je trošak korištenje resursa odjela prodaje, implementacije i edukacije, dok postojeći klijent samostalno stvara dodatnu dobit poduzeću povjerujući sustavu korištenje više aplikacija, a poznavajući korištenje sustava minimizira zahtjeve za pomoći. Baš iz tog razloga je važno obratiti pozornost i na održivost sustava, kako se ne bi dogodila neželjena zagušenja i opterećujući servere smanjila performanse korisnicima. Spominjući efikasnost servera i performansi ovdje se može primijetiti i primjena koncepta održivosti kroz bolje upravljanje kodom čija kvaliteta smanjuje potrošnju energije rada i hlađenja servera podataka. Također, po pitanju održivosti valja spomenuti i samu ideju Support aplikacije za pružanje pomoći i implementaciju, smanjujući tako potrebe za fizičkim posjetima klijentima pri postavljanju i edukaciji u sustavu i štedeći potencijalno potrošeno gorivo i amortizaciju poslovnog auta kao imovine poduzeća na put do klijenta.

4.8. Daljnji koraci i planovi

Pod održavanjem se smatra i pitanje nakon što je aplikacija napravljena koji je njezin plan za unaprjeđenjem i održavanjem u budućnosti. Kad poduzeće ima više projekata i aplikacija u portfelju, onda se tu mora paziti na prioritete i tako odlučiti u kojem slučaju se treba pružiti više pozornosti nekoj aplikaciji, da li je to po broju ukupnih različitih zahtjeva, pa da se u nekoliko sprintova odradi sve što se skupilo do tad, da li je to neka veća funkcija pa će se razbiti u više manjih podzadataka ili po broju traženih zahtjeva za istu funkciju što ju automatski čini važnom. Točnog odgovora na ovo nema, jer su zahtjevi veoma specifični situaciji u kojoj se nađu, te je neko proaktivno rješenje kad se osjeti da se alarmi za uzbunu podignu zapravo najbolja solucija uz zapisivanje svih zahtjeva u backlog i održavanjem dokumentacije da se zna kad je zapravo dobro vrijeme za ažuriranjem aplikacije. Tako u slučaju Supporta, svi manji zahtjevi se zapisuju te svakih nekoliko mjeseci naprave u sustavu, no i dalje stoje ideje većih funkcija poput direktno integriranog chat sustava u svakoj aplikaciji sa automatskim kreiranjem ticketa, integracija sa mobilnim aplikacijama, desktop notifikacije i zasebna sekcija za sve video upute kroz aplikacije jednako kako postoji sekcija za tekstualne upute. Trenutni model funkcionira zadovoljavajuće, no pitanje i dalje stoji kada će te funkcije biti od dovoljno velike važnosti da se u njih uloži dodatne resurse tako održavajući konkurentnost i zadovoljstvo korisnika. U manjim razvojnim timovima ovaj proces inicijacije promjene dolazi intuitivno pošto su svi članovi ujedno i korisnici aplikacija te tako poznaju kada je vrijeme za promjenu zajedno sa reevaluacijom prioriteta što se od trenutnih projekata mora odgoditi kako bi se potrebna promjena u trenutnoj aplikaciji realizirala. Pošto se radi o brzorastećem poduzeću, ta redefinicija procesa i zadovoljavajuće stanje aplikacije zahtijevanja pomoći koja drži sve podatke i statistike o načinu korištenja i problemima svih aplikacija u sustavu je od visoke važnosti kako bi bili pripremljeni za budući rast, koliko god rapidan on bio. Tako se zaustavlja problem retroaktivnog djelovanja onog trenutka kad je za promjenu kasno, kada se mora žrtvovati kvalitetu izrade zbog požurenih ideja i načina njihovog realiziranja što je onda gotovo nemoguće ispraviti, ili je moguće uz neke rizike i gubitke, poput smanjenog zadovoljstva klijenata, nedostatka informacija u bazi podataka o načinu korištenja aplikacija i slično. Osim samog razvoja aplikacije, planovi se dotiču i samog procesa razvoja aplikacije, poput premještanja na druge bolje alate za projektni menadžment o čemu će biti više riječi u nastavku, pošto se radi o promjeni koja se veže za kompletan sustav aplikacija u poduzeću, ne samo Support.

4.9. Procesi razvoja u drugim Hotel's Touch aplikacijama

Support aplikacija je sustav koji je bio u nastajanju dvije godine prije pisanja ovog rada te se od tada proces razvoja promijenio kako u Supportu, tako i u svim drugim aplikacijama. Generalna ideja svih navedenih procesa u ovom poglavlju je i dalje ista - zapisuju se ideje, pa kad dođe do potrebe za promjenom se postave zahtjevi kako bi njezina funkcija trebala raditi, dizajnira se izgled funkcije, razvoj i testiranje odrade svoj posao kreiranja te funkcije, te se nova promjena postavi na server i javi zainteresiranim korisnicima kako je funkcija sad dostupna. Razlika je najvećim dijelom u upravljanju tim razvojem kroz korištenje alata poput JIRA-e za pojednostavljivanje procesa dokumentacije i praćenja samog razvoja kroz neke samostalno kreirane faze. Aplikacija Projects sa svojim jednostavnim zapisivanjem problema kako bi to bilo u jednostavnom Word dokumentu je bila dovoljno dobra do jednog trenutka, ali sa određenim rastom broja korisnika i ukupnog broja funkcija po aplikacijama, novi *feature*-i i bugovi također rastu, te je tako nastala potreba za alatom JIRA koji je specijaliziran po tom pitanju i ima predefinirane postavke za upravljanje sličnim informatičkim projektima kroz SCRUM metodu što je već dokazan način učinkovitosti upravljanjem takvog razvoja. Svaka aplikacija tako sada ima svoj projekt na JIRA-i sa svojim backlogom programerskih zahtjeva. Sa svim tim aplikacijama i zahtjevima tako je najbitnija konstantna prioritizacija zadataka, te samim time definiranje načina kroz koji se odlučuje što će biti idući prioriteti kako u kratkom razdoblju kroz idućih par sprintova, ili za dugo razdoblje. Tako se prilikom kreiranja zahtjeva u backlogu može birati o kojem se tipu zahtjeva radi:

- **Bug** - neželjeni problem u sustavu kojeg se mora riješiti.
- **Task** - manji točno definiran zadatak.
- **Story** - veći zadatak za zajedničku raspravu kako ga napraviti.

JIRA ima mnogo tipova zadataka te pravila na koji se način koriste, ali to nikad nije mjerilo stvarnosti kako se to u nekom timu prati, sve dok svi znaju na koji način nešto funkcionira i da je svima logičan način praćenja takvog procesa. Isto tako, JIRA ima mnoštvo alata kroz različite modele pretplate, ali za manji tim sama funkcija jednostavnog upisa problema u backlog, postavljanje sprintova te praćenje cijelog procesa kroz kanban ploču je sam po sebi veoma moćno sredstvo upravljanjem razvoja. Baš tako i kanban ploča kao alat sa jednostavnom funkcijom drag and drop tehnike omogućava jednostavnu organizaciju i vizualizaciju cijelog procesa svakom članu u timu gdje se neki zadatak nalazi i tko je trenutno zadužen za njegovo rješavanje. Tako, na primjer, iako svaka aplikacija ima svoj projekt, neke aplikacije imaju manji prioritet te se zadaci skupljaju u backlogu dok ih se ne skupi dovoljno za jedan

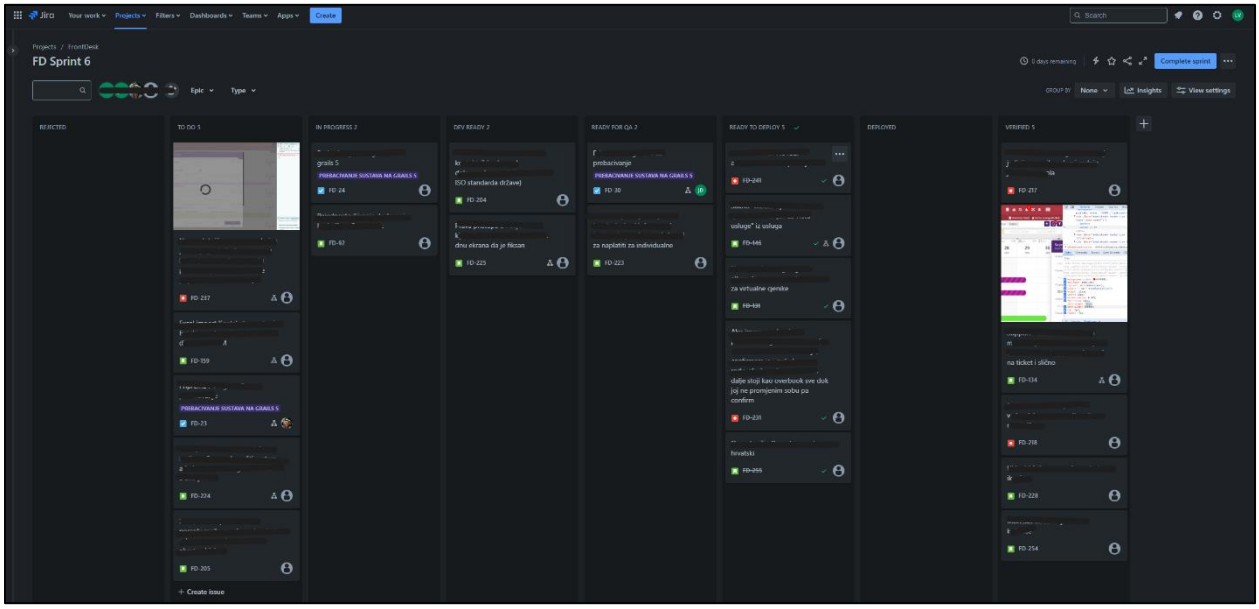
sprint ili dok nema stvarne potrebe za promjenom gdje se pokrene sprint za tu aplikaciju te se tako po potrebi reaktivira, dok kompleksniji sustavi poput Front Desk aplikacije imaju regularne dvotjedne sprinteve sa listom zadataka za odraditi taj tjedan. Pošto se radi o fleksibilnom manjem timu, sprintevi nikad nisu mjerilo što se mora napraviti u iduća dva tjedna, nego nekakva ideja što se misli da bi trebalo, te kao takva nikad nije rigidna sa rokovima da li je stvarno napravljeno što je planirano, već ono što se nije stiglo se jednostavno prebaci na idući sprint zajedno sa par novih zadataka iz backloga. Tako jednostavno korištenje sprinteva je poboljšalo produktivnost pojednostavljavajući proces definiranja kad, što i u koliko vremena nešto treba napraviti te se sad u prosjeku za tu aplikaciju riješi oko 30 zadataka u jednom mjesecu, dok je prije JIRA-e to bilo oko 15 neorganiziranih zadataka po mjesecu.

Tako svaka aplikacija ima svoju dogovorenu ideju kroz koje bi faze zadatak trebao proći dogovorom između članova tima što misle da je najbolje za sve. Za prethodno spomenutu aplikaciju Front Desk, na primjer, to izgleda na idući način:

- **To do** - zadatak iz backloga ide u *„to do“* listu unutar sprinta, što znači da ga programer treba riješiti. Ponekad se ova faza koristi i za testera za zadatke poput pronaći način reproducirati neki bug u sustavu kako bi se otkrio njegov problem te se krenulo u njegovo rješavanje.
- **In progress** - Pošto ima više programera na jednom projektu, svaki uzme neki zadatak i stavi ga u *in progress* status kako bi se znalo da na svojoj grani trenutno rješava taj zadatak.
- **Dev ready** - informacija koju programer postavi sam sebi čisto da zna da je odradio zadatak, ali da mu još stoji u development okruženju, odnosno da još nije postavio taj zadatak na testni server kako bi tester provjerio da li sve valja.
- **Ready for QA** - kad programer deploya napravljene promjene na testni server tester ga preuzima te provjerava da li su funkcionalnosti kakve trebaju biti i da li je slučajno nešto drugo prestalo poraviti zbog nove promjene. Tako ako zadatak ne prođe QA provjeru, razlog se upisuje unutar tog zadatka kao “*subtask*” ili podzadatak tog zadatka, te ga se postavlja nazad u *„to do“* status dok ponovo ne dođe u *„ready for QA“* na provjeru.
- **Ready for deploy** - ako je QA tester ocijenio da je sve u redu sa zadatkom tada ga postavlja u “*ready for deploy*” status kao informaciju programeru da je funkcija spremna da se stavi na produkcijski server. Ako je veoma hitno, taj jedan zadatak se može samostalno deployati na produkciju, no ako nije hitno, čeka se da svi zadaci dođu u “*ready for deploy*” status te se svi zajedno puštaju kroz jedan deploy.

- **Deployed** - nakon što je određeni zadatak postavljen na produkcijski server, stavlja mu se status "*deployed*".
- **Verified** - zadnji korak nakon što se zadatak postavi na produkciju je da QA testira još jednom da li je sve u redu sa tim zadatkom ili ima nekih novih bugova, pošto se baze i grane između produkcijskih servera i testnih servera mogu razlikovati, te tako neka funkcija može raditi na testnom serveru, ali ne i na produkcijskom, te se zato još jednom mora pregledati da li sve radi kako bi trebalo biti.

Slika ispod prikazuje radnu ploču jednog sprinta za Front Desk aplikaciju gdje je po stupcima prikazana svaka razvojna faza, te u svakoj fazi kartice zahtjeva koje se pomiču iz jedne faze u drugu, dok sve kartice ne završe ciklus te se sprint zatvori:



Slika 8 - Prikaz jednog sprinta unutar programa JIRA u razvoju aplikacije

Izvor: izrada autora, prema internim informacijama sa https://www.atlassian.com/software/jira

Tako se u Hotel's Touchu koriste razni alati i metode za razvoj sustava, svaki od kojih se situacijski improvizirano, ali sustavno planirano razvija kako za trenutne potrebe pojednostavljivanja nepotrebnih procesa ili koraka za cijeli tim, tako i organizirano kako bi za buduće potrebe bili pripremljeni za prilagodbu promjenama u tom cijelom procesu.

5. DISKUSIJA

U nastavku će ponovo biti navedena istraživačka pitanja zajedno sa odgovorom na svako pitanje.

- **Kako izgleda proces razvoja i ažuriranja aplikacija, tko su sudionici u tim procesima, koje su njihove funkcije, zadaci, odgovornosti i važnost u procesu te koja je važnost održavanja softvera nakon što je kreiran?**

Razvoj aplikacija je proces koji se očituje kroz neke faze koje prati tijekom tog ciklusa. Tako se nakon otkrivanja neke potrebe za promjenom na tržištu ili unutar postojeće aplikacije stvara plan koji mora pokrivati neke postavljene zahtjeve kao funkcionalnosti koje bi aplikacija trebala imati zajedno sa definiranjem resursa, vremenskih rokova i drugih ograničenja za ostvarenje tog projekta. Iduća faza je dizajn tih zahtjeva kao vizualni dizajn i kao dizajn iskustva korištenja tih funkcionalnosti gdje se stvori ideja kako bi ta cijela aplikacija trebala izgledati i funkcionirati. Tada aplikacija kreće u fazu razvoja gdje programeri zajedno sa odjelom testiranja zajedno zapravo razvijaju tu cijelu aplikaciju osiguravajući da se prate svi postavljeni zahtjevi te da se putem pronađu još neke nove potencijalne funkcionalnosti i riješe problemi sa loše definiranim ili krivo prioritiziranim postojećim zahtjevima. Kad je sve spremno, aplikacija se postavlja na produkcijski server te se korisnicima omogućava korištenje aplikacije gdje se na stvarnom slučaju počnu graditi novi zahtjevi i pronalaziti bugovi sa trenutnom verzijom sustava gdje sustav prelazi u fazu održavanja i potrebne regularne optimizacije aplikacije. U tom procesu direktni sudionici su oni koji sastavljaju projektni tim, a to su projekt menadžer kao vlasnik cijelog procesa, dizajneri i pisci sadržaja kao kreativni voditelji same privlačnosti funkcionalnosti aplikacije, programeri i testeri koji zajedno zapravo stvaraju taj cijeli sustav kroz pisanje koda te pronalazaka problema unutar tog koda, te tehnička podrška, marketing i ostali sudionici u procesu održavanja koji osiguravaju da aplikacija dođe do što većeg broja korisnika te da njihovo zadovoljstvo korištenja bude na najvišoj razini. Indirektno, i sami korisnici su sudionici procesa, jer njihove ideje i problemi sa kojima se suoče prilikom korištenja aplikacije pomažu usavršiti taj cijeli sustav ako se njihovim zahtjevima kvalitetno rukuje kroz dobru prioritizaciju svih tih ideja kroz redovno održavanje sustava kao alat za dugoročno zadovoljstvo korisnika.

- **Koje metode razvoja i ažuriranja aplikacija su najpoznatije te koje od njih pružaju najbolje rezultate?**

Prethodno navedene faze razvoja moraju pratiti neki način izvršavanja postavljenih zadataka u svakoj od njih, te za to mora postojati neka metodologija koja se prati prilikom tog procesa. Tradicionalne metode bi to ostvarivale kroz ideju da kad jedna faza završi da se ona smatra završenom do kraja procesa, a zbog korisnicima u prethodno navedenom pitanju stalno mijenjajućih potreba za redovnim održavanjem, između ostalih razloga, narasla je potreba za novom, fleksibilnijom metodom. Tako je nastala agilna metodologija čija je primjena skoro u potpunosti preuzela razvoj informatičkih sustava u zadnjih 10 godina kroz ideju inkrementalnog i iterativnog razvoja, odnosno čestog razvoja manjeg broja zahtjeva tako da idealno svakih nekoliko tjedana najprioritiziraniji zadaci prođu kroz sve faze razvoja te se postave na server za korištenje, te tako u krug. Najpoznatija agilna metoda tako je SCRUM, koju koristi poduzeće iz empirijskog dijela istraživanja koje je preseljenjem iz improvizirane vrste agilne metode prešla na SCRUM kao metodu sa određenim pravilima za najbolje rezultate te tako poduplala broj zahtjeva koji razvije u mjesec dana bez dodatnog napora za postojeći tim.

- **Koja je važnost održivosti informacijskih sustava u budućnosti, kako se implementira te iz kojih razloga se mora osigurati?**

Održavanje i održivost zajedno idu ruku uz ruku zbog potrebe da se održi poslovna stabilnost kako postojećim korisnicima, ali isto tako dugoročna održivost kroz društvenu odgovornost svim budućim generacijama kroz smanjenje potrošene energije i recikliranje postojećih resursa. Tako osim smanjenja negativnog utjecaja na okoliš kroz smanjenje energetske potrošnje i stvorenog otpada, ideja održivosti unutar informacijskih sustava poprima i ideju optimizacije pisanog koda za manje iskorištavanje energije fizičkih sustava procesuiranja i pristupa informacijama, ali i dokumentaciji kroz kvalitetno upravljanje postojećim znanjem poduzeća da se ti artefakti informacija iskoriste i u budućim projektima te tako smanje potrošenu energiju na ponovno sastavljanje tog koda. Osim smanjenja troškova kroz stvaranje više održivog informacijskog sustava, poduzeće osigurava i tržišnu, odnosno konkurentsku prednost svojim signaliziranjem djelovanja ka održivosti te osigurava jednostavnije praćenje regulatornih zahtjeva u budućnosti, zakoni kojih postaju sve stroži. Samo upravljanje razvojem kvalitetnih informacijskih sustava, posebno onih koji koriste umjetnu inteligenciju za automatiziranim provođenjem boljih odluka u energetskom, željezničkom, prometnom i drugim sektorima dovodi do veće održivosti kroz korištenje manje energije za takve strojeve i mehanizme.

- **Kako u praksi na primjeru poduzeća Hotel's Touch izgleda, prati li neki od postojećih modela, odnosno kojem modelu je najsličniji te koliko dobro funkcionira taj model razvoja i ažuriranja softvera?**

Hotel's Touch kao srednje veliko poduzeće koje je u konstantnoj komunikaciji sa vlastitim korisnicima povijesno koristi improvizirani *ad hoc* agilni pristup razvijanju novih aplikacija i ažuriranju postojećih, što se zadnjih par godina sve više oblikuje u pravu SCRUM metodu sa svojim zacrtanim pravilima zajedno sa promjenom načinom pristupa razvijanju i zapisivanju dokumentacije, sve u ideji pružanja najboljeg iskustva korisniku kroz najbrži razvoj što više funkcija bez neželjenih bugova ili zastoja u sustavu. Te promjene u procesu praćenja metode razvoja ne nastupaju korektivno zato što je otkriveno da je trenutni proces neoptimalan, već preventivno kako bi za ubuduće poduzeće bilo spremno na veće promjene i očekivani rast. To se očituje i kroz zadovoljstvo korisnika, većina od kojih su dugogodišnji korisnici koji kao najveću kvalitetu unutar sustava izlažu količinu promjena u sustavu te brzinu djelovanja na njihove zahtjeve i probleme što je najbolji dokaz funkcionalnosti tog modela razvoja. Ovdje valja još jedan put napomenuti kako je ovaj rad studija slučaja poduzeća u kojem je autor zaposlenik te tako ovi podaci mogu sadržavati određenu razinu pristranosti u interpretaciji i prezentaciji podataka, te se zato ne mogu uzeti kao generalizacija razvoja u poduzećima sličnog profila. Iz tog razloga preporuča se daljnje istraživanje na većem uzorku kako bi se dobili reprezentativniji rezultati koji bi mogli pružiti širu sliku i omogućiti općenitije zaključke.

6. ZAKLJUČAK

Projektni menadžment informatičkih sustava kao skup aktivnosti kvalitetno odabranog tima ljudi iz različitih odjela svaki sa svojim vještinama i znanjima razvija nove aplikacije, usluge i procese te poboljšava postojeće. Prilikom planiranja izrade takvog projekta mnogi propadaju zbog lošeg dizajna, neorganiziranog razvoja i lošeg upravljanja. Uz ograničene resurse i sve veću konkurenciju pitanje razvoja postaje kako optimizirati taj cijeli proces i sa najmanje troškova izgraditi najbolji sustav u što manjem vremenu. Kako bi se cijeli taj pothvat odradio, mora se poznavati što je uopće ono što sačinjava taj softverski proizvod, odnosno što se mora izgraditi stepenicu po stepenicu kako bi se taj proizvod smatrao funkcionalnim. Tako životni ciklus razvoja proizvoda (SDLC) opisuje cijeli proces, odnosno faze razvoja proizvoda koje moraju biti zadovoljene kako bi softver bio uspješan. Te faze obuhvaćaju planiranje i analizu tržišta, potreba i zahtjeva, zatim fazu dizajna funkcionalnog i vizualnog iskustva aplikacije, pa razvoj i testiranje svih prvotno navedenih zahtjeva, isporuku samog softvera te na kraju njegovo smisleno održavanje. Postoje različiti modeli SDLC-a, no izbacujući tradicionalne metode kao danas irelevantne gledajući kroz udio poduzeća koji ih još prakticiraju, agilni modeli danas se ističu kao najfleksibilniji i najpopularniji. Tako agilne metode, osobito SCRUM, naglašavaju suradnju i komunikaciju između projektnog tima i korisnika, smanjuju projektnu dokumentaciju na najvažnije informacije, uvode krajnjeg korisnika kao člana tima razvoja aplikacije te iterativno barem na mjesečnoj razini pružaju inkrementalne, odnosno male promjene u tom cijelom sustavu radi bržeg prilagođavanja i boljeg upravljanja budućim planovima i prioritetima razvoja. Razvoj web aplikacija i mobilnih aplikacija zahtjeva prilagodbu postojeće SDLC agilne metodologije, uvodeći neke specifičnosti razvoja takvih proizvoda, od kojih se najveći dio veže za načine upravljanja fazom održavanja kroz potrebna sigurnosna ažuriranja, konstantna usklađivanja sa tržištima i platformama na kojima se nalaze zajedno sa regulativama politika privatnosti GDPR-a i ostalih zakona.

To održavanje softvera ključno je i za očuvanje funkcionalnosti i stabilnosti softverskih sustava. Pravovremeno planirana modifikacija i održavanje softvera u skladu s promjenjivim potrebama korisnika i tehnološkim napretkom osiguravaju dugotrajnost i relevantnost softverskog proizvoda. Kategorije aktivnosti održavanja, kao što su savršeno, prilagodljivo, korektivno i preventivno održavanje, omogućuju kontinuirano unapređenje kvalitete, performansi, sigurnosti i fleksibilnosti softvera. Migracije na moderne tehnologije i restrukturiranje koda su neophodni za smanjenje složenosti i poboljšanje održivosti softverskih sustava. Posebna pažnja posvećuje se sigurnosnim aspektima održavanja, uključujući implementaciju sigurnosnih ažuriranja i zaštitu osjetljivih podataka.

Održavanje i održivost softvera su međusobno povezani koncepti. Održivi softver karakterizira produžen vijek trajanja, optimizacija resursa i prilagodljivost promjenama, a usvajanje održivih praksi u razvoju softvera omogućava postizanje dugoročne ekonomske, društvene i ekološke koristi. Upravljanje razvojem digitalnih tehnologija ima potencijal u budućnosti značajno smanjiti globalne emisije u mnogim sektorima pojednostavljući donošenje dobrih odluka, a uključujući ideju održivosti u taj proces razvoja samom poduzeću može smanjiti troškove, pojednostaviti prilagodbu na buduće regulative i povećati zadovoljstvo korisnika pružajući kvalitetan proizvod.

Kroz poduzeće iz studije slučaja ovog rada predočeno je upravljanje projektom razvoja jednog IT sustava za ticketing support kao dodatan alat za traženje pomoći u aplikacijama za postojeće korisnike aplikacija navedenog poduzeća. Opisane su sve faze razvojnog ciklusa kompletne aplikacije zajedno sa detaljnim razlaganjem kako je poduzeće došlo na ideju to razviti, kako je planiran cijeli projekt, koji su problemi pronađeni i otklonjeni putem, te kako nakon isporuke aplikacije poduzeće osigurava provođenje ideje redovnog održavanja, kojim alatima te iz kojeg razloga. Praćenje pravila SCUM metode za održavanje i daljnji razvoj omogućilo je poduzeću da podupla broj riješenih zahtjeva mjesečno, ali ovo se ne može smatrati kvantitativnom mjerom pošto nema javno dostupnih podataka koje bi potvrdile tu tvrdnju osim vlastitog iskustva sudjelovanja u tom projektu, što je osim objektivnosti pristupa prezentiranju informacija limit ove studije slučaja, no zato iz prve ruke kvalitativno pokazuje cjelokupni razvoj jednog informacijskog sustava i kako na pravom slučaju kroz nekoliko godina razvoja IT projekta taj sustav poprima svoj ciljani oblik.

7. LITERATURA

1. Abdul-Aziz, A., Koronios, A., Gao, J., & Suhaizan, M. (2012). Towards Effective Development of Web-based Business Applications. Journal of Internet and e-Business Studies, 1–13. https://doi.org/10.5171/2012.944600
2. Abdul-Aziz, A., Koronios, A., Gao, J., & Sulong, M. S. (2012). Association for Information Systems AIS Electronic Library (AISeL) A Methodology for the Development of Web-based Information Systems: Web Development Team Perspective Recommended Citation. http://aisel.aisnet.org/amcis2012/proceedings/SystemsAnalysis/11
3. Ahmed, A. (2022). Digital Information World: Google Play Store dropped 1 million apps, but it's nothing to worry about, [Internet], https://www.digitalinformationworld.com/2022/04/google-play-store-dropped-1-million.html# , [11.09.2022.]
4. Altynpara, E. & Rovnaya, A, (2022). Cleveroad: Agile software development methogolody: definition, types, workflows, [Internet], https://www.cleveroad.com/blog/agile-software-development/ , [13.09.2022.]
5. Amri, R., & Bellamine Ben Saoud, N. (2014). Towards a generic sustainable software model. Proceedings - 2014 4th International Conference on Advances in Computing and Communications, ICACC 2014, 231–234. https://doi.org/10.1109/ICACC.2014.62
6. Anthony, B., Majid, M. A., & Romli, A. (2017). A model for adopting sustainable practices in software based organizations. 2017 8th International Conference on Information Technology (ICIT), 26–35. https://doi.org/10.1109/ICITECH.2017.8079911
7. Aurum, A., Daneshgar, F., & Ward, J. (2008). Investigating Knowledge Management practices in software development organisations – An Australian experience. Information and Software Technology, 50(6), 511–533. https://doi.org/https://doi.org/10.1016/j.infsof.2007.05.005
8. AWS Amazon (2024). What is SDLC (Software Development Lifecycle), [Internet], https://aws.amazon.com/what-is/sdlc/ , [26.05.2024.]
9. Baa, R. (2022). Role and its Impacts of Computer Application in Management and Business. 2022 5th International Conference on Computers in Management and Business (ICCMB), 44–49. https://doi.org/10.1145/3512676.3512684
10. Bennett, K., & Rajlich, V. (2000). Software Maintenance and Evolution: a Roadmap. https://doi.org/10.1145/336512.336534

11. Bower, A. (2023). Guide to the Website Development Process, [Internet], https://www.capterra.com/resources/website-development-process/ , [24.05.2024.]
12. Çetin, E., & Durdu, P. (2018). Blended Scrum model for software development organizations. Journal of Software: Evolution and Process, 31, e2147. https://doi.org/10.1002/smr.2147
13. Clark, H. (2024). The Software Development Life Cycle (SDLC): 7 Phases and 5 Models, [Internet], https://theproductmanager.com/topics/software-development-life-cycle/ [26.05.2024.]
14. Cleland, D., & Ireland, L. (2002). Project Management: Strategic Design and Implementation.
15. Čendo Metzinger, T. & Toth, M. (2020). Metodologija istraživačkog rada za stručne studije. Velika Gorica. Veleučilište Velika Gorica., [Internet], https://www.bib.irb.hr/1058026 , [11.09.2022.]
16. eVisitor (2022). O sustavu, [Internet], https://www.evisitor.hr/info/hr-HR/ , [11.09.2022.]
17. Figma (2024). What is Wireframing?, [Internet], https://www.figma.com/resource-library/what-is-wireframing/ , [24.05.2024.]
18. Ghandi, L., Silva, C., Martinez, D., & Gualotuña, T. (2017). Mobile application development process: A practical experience. 2017 12th Iberian Conference on Information Systems and Technologies (CISTI), 1–6. https://doi.org/10.23919/CISTI.2017.7975825
19. Google, (2023). Target API level requirements for Google Play apps, [Internet], https://support.google.com/googleplay/android-developer/answer/11926878 , [24.05.2024.]
20. Gurung, G., Shah, R., & Jaiswal, D. (2020). Software Development Life Cycle Models-A Comparative Study. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 30–37. https://doi.org/10.32628/CSEIT206410
21. Hagues, A. (2022). Nintendo life: Stardew Valley Sells Over 20 Million Copies, [Internet], https://www.nintendolife.com/news/2022/05/stardew-valley-sells-over-20-million-copies , [11.09.2022.]
22. Huang, W., Li, R., Maple, C., Yang, H. J., Foskett, D., & Cleaver, V. (2010). A Novel Lifecycle Model for Web-based Application Development in Small and Medium Enterprises. International Journal of Automation and Computing, 7(3), 389–398. https://doi.org/10.1007/s11633-010-0519-3
23. Hummel, M. (2014). State-of-the-art: A systematic literature review on agile information systems development. Proceedings of the Annual Hawaii International Conference on System Sciences, 4712–4721. https://doi.org/10.1109/HICSS.2014.579
24. Huotari, P., & Ritala, P. (2021). When to switch between subscription-based and ad-sponsored business models: Strategic implications of decreasing content novelty. Journal of Business Research, 129, 14–28. https://doi.org/10.1016/j.jbusres.2021.02.037

25. IBM (2024), What is knowledge management?, [Internet], https://www.ibm.com/topics/knowledge-management , [24.05.2024.]
26. IBM (2024). Why data privacy is much more than compliance, [Internet], https://www.ibm.com/security/digital-assets/data-privacy-matters/ , [24.05.2024.]
27. IEEE Computer Society (2024). What is Software Maintenance?, [Internet], https://www.computer.org/resources/software-maintenance , [24.05.2024.]
28. International Institute for Sustainable Development (2024). Sustainable Development, [Internet], https://www.iisd.org/mission-and-goals/sustainable-development , [24.05.2024.]
29. Ionel, N. (2009). AGILE SOFTWARE DEVELOPMENT METHODOLOGIES: AN OVERVIEW OF THE CURRENT STATE OF RESEARCH. Annals of Faculty of Economics, 4, 381–385.
30. ISO (2006). Software engineering – Software life cycle processes - Maintenance [Internet], https://mireilleblayfornarino.i3s.unice.fr/lib/exe/fetch.php?media=teaching:reverse:10.1109_ieeestd.2006.235774.pdf , [24.05.2024.]
31. Jabangwe, R., Edison, H., & Duc, A. N. (2018). Software engineering process models for mobile app development: A systematic literature review. Journal of Systems and Software, 145, 98–111. https://doi.org/10.1016/j.jss.2018.08.028
32. Jeremiah, J. (2024). Survey: Is agile the new norm?, [Internet], https://techbeacon.com/app-dev-testing/survey-agile-new-norm , [26.05.2024.]
33. Jrad, R., & Sundaram, D. (2015). Inter-Organizational Information and Middleware System Projects: Success, Failure, Complexity, and Challenges.
34. KMS Solutions (2022). Mobile Application Development Lifecycle: 5 Key Phases, [Internet], https://blog.kms-solutions.asia/mobile-app-development-lifecycle , [24.05.2024.]
35. Levin, T. (2019). Business Insider: Car companies stand to make billions by charging you monthly fees for add-on features like heated seats, [Internet], https://www.businessinsider.com/car-feature-subscriptions-add-ons-bmw-ford-toyota-gm-2022-2 , [11.09.2022.]
36. Mailchimp (2024). What is SEO?, [Internet], https://mailchimp.com/marketing-glossary/seo/ , [24.05.2024.]
37. Mailchimp (2024). What is Website Maintenance and Why is it Necessary?, [Internet], https://mailchimp.com/resources/website-maintenance/ , [24.05.2024.]
38. Martins, J. (2024). Understanding the iterative process, with examples, [Internet], https://asana.com/resources/iterative-process , [26.05.2024.]

39. Nowduri, S., & DBA, S. A.-D. (2012). Management Information Systems and Its Support to Sustainable Small and Medium Enterprises. International Journal of Business and Management, 7(19). https://doi.org/10.5539/ijbm.v7n19p125
40. Priya, A. (2020). Case Study Methodology of Qualitative Research: Key Attributes and Navigating the Conundrums in Its Application. Sociological Bulletin, 70(1), 94–110. https://doi.org/10.1177/0038022920970318
41. Saeed, S., Jhanjhi, N. Z., Naqvi, M., & Humayun, M. (2019). Analysis of software development methodologies. International Journal of Computing and Digital Systems, 8(5), 445–460. https://doi.org/10.12785/ijcds/080502
42. Salesforce (2024). What is upselling?, [Internet], https://www.salesforce.com/eu/learning-centre/sales/upselling/ , [24.05.2024.]
43. Schlauderer, S., & Overhage, S. (2012). Investigating the Long-Term Acceptance of Agile Methodologies: An Empirical Study of Developer Perceptions in Scrum Projects. 2014 47th Hawaii International Conference on System Sciences, 5452–5461. https://doi.org/10.1109/HICSS.2012.387
44. SCRUM (2024). What is ScrumBut?, [Internet], https://www.scrum.org/resources/what-scrumbut , [26.05.2024.]
45. Simplified science publishing (2022). Scientific Data Visualization Techniques and Tools, [Internet], https://www.simplifiedsciencepublishing.com/resources/scientific-data-visualization-tools-and-techniques , [11.09.2022.]
46. Siteefy (2024). How Many Websites Are There in the World?, [Internet], https://siteefy.com/how-many-websites-are-there/ , [26.05.2024.]
47. STATISTA (2024). Mobile Internet & Apps, [Internet], https://www.statista.com/markets/424/topic/538/mobile-internet-apps/#overview , [26.05.2024.]
48. Subedi, B., Alsadoon, A., Prasad, P. W. C., & Elchouemi, A. (2016). Secure paradigm for web application development. 2016 15th RoEduNet Conference: Networking in Education and Research, 1–6. https://doi.org/10.1109/RoEduNet.2016.7753243
49. Torres, R. (2019). Enterprise app sprawl swells, with most apps outside of IT control, [Internet], https://www.ciodive.com/news/app-sprawl-saas-data-shadow-it-productiv/606872/ , [11.09.2022.]
50. Trumbach, C., Walsh, K., & Mahesh, S. (2022). A Historical and Bibliometric Analysis of the Development of Agile (str. 107–121). https://doi.org/10.4018/978-1-6684-3702-5.ch007
51. Uptime institute (2022). Uptime Institute's 2022 Outage Analysis Finds Downtime Costs and Consequences Worsening as Industry Efforts to Curb Outage Frequency Fall Short, [Internet],

https://uptimeinstitute.com/about-ui/press-releases/2022-outage-analysis-finds-downtime-costs-and-consequences-worsening , [11.09.2022.]

52. Vithani, T., & Anandkumar, A. (2014). Modeling the Mobile Application Development Lifecycle. https://api.semanticscholar.org/CorpusID:17577272
53. West, D. (2024). Agile scrum roles and responsibilities, [Internet], https://www.atlassian.com/agile/scrum/roles , [26.05.2024.]
54. World Economic Forum (2022). Digital solutions can reduce global emissions by up to 20%. Here's how , [Internet], https://www.weforum.org/agenda/2022/05/how-digital-solutions-can-reduce-global-emissions/ , [24.05.2024.]
55. Zabardast, E., Frattini, J., Gonzalez-Huerta, J., Mendez, D., Gorschek, T., & Wnuk, K. (2022). Assets in Software Engineering: What are they after all? Journal of Systems and Software, 193, 111485. /https://doi.org/10.1016/j.jss.2022.111485

8. POPIS SLIKA

9. SAŽETAK

Projektni menadžment informatičkih sustava predstavlja koordinaciju aktivnosti tima stručnjaka iz različitih područja kako bi se stvorili i unaprijedili softverski proizvodi. Planiranje i izvedba ovakvih projekata često nailazi na izazove poput lošeg dizajna, nedostatka organizacije i lošeg upravljanja, dodatno otežane ograničenim resursima i sve većom konkurencijom. Za uspješno upravljanje takvim informatičkim projektima ključno je razumijevanje životnog ciklusa razvoja proizvoda (SDLC) koji opisuje sve razvojne faze, uključujući planiranje, dizajn, razvoj, testiranje, isporuku i održavanje softvera. Agilni modeli razvoja tim fazama, poput SCRUM-a, postaju sve popularniji jer naglašavaju suradnju, komunikaciju i inkrementalne promjene radi bržeg prilagođavanja i boljeg upravljanja projektima. Osim razvoja, važno je upravljati i održavanjem postojećih sustava radi očuvanja funkcionalnosti i stabilnosti softvera u skladu sa tehnološkim napretkom i promjenjivim potrebama korisnika, te sve više važnih sigurnosnih ažuriranja i usklađivanja sa politikama zaštite osjetljivih podataka kao bitnih tema današnje međunarodne regulative. Proces održavanja softvera uključuje i osiguranje održivosti, odnosno razvoja društveno važne karakteristike softvera kao proizvoda sa produženim vijekom trajanja i prilagodljivosti promjenama. Održavanjem softvera zajedno sa usvajanjem održivih praksi u razvoju postižu se dugoročne ekonomske, društvene i ekološke koristi, kako za krajnjeg korisnika, tako i za cijelo društvo. Kroz metodu studije slučaja u ovom radu je opisan stvarni primjer cjelokupnog razvoja jedne aplikacije detaljno opisujući aktivnosti sve faze razvojnog ciklusa, zajedno sa pregledom prakse njezinog održavanja nakon isporuke, sve u cilju isporuke najveće koristi za korisnike.

Ključne riječi: razvoj informacijskih sustava, održavanje informacijskih sustava, održivost informacijskih sustava

10. SUMMARY

Information systems project management involves coordinating the activities of a team of experts from various fields to create and improve software products. Planning and executing such projects often encounter challenges such as poor design, lack of organization and ineffective management, further compounded by limited resources and increasing competition. Successfully managing such IT projects crucially depends on understanding the Software Development Life Cycle (SDLC), which describes all development phases including planning, design, development, testing, delivery, and maintenance of the software. Agile development models, such as SCRUM, are becoming increasingly popular as they emphasize collaboration, communication, and incremental changes for faster adaptation and better project management. In addition to development, it is important to manage the maintenance of existing systems to preserve the functionality and stability of software in line with technological advancements and changing user needs, as well as increasing security updates and compliance with sensitive data protection policies which are significant topics in today's international regulations. Software maintenance process also includes ensuring sustainability, involving the development of socially important characteristics of software as a product with extended lifespan and adaptability to changes. By maintaining software alongside adopting sustainable development practices, long-term economic, social and environmental benefits are achieved, both for end-users and society as a whole. Using the case study method, this paper describes true example of the entire development process of an application in detail, outlining the activities of each phase of the development cycle, along with a review of maintenance practices post-delivery, all aimed at maximizing benefits for users.

Keywords: information systems development, information systems maintenance, information systems sustainability